Univerza v Mariboru

Fakulteta za elektrotehniko,
računalništvo in informatiko

23.
konferenca

OTS 2018

sodobne informacijske tehnologije in storitve

Urednika:
Marjan Heričko,
Katja Kous

Univerzitetna založba
Univerze v Mariboru

Univerzitetna založba
Univerze v Mariboru

OTS 2018
Sodobne informacijske tehnologije in storitve

Zbornik triindvajsete konference

Maribor, 19. in 20. junij 2018

Urednika:
red. prof. dr. Marjan Heričko
dr. Katja Kous

Junij 2018

Naslov: OTS 2018 Sodobne informacijske tehnologije in storitve

Podnaslov: Zbornik triindvajsete konference, Maribor, 19. in 20. junij 2018

Urednika: red. prof. dr. Marjan Heričko
(Univerza v Mariboru, Fakulteta za elektrotehniko, računalništvo in informatiko)

dr. Katja Kous
(Univerza v Mariboru, Fakulteta za elektrotehniko, računalništvo in informatiko)

Tehnična urednica: asist. Lucija Brezočnik (Univerza v Mariboru, Fakulteta za elektrotehniko, računalništvo in informatiko)

Oblikovanje ovitka: asist. Lucija Brezočnik (Univerza v Mariboru, Fakulteta za elektrotehniko, računalništvo in informatiko)

Grafične priloge: Avtorji prispevkov

Konferenca: OTS 2018 23. konferenca sodobne informacijske tehnologije in storitve

Datum konference: 19. – 20. junij 2018

Kraj konference: Maribor

Izdajateljica:
Univerza v Mariboru, Fakulteta za elektrotehniko, računalništvo in informatiko
Koroška cesta 46, 2000 Maribor, Slovenija
http://feri.um.si, feri@um.si

Založnik:
Univerzitetna založba Univerze v Mariboru
Slomškov trg 15, 2000 Maribor, Slovenija
http://press.um.si, zalozba@um.si

Izdaja: Prva izdaja

Vrsta publikacije: e-publikacija

Dostopno na: http://press.um.si/index.php/ump/catalog/book/338

Izid: Maribor, junij 2018

CIP - Kataložni zapis o publikaciji
Univerzitetna knjižnica Maribor

004.946.5:004.7(082)

KONFERENCA Sodobne informacijske tehnologije in storitve (23 ; 2018 ; Maribor)
Sodobne informacijske tehnologije in storitve [Elektronski vir] : OTS 2018 : zbornik triindvajsete konference, Maribor, 19. in 20. junij 2018 / urednika Marjan Heričko in Katja Kous. - 1. izd. - Maribor : Univerzitetna založba Univerze, 2018

Način dostopa (URL): http://press.um.si/index.php/ump/catalog/book/338

ISBN 978-961-286-162-9 (pdf)

doi: 10.18690/978-961-286-162-9
1. Heričko, Marjan
COBISS.SI-ID 94690049

ISBN: 978-961-286-162-9 (PDF)
978-961-286-163-6 (Tiskana izdaja)

DOI: https://doi.org/10.18690/978-961-286-162-9

Cena: Brezplačen izvod

Odgovorna oseba založnika: red. prof. dr. Žan Jan Oplotnik, prorektor Univerze v Mariboru

http://www.ots.si

Spoštovane in spoštovani,

tudi tokrat prispevki, zbrani v zborniku že 23. strokovne konference Sodobne informacijske tehnologije in storitve, naslavljajo izjemno aktualne izzive, s katerimi se informatiki, programski inženirji, računalničarji, podatkovni znanstveniki, arhitekti, razvijalci ter upravljalci informacijskih rešitev in storitev srečujemo pri svojem vsakdanjem delu. Avtorji predstavljajo inovativne rešitve in skozi konkretne projekte pridobljene izkušnje:

- z uporabo tehnologij in platform veriženja blokov,
- oblikovanjem novih ekosistemov, poslovnih modelov in storitev ter digitalno preobrazbo,
- vpeljavo in udejanjanjem mikrostoritvenih arhitektur,
- zagotavljanjem varnosti, zaupnosti in zasebnosti ter s tem povezanimi tehnikami obvladovanja tveganj,
- popolno virtualizacijo in izkoriščanjem infrastrukture,
- posodobitvijo in nadgradnjo obstoječih informacijskih sistemov,
- integracijo algoritmov strojnega učenja in inteligentnih storitev ter platform za upravljanje in obdelavo velepodatkov,
- spletnimi komponentami in tehnologijami ter
- agilnimi pristopi, ki omogočajo hiter in učinkovit razvoj tako prototipov kot produkcijskih uporabniško usmerjenih rešitev v sklopu avtomatiziranih in neprekinjenih procesov razvoja, integracije in dostave.

Kar še posebej veseli in navdušuje, je dejstvo, da se še v preteklem letu zgolj nakazane namere in v sklopu pilotnih projektov preizkušane tehnologije in pristope, sedaj že manifestirajo v rezultatih konkretnih projektov in rešitvah, namenjenih svetovnemu trgu. Porajajoče tehnologije in pristopi pa že prinašajo nove izzive!

Zato je toliko bolj pomembno, da skozi izmenjavo spoznanj ter izkušenj skupaj oblikujemo dobre prakse in prispevamo k njihovi uveljavitvi v svojih okoljih.

dr. Katja Kous
vodja organizacijskega odbora OTS 2018

prof. dr. Marjan Heričko
predsednik 23. konference OTS 2018

KAZALO

Univerzitetna založba Univerze v Mariboru

SLEDENJE UKRADENIM BITCOINOM PO VERIGI BLOKOV

MARKO GAŠPARIČ IN BOŠTJAN VESNICER

Povzetek: Mešanje bitcoin kovancev z drugimi se pogosto uporablja za prikrivanje izvora kovancev. Pri kripto-valutah, kjer so vse transakcije javne, se ta metoda uporablja za legitimne namene, npr. za zaščito zasebnosti, in za nelegitimne namene, npr. kot pranje denarja. Razvili smo orodje za zaznavo menjav bitcoin kovancev na spletnih menjalnicah za kripto-valute in pologe ukradenih kovancev na NiceHash platformi. Orodje temelji na seznamu "pomembnih" transakcij, na katerega je transakcija uvrščena, če vsebuje dovolj "umazanih" kovancev. Analiza kovancev deluje dovolj hitro, da se seznam transakcij osveži z vsakim novim bitcoin blokom in tako omogoča sprotno preverjanje transakcij.

Ključne besede: • veriga blokov • sledenje • transakcija • bitcoin • NiceHash

NASLOVA AVTORJEV: Marko Gašparič, MaG IT d.o.o., Volkmerjeva cesta 56a, 2250 Ptuj, Slovenija, e-pošta: m.gasparic@gmail.com. Boštjan Vesnicer, H-BIT d.o.o., Obrežna ulica 3, 2312 Orehova vas, Slovenija, e-pošta: bostjan@nicehash.com.

DOI https://doi.org/10.18690/978-961-286-162-9.1 ISBN 978-961-286-163-6

Dostopno na: http://press.um.si.

1. UVOD

NiceHash[1] je spletna tržnica za trgovanje s procesno močjo, ki se uporablja za rudarjenje različnih kripto-valut. Ponudniki procesne moči so ponavadi lastniki zmogljivih grafičnih kartic ali drugih naprav, namenjenih za rudarjenje kripto-valut. Kupci procesne moči so investitorji, ki so pripravljeni vložiti določen znesek za nakup ponujene procesne moči, ki jo nato uporabijo za rudarjenje specifične kripto-valute, v želji, da se jim bo vložek obrestoval. Osnovni namen NiceHash platforme je enostavna povezava ponudnikov in kupcev, ki lahko svoja sredstva na platformi tudi hranijo. Za trgovanje na tržnici se uporablja kripto-valuta bitcoin (BTC).

Šestega decembra 2017 se je zgodil sofisticiran napad na NiceHash platformo. Med napadom je bilo ukradenih več kot 4700 bitcoinov; večina jih je bila v lasti uporabnikov NiceHasha. Z zlorabo sistema za izplačevanje so se sredstva najprej prenesla iz interne denarnice na pet različnih bitcoin naslovov. Nato pa se je majhen del teh kovancev razpršil po različnih naslovih, večji del pa je še isti dan končal na enem naslovu, ki pred napadom ni bil še nikoli uporabljen. Oznaka tega centralnega naslova je *1EnJHhq8Jq8vDuZA5-ahVh6H4t6jh1mB4rq*.

En teden po napadu, 13.12.2017, se je zgodila prva izhodna transakcija iz *1EnJHhq8Jq8vDuZA5ahVh6H-4t6jh1mB4rq*. Na naslov *1EXvoGyYsbazaejgQPFrSKu8nh2joU6mdm* je bilo prenesenih 614,24212398 BTC, na naslov *1P1bSFBeuWy7PPLk6nKEkWHx4qAc8safLi* pa točno 50 BTC. Ta transakcija ima oznako *b82dc-42650688bee5f67273adb3f224ea2c96b71ebeaf6c4411d09f3903c4d69*.

14.12.2017 smo zaznali prvo združevanje ukradenih bitcoin kovancev, ki so se nahajali na *1EnJHhq8Jq8v-DuZA5ahVh6H4t6jh1mB4rq* naslovu, z bitcoini kovanci iz drugega naslova. V transakciji z oznako *23dc3-1e528d3c4cde2e4dcd298f08b0366ee3f901f7c795782ba8baa6da15544* je bilo združenih 72,1737975 BTC iz *1EnJHhq8Jq8vDuZA5ahVh6H4t6jh1mB4rq* in 614,24212398 BTC iz *1EXvoGyYsbazaejgQPFrSKu8nh2jo-U6mdm* naslovov, nakazani pa so bili na *19pufhHX37EM2aDawtr4mp5zqiRkVEArSk* in *1FQwzBR6X-pzCmULZHZ16NUBHi5W9cHA7u2*. Tej transakciji so sledile še štiri transakcije v vrednosti 1000 BTC, izključno iz *1EnJHhq8Jq8vDuZA5ahVh6H4t6jh1mB4rq* naslova, po katerih je bil ta naslov praktično izpraznjen; 22.12.2017 je *1EnJHhq8Jq8vDuZA5ahVh6H4t6jh1mB4rq* vseboval manj kot 0,0002 BTC. Preostanek je bil porabljen v štirih transakcijah, ki so se zgodile leta 2018 in so vsebovale tudi naslove, ki so pred tem že prejeli večje količine ukradenih bitcoinov, tudi preko *1EnJHhq8Jq8vDuZA5ahVh6-H4t6jh1mB4rq*. Z vidika sledenja bitcoinov po naslovih je to ustvarilo cikle, ki močno otežijo sledenje.

Ko so ukradeni bitcoini zapustili centralni naslov, so se razpršili, napadalci pa so jih na različne načine združili z drugimi bitcoini ali zamenjali za bitcoine, ki niso direktno povezani z NiceHash denarnico. Tej metodi se reče "mešanje", pogosto pa se uporablja za prikrivanje izvora kovancev. Na spletu obstajajo tudi ponudniki storitev, ki opravljajo mešanje kripto-valut v zameno za plačilo [1]. Pri kripto-valutah, kjer so vse transakcije javne, se ta metoda uporablja za legitimne namene, npr. za zaščito zasebnosti, in za nelegitimne namene, npr. kot pranje denarja. Do določene mere poteka mešanje tudi spontano: kot je razvidno iz zgornjih opisov transakcij, lahko transakcije v verigi blokov vsebujejo izhode več transakcij, ki se lahko razdelijo po več naslovih; vse to pa znatno oteži sledenje.

Kljub vsemu je za okradeno podjetje vzpostavitev ali uporaba obstoječega sistema za sledenje bitcoinom obvezna. Prvi razlog je dejstvo, da je pri kraji bitcoinov najbolj verjetno, da se bo ugotovilo identiteto roparja pri izplačilu. Drugi razlog je nujnost detekcije uporabe ukradenih kovancev na lastni platformi. Čeprav lastnik naslova ne more preprečiti nakazila na svoj naslov, je potrebno takšne dogodke nadzorovati, saj bi v nasprotnem primeru lahko stranke domnevale, da se napad v resnici ni zgodil oz. je podjetje samo ukradlo kovance, kar bi očrnilo ugled podjetja.

[1] https://www.nicehash.com

V tem prispevku bomo najprej predstavili kako izgleda pošiljanje in mešanje bitcoinov. Nato bomo opisali problem sledenja bitcoinom po verigi blokov in predstavili orodje, ki smo ga razvili za zaznavo menjav kovancev na spletnih menjalnicah za kripto-valute in pologe ukradenih kovancev na NiceHash platformi. To orodje temelji na seznamu "pomembnih" transakcij, na katerega je transakcija uvrščena, če vsebuje dovolj "umazanih" kovancev. Analiza deluje dovolj hitro, da se seznam transakcij osveži z vsakim novim bitcoin blokom in tako omogoča sprotno preverjanje transakcij, ki prihajajo na naš sistem. Možno je tudi dokaj hitro preverjanje poljubnih transakcij, npr. takšnih, ki bi lahko bile izvedene na menjalnicah za kripto-valute. V zaključku prispevka bomo primerjali naš pristop z določenimi obstoječimi orodji in metodami.

2. POŠILJANJE IN MEŠANJE BITCOINOV

Bitcoin temelji na tehnologiji veriženja blokov. Vsak prenos kovancev je predstavljen z unikatno transakcijo, ki je vključena v blok. Če blok z določeno transakcijo ni vključen v javno objavljeno in splošno sprejeto verigo blokov, se smatra, da ta transakcija ni bila izvedena. Zato mora biti blok, ki vsebuje množico transakcij, vedno povezan s predhodnim blokom. V povprečju je nov bitcoin blok ustvarjen vsakih deset minut. Vsi bloki in vse transakcije pa so kriptografsko zaščiteni.

V tabeli št. 1 je predstavljen primer transakcije, kjer smo izvorni heksadecimalni zapis pretvorili v berljivo obliko s pomočjo Blockchain.info[2]. Kot primer je uporabljena transakcija z oznako *3b5e91d24baf8a91b-f2f6af5c6a2db9f3307158d445cbabb7996072beb7fd91e*. Ta transakcija vsebuje prenos zadnjih 0,0000235 BTC iz *1EnJHhq8Jq8vDuZA5ahVh6H4t6jh1mB4rq* naslova, ki so bili pomešani s 499,984557 BTC iz *1HJWcM9o5D8aRSbeeAZ5G5E1f9gToATTFn* naslova. Od skupno 499,9845805 BTC je bilo 100 BTC nakazanih na naslov z oznako *1GqmHHy2uF3cZ6gHshuFSFZJeo9binA1AR*, preostalih 399,98394214 BTC pa na naslov z oznako *1qbfx8J9xnhpMA6FbRmny86UWKkgL2xX9*.

Tabela 1. Primer transakcije v izvorni obliki

{"lock_time":0,"size":520,
"inputs":[

{"prev_out":{"index":0,"hash":"5a49179cfc9f277de5b6abc51a71e24140131b757321ee4a6de8a06ff4b1e20c"
},
"script":"4730440220230896b58a27985a1058159255376e219076dfcd9d906b8db802a2845383b29402200a9e
3fef1ec4e95cb72170348d33c8a12dabd00daab9d9bfd4a2d549cfff7f56012103d15bc50e9a1309642f1673619c4
60b0b6fbb716940d2bd6c8c15e363f80feea3"},

{"prev_out":{"index":0,"hash":"66ed56f516bd6c40dd1df751b3044de4a21164a4bfde37a3112700c9298dbfd0
"},
"script":"473044022063a46df0e614b41d8937af821c8b883bf55388343fa8d20ea2d2fb5778c0e1d5022061d7fd
6a1a9d11f6e7818fe06af80726dce494a0ec14338c6b20f5dae2625696012103d15bc50e9a1309642f1673619c46
0b0b6fbb716940d2bd6c8c15e363f80feea3"},

{"prev_out":{"index":1,"hash":"e5cfd70c1134cb7bc0eaec6a9b7b112ac33cabb70de46a42de08b1ce737f274c"
},
"script":"483045022100b49d23b8fd59749d9f3370489c61aeefae5aa815aacd506483526f644e04be2902205f3e
57e4b3760b8d844b6ac6d254460b866e673d3e0c996f3cb1a64ae9b76e6d012103c0d8a5b75364981b87842505
3e57060e512aebd0499913621c34c5d8547a50f2"}],
"version":1,
"vin_sz":3,
"hash":"3b5e91d24baf8a91bf2f6af5c6a2db9f3307158d445cbabb7996072beb7fd91e",

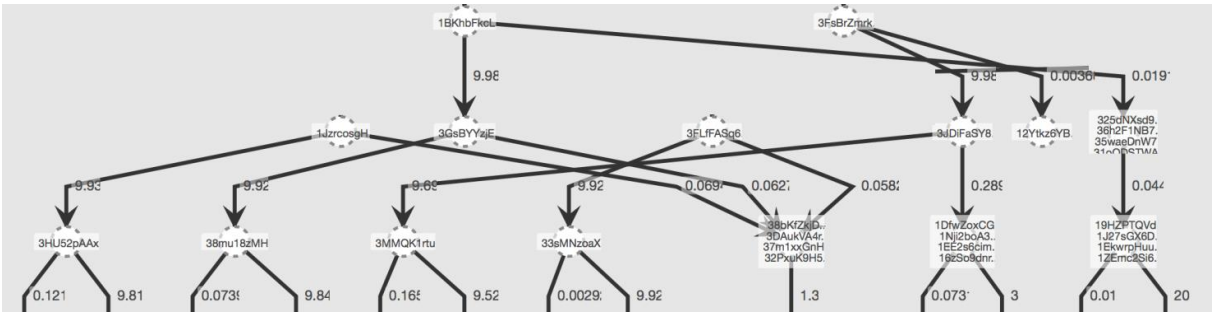
[2] https://blockchain.info/decode-tx

"vout_sz":2,
"out":[
{"script_string":"OP_DUP OP_HASH160 adc2d1173f363b6eaad2a50bc827a26a58b0399f OP_EQUALVERIFY OP_CHECKSIG",
"address":"1GqmHHy2uF3cZ6gHshuFSFZJeo9binA1AR","value":10000000000,
"script":"76a914adc2d1173f363b6eaad2a50bc827a26a58b0399f88ac"},
{"script_string":"OP_DUP OP_HASH160 0930e8952b3dbaa306f3363cc552bdc825ff93ff OP_EQUALVERIFY OP_CHECKSIG",
"address":"1qbfx8J9xnhpMA6FbRmny86UWKkgL2xX9","value":39998394214,
"script":"76a9140930e8952b3dbaa306f3363cc552bdc825ff93ff88ac"}]}

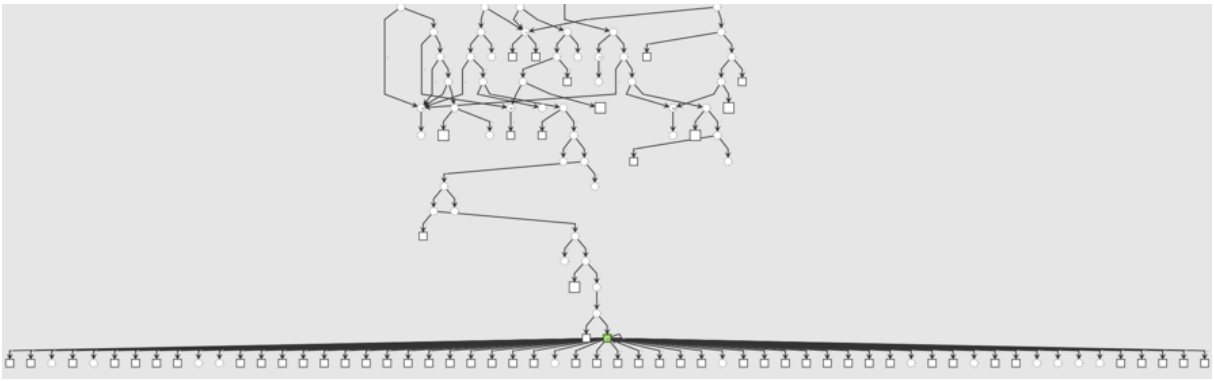
Kot je razvidno iz primera, ima dejanska transakcija tri vhodne elemente (glej: "prev_out"). To je zato, ker vhodni elementi transakcij niso kovanci, ki se nahajajo na določenem naslovu, temveč izhodi izbranih predhodnih transakcij. Bitcoin kovanec je v resnici metafora za prenos vrednosti, sam koncept pa v verigi blokov ni udejanjen. V bistvu je naslov *1EnJHhq8Jq8vDuZA5ahVh6H4t6jh1mB4rq* prvi izhod ("index":0) transakcije ("hash") *5a49179cfc9f277de5b6abc51a71e24140131b757321ee4a6de8a06ff4b1e20c*. Takrat je ta naslov prejel 786 satoshijev oz. 0,00000786 BTC. Ista vrednost se je zdaj prenesla naprej; skupaj s 1564 satoshiji, ki so prvi izhod transakcije z oznako *66ed56f516bd6c40dd1df751b3044de4a21164a4bfde37a311-2700c9298dbfd0*, in s 49.998.455.700 satoshiji, ki so drugi izhod transakcije z oznako *e5cfd70c1134cb7-bc0eaec6a9b7b112ac33cabb70de46a42de08b1ce737f274c*.

Iz primera v tabeli št.1 se tudi vidi, da je naslov *1GqmHHy2uF3cZ6gHshuFSFZJeo9binA1AR* prvi izhod transakcije *3b5e91d24baf8a91bf2f6af5c6a2db9f3307158d445cbabb7996072beb7fd91e*, njen drugi izhod pa je naslov *1qbfx8J9xnhpMA6FbRmny86UWKkgL2xX9* (glej: "out").

Mešanje bitcoinov je metoda, ki se uporablja za prekinitev povezave med resničnim pošiljateljem bitcoinov in resničnim prejemnikom. Pri tej metodi se kovanci pošiljajo na različne naslove, ki so na novo ustvarjeni ali že obstoječi, hkrati pa se jim dodajajo kovanci iz drugih virov. Ta postopek traja tako dolgo, da se sled zabriše in končni prejemnik dobi na svoj bitcoin naslov kovance, za katere je nemogoče ugotoviti natančen izvor. Kako rezultat mešanja zgleda od blizu, je prikazano na sliki št. 1. Kako mešanje zgleda od daleč, je prikazano na sliki št. 2.



Slika 1. Primer mešanja bitcoinov od blizu



Slika 2. Primer mešanja bitcoinov od daleč

V prvem primeru so v obliki krogov in s črtkano obrobo prikazani samo naslovi in množice naslovov ter povezave med njimi. V drugem primeru so v obliki kvadratov in s polno obrobo prikazane tudi transakcije, ki še niso razširjene, saj bi bil sicer graf prevelik. Na dnu slike št. 2 se vidi kaj se zgodi v primeru, da transakcija vključuje naslov, ki je pogosto uporabljan, npr. naslov menjalnice za kripto-valute. Za ponazoritev naj omenimo, da smo samo dva tedna po napadu na NiceHash zaznali 3,139,890 naslovov, ki bi lahko prejeli vsaj en ukraden satoshi.

3. OPIS PROBLEMA SLEDENJA

Prav ogromno število možnih naslovov in vhodnih transakcij je največja težava sledenja bitcoinom po verigi blokov. Teoretično lahko to število raste eksponentno, v praksi pa je omejeno s hitrostjo izvajanja nakazil. Končno število transakcij, ki se bodo zgodile pred dejanskim izplačilom, je predvsem odvisno od taktike napadalcev in od tega koliko sredstev so pripravljeni vložiti v zakrivanje sledi, saj transakcije niso brezplačne.

Implementacija naše rešitve za sledenje bitcoinom mora biti predvsem hitra, saj drugače sprotno preverjanje pologov na NiceHash platformi ni mogoče. Osnovni pogoj je, da se ena transakcija preveri v roku nekaj milisekund. Na podlagi te analize se potem sistem odloči ali bo dovolil uporabo naloženih kovancev ali ne. Drugi pogoj je, da v kolikor je potrebna analiza novega bitcoin bloka, ta ne sme trajati več kot 10 minut, saj bi se v nasprotnem primeru bloki dodajali hitreje kot bi jih analizirali, torej bi se zaostanek povečeval.

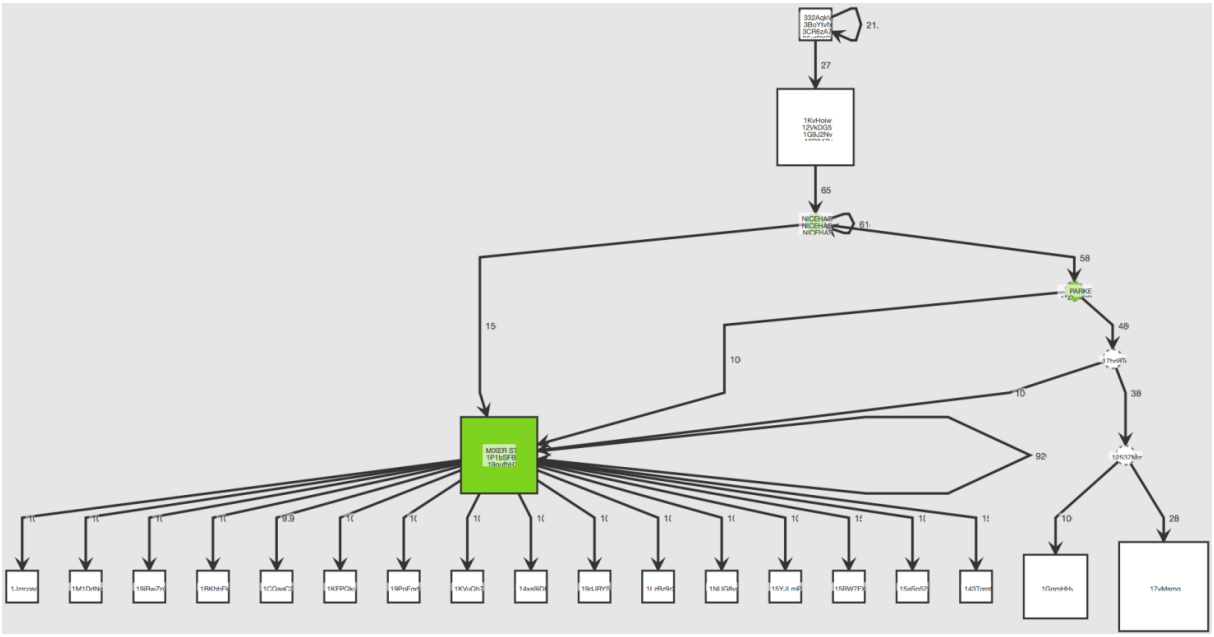
Ker se določen izhod bitcoin transakcije lahko uporabi samo enkrat kot vhod v drugo transakcijo, je struktura transakcij dejansko usmerjen acikličen graf. Sledenje prenosom ukradenih sredstev po takšni strukturi je precej enostavnejše in hitrejše kot bi bila navigacija po naslovih. Slabost takšnega pristopa je, da v primeru, ko se lastništvo bitcoinov na določenem naslovu zamenja, se tudi sled za roparji izgubi, saj sistem prenosom, ki jih izvajajo roparji ne sledi več; namesto tega pa sledi prenosom, ki jih izvajajo novi lastniki ukradenih kovancev.

4. PREDLAGANA REŠITEV ZA SLEDENJE

Po našem mnenju obstajata dva osnovna, naivna pristopa k analizi izvora kovancev, če za sledenje uporabimo graf transakcij. Pri prvem pristopu opazujemo transakcije, ki izhajajo iz začetne množice in shranimo seznam končnih transakcij. V tem primeru se po grafu, oz. po verigi blokov, premikamo "naprej". Izhodiščni seznam bi vseboval vse transakcije, ki so bile izvedene med krajo, seznam končnih transakcij pa bi se osvežil z vsakim novim blokom. Ko bi želeli preveriti ali določena nova transakcija vsebuje ukradene kovance, bi samo pogledali ali so njeni vhodni elementi že na našem seznamu. Npr., če opazujemo transakcijo *ccee1babda611f244aafafe-3b9efd9cb2d8fa9633fea012697d25dc7b1f3b3cd,* ki je ena od začetnih transakcij, potem v določenem trenutku njen celoten graf izgledal tako kot prikazuje slika št. 3, končne transakcije pa so prikazane na dnu slike.

Pri drugem pristopu je izhodiščna točka transakcija, ki jo analiziramo, zanimajo pa nas vhodi v to transakcijo. Po grafu transakcij, oz. po verigi blokov, bi se premikali "nazaj" in sproti preverjali, če je katera od vhodnih transakcij bila izvedena med krajo ali če smo že prispeli do transakcije, ki se je zgodila pred vdorom.

Za prvi pristop je potrebno veliko prostora za hrambo seznama, s povečevanjem števila transakcij, ki so na seznamu, pa se povečuje tudi čas potreben za analizo. Drugi pristop ni prostorsko zahteven, vendar je izjemno počasen. Ker je pri pologih potrebno hitro analizirati vsako vhodno transakcijo, je bila za nas edina možnost nadgradnja prvega pristopa.



Slika 3. Primer prostorsko intenzivnega naivnega pristopa k sledenju bitcoinov

Kot smo že omenili, število naslovov, po katerih se prenašajo ukradeni NiceHash kovanci, narašča zelo hitro. Število vseh transakcij, ki vključujejo vsaj kakšen ukraden satoshi, pa raste še hitreje. Posledično smo morali sprejeti kompromis, da je določena transakcija uvrščena na seznam "pomembnih" transakcij le v primeru, da vsebuje dovolj "umazanih" kovancev. Koncept "umazanih" kovancev predstavlja vsoto vhodnih vrednosti transakcij, ki so že na seznamu "pomembnih" transakcij. V primeru, da je delež teh kovancev v novi transakciji dovolj visok, je tudi ta transakcija uvrščena na seznam, vsi njeni izhodi - vključno s "čistimi" kovanci, ki so bili primešani k "umazanim" - pa se smatrajo kot "umazani". Npr., če bi bil prag za uvrstitev transakcije na seznam določen pri 10% "umazanih" kovancev in transakcija *e5cfd70c1134cb7bc0eaec6a9b7-b112ac33cabb70de46a42de08b1ce737f274c* ne bi bila na seznamu, potem tudi transakcija *3b5e91-d24baf8a91bf2f6af5c6a2db9f3307158d445cbabb7996072beb7fd91e* ne bi bila uvrščena na seznam (glej: tabela št. 1 in sekcija št. 2). Glede na to, da transakcija *e5cfd70c1134cb7bc0eaec6a9b7b112ac33-cabb70de46a42de08b1ce737f274c* je na seznamu, saj v resnici vsebuje veliko količino ukradenih kovancev, poleg tega pa sta na seznamu tudi drugi dve vhodni transakciji, bo posledično tudi *3b5e91d24baf8a91-bf2f6af5c6a2db9f3307158d445cbabb7996072beb7fd91e* uvrščena na seznam "pomembnih" transakcij.[3] Moser et al. ta pristop k označevanju transakcij imenujejo "strup" [2].

Manjšanje deležev "umazanih" kovancev v transakcijah je povezano z večjimi stroški za napadalce. Tudi zamenjava večjih količin kovancev ni enostavna, saj je bilo število ukradenih bitcoinov zelo veliko.

[3] V tem članku ne moremo razkriti dejanskega deleža "umazanih" kovancev, ki se uporablja kot prag za uvrstitev transakcije na seznam "pomembnih" transakcij.

Določitev praga za uvrstitev transakcije na seznam nam je omogočila fokusiranje na pomembnejše transakcije in zmanjšanje velikosti seznama iz nekaj milijonov zapisov na nekaj tisoč. Takšen seznam je dovolj majhen, da je iskanje po njem hitro. Preverjanje, če je določena transakcija že na seznamu, traja le nekaj milisekund, osvežitev seznama, ko je nov blok priključen verigi blokov, pa je tudi dovolj hitra, da je aktualni seznam vedno ažuren. Preverjanje, katere transakcije na seznamu predstavljajo pologe na kripto-menjalnicah ali podobnih platformah, z uporabo javno dostopnih spletnih storitev, kot je na primer WalletExplorer[4], traja le nekaj minut.

5. ZAKLJUČEK

V tem prispevku smo predstavili potrebo po uporabi ali razvoju orodja za sledenje bitcoinom po verigi blokov. Opisali smo, kako poteka pošiljanje in mešanje bitcoinov in kakšne so zahteve za analizo transakcij na NiceHash tržnici. Nazadnje smo predstavili našo rešitev, ki je tudi implementirana, v obliki spletne storitve, in omogoča sprotno preverjanje depozitov ter hitro analizo poljubnih transakcij.
Hitrost analize je dosežena na račun njene natančnosti. Pri poglobljenih analizah gibanja kovancev se ponavadi uporabljajo metode, ki grozdijo naslove, ki se uporabljajo pri prenosih opazovanih kovancev [3][4]. Te metode so bolj natančne, vendar tudi prepočasne za potrebe NiceHasha. Pri našem pristopu se zanašamo na hevristiko označevanja transakcij, ki se imenuje "strup". Temelji na predpostavki, da se lahko označi vse izhode določene transakcije kot "umazane", se pravi kot predstavnike ukradenih kovancev, če je takšen tudi dovolj velik delež vhodov v transakcijo. Obstajajo tudi drugačne hevristike, npr. "prvi-noter-prvi-ven" [5], kjer se opazuje pologe in dvige kovancev iz določenega bitcoin naslova, kot ukradene pa se vedno označi točno določen del kovancev, odvisno od tega kdaj se je zgodil ustrezen polog in kdaj ustrezen dvig. V kolikor se bodo v prihodnosti določila natančna pravila označevanja kovancev, nameravamo naš sistem ustrezno prilagoditi.

6. LITERATURA

[1] MOSER Malte, BOHME Rainer, BREUKER Dominic "Inquiry into Money Laundering Tools in the Bitcoin Ecosystem", eCrime Researchers Summit 2013

[2] MOSER Malte, BOHME Rainer, BREUKER Dominic "Towards Risk Scoring of Bitcoin Transactions", International Conference on Financial Cryptography and Data Security 2014

[3] ERMILOV Dmitry, PANOV Maxim, YANOVICH Yury "Automatic Bitcoin Address Clustering", IEEE International Conference on Machine Learning and Applications 2017

[4] HARRIGAN Martin, FRETTER Christoph "The Unreasonable Effectiveness of Address Clustering", IEEE Conferences on Ubiquitous Intelligence & Computing, Advanced and Trusted Computing, Scalable Computing and Communications, Cloud and Big Data Computing, Internet of People, and Smart World Congress 2016

[5] ANDERSON Ross, SHUMAILOV Ilia, AHMED Mansoor, Making Bitcoin Legal, 2018

[4] https://www.walletexplorer.com

Univerzitetna založba Univerze v Mariboru

IMPLEMENTACIJA NADGRADLJIVOSTI IN ZAMENLJIVOSTI PAMETNIH POGODB NA PLATFORMI ETHEREUM

BLAŽ PODGORELEC IN MUHAMED TURKANOVIĆ

Povzetek: Pametne pogodbe se trenutno nahajajo v fazi zasnove koncepta, kar proži veliko zanimanja javnosti. Tehnologija veriženja blokov je omogočila oživitev in razvoj pametnih pogodb, ki sedaj predstavljajo dodano vrednost tej tehnologiji. Zaradi omenjene zgodnje faze razvoja pametnih pogodb se z njimi povezani vzorci in dobre prakse šele oblikujejo. Za pametne pogodbe, nameščene v omrežje Ethereum, velja, da jih je zaradi tehnologije veriženja blokov nemogoče naknadno spreminjati, kar lahko v primeru napačne implementacije privede do nepredvidenih anomalij v delovanju le teh. S tem se odpirajo vprašanja, kako je z nadgradnjo ali spremembo že definiranega poslovnega procesa. Namen prispevka je predstaviti arhitekturo in s tem predlog dobre prakse razvoja pametnih pogodb, ki omogoča učinkovito zamenljivost in nadgradljivost že nameščenih pametnih pogodb v omrežje verig blokov Ethereum. Predstavili bomo modelirane koncepte in podali primere v visokonivojskem programskem jeziku Solidity.

Ključne besede: • verige blokov • pametne pogodbe • Ethereum • nadgradljivost • zamenljivost

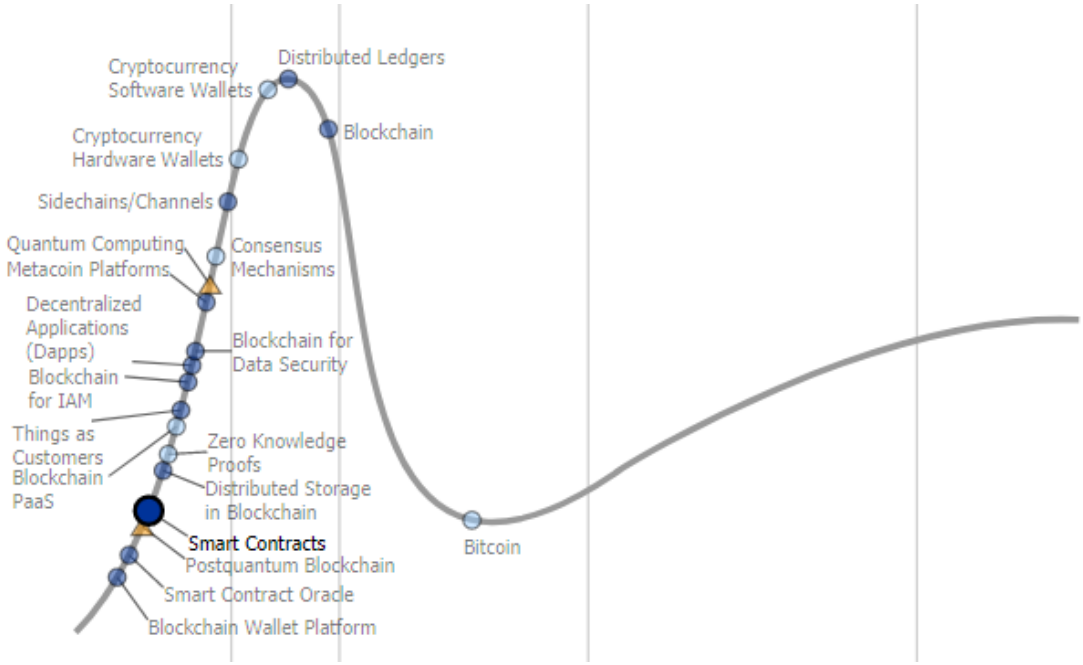
NASLOVA AVTORJEV: Blaž Podgorelec, Univerza v Mariboru, Fakulteta za elektrotehniko, računalništvo in informatiko, Koroška cesta 46, 2000 Maribor, Slovenija, e-pošta: blaz.podgorelec@um.si. dr. Muhamed Turkanović, docent, Univerza v Mariboru, Fakulteta za elektrotehniko, računalništvo in informatiko, Koroška cesta 46, 2000 Maribor, Slovenija, e-pošta: muhamed.turkanovic@um.si.

DOI https://doi.org/10.18690/978-961-286-162-9.2 ISBN 978-961-286-163-6

Dostopno na: http://press.um.si.

1. UVOD

Pametne pogodbe (angl. Smart Contracts) se glede na poročilo Gartnerjevega grafa navdušenja za tehnologijo veriženja blokov (angl. Hype Cycle for Blockchain Technologies) iz leta 2017, prikazanem na Sliki 1, nahajajo v fazi zasnove koncepta, kar proži veliko zanimanja javnosti.

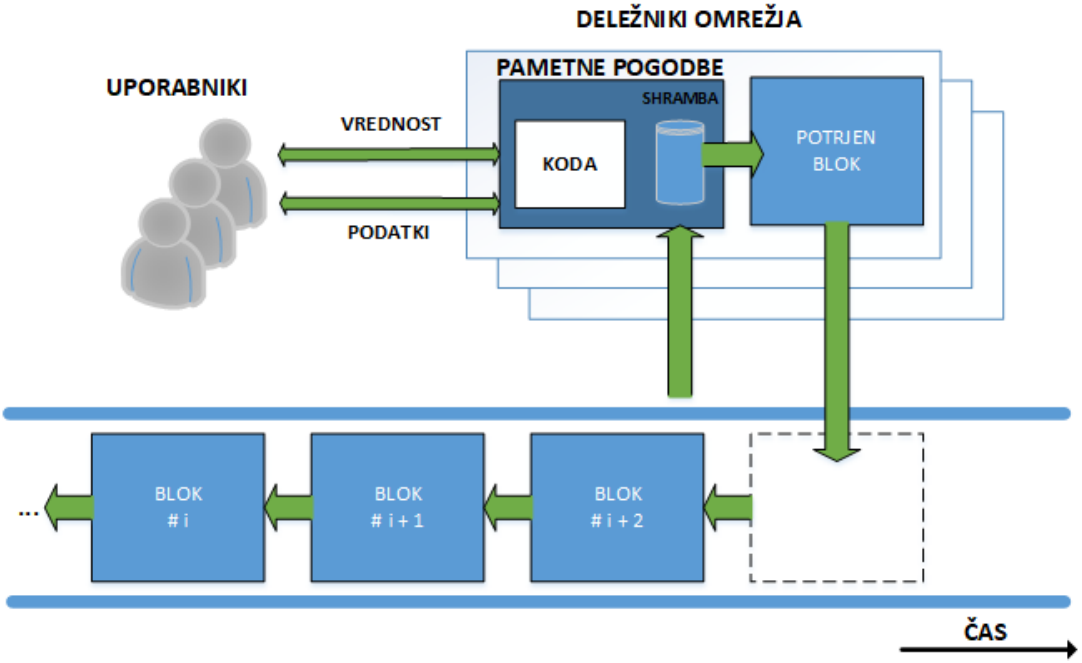


Slika 1: Gartnerjev graf navdušenja za tehnologijo veriženja blokov [1].

Osnovni koncepti pametnih pogodb (v nadaljevanju PP) so bili predstavljeni že leta 1997 s strani Nick Szaboja [2]. Definiral jih je kot računalniško transakcijski protokol, ki avtomatizirano izvršuje pogodbene pogoje in zmanjšuje potrebo po posrednikih, ki imajo vlogo izvrševalca pravil pogodb [3]. Predlagani koncepti zaradi neobstoja ustrezne tehnologije v preteklosti niso imeli možnosti ustreznega razvoja. Vzpostavitev soglasja med neznanimi deležniki, ki ga ponuja tehnologija veriženja blokov (angl. Blockchain), je omogočila oživitev in razvoj že predlaganih konceptov, ki sedaj predstavljajo dodano vrednost tej tehnologiji. Tako je leta 2013, za razliko od platforme verig blokov Bitcoin, platforma Ethereum ponudila možnost razvoja PP na verigi blokov omejeno zgolj s Turingovo polnostjo. Omenjena platforma ni edina, ki ponuja možnost razvoja in uporabe PP na verigah blokov, vendar še danes velja za najbolj priljubljeno in uporabljeno platformo za razvoj le teh [4]. Pojmovanje »pametna pogodba« je lahko zavajajoče, saj je programska logika PP predpisujoča in nima nikakršnih lastnosti umetne inteligence. Zaradi že omenjene zgodnje faze razvoja PP so izoblikovani vzorci dobrih praks in arhitektur decentraliziranih aplikacij, ki temeljijo na PP, zelo okrnjeni. Posledica tega je, da razvijalci PP prevzemajo veliko odgovornost, ki zahteva skoraj ničelno toleranco do napak v sami implementaciji PP. Odgovornost je toliko večja zaradi domorodne kriptovalute Ether, katere skupna vrednost je več milijard dolarjev [4]. V nadaljevanju bomo predstavili osnovne gradnike tehnologije veriženja blokov, pametne pogodbe in platformo Ethereum, kot najbolj uporabljeno platformo za razvoj PP na verigah blokov [4]. Predstavili bomo izzive zamenljivosti in nadgradljivosti PP platforme Ethereum. Za konec bomo kot osrednji del prispevka predstavili predlog koncepta dobrih praks razvoja PP z namenom učinkovitega doseganja zamenljivosti in nadgradljivosti že nameščenih PP v omrežje Ethereum.

2. PAMETNE POGODBE NA VERIGAH BLOKOV

Tehnologija veriženja blokov omogoča vzpostavitev decentraliziranega in porazdeljenega omrežja vozlišč, ki hranijo vse transakcije, izvedene znotraj omrežja, v med seboj povezanih blokih, ki jih poimenujemo *glavna knjiga* (ang. ledger) [5]. Doseganje varnosti s pomočjo kriptografskih mehanizmov je tisto, kar ob prenosu izvajanja PP znotraj omrežja, temelječega na tehnologiji veriženja blokov, omogoča uporabo PP znotraj porazdeljenega in decentraliziranega okolja [3]. Kljub nepoznavanju nasprotne strani lahko zaupamo transakciji, ki izvira z druge strani, saj je le ta podpisana s pomočjo njenega zasebnega kriptografskega ključa. Z vidika tehnologije veriženja blokov pomemben gradnik za doseganje enotnosti deležnikov omrežja v porazdeljenem okolju predstavljajo algoritmi soglasja. Vozlišča s pomočjo algoritmov porazdeljenega soglasja potrjujejo nove, še nepotrjene transakcije ter tako ustvarjajo kronološki vrstni red ustvarjenih transakcij in podatkov v obliki medsebojno povezanih blokov [5]. Omogočanje oddaljenega decentraliziranega dogovora, zagotavljanje varnosti v porazdeljenem okolju in povečevanje učinkovitosti z avtomatizacijo procesov so poglavitne prednosti PP. Cilj PP je doseganje vnaprej programljivih transakcij, ki manipulirajo z vrednostmi ali stanji znotraj le teh [3]. Pametne pogodbe, ki so zasnovane na principih programljivosti ter kriptografskih varnostnih mehanizmih, tako nudijo možnost prenosa izvajalske in pogojne logike realnega problema v avtomatizirano obliko, ki se bo izvedla ob točno definiranih pogojih. Pametne pogodbe na verigi blokov si lahko predstavljamo kot instanco računalniškega programa, ki se izvaja na vseh vozliščih znotraj omrežja verig blokov, kot prikazuje Slika 2 [6].



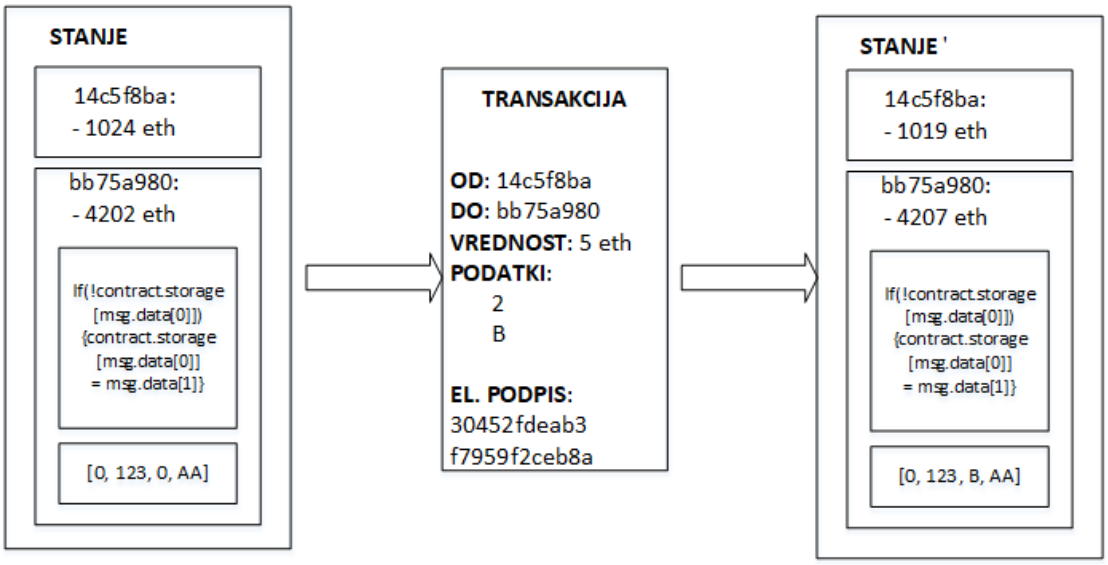
Slika 2: Shema decentraliziranega sistema in pametnih pogodb [6].

Slika 2 prikazuje interakcijo zunaj-lastniškega računa s PP, nameščeno znotraj omrežja verige blokov. Klic funkcije, definirane znotraj PP, lahko rezultira v premik določenih sredstev domorodne kripto valute Ether v ali izven PP in izvrševanje računskih operacij glede na vhodne parametre funkcij [7].

2.1. PLATFORMA ETHEREUM

Ethereum je porazdeljena računalniška platforma, ki uporablja tehnologijo veriženja blokov za shranjevanje, ne samo stanja zunaj-lastniških računov (angl. External Owned Accounts), ampak tudi stanj, povezanih s posebno vrsto pogodbenih računov (angl. Contract Accounts). Platformo Ethereum

si lahko predstavljamo kot transakcijsko temelječ stroj stanj, ki svoje delovanje začne z izvornim stanjem, ki ga s postopnim izvajanjem transakcij spreminja v neko končno stanje. Iz tega razloga in zaradi porazdeljenosti sistema se omrežje Ethereum imenuje tudi svetovni računalnik [8]. Pametne pogodbe na platformi Ethereum vsebujejo trenutno vrednost stanja računa v domorodni kripto valuti Ether, shrambo podatkov in programsko kodo. Programska koda omogoča branje in pisanje v shrambo računa, pošiljanje sporočil in ustvarjanje novih PP [9]. Vsa vozlišča znotraj omrežja Ethereum poganjajo Ethereum virtualni stroj (angl. Ethereum Virtual Machine – EVM), ki služi kot izvajalno okolje PP na platformi Ethereum. EVM omogoča izvajanje PP, ki so skupek spremenljivk in funkcij. PP so najbolj pogosto kodirane z visokonivojskim programskim jezikom Solidity [10] [3]. Klici funkcij PP, ki spreminjajo stanje v verigi blokov, se prožijo s pomočjo transakcij, ki se procesirajo znotraj EVM. Za procesiranje takšnih funkcij PP vozlišča v omrežju porabljajo določeno računsko moč, ki je odvisna od zahtevnosti definiranih operacij znotraj funkcije PP. Izvajanje takih funkcij je potrebno plačati s t.i. gorivom (angl. gas), katero ima znotraj omrežja definirano konverzno vrednost z Ethereum domorodno kripto valuto Ether. Klic funkcije znotraj PP, ki ne spreminja stanja v omrežju verige blokov, ampak po stanju samo poizveduje, se izvede na najbližjem vozlišču v omrežju in za procesiranje ne zahteva plačila.



Slika 3: Primer delovanje funkcije tranzicije stanja [11].

Slika 3 prikazuje vpliv klica transakcije na spremembo stanja, zapisanega na omrežju verige blokov z delovanjem funkcije tranzicije stanja. Po izvedbi transakcije se z računa z naslovom *»14x5f8ba«* na račun *»bb75a980«* prenese vrednost pet Etherjev. Transakcija hkrati vsebuje dva podatka »2« in »B«, ki prožita izvedbo programske kode, definirane v PP z naslovom *»bb75a980«*. Programska koda preveri, ali je mesto z indeksom dve v shrambi zasedeno. Ker je omenjeno mesto prosto, se na tem mestu nastavi vrednost *B,* definirana v poslanih podatkih transakcije.

Pametne pogodbe v celoviti arhitekturi informacijskih sistemov, temelječih na verigah blokov, predstavljajo zaledni sistem, ki je tako zamenjal centralizirane strežnike v tradicionalnih rešitvah. Oblični del nove arhitekture predstavljajo t.i. decentralizirane aplikacije (dApp), ki so komunikacijo s strežnikom zamenjale s komunikacijo s pametnimi pogodbami na verigah blokov. Da decentralizirana aplikacija zaganja programljivo poslovno logiko znotraj pametnih pogodb, mora ob sklicevanju na naslov pametne pogodbe uporabiti tudi sklicevanje na definirane funkcije (imena) znotraj le teh. Na ta način lahko razvijalci decentraliziranih aplikacij uporabniku ponudijo uporabniško prijazen vmesnik, ki v ozadju s pomočjo EVM izvaja poslovno logiko proženih funkcij pametnih pogodb. Na ta način

omogočimo uporabo pametnih pogodb oz. natančneje spreminjanje definiranih stanj znotraj pametnih pogodb.

3. IZZIV

Ključna lastnost tehnologije veriženja blokov je nespremenljivost zapisov, kar obenem predstavlja dodano vrednost same tehnologije. Z vidika PP na verigah blokov omenjena dodana vrednost lahko predstavlja izvedbene ter varnostne težave. Izvorna koda PP z namestitvijo v omrežje verige blokov (transakcija s programsko kodo je del potrjenega bloka) postane nespremenljiva, kar pomeni, da je v primeru napačno implementirane programske logike le ta tudi nezamenljiva [6]. Nepredvidljiva obnašanja PP z napakami v programski logiki so v preteklosti že rezultirala v poslovni škodi udeležencev poslovnega procesa, implementiranega s pomočjo PP na verigah blokov. Eden izmed uspešnih napadov na programske napake v implementaciji PP je bil t.i. napad TheDAO [12]. Pametna pogodba TheDAO je izvajala decentraliziran avtonomni sklad tveganega kapitala in je napadalcem zaradi programske napake v implementaciji PP omogočila krajo več kot 50 milijonov dolarjev [13]. Iz podobnih razlogov razvijalci tako prevzemajo veliko odgovornosti pri razvoju programske logike PP, saj je zahtevana skoraj ničelna toleranca do napak. Težava nespremenljivosti PP znotraj platforme Ethereum se pojavlja ne samo na javnih omrežjih verig blokov, temveč tudi na omrežjih, uporabljenih za zasebno in konzorcijsko rabo. Zaradi nezmožnosti zamenljivosti PP znotraj verige blokov se prav tako pojavljajo tudi izzivi, do katerih prihaja v primeru nadgradnje ali spremembe definiranega poslovnega procesa z vidika PP, kot osnovnih gradnikov decentraliziranih aplikacij. Zaradi nezmožnosti zamenljivosti in nadgradljivosti PP, nameščenih znotraj omrežja verig blokov, so potrebni drugačni načini in pristopi gradnje arhitekture ekosistema PP, ki bodo sestavljale decentralizirano aplikacijo, kot v primeru tradicionalnih aplikacij. V primeru tradicionalnih aplikacij je morebitne napake v delovanju brez vpliva na izgubo podatkov mogoče odpraviti ob samem odkritju. Prav tako se pojavi težava s sklopljenostjo obličnega dela z zalednim delom, temelječim na verigah blokov, saj smo v primeru napak primorani nameščati nove pametne pogodbe v verigo blokov, ki bodo zamenjale stare, nakar porušimo povezavo med prej omenjenima komponentama celotne arhitekture decentraliziranih aplikacij. Osnovne koncepte in gradnike predlagane arhitekture, ki omogoča zanesljivost in nadgradljivost, bomo v nadaljevanju predstavili z osnovnim primerom PP *Parkomat*.

4. OPIS PREDLAGANEGA KONCEPTA

V tem poglavju bomo predstavili osnovne koncepte in gradnike predlagane arhitekture, ki omogoča zanesljivost in nadgradljivost na osnovi primera PP *Parkomat*, katere naloga bo beleženje dodeljenih dovoljenj vozilom za parkiranje na določenem parkirišču. Pomembno poslovno pravilo, ki ga mora implementirati pogojna logika PP, je tudi omejitev, ki implementira možnost dodeljevanja pravic parkiranja samo lastniku PP. Izvorna koda 1 prikazuje implementacijo PP *Parkomat* v visokonivojskem programskem jeziku Solidity.

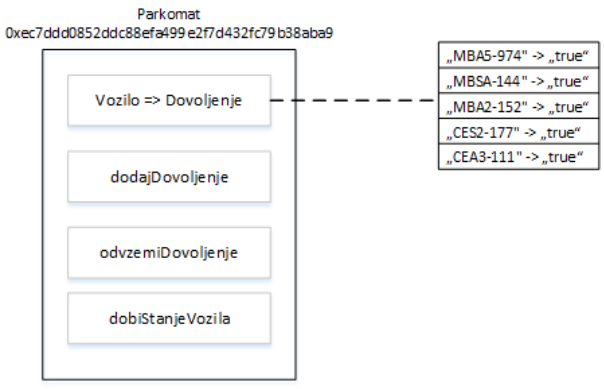
```
contract Parkomat {
    mapping (string => bool) private voziloDovoljenje;

    function dodajDovoljenje(string reg) external {
        voziloDovoljenje[reg] = true;
    }

    function odvzemiDovoljenje(string reg) external {
        voziloDovoljenje[reg] = false;
    }

    function dobiStanjeVozila(string reg) external view returns(bool) {
        return voziloDovoljenje[reg];
    }
}
```

Izvorna koda 1: Pametna pogodba Parkomat.

Pametna pogodba *Parkomat* definira spremenljivko podatkovnega tipa preslikave, imenovano *voziloDovoljenje*, ter tri funkcije:

- *dodajDovoljenje*, ki kot vhodni argument sprejme registrsko številko vozila, kateremu dodeli pravico do parkiranja, in le to zapiše v podatkovno strukturo *voziloDovoljenje*.
- *odvzemiDovoljenje*, ki kot vhodni argument sprejme registrsko številko vozila, ki se mu odvzame pravica do parkiranja, in le to zapiše v podatkovno strukturo *voziloDovoljenje*.
- *dobiStanjeVozila*, ki kot vhodni argument sprejme registrsko številko vozila in iz podatkovne strukture *voziloDovoljenje* prebere stanje dovoljenja parkiranja.

Stanje zapisov podatkovne strukture *voziloDovoljenje* v shrambi PP *Parkomat*, nameščene v omrežje verig blokov, po vnosu petih vozil s pomočjo funkcije *dodajDovoljenje* vsebuje pet zapisov, prikazanih na Sliki 4.



Slika 4: Stanje zapisov v shrambi PP *Parkomat*.

Če podrobneje pogledamo Programsko kodo 1, lahko ugotovimo, da ima prav vsak uporabnik sistema možnost dodajanja in odvzemanja dovoljenj za parkiranje posameznim vozilom. Implementacija tako ne sovpada z našim poslovnim pravilom, ki zahteva, da ima možnost dodajanja in odvzemanja dovoljen za parkiranje posameznim vozilom samo lastnik parkomata. Glede na opisano je nujno potreben popravek izvorne kode PP *Parkomat*. Kot smo predhodno omenili, pametnih pogodb, nameščenih v

omrežje, ni mogoče spreminjati, zato smo primorani implementirati novo PP, ki bo v implementaciji programske logike upoštevala, da lahko samo lastnik parkomata dodaja in odvzema dovoljenja za parkiranje vozil. Programska koda 2 prikazuje novo implementacijo PP *Parkomat2*, ki ob namestitvi v omrežje verige blokov v shrambo PP zapiše naslov tistega, ki PP namešča v omrežje in se smatra za lastnika parkomata.

```
import "./Lastnik.sol";

contract Parkomat2 is Lastnik {
    mapping (string => bool) private voziloDovoljenje;

    function dodajDovoljenje(string reg) external samoLastnik {
        voziloDovoljenje[reg] = true;
    }

    function odvzemiDovoljenje(string reg) external samoLastnik {
        voziloDovoljenje[reg] = false;
    }

    function dobiStanjeVozila(string reg) external view returns(bool) {
        return voziloDovoljenje[reg];
    }
}
```

Programska koda 2: Popravljena implementacija PP Parkomat2.

Posodobljena PP *Parkomat2* deduje od novo ustvarjene PP *Lastnik*, katere implementacija je prikazana v Programski kodi 3. Pametna pogodba *Lastnik* definira funkcijski modifikator, ki preveri, ali je klicatelj funkcije določen kot lastnik PP *Lastnik*, ter zaradi dedovanja posledično tudi PP *Parkomat2*. V primeru, da temu ni tako, funkcijski modifikator vrača napako pri izvajanju funkcije. S pomočjo funkcijskega modifikatorja dosežemo, da lahko klice funkcij izvaja samo vnaprej določen uporabnik sistema, ki je v našem primeru lastnik parkomata.

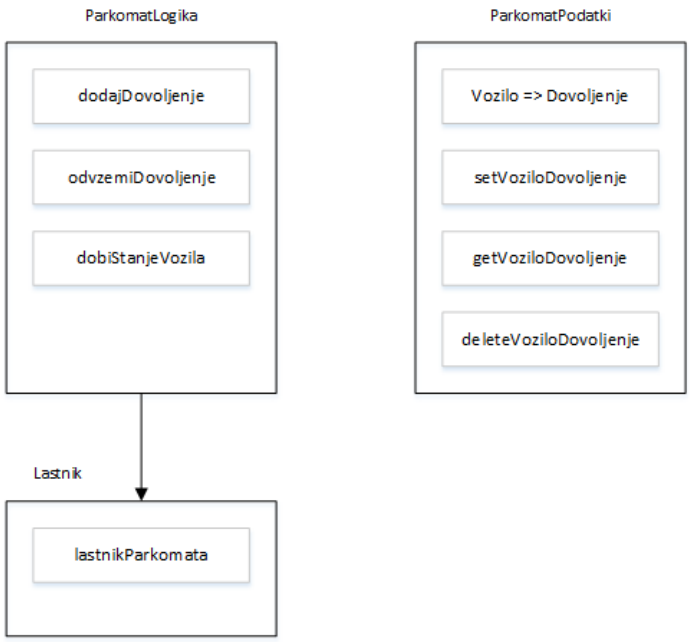
```
contract Lastnik {
    address public lastnikParkomata;

    constructor() public {
        lastnikParkomata = msg.sender;
    }

    modifier samoLastnik() {
        require(msg.sender == lastnikParkomata);
        _;
    }
}
```

Programska koda 3: Pomožna PP, ki definira lastništvo PP.

Pametno pogodbo *Parkomat2* namestimo v omrežje verige blokov, pri čemer lahko ugotovimo, da je spremenljivka oz. podatkovna struktura *voziloDovoljenje* v shrambi PP *Parkomat2* prazna, saj je nameščena PP *Parkomat2* popolnoma neodvisna PP v omrežju, ki nima logične povezave s PP *Parkomat*. Pametna pogodba *Parkomat2* tako vsebuje popravljeno implementacijo programske logike, vendar nima dostopa do podatkov PP *Parkomat,* ki je implementirala poslovni proces pred uvedbo popravkov izvorne kode.

Z namenom ponovne uporabe spremenljivk oz. podatkovnih struktur, ki se nahajajo v shrambi posameznih PP, predlagamo koncept ločevanja programske logike PP in shrambe PP, kar privede do dveh neodvisnih PP. Z ločevanjem programske logike PP in shrambe PP dosežemo šibko sklopljenost programske logike PP in podatkov, s katerimi le ta operira, kar omogoča možnost implementacije popravkov programske logike brez izgube podatkov, shranjenih v shrambi PP. Vzorec ločevanja programske logike in podatkovnih struktur smo povzeli iz mikrostoritvene arhitekture.

Slika 5 prikazuje predhodno PP *Parkomat*, sedaj razdeljeno na dve neodvisni pametni pogodbi *ParkomatLogika* in *ParkomatPodatki*. Pametna pogodba *ParkomatPodatki* tako definira podatkovno strukturo *voziloDovoljenje* in funkcije, s katerimi je omogočena manipulacija podatkovne strukture (set, get). Pametna pogodba *ParkomatLogika* definira funkcije, s katerimi dosežemo funkcionalnosti, definirane s poslovnimi pravili.

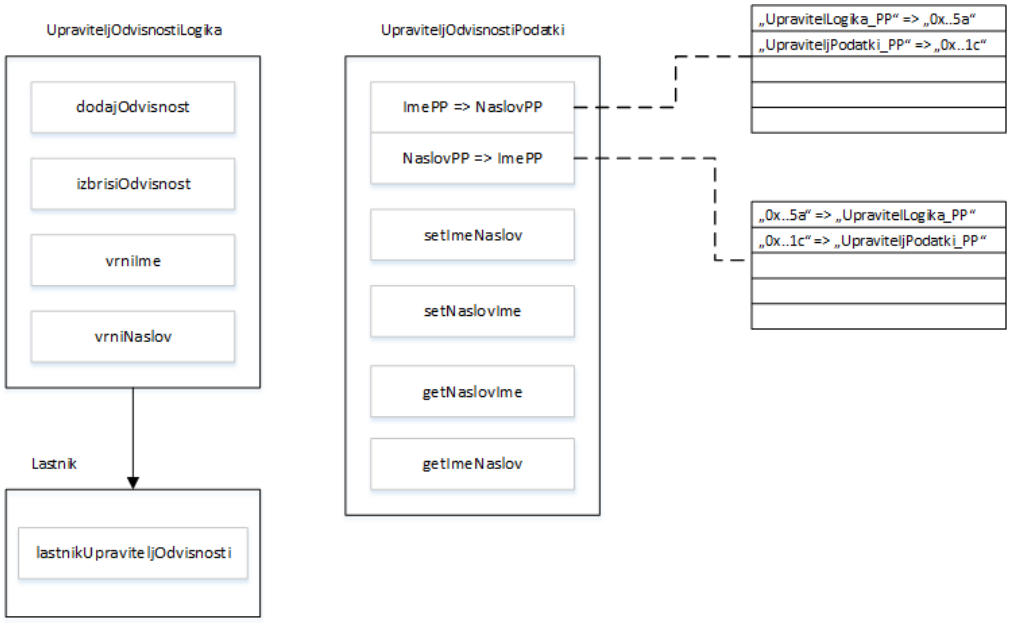


Slika 5: Ločevanje programske logike od shrambe podatkov na primeru PP Parkomat.

Kot smo že omenili, sta pametni pogodbi *ParkomatLogika* in *ParkomatPodatki* v primeru namestitve v omrežje verig blokov popolnoma neodvisni, za implementacijo poslovnega procesa pa je potrebno povezovanje prej omenjenih pametnih pogodb na način, da bodo podatki v primeru morebitnih hroščev v programski logiki ostali neodvisni in na voljo za uporabo pametni pogodbi, ki bo implementirala popravke programske logike.

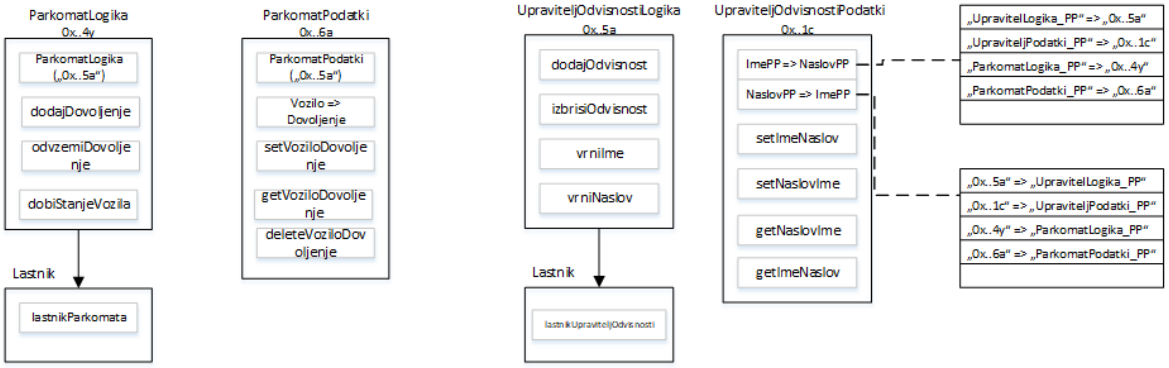
Za doseganje povezovanja pametnih pogodb v ekosistemu arhitekture, ki omogoča zamenljivost in nadgradljivost pametnih pogodb v omrežju Ethereum se pojavlja potreba po neodvisnem gradniku, katerega namen je hramba vseh imen in naslovov pametnih pogodb, nameščenih v omrežje, ki so relevantne za gradnjo decentralizirane aplikacije. Predlagan gradnik smo poimenovali upravitelj

odvisnosti (ang. dependency manager). Konceptualna shema predlaganega gradnika je prikazana na Sliki 6.



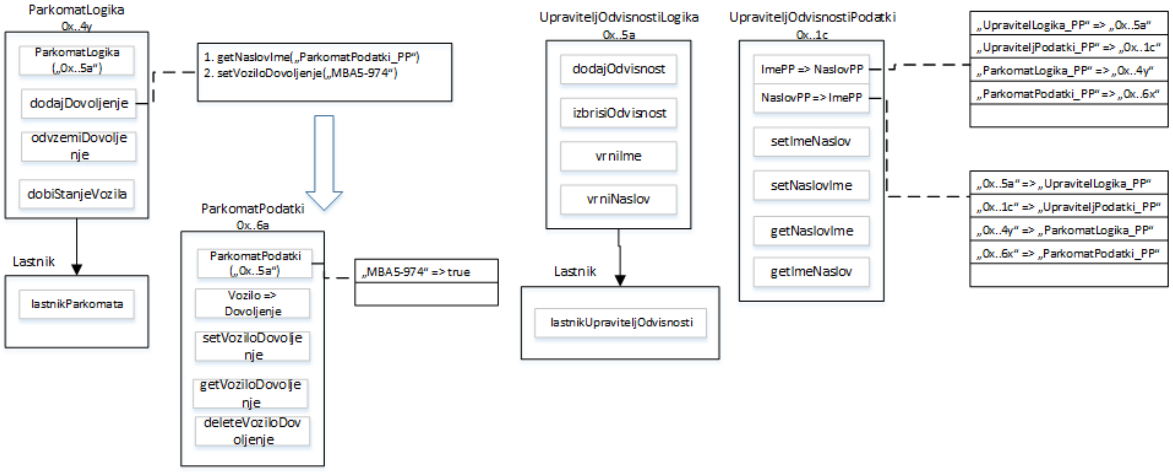
Slika 6: Upravitelj odvisnosti v ekosistemu PP.

Omenjen gradnik je PP, ki ob namestitvi v omrežje kreira dve ločeni pametni pogodbi, imenovani *UpraviteljOdvisnostiLogika* in *UpraviteljOdvisnostiPodatki.* V shrambo PP *UpraviteljOdvisnostiPodatki* se ob kreiranju zabeležita imeni in naslova novo kreiranih pametnih pogodb. Za doseganje povezovanja je potrebno omenjen gradnik vključiti v prav vsako PP, ki bo del ekosistema pametnih pogodb, ki bodo tvorile decentralizirano aplikacijo. Če se navežemo na primer PP *ParkomatLogika* in PP *ParkomatPodatki*, je potrebno naslov PP *UpraviteljOdvisnostiLogika* vključiti v prej omenjeni pametni pogodbi kot parameter konstruktorja, ki se proži ob namestitvi PP v omrežje verige blokov in s tem ustvari povezavo med omenjenima pametnima pogodbama in PP *UpraviteljOdvisnostiLogika*. Prav tako se v seznam odvisnosti dodata pametni pogodbi, ki vključujeta PP *UpraviteljOdvisnostiLogika.* Na ta način je omogočena komunikacija med PP *ParkomatLogika* in PP *ParkomatPodatki* preko funkcij, ki jih omogoča PP *UpraviteljOdvisnostiLogika.* Omenjena visokonivojska shema nameščenih PP v omrežje verige blokov je vidna na Sliki 7.



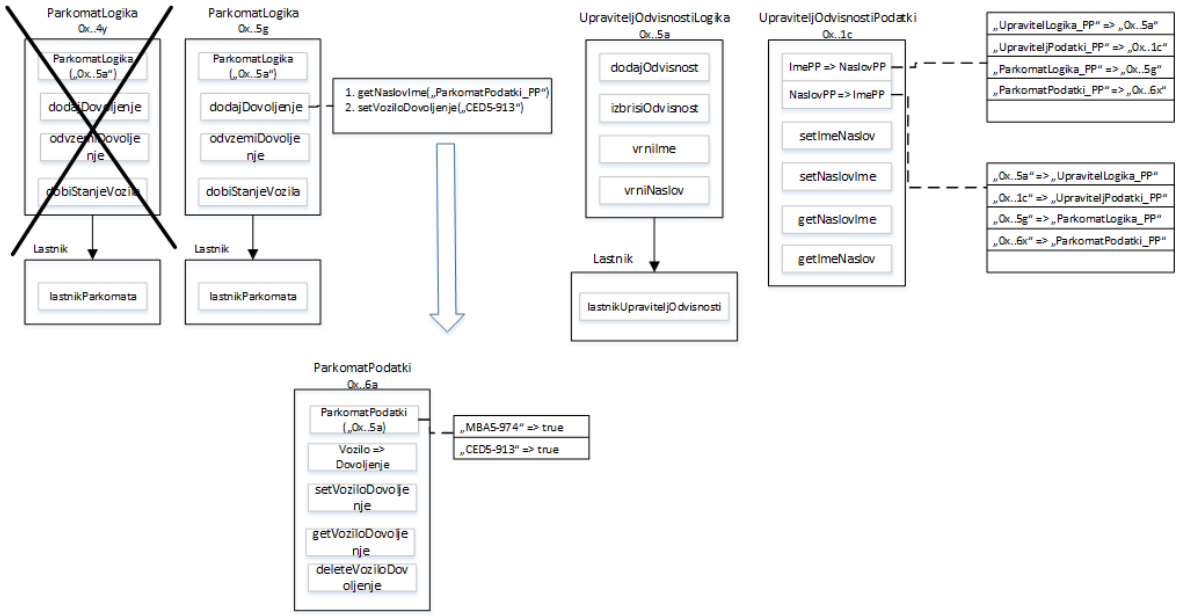
Slika 7: Shema ekosistema pametnih pogodb po namestitvi v omrežje verige blokov z gradnikom upravitelj odvisnosti.

Funkcije PP *ParkomatLogika* v svoji programski logiki kličejo funkcijo *getNaslovIme*, definirano znotraj PP *UpraviteljOdvisnostiLogika*, ki glede na podano ime PP vrača naslov PP, nameščene v omrežje verige blokov. Na Sliki 8 je prikazano delovanje funkcije *dodajDovoljenje*:*ParkomatLogika* v navezavi s PP *UpraviteljOdvisnostiLogika* in PP *ParkomatPodatki*. V primeru napačne implementacije programske logike PP *ParkomatLogika* opisan koncept razdelitve programske logike ter shrambe PP in uporabe gradnika *UpraviteljOdvisnosti* omogoča zamenljivost PP *ParkomatLogika*. Namestitev nove PP *ParkomatLogika* v verigo blokov bo povzročila kreiranje novega naslova omenjene PP, katerega sprememba se bo ob vključitvi naslova PP *UpraviteljOdvisnostiLogika* v konstruktorju PP *ParkomatLogika* odražala tudi v PP *UpraviteljOdvisnostiPodatki*. Na ta način bo novo implementirana programska logika PP *ParkomatLogika* preko PP *UpraviteljOdvinostiLogika* lahko dostopala do shrambe podatkov, ki jo je pred tem uporabljala starejša različica PP *ParkomatLogika*.



Slika 8: Delovanje funkcije dodajDovoljenje:ParkomatLogika.

Visokonivojska predstavitev zamenjave PP *ParkomatLogika* in rezultat proženja funkcije *dodajDovoljenje:ParkomatLogika* je vidna na Sliki 9.



Slika 9: Delovanje funkcije dodajDovoljenje:ParkomatLogika v zamenjani PP ParkomatLogika.

Za konec poudarimo še, da so funkcije, ki omogočajo manipulacijo spremenljivk v shrambah PP, javne, kar pomeni, da obstaja možnost klica PP s strani za to neavtoriziranega računa. Zaradi tovrstnega problema predlagamo dodaten gradnik v ekosistem zamenljivosti in nadgradljivosti PP, ki služi kot hramba prej definiranih zaupanja vrednih povezav s strani lastnika ekosistema PP. Predlagan gradnik je prav tako PP, ki definira funkcijo, s katero pametne pogodbe, namenjene hrambi podatkov, pred vsako manipulacijo podatkov preverijo pravico računa klicatelja funkcije do spremembe le te. Dodajanje dovoljenj za komunikacijo je možno implementirati že v konstruktorju PP pred namestitvijo v arhitekturo ekosistema PP ali pa s klicem funkcije, ki jo za to namensko definira PP. Zaradi kompleksnosti dodatnega gradnika bomo podrobnosti le tega izpustili in predstavili v drugih prispevkih.

5. ZAKLJUČEK

Pametne pogodbe so s tehnologijo veriženja blokov, natančneje s platformo Ethereum, v tehničnem smislu polno zaživele. Razlog temu je tudi možnost uporabe takšnih pametnih pogodb za namen začetnih ponudb kovancev (ang. initial coin offering – ICO), ki so v zadnjih letih v razmahu, saj so omogočile zbiranje več milijard evrov ustanovnega kapitala za različne projekte. Čeprav jih poznamo že več kot 20 let, so njihova tehnična izvedljivost in koncepti, na katerih temeljijo, še zelo sveži. Zaradi omenjene zgodnje faze razvoja pametnih pogodb se vzorci in dobre prakse razvoja pametnih pogodb šele oblikujejo in vzpostavljajo. Hkrati prevzemajo razvijalci pametnih pogodb odgovornost, ki zahteva skoraj ničelno toleranco do napak, kar je neizvedljivo v popolnosti. Ker temeljijo na konceptih tehnologije veriženja blokov, kot so decentralizacija, porazdeljenost, nespremenljivost, se te lastnosti prenašajo tudi na delovanje pametnih pogodb. Posledica je nezmožnost spreminjanja pametnih pogodb, kar lahko v primeru napačne implementacije privede do nepredvidenih anomalij v delovanju le teh. V svojem delu smo se lotili raziskovanja omenjene problematike ter predstavili način gradnje pametnih pogodb, ki bi v primeru zahtev po spremembah poslovne logike ali ugotovljenih hroščev v implementaciji posameznih že nameščenih pametnih pogodb, omogočal nadgradnjo oz. odpravo le teh. V prispevku smo na kratko predstavili arhitekturo in s tem predlog dobre prakse gradnje pametnih pogodb, ki omogoča učinkovito zamenljivost in nadgradljivost že nameščenih pametnih pogodb omrežja Ethereum. Zaradi obširnosti same tematike smo izpostavili le del ključnih gradnikov predlagane arhitekture in le te, za namene lažjega razumevanja, podkrepili s predstavitvijo in kratkimi primeri. V nadaljnjih delih bomo arhitekturo razširili in vzpostavili ogrodje (ang. framework), ki bo razvijalcem še dodatno olajšalo uporabo predstavljenih dobrih praks.

6. LITERATURA

[1] D. Furlonger, R. Valdes, and R. Kandaswamy, “Hype Cycle for Blockchain Technologies, 2017,” *Gartner*, Julij, p. 3775165, 2017.

[2] N. Szabo, “Formalizing and Securing Relationships on Public Networks,” *First Monday*, vol. 2, no. 9, Sep. 1997.

[3] M. Turkanović, A. Kamišalić, and B. Podgorelec, “DSI 2018 - Pametne pogodbe na verigah blokov,” *Dnevi Slov. Inform.*, April 2018, pp. 1–8, 2018.

[4] www.coinmarketcap.com, Cryptocurrency Market Capitalizations | CoinMarketCap, obiskano 12.5.2018.

[5] M. Hölbl, L. Hrgarek, M. Kompara, and M. Turkanović, “Varnostni koncepti tehnologije veriženja blokov,” no. april 2018, pp. 1–10, 2018.

[6] K. Delmolino, M. Arnett, A. E. Kosba, A. Miller, and E. Shi, “Step by Step Towards Creating a Safe Smart Contract: Lessons and Insights from a Cryptocurrency Lab.,” *IACR Cryptol. ePrint Arch.*, vol. 2015, p. 460, 2015.

[7] S. Luck, “Design and Implementation of a Smart Contract Creator Framework for IoT Devices,” no. August, 2017.

[8] M. Dameron, "Beigepaper: An Ethereum Technical Specification," pp. 1–24.

[9] G. Wood, "Ethereum: a secure decentralised generalised transaction ledger," *Ethereum Proj. Yellow Pap.*, pp. 1–32, 2014.

[10] www.solidity.readthedocs.io/en/latest/index.html, Solidity — Solidity 0.4.21 documentation, obiskano 15.5.2018.

[11] V. Buterin, "A next-generation smart contract and decentralized application platform," *Etherum*, no. Januar, pp. 1–36, 2014.

[12] N. Atzei, M. Bartoletti, and T. Cimoli, "A Survey of Attacks on Ethereum Smart Contracts (SoK)," vol. 10204, no. Marec, 2017, pp. 164–186.

[13] K. Bhargavan *et al.*, "Formal Verification of Smart Contracts Short Paper," *Proc. 2016 Acm Work. Program. Lang. Anal. Secur.*, pp. 91–96, 2016.

Univerzitetna založba
Univerze v Mariboru

RAZVOJ PROGRAMSKIH REŠITEV VERIŽENJA BLOKOV ZA ENERGETSKO POSLOVNO DOMENO

ANDREJ BREGAR, CIRIL KAFOL, JURE TRILAR IN MATEJ NOSAN

Povzetek: Ena aktualnejših tehnologij zadnjega časa je veriženje blokov, ki zagotavlja sledljiv in zaupanja vreden konsenz glede velikih množic transakcij. Ta tehnologija pridobiva vse več privržencev in primerov uporabe v različnih poslovnih domenah. V prispevku analiziramo in predstavimo možnosti in scenarije celovite uporabe mehanizmov veriženja blokov v povezavi s poslovnimi ter reguliranimi procesi energetskih sistemov. Kot enega prvih vzorčnih primerov v tej domeni realiziramo predlog pilotskega projekta razvoja in uvedbe programske rešitve veriženja blokov v izbranem procesu transparentnega, varnega ter sporazumnega prenosa merilnih podatkov na trgu z električno energijo. Prototipna rešitev je razvita na platformi Ethereum in zajema vse ključne elemente veriženja blokov, kot so poslovna pravila na osnovi pametnih pogodb, zapisovanje in persistenca transakcijskih podatkov v blokih, porazdeljena večvozliščna arhitektura, uporabniška identiteta, varnostni mehanizmi na podlagi certifikatov in podpisovanje z žetoni. Varen uporabniški vmesnik je razvit kot decentralizirana aplikacija. Model je zaseben, kar pomeni, da končni odjemalci dostopajo s svojimi identitetami, ki se vodijo z javnimi ključi, prek varnega spletnega uporabniškega vmesnika do vozlišč distributerjev, na katerih infrastrukturo so vezana merilna mesta. V prispevku opišemo implementirani poslovni scenarij, pametne pogodbe s procesnimi pravili, podatkovni model, arhitekturo sistema, varnostne mehanizme in uporabniški vmesnik. Ker temelji programska rešitev na specifični izbrani platformi, opravimo tudi primerjavo razpoložljivih tehnologij veriženja blokov in izbiro najprimernejše tehnologije za aplicirani scenarij v energetski domeni. Članek zaključimo z analizo pridobljenih izkušenj z uporabo in zrelostjo tehnologije. Izpostavimo tako prednosti in priložnosti kakor tudi pomanjkljivosti in dvome.

Ključne besede: • veriženje blokov • platforma Ethereum • primerjalna študija • informatizacija poslovnih procesov energetske domene • razvoj decentraliziranih aplikacij

NASLOV AVTORJEV: dr. Andrej Bregar, Informatika d.d., Vetrinjska ulica 2, 2000 Maribor, Slovenija, e-pošta: andrej.bregar@informatika.si. dr. Ciril Kafol, Informatika d.d., Vetrinjska ulica 2, 2000 Maribor, Slovenija, e-pošta: ciril.kafol@informatika.si. Jure Trilar, Informatika d.d., Vetrinjska ulica 2, 2000 Maribor, Slovenija, e-pošta: jure.trilar@informatika.si. Matej Nosan, Informatika d.d., Vetrinjska ulica 2, 2000 Maribor, Slovenija, e-pošta: matej.nosan@informatika.si.

DOI https://doi.org/10.18690/978-961-286-162-9.3 ISBN 978-961-286-163-6

Dostopno na: http://press.um.si.

1. UVOD

Podobno kot v vseh kompleksnejših poslovnih in reguliranih okoljih, se tudi v (elektro)energetski domeni izvajajo številni temeljni procesi, kot so postopki priključevanja odjemalcev električne energije, izbire in menjave dobaviteljev na energetskem trgu ter pridobivanja, obdelave in izmenjave merilnih podatkov. Ti procesi so aplikativno-informacijsko podprti, praviloma pa tudi standardizirani, na primer po priporočilih foruma ebIX [32]. Za tovrstne poslovne procese je ključno, da ustrezno zaščitimo podatke in podatkovne transakcije ter da dosežemo konsenz glede veljavnosti podatkov s strani številnih akterjev, ki so udeleženi v procesih. V ta namen lahko posežemo po zelo aktualni tehnologiji veriženja blokov [12]. V zadnjem času so se pojavile prve ideje o uporabi veriženja blokov v energetiki [13], vendar je dejanskih raziskav in praktičnih aplikacij razmeroma malo, zaradi česar ostaja odprtih veliko možnosti. Namen pričujočega prispevka je tako raziskati možnosti uporabe ustreznih mehanizmov veriženja blokov za podporo poslovnim, informacijskim in tehničnim procesom v energetski domeni ter izpeljati realizacijo pilotskega projekta uvedbe veriženja blokov v procesih zagotavljanja merilnih podatkov s trga električne energije.

Tehnologija veriženja blokov je v zadnjem času po zaslugi pretežne rasti cene kriptovalut, kot so Bitcoin, Ether in množica ostalih alternativnih kriptovalut (*altcoins*), na vrhu pričakovanj glede bodočega razvoja in uporabnosti [8]. Vendar pa smo šele v stopnji pred dejansko komercializacijo, na kar nakazuje dejstvo, da gre v veliki meri za nekritično prevzemanje novosti kot takšnih in ne prevzemanje tehnologije po načelu uporabnosti, kar je značilnost kasnejših stopenj krivulje življenjskega cikla proizvoda ali storitve. Zato je v tem trenutku edini res oprijemljivi koncept kriptovalutni trgovalni ekosistem, medtem ko so drugi primeri uporabe precej omejeni, zlasti z vidikov ergonomije in umestitve v obstoječe procese. Zakaj omenjamo umestitev v obstoječe procese, če naj bi blokovne verige prinašale popolnoma nove, boljše procese? Ker je sprejemanje tehnologije širše v družbi precej verjetno bolj odvisno od zmožnosti za umestitev v obstoječe procese kot pa od nekritične zamenjave le-teh. Še kako velja maksima, da »so blokovne verige več kot le tehnologija, so strategija«, kajti uporaba tehnologije predvideva tudi pravila, standarde in procese, ki so specifični zgolj zanjo. Zato je ena naših dodatnih nalog ugotoviti, katera platforma omogoča poleg tehnične učinkovitosti tudi prilagoditev obstoječim procesom. Samo tako bo sčasoma tehnologija veriženja blokov postala bolj zrela in uporabna za širšo javnost.

Preostanek članka sestoji iz petih poglavij. V 2. poglavju opišemo možnosti uporabe tehnologije veriženja blokov v energetski poslovni domeni. V 3. poglavju opravimo primerjavo nekaterih ključnih razpoložljivih platform in izberemo tisto, za katero menimo, da je najprimernejša za razvoj pilotske rešitve. Poglavje 4 predstavi pilotsko aplikacijo veriženja blokov v energetski domeni ter tehnološke, metodološke in razvojne rešitve, po katerih smo posegli. V 5. poglavju povzamemo pridobljene izkušnje ter izpostavimo prednosti in slabosti tehnologije. Članek zaključimo s 6. poglavjem, v katerem ocenimo potenciale in zmožnosti za uporabo in nadgradnjo tehnologije v prihodnosti.

2. MOŽNOSTI UPORABE TEHNOLOGIJ VERIŽENJA BLOKOV V INFORMACIJSKIH IN POSLOVNIH DOMENAH ENERGETSKIH SISTEMOV

Aplikacije in tehnologije veriženja blokov dobivajo v zadnjem času vse več primerov uporabe v različnih poslovnih domenah, tudi v energetiki [13]. (Elektro)energetska domena ima nekatere specifične značilnosti. Vzpostavljenih je veliko poslovnih in reguliranih procesov [14, 32] ter tržnih modelov in storitev, v katerih so v medsebojnih pogodbenih odnosih številni deležniki na energetskem trgu, kot so distributerji, regulatorji, dobavitelji, odjemalci, proizvajalci in drugi. Ti deležniki v sklopu standardiziranih in lastniških procesov shranjujejo, obdelujejo in izmenjujejo velike množice zaupnih elementarnih, agregiranih ter transakcijskih podatkov. V zadnjem času dobivajo pomembno vlogo tudi pametne naprave, pametna omrežja, integracije informacijskih in operacijskih tehnologij ter koncepti interneta stvari, ki so podvrženi nezanemarljivim kibernetskim varnostnim tveganjem [3]. Za razliko od

finančnega sektorja, ki je primarno osredotočen na transakcije kot informacijske entitete, trži energetski sektor energijo v obliki fizičnega proizvoda, katerega na osnovi transakcij dobavlja odjemalcem prek infrastrukture energetskega omrežja.

Tehnologija veriženja blokov skuša zagotoviti zaupanja vreden in sledljiv konsenz glede množic transakcij, shranjenih v dokončnih blokih v mreži vozlišč. Bloki so replicirani med različnimi lastniki, kar omogoči porazdeljeno persistenco transakcij. Sklenjene so tudi pametne pogodbe, ki vzpostavijo zaupanje in pravila med udeleženimi partnerji. Z decentralizacijo in pametnimi pogodbami je vsaj teoretično možno izboljšati nivo varnosti ter zagotoviti večjo neodvisnost od osrednje avtoritete. Za energetsko domeno to pomeni, da lahko zmanjšamo potrebo po posrednikih in regulatorjih. S kriptovalutami lahko udejanjimo tudi modele neposrednega plačevanja porabe energije na podlagi pravil, ki jih določajo členi pogodb.

Kljub predpostavkam o tehnološki in konceptualni primernosti je zlasti v ožjem regionalnem okolju veliko konceptov in potencialov nepreverjenih ter neizkoriščenih. Ocenjujemo, da bi bilo mogoče na področju energetike aplicirati mehanizme veriženja blokov v več scenarijih v povezavi s poslovnimi in reguliranimi procesi. Osnovni scenariji uporabe so zbrani v tabeli 1, v kateri so za reprezentativnejše med njimi podana tudi dodatna pojasnila.

Tabela 1. Scenariji uporabe veriženja blokov v energetiki

Scenarij uporabe	**Dostop**	**Obnašanje deležnikov**	**Domena**	**Aplikativni model**
Izmenjava podatkov na energetskem trgu • Zasebni pogodbeni dogovor med deležniki • Regulirane storitve • Varna posredna izmenjava podatkov	Zaprt/zaseben ali odprt/javen	Sodelovalno	Regulirana	Pogodbeni, upravljanje lastništva
Registracija podatkov o lastništvu virov (merilne naprave, proizvodne naprave)	Zaprt/zaseben ali odprt/javen	Sodelovalno	Regulirana	Upravljanje lastništva
Certifikacija in garancije (obnovljivi viri energije, emisije)	Zaprt/zaseben ali odprt/javen	Sodelovalno	Regulirana	Upravljanje lastništva
Optimizacija modelov nakupa in trženja energije • Odobritev dostopa do merilnih podatkov • Persistenca transakcij o proizvodnji in dobavi • Optimizacija pri oblikovanju ponudb • Optimizacija pri iskanju ponudnikov • Pogajanja med ponudniki in odjemalci • Plačevanje s kriptovalutami	Zaprt/zaseben ali odprt/javen	Tekmovalno	Poslovna	Transakcijski
Upravljanje in optimizacija kapacitet energetskega omrežja na podlagi ponudbe in povpraševanja • Odobritev dostopa do merilnih podatkov • Persistenca transakcij o proizvodnji in dobavi • Dinamično načrtovanje topologije omrežja • Dinamično planiranje obsega proizvodnje • Dinamično planiranje hrambe presežne energije • Pogodbe o proizvodnji in hrambi energije	Zaprt/zaseben ali odprt/javen	Sodelovalno, tekmovalno	Regulirana, poslovna	Pogodbeni, transakcijski
Neposredno decentralizirano trženje in distribucija energije • Odprt energetski trg ali pametne skupnosti s proizvodnimi napravami na osnovi obnovljivih virov energije • Proizvajalci neposredno tržijo energijo • Ni posredniških dobaviteljev • Ni reguliranega procesa menjave dobavitelja • Pogodbene relacije o dobavi in hrambi energije • Shranjevanje presežne proizvedene energije	Odprt/javen	Sodelovalno, tekmovalno	Poslovna	Pogodbeni, transakcijski

• Persistenca transakcij o proizvodnji in dobavi • Plačevanje s kriptovalutami				
Merjenje in obračun porabe energije	Odprt/javen	Sodelovalno, tekmovalno	Poslovna	Pogodbeni, transakcijski
Obračun energije za mobilnost (polnilnice električnih vozil) • Enkratne tržne storitve • Persistenca transakcij odjema za vpogled	Odprt/javen	Sodelovalno, tekmovalno	Poslovna	Transakcijski
Komunikacija pametnih naprav	Odprt/javen	Sodelovalno	Poslovna	Transakcijski

Glede na trenutno stanje in stopnjo razvitosti tehnologije veriženja blokov vsi identificirani scenariji še niso neposredno in v celoti izvedljivi ali so izvedljivi z omejitvami. Poglavitni težavi sta skalabilnost in hitrost transakcij, o čemer podrobneje pišemo v petem poglavju članka. Prav tako bo večina scenarijev zahtevala regulativne spremembe na energetskem trgu, ki jih ne bo zlahka zagotoviti, čeprav so se nekatere aktivnosti in pobude v tej smeri tako na nacionalni kot mednarodni ravni že pričele [15]. Te spremembe bodo morale upoštevati regulirane procese, standardizacijo podatkovnih struktur in model vlog na energetskem trgu. Ker pa je tehnologija veriženja blokov še v fazi razvoja, velikih pričakovanj in pridobivanja aplikativne zrelosti, lahko pričakujemo za večino identificiranih scenarijev višjo stopnjo izvedljivosti v prihodnjih letih.

Scenariji se razlikujejo po stopnji kompleksnosti. Najpreprostejši za izvedbo so tisti, ki urejajo lastništvo podatkov, naprav, dokumentov in certifikatov. Ti scenariji ne izkoriščajo nujno vseh potencialov veriženja blokov, še zlasti ne v duhu udejanjanja inovativnih in aktivnih poslovnih modelov. Veliko možnosti se pokaže, če podpremo tekmovalne in sodelovalne modele med dobavitelji in/ali distributerji, neposredno shranjevanje agregiranih merilnih podatkov v verige blokov ali pa javne modele, v katerih lahko tudi proizvajalci in odjemalci od centralne avtoritete pridobijo certifikate ter vzpostavijo svoja vozlišča. V slednjem primeru bi bilo možno oblikovati odprt energetski trg, na katerem lahko odjemalci proizvajalci (tako imenovani »prosumerji«) neposredno tržijo proizvedeno energijo brez posredniških dobaviteljev, pri čemer ta koncept razmeroma enostavno in naravno nadgradijo v smeri plačevanja s kriptovalutami.

Prvi primer aplikacije odprtega decentraliziranega sistema trženja in distribucije električne energije je bil s pilotskim projektom v okviru manjše pametne samooskrbujoče energetske skupnosti že udejanjen v praksi [6]. Gre za mikroomrežje v newyorški soseski Brooklyn, ki obratuje neodvisno od centralnega regionalnega omrežja, tako da je samostojno odgovorno za celotno proizvodnjo in porabo energije, ki je pridobljena iz obnovljivih virov. Odjemalci, ki nimajo lastnih proizvodnih kapacitet, tako kupujejo energijo neposredno od sosedov proizvajalcev, pri čemer niso potrebni vmesni dobavitelji. Cene se oblikujejo dinamično glede na ponudbo in povpraševanje. Vsi transakcijski podatki se shranjujejo v blokovni verigi na platformi Ethereum, ki omogoča tudi sklepanje pogodb med lokalnimi proizvajalci in odjemalci. Takšen decentralizirani sistem ima več temeljev [13]:

- Pametne pogodbe uravnovešajo ponudbo in povpraševanje ter tako nadzirajo energetska omrežja v smislu proizvodnje, odjema in shranjevanja energije.
- Transakcijski podatki se shranjujejo v decentralizirani varni blokovni verigi, do katere deležniki dostopajo s svojimi digitalnimi identitetatmi.
- Transakcije se udejanjajo samodejno na osnovi pametnih pogodb in vplivajo na dobavo energije prek fizičnega energetskega omrežja.
- Odjemalci lahko plačujejo transakcije s kriptovaluto.
- V blokovni verigi so shranjeni tudi lastniški podatki, kot so potrdila o lastništvu proizvodnih virov, certifikati itd.
- Spremljajoča infrastruktura energetskih sistemov, ki delujejo na osnovi blokovnih verig, vključuje pametne merilne naprave, pametne električne naprave, senzorje, mobilne aplikacije itd.

Eden od možnih izzivov je konsolidacija konceptov veriženja blokov ter teorije večkriterijskega odločanja in pogajanj. Veriženje blokov namreč v svoji zasnovi ne obravnava konfliktnih transakcij tekmovalnih strani, večkriterijskih preferenčnih struktur, pogajanj in popuščanj za maksimizacijo ali minimizacijo koristnosti sodelujočih partnerjev v situacijah tipa »win-win« ali nasprotujočih si entitet v situacijah tipa »win-lose«, prav tako pa tudi ne drugih konceptov, ki jih podpirajo mehanizmi tradicionalnih večkriterijskih pogajalskih procesov. Konsolidacija teh dveh področij odpira vrata za inovativne poslovne modele, v katerih ponudniki storitev na podlagi zmožnosti vpogleda v zaščitene transakcijske podatke dinamično maksimizirajo svoje večkriterijske koristnosti, oblikujejo in ponudijo optimalne storitve in storitvene strategije, izberejo najboljše razpoložljive stranke ali odjemalce ter presežejo svoje tekmece. Hkrati pa stranke ali odjemalci prav tako dinamično maksimizirajo svoje koristnosti z izbiro najboljših razpoložljivih storitev in ponudnikov storitev.

3. PRIMERJAVA RAZPOLOŽLJIVIH TEHNOLOGIJ VERIŽENJA BLOKOV IN IZBIRA NAJPRIMERNEJŠE TEHNOLOGIJE ZA APLICIRANI SCENARIJ

3.1. Primerjava tehnologij

Pri določitvi nabora platform za primerjavo smo izhajali iz subjektivne ocene priljubljenosti posameznih platform v razvijalski skupnosti. V danem obdobju (prva četrtina leta 2018) se je ta priljubljenost odražala tudi na lestvici tržne kapitalizacije primarne kriptovalute posamezne platforme [30], z izjemo tehnologije Hyperledger Fabric [35], ki ne uporablja enotne primarne valute. Bitcoin [28] je kljub tehnološki zastarelosti v določenih parametrih upoštevan kot orientacija, saj gre za najbolj prepoznavno in prvo kriptovaluto, ki je v javnosti sinonim za veriženje blokov.

Pri primerjavi je zajet širši nabor lastnosti, ki so pomembne v okviru tehničnih, konceptualnih, razvojnih in stroškovnih okvirov posamezne platforme blokovnih verig. Primerjavo podaja tabela 2, v kateri poskušamo prikazati tiste vidike, ki so pomembni za produkcijske izvedbe decentraliziranih aplikacij, kot tudi tiste, ki so relevantnejši za uspešno prototipiranje in preizkušanje konceptov, na katerih temeljijo posamezne platforme. Realna ocena je, da so tiste platforme, ki so najbolj primerne za produkcijo, zaradi svoje nazivne hitrosti in ostalih želenih lastnosti, še v razvoju, tiste, ki so primerne za prototipiranje, pa so bolj robustne in počasne. Menimo, da to pri prototipih ne predstavlja težav, saj se bo v naslednjih nekaj letih bržkone dalo prenesti koncepte in izvedbene načrte decentraliziranih aplikacij iz manj zmogljivih na bolj zmogljive platforme.

Pri nekaterih primerjalnih kriterijih je uporabljena subjektivna kvalitativna lestvica, pri kateri pa smo skušali ocene objektivizirati na osnovi preseka zaupanja vrednih virov. Prav tako smo pri oceni hitrosti transakcij upoštevali povprečje različnih virov, pri čemer v tabeli ni navedena absolutna hitrost, saj je prepustnost pogojena tudi z obremenitvami omrežij. Omrežja, ki se intenzivneje uporabljajo, npr. Ethereum ob večjih zbiranjih prispevkov (*ICO – Initial Coin Offering* [18]), večkrat doživijo upočasnitve. Platforma Bitcoin zato v zadnjem času uvaja izboljšavo v obliki protokola »*lightning network*«, ki pohitri in poceni transakcije tako, da le-te niso javne v verigi blokov [19].

Tabela 2. Primerjava lastnosti platform za veriženje blokov

	Bitcoin [28]	**Ethereum [33]**	**Hyperledger Fabric [35]**	**NEO [27]**	**IOTA [36]**	**EOS [20]**	**Cardano [29]**
Lastna kriptovaluta	Da (BTC)	Da (ETH)	Ne	Da (NEO, GAS)	Da (IOTA)	Da (EOS)	Da (ADA)
Pametne pogodbe	Omejeno (RSK)	Da (Solidity)	Da (Go, Java, Chaincode)	Da (C#, Java, Python)	Ne	Da (C++)	Da (CCL, REPL)

Tip konsenza in nagrajevanja	PoW	PoW, PoS (Casper)	Modularno	dBFT, PoS	Tangle, PoW	PoS	PoS (Ouroboros)
Nosilec razvoja, upravljanje	Skupnost, Bitcoin foundation	Skupnost, Ethereum foundation	IBM	OnChain ltd, CoZ	IOTA foundation	Block.One	Cardano foundation, IOHK, Emurgo
Stroški transakcij	Da – veliki	Da – srednji	Ne	Da – majhni	Ne	Ne	Da – majhni
Hitrost transakcij	7 tx/s	25 tx/s	3500 tx/s	1000 tx/s	500 tx/s	1mio tx/s	260 tx/s
Decentralizacija	Večja	Večja	Minimalna	Manjša	Večja	Manjša	Večja
Tip omrežja	Javno	Javno ali zasebno	Zasebno	Javno ali zasebno	Javno	Javno	Javno ali zasebno
Dostop	Odprt (permissionless)	Odprt (permissionless)	Zaprt (permissioned)	Oboje	Oboje	Odprt (permissionless)	Oboje
Anonimni računi	Da	Da	Ne	Oboje	Oboje	Da	Da
Hitri kanali (state channels)	Lightning Network	Raiden, Generalized	Ne potrebujemo	Trinity	Off Tangle	Ne	Ne
Primerno za IoT, M2M	Ne	Da, z omejitvami	Da	Da, z omejitvami	Da	Da, z omejitvami	Da
Primerno za decentralizirane aplikacije	Ne	Da	Da	Da	Da	Da	Da
Obratovalni stroški	Ne	Ne	Veliki	Ne	Ne	Ne	Ne
Podpora razvijalske skupnosti	Velika	Zelo velika	Srednja	Velika	Manjša	Manjša	Manjša
Komentar	Ni primerno za lastne odjemalce	Primerno za prototipiranje	Visoki stroški (TCO)	Centralizira-nost	Še v razvoju	Še v razvoju	Še v razvoju

3.2. Izbira tehnologije

Na podlagi opravljene primerjave tehnologij, analize virov, pregleda razvojne dokumentacije ter razvojnih in tržnih vizij ocenjujemo, da v tem trenutku nobena platforma tehnologij blokovnih verig še ni v zadostni meri pripravljena za produkcijske rešitve zaradi tehničnih kot tudi netehničnih značilnosti. Zadržki se nanašajo predvsem na težave s preformančnimi parametri posameznih platform, kot so število transakcij na sekundo (v realnih scenarijih) in skalabilnost, ter na konceptualne izzive, kakršni so centraliziranost, dostopnost in odprtost razvojne podpore ter tržni vidiki. Med slednje spada odsotnost pristopov z dokazljivo uporabnostjo v prihodnosti (»future-proof«), saj se lahko potrebe trga hitro spreminjajo, zaradi česar obstaja tveganje, da postanejo koncepti, ki so trdno določeni v posameznih platformah, zastareli. Pri izbiri najprimernejše platforme tehnologije blokovnih verig za izvedbo prototipnih rešitev smo tako upoštevali naslednje kriterije:

- zadosten dostop do razvojnih virov, velikost in odzivnost aktivne razvijalske skupnosti, dostopnost dokumentacije in dobri primeri obstoječih izdelkov, na osnovi česar lahko minimiziramo možnost, da se razvoj ustavi, ker temeljne infrastrukturne rešitve niso dobro dokumentirane ali morebitne težave še niso bile obravnavane, ter se izognemo kritičnim tehničnim oviram, ki nastanejo v primeru napak na jedrni platformi, in težavam z vzpostavitvijo lastne informacijske hrbtenice, ki bi podpirala to platformo;
- možnost prototipiranja in testiranja širokega nabora uporabniških scenarijev, česar ne omogočajo vse platforme, saj so nekatere med njimi specializirane zgolj za specifične tehnične in poslovne pogoje;
- možnost prenosa realiziranih konceptov in uporabne vrednosti na druge platforme, ko bodo le-te bolj zrele za produkcijsko uporabo;

- uporaba javnega okolja, pri čemer se za shranjevanje podatkov in procesne potrebe integracije lahko zanašajo na javno infrastrukturo, ki je ni potrebno vzdrževati.

Javno okolje resda omogoča manjšo prilagodljivost različnih tehničnih parametrov, vendar primerjava z zasebnimi (lastniškimi) integracijami tehnologije blokovnih verig pokaže, da je potrebno upoštevati tudi stroške vzpostavitve, predvsem pa vzdrževanja takega sistema, ki zahteva specialistična znanja in stroškovno manj učinkovite pristope. Bržkone ni potrebno posebej izpostaviti javnega značaja informacij in zapisovanja v porazdeljeno bazo podatkov, ki sama po sebi omogoča nadzor ter s tem vzbuja zaupanje uporabnikov in širše javnosti tako do aplikacije kot tudi do ponudnika storitve. Marsikateri avtor opaža, da je tehnologija blokovnih verig začetek protokolov zaupanja in konsenza, ki bodo temelj bodočim (bolj) decentraliziranim omrežjem.

Na osnovi kriterijev smo za potrebe razvoja prototipne rešitve izbrali tehnologijo Ethereum [25, 33]. Specifikacija protokola za varen prenos podatkov v tehnologiji blokovnih verig na omrežju Ethereum omogoča:

- hitro pridobitev podatkov iz platform Ethereum, ki praviloma ne presega 300 milisekund;
- manjšo hitrost zapisovanja in potrjevanja zapisov, ki praviloma zahteva od 3 do 5 minut;
- bolj konstantno generiranje blokov v primerjavi z omrežjem Bitcoin;
- uporabo različnih razvojnih okolij in knjižnic, kot so Truffle, Remix Web IDE itd.;
- procesiranje programske logike v izvajalnem okolju Ethereum Virtual Machine [10], ki omogoča visoko decentralizacijo na 27.500 vozliščih po vsem svetu;
- razvoj pametnih pogodb v namenskem jeziku Solidity [10];
- integracijo s pametnimi pogodbami iz spletnih aplikacij na osnovi aplikacijskega programskega vmesnika Ethereum JavaScript API (Web3.js) [23], kar predstavlja široko razvojno okolje;
- domorodno sredstvo za potrjevanje transakcij v obliki kriptovalue Ether v »agregatnem stanju« Gas [31], kar predstavlja mikro enote primarne valute in se nanaša na obremenitev »največjega in najpočasnejšega« računalnika za svetu, pri čemer so v praksi stroški transakcij manjši kot provizije za transakcije Bitcoina;
- transformacijo iz energetsko potratnega protokola nagrajevanja PoW (*Proof-of-Work*) v protokol PoS (*Proof-of-Stake*) [17, 24];
- razvoj v različnih razvojno-testnih omrežjih (Ropsten, Rinkeby, Kovan ali RPC po meri), v katerih se za testiranje ne porablja prava vrednost Ethra, kot je to na glavni hrbtenici (*main-net*);
- največji konzorcij podpornikov iz industrije (*Enterprise Ethereum Alliance*) [21].

4. RAZVOJ PILOTSKE REŠITVE VERIŽENJA BLOKOV V ENERGETSKI DOMENI

4.1. Scenarij uporabe

Scenarij uporabe rešitve veriženja blokov smo oblikovali na dveh internih delavnicah. Prve delavnice se je udeležilo večje število domenskih strokovnjakov. Na njej smo identificirali scenarij in opredelili njegove osnovne značilnosti. Delavnico smo izvedli s pomočjo polstrukturiranih intervjujev in ustvarjalnega zbiranja idej s tehniko možganske nevihte [9]. Analizirali smo pričakovanja in potrebe ciljnega sistema ter zmožnosti tehnologije. Pri tem smo upoštevali številne vidike in kriterije:

- Splošno uredbo o varstvu osebnih podatkov (GDPR) [16], katere implikacija je omejevanje na obdelavo domenskih podatkov, ki niso kritični s stališča pravne varnosti;
- umestitev v obstoječe procese v energetski domeni;
- preprečevanje manipulacij procesov, vpogleda v poslovanje organizacij in kibernetskih tveganj;
- tehnično izvedljivost, zlasti z vidikov obsega podatkov za obdelavo in persistenco, hitrosti in števila transakcij ter števila samostojnih vozlišč omrežja;

- neposredno uporabnost v sklopu obstoječe informacijske podpore poslovanju energetskega sektorja;
- primernost za demonstracijske namene krepitve internih kompetenc ter predstavitve tehnoloških in konceptualnih zmožnosti in potencialov v okviru širšega okolja;
- relativno enostavno in hitro izvedljivost;
- primernost za hitro prototipiranje in implementacijo na javni platformi (Ethereum).

Druge delavnice se je udeležila omejena skupina domenskih strokovnajakov. S tehniko sortiranja kartic [11] smo oblikovali natančen nabor konceptov, na katerih temelji aplikacija, ter definirali splošno arhitekturno zasnovo sistema, osnovne podatkovne strukture, temeljna pravila pametne pogodbe in deležnike procesa.

Na podlagi identificiranih omejitev in neizpolnjevanja definiranih pogojev smo izločili vse scenarije, ki vključujejo neposredno shranjevanje in obdelavo merilnih podatkov, pridobljenih na posameznih merilnih mestih odjemalcev električne energije, čeprav predstavljajo nekateri od teh scenarijev tehnološki, regulativni, funkcionalno-konceptualni in razvojno-inovativni izziv. Kot enega prvih realnih primerov smo tako v okviru predloga pilotske rešitve uvedli koncepte veriženja blokov v procesih zagotavljanja posrednega dostopa do merilnih podatkov dobaviteljem električne energije. V pristojnosti naprednega merilnega centra je možnih več scenarijev dostopa do merilnih podatkov odjemalca in/ali proizvajalca električne energije prek storitev za izmenjavo podatkov [15]. Izbrali smo podmnožico scenarijev, ki jih opredeljujejo členi pogodbe o uporabi podatkovnih storitev dostopa do merilnih podatkov odjemalca/proizvajalca električne energije. Členi so povzeti v tabeli 3.

Tabela 3. Pogodba o uporabi podatkovnih storitev dostopa do merilnih podatkov

1. člen:
Dobavitelju električne energije *X* bo omogočen spletni dostop do četrturnih merilnih in obračunskih podatkov za merilno mesto št. *Y* podpisanega odjemalca/proizvajalca električne energije za obdobje zadnjih 12 mesecev. Zajeti bodo mesečni četrturni podatki o odjemu in/ali prevzemu delovne in jalove energije ter mesečni količinski podatki po tarifah.
2. člen:
Dobavitelju električne energije *X* bo omogočen spletni dostop do četrturnih merilnih podatkov za merilno mesto št. *Y* podpisanega odjemalca/proizvajalca električne energije za pretekli mesec. Zajeti bodo mesečni četrturni podatki o odjemu in/ali prevzemu delovne in jalove energije.
3. člen:
Dobavitelju električne energije *X* bo omogočeno zagotavljanje statističnih merilnih in obračunskih podatkov za merilno mesto št. *Y* podpisanega odjemalca/proizvajalca električne energije za obdobje zadnjih 24 mesecev. Zajeti bodo mesečni količinski podatki o odjemu in/ali prevzemu delovne in jalove energije po tarifah.

Merjeni podatki o porabi električne energije pripadajo posameznim končnim odjemalcem in so v lasti distribucijskih podjetij. Dostop do teh podatkov je za dobavitelje elektrike posledično omejen, hkrati pa zanje predstavlja neprecenljiv vir informacij za oblikovanje optimalnih storitev in utrditev konkurenčne sposobnosti na trgu. Informacijska rešitev na temelju veriženja blokov v skladu z zakonodajo omogoča odjemalcu, da posredno prek distributerja odobri dobavitelju vpogled v osebne merilne podatke, s čimer pridobi dobavitelj možnost analize 15-minutnih odbirkov in na osnovi le-teh tudi možnost oblikovanja najustreznejše poslovne ponudbe. S pametno pogodbo so pri tem določena pravila, ki so sprejemljiva za vse udeležene strani.

Ustvarjalec pogodbe je dobavitelj ali distributer električne energije. Končni odjemalci ne morejo vzpostaviti svojih lastnih vozlišč, temveč dostopajo s svojimi identitetami, ki se vodijo z javnimi ključi, prek varnega spletnega uporabniškega vmesnika do vozlišč dobaviteljev ali vozlišč distributerjev, na katerih infrastrukturo so vezana merilna mesta. Ko odjemalec z zasebnim ključem podpiše enega ali več opredeljenih členov pametne pogodbe, se v bloku ustvari nova transakcija, ki je ni mogoče spreminjati in je kadarkoli na voljo za vpogled tako temu odjemalcu kakor tudi lastnikom vozlišč (administratorjem

sistema). Podatki transakcije identificirajo merilno mesto, distributerja, dobavitelja in časovno obdobje merilnih podatkov. Dobavitelj lahko pridobi odobrene podatke od distributerja na osnovi storitveno usmerjene integracije in jih uporabi za potrebe poslovnih analiz.

Ustvarjalec pogodbe ima še nekaj dodatnih možnosti. Lahko dodaja nove člene pogodbe, razporeja žetone med uporabnike in vidi, katere identitete so podpisale posamezne člene. Možna razširitev je, da podpisovanje članov ni dovoljeno vsem uporabnikom, temveč le tistim, ki imajo zadostno število žetonov. Pri tem ima ustvarjalec pogodbe oziroma administrator sistema nadzor, komu lahko dodeli pravice dostopa.

4.2. Tehnična izvedba

Prototip sestoji iz zalednega (*back-end*) dela, katerega osnova je pametna pogodba, ki ureja glavno logiko in zapis podatkov o uporabnikih in potrjenih členih pametne pogodbe v strukture polj, ter začelnega (*front-end*) uporabniškega vmesnika, ki skrbi za prijazno interakcijo med uporabnikom in blokovno verigo Ethereum. Programska koda je javno objavljena na portalu GitHub na naslovu http://www.github.com/juretrilar/dapp-informatika. Upoštevani so nekateri uveljavljeni vzorci zasnove decentraliziranih aplikacij [1, 2].

4.2.1. Avtentifikacija uporabnika

Avtentikacija uporabnika, ki ima lahko tudi vlogo lastnika (administratorja) pogodbe, se vrši prek denarnice platforme Ethereum, natančneje vtičnika MetaMask. Ta omogoča prijazno komuniciranje z blokovno verigo, saj ga uporabniki preprosto dodajo v brskalnik. To je najbolj splošno sprejet način interakcije med spletnimi aplikacijami in platformo Ethereum. Predpostavka je, da se samo edinstveni uporabnik lahko avtenticira in podpisuje transakcije s svojim zasebnim ključem. En vidik takšne oblike avtentikacije, ki je v nasprotju s tradicionalnimi, kot so certifikati ali kombinacija uporabniškega imena in gesla, je, da izključujemo osebne podatke, tako iz denarnice kot tudi iz vseh transakcij. Seveda je možno povezati javni ključ denarnice z bazami osebnih podatkov, vendar se v luči uredbe GDPR temu izogibamo, s čimer zagotovimo skladnost z uredbo ter ustvarimo nastavke za z GDPR skladne rešitve drugih ponudnikov, ki bodo lahko bolje prilagajali svoje nišne pristope za identifikacijo (KYC/AML [26]), da bodo pravno ustrezni. Ti nastavki bodo v obliki interakcije med javnim in zasebnim ključem uporabnikove denarnice ter ponudnikom storitve identifikacije.

4.2.2. Zaledje – pametna pogodba za shranjevanje podatkov

Za implementacijo in testiranje pametne pogodbe, ki upravlja z logiko in shranjuje zapise, smo uporabili spletno razvojno okolje Remix (remix.ethereum.org), ki je dostopno na glavni strani združenja Ethereum. Omogoča relativno hitro testiranje in objavo pametnih pogodb na testnem omrežju Ropsten [7], s čimer smo se izognili precej bolj kompleksnemu postopku objave pametne pogodbe iz lastnega razvojnega vozlišča. Ko podatke zapisujemo, traja potrditev zapisa od 3 do 5 blokov, kar pomeni nekaj minut. Če želimo podatke brati, pa je čas za to zanemarljiv, praviloma pod 300 milisekund.

Struktura pametne pogodbe je relativno razumljiva vsakomur, ki pozna osnove programiranja. V prvem delu so definirane spremenljivke in polja (*array*) ter podatkovne strukture (*struct*), ki se nanašajo na globalne zunanje lastnosti pogodbe, objekte uporabnikov, ki vsebujejo podpisane člene in stanje žetonov, ter objekte posameznih členov pogodbe, ki zajemajo besedilo in rezultate zgoščevalnih funkcij za napredno preverjanje. Sledijo funkcije, ki se nanašajo na uporabniške operacije. To so funkcije zasebnega (samo za uporabo znotraj pogodbe) ali javnega tipa (za zunanje klice), ki vračajo podatke uporabnikov in omogočajo uporabniško podpisovanje členov. Nazadnje so implementirane še funkcije za upravljanje z vsebinskimi členi pogodbe ter upravljanje identitet, ki podpisujejo posamezene člene. Del implementacije pogodbe je razviden iz izseka programske kode.

Programska koda 1. Izsek pametne pogodbe

```
contract FinalContract {

/* Definicije */

address public contractCreatorAddress = msg.sender;
string public contractName = "Pogodba o uporabi podatkovnih storitev dostopa do merilnih podatkov";

struct User {
      bytes32[] activeArticles; /* Cleni pogodbe, ki jih podpise uporabnik */
      uint balanceOf; /* Stanje zetonov */
}

mapping (address => User) public users; /* Preslikava uporabnikov */
address[] private usersList;

mapping(bytes32 => string) articles; /* Preslikava clenov pogodbe */
bytes32[] private articleList;

/* Funkcije */

function userSignArticles(address _address, bytes32[] _articles) public {
      for (uint i = 0; i < _articles.length; i++) userSignArticle(_address, _articles[i]);
      usersList.push(_address);
}

function setUserTokenBalance(address _address, uint _value) public {
      users[_address].balanceOf = _value;
      usersList.push(_address);
}

function addHashValue(string value) public returns(bool) {
      articleList.push(keccak256(value));
      return addKeyValueByHash(keccak256(value), value);
}

function addKeyValueByHash(bytes32 hash, string value) private returns(bool) {
      if(bytes(articles[hash]).length != 0) {
         return false;
      }
      articles[hash] = value;
      return true;
} }
```

Ko pogodbo objavimo, pridobimo njen unikaten naslov in vmesnik ABI (*Application Binary Interface*), s katerim lahko to pogodbo povežemo z uporabniško aplikacijo. Sedaj je pogodba dovzetna za proženje njenih funkcij iz zunanjega sveta. Čim nekaj shranimo v pogodbo, tega ni moč izbrisati. Pogodba deluje samostojno na unikatnem naslovu in če jo želimo dopolniti, jo lahko shranimo samo na nov naslov, vendar pri tem izgubimo podatke, ki se nanašajo na prejšnjo verzijo. Tako zagotovimo, da sta pogodba in obstoj podatkov vezana na unikaten naslov, kar zaznamo v primeru preusmeritve administratorja ali tretje osebe na morebitno alternativno pogodbo. Unikaten naslov se nahaja samo na omrežju (npr. Ropsten), na katerem smo pogodbo objavili. Če bi objavili produkcijsko verzijo na glavnem omrežju (*mainnet*), ki dejansko porablja sredstva s pravimi vrednostmi, nas aplikacija na to opozori.

4.2.3. Začelje – uporabniški del decentralizirane aplikacije

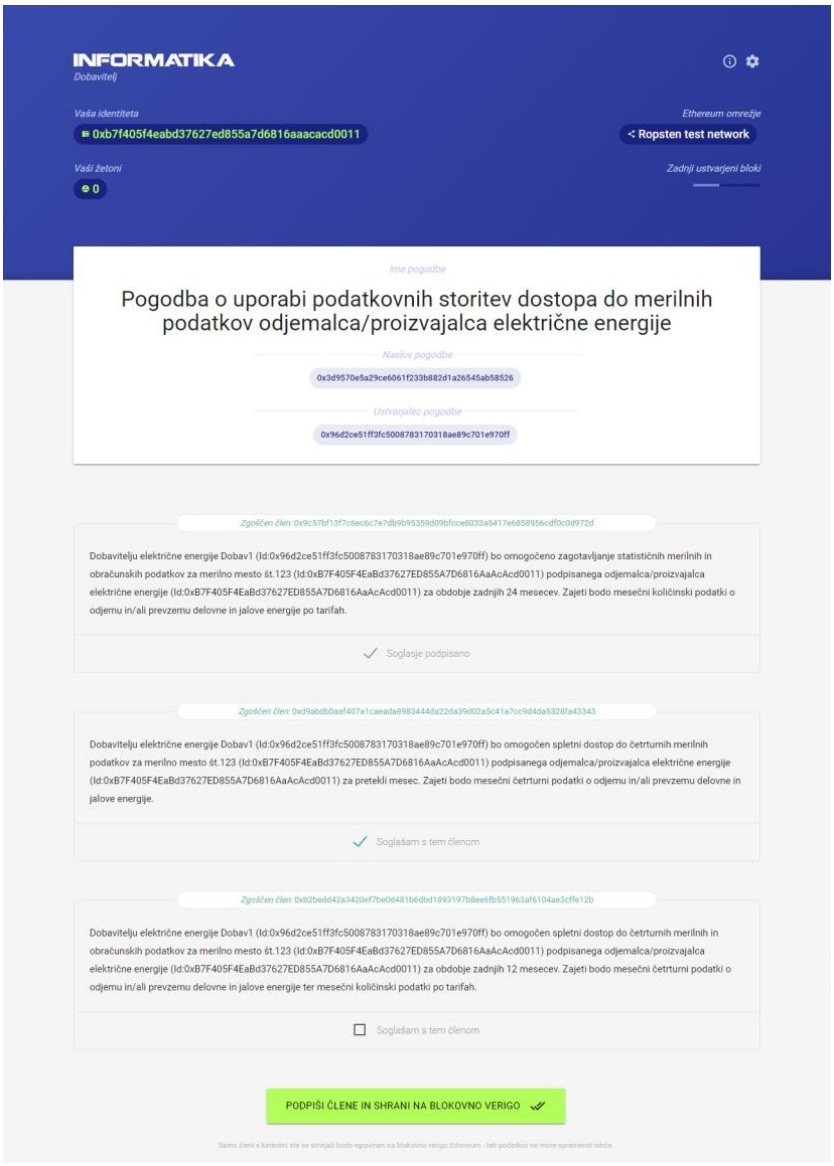
Uporabniški odjemalec temelji na osnovnih gradnikih spletnih aplikacij (HTML5, CSS3) z razširitvijo Materialize CSS, ki mu daje modernejši izgled. Za manipulacijo z vmesnikom DOM (*Document Object Model*) služi knjižnica jQuery, vse pa je nameščeno v ogrodju Express [22], ki se izvaja na mini strežniku Node.js za lokalno testiranje. Za javno objavo v spletu smo vzpostavili odlagališče Github, ki je povezano s platformo HerokuApp [34] in prikaže prototip na javnem spletnem naslovu.

Uporabnik lahko nastopa v dveh vlogah – kot navadni uporabnik ali kot ustvarjalec pametne pogodbe. S tem ima na voljo dva različna vmesnika. Ponazorjena sta v naslednjem podpoglavju.

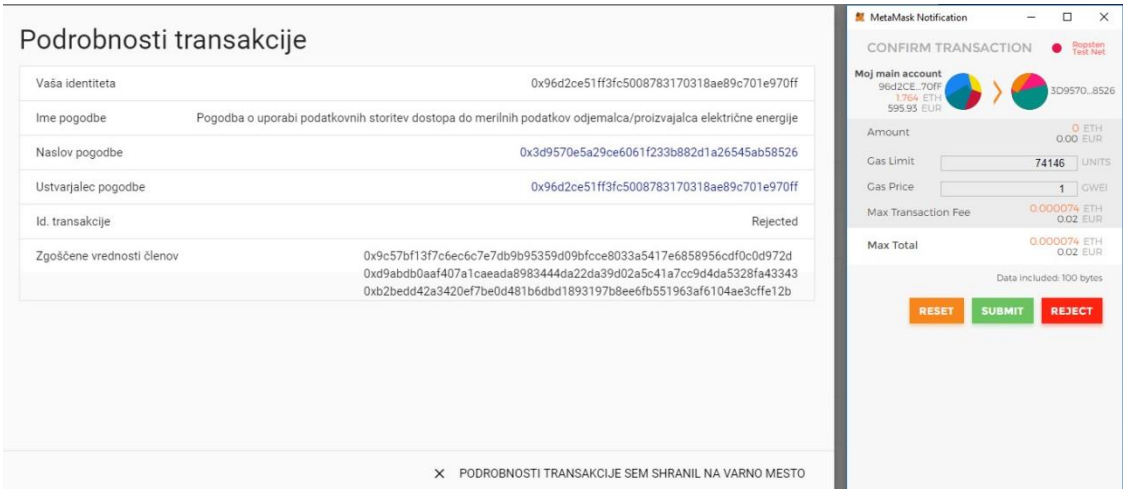
4.3. Uporabniška izkušnja

Navadni uporabnik z aktivirano denarnico v vtičniku MetaMask obišče decentralizirano aplikacijo (*dApp*), v kateri vidi svojo identiteto in stanje sredstev (žetonov) ter aktivnosti omrežja s prikazom

pisanja v zadnje bloke. Tiste člene pogodbe, ki jih ni že prej podpisal, potrdi, če se z njimi strinja. To prikazuje slika 1. Kot dodaten varnostni ukrep je zraven posameznega člena dodan tudi zgoščen zapis tega člena (*keccak256*). Ko želi uporabnik oddati označene člene, to stori v dialogu za podpis členov. V potrdilu o transakciji so zapisani pomembni podatki o členih, identifikaciji transakcije ter identiteti lastnika in podpisovalca pametne pogodbe, kot je razvidno iz slike 2. Uporabnik ob kasnejših obiskih aplikacije vidi tudi že podpisane člene pogodbe, katerih pa ne more spremeniti.

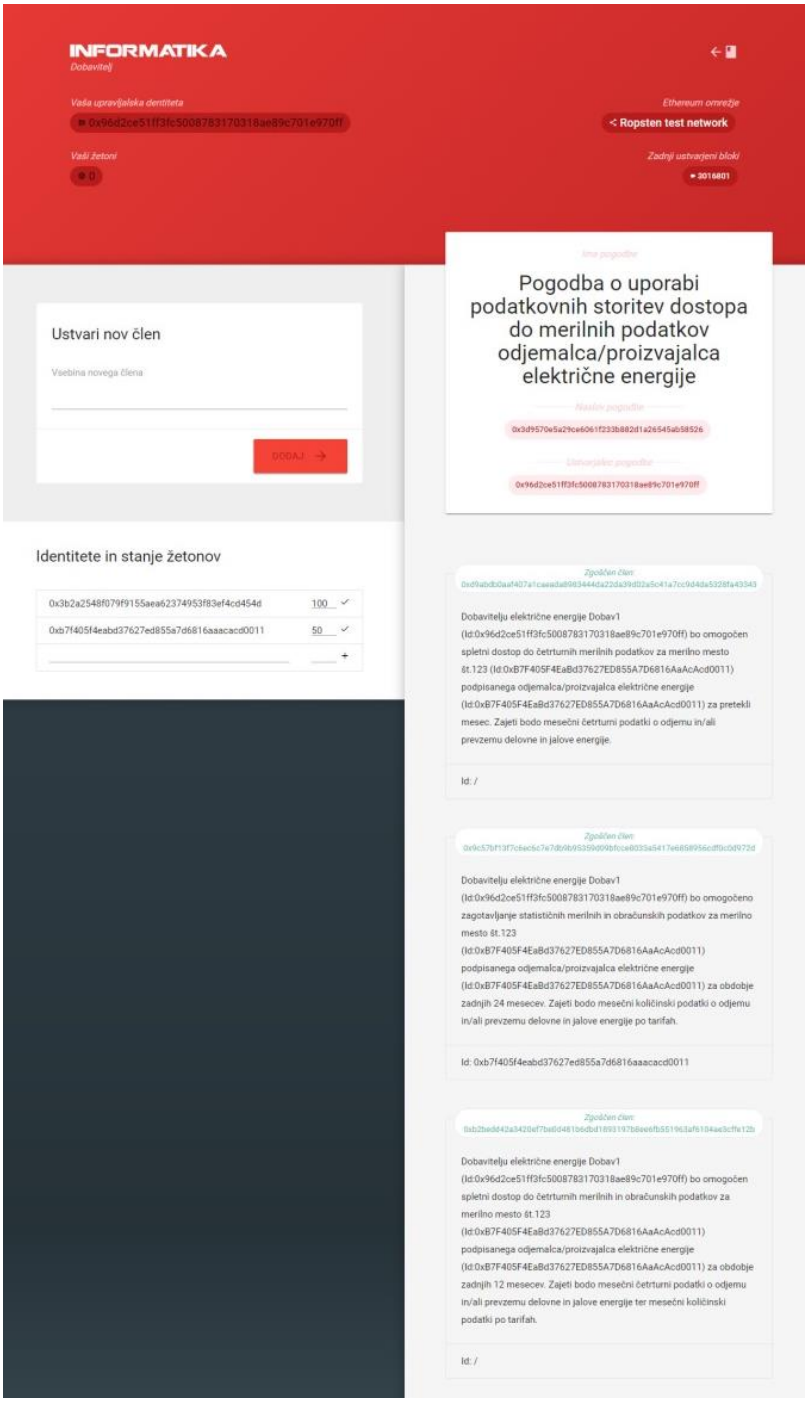


Slika 1. Spletni vmesnik za navadnega uporabnika s prikazom pametne pogodbe



Slika 2. Podpisovanje členov pametne pogodbe z vtičnikom MetaMask

Ustvarjalec, lastnik, ponudnik oziroma administrator pogodbe je tista identiteta, ki pogodbo objavi v omrežju Ethereum. Pozoren mora biti na to, da unikaten naslov pametne pogodbe skopira v aplikacijo, ki jo objavi na želenem javno dostopnem spletnem naslovu. Ko odpre aplikacijo, ta zazna njegovo identiteto. V kolikor se ta sklada z identiteto ustvarjalca pogodbe, dobi na voljo tudi upravljanje v administrativnem načinu, ki ga kaže slika 3. Tu lahko dodaja člene pogodbe in spremlja, katere identitete so podpisale posamezne člene, ne more pa jih spreminjati ali brisati. Kadar je za dostop do členov zahtevano določeno število žetonov, ki jih mora imeti navadni uporabnik v lasti, lahko določa te količine in dodeljuje žetone. Ker s tem zagotovimo nadzor nad dostopom, ni predvidena funkcija, da si uporabniki med seboj pošiljajo žetone.



Slika 3. Uporabniški vmesnik za dobavitelja električne energije

5. ANALIZA IZKUŠENJ Z UPORABO IN ZRELOSTJO TEHNOLOGIJE

Z uporabo tehnologije blokovnih verig na platformi Ethereum smo razmeroma hitro in brez večjih težav implementirali izbrani scenarij. V okviru tega scenarija se je tehnologija izkazala kot primerna in uporabna. Kljub temu bi lahko hitro naleteli na ovire, če bi za implementacijo izbrali katerega od kompleksnejših scenarijev, ki so identificirani v drugem poglavju članka. Kot kaže tabela 4, ki je delno povzeta iz literature [5, 13] in delno oblikovana na osnovi pridobljenih lastnih izkušenj, je ob

upoštevanju trenutne stopnje zrelosti uporaba tehnologije blokovnih verig kompromis med številnimi prednostmi in slabostmi.

Tabela 4. Prednosti in slabosti uporabe tehnologije veriženja blokov v energetskih informacijskih sistemih

Prednosti	**Slabosti**
• Nižji stroški transakcij in nižji računi za odjem energije zaradi možnosti trženja brez posredniških dobaviteljev • Padanje tržnih cen zaradi boljše transparentnosti transakcij v okviru veljavnih regulativ • Boljši in širši vpogled v transakcijske podatke o proizvodnih virih in odjemu zaradi decentralizacije v okviru regulativ • Poenostavitev poslovnih domen, pravil in procesov v modelih blokovnih verig, ki temeljijo na porazdeljenih transakcijah in pametnih pogodbah, v primerjavi z obstoječimi reguliranimi domenami in procesi • Poenostavitev in krepitev vloge odjemalcev proizvajalcev kot ponudnikov energije na trgu zaradi zmanjšanja pristojnosti posredniških dobaviteljev kot vmesnih členov • Boljši modeli samooskrbe v pametnih energetskih skupnostih • Boljši in enostavnejši modeli dobave energije za mobilnost • Fleksibilnejši tržni modeli in storitve • Preverljivost, varnost, redundantnost in integriteta podatkov ter nezmožnost njihovega ponarejanja • Uvedba mehanizmov za splošni konsenz in opustitev potrebe po zaupanju osrednji avtoriteti	• Popolna izguba podatkov v primeru izgube identitete • Nizka skalabilnost, počasnost in visoki stroški transakcij na nekaterih platformah veriženja blokov • Neprimernost tehnologije za nekatere scenarije uporabe • Neustrezen oziroma omejujoč regulativni okvir za nekatere scenarije uporabe • Funkcionalne pomanjkljivosti in varnostna tveganja zaradi nezadostne standardizacije in regulative • Pomanjkanje razvojnih izkušenj in dokumentacije • Možnost tehničnih težav pri vpeljavi tehnologije in uvedbi aplikativnih rešitev • Morebitna slaba sprejetost tehnologije pri uporabnikih zaradi kompleksnosti, nerazumevanja in nezaupanja • Ni osrednje avtoritete, ki bi reševala nesporazume • Zahtevana je večja fleksibilnost omrežij • Kibernetska tveganja zaradi možnosti zlorab na omrežnem nivoju, npr. pri upravljanju s pametnimi merilnimi napravami

Ker je tehnologija blokovnih verig v zadnjih letih pridobila na popularnosti, se pogosto uporablja kot sinonim za univerzalno rešitev za vse probleme v širokem naboru industrij. Vendar moramo pri tem biti pozorni, saj je tehnologija v zgodnji fazi razvoja, obenem pa ima številne pomanjkljivosti, ki se jih entuziasti sicer zavedajo, a vseeno premalo upoštevajo pri načrtovanju razvoja aplikacij in reševanju problemov. Frisby [4] in Marr [5] sta izpostavila nekaj ključnih pomanjkljivosti, na katere moramo biti pozorni pri razmisleku o reševanju problemov s pomočjo tehnologije blokovnih verig.

Tehnologija ne more biti vse, kar od nje zahtevamo. Kot vsaka tehnologija, je tudi ta ujeta med tremi ključnimi cilji, in sicer: hitrostjo, nizko ceno in decentraliziranostjo. Od tehnologije se pričakuje pokrivanje vseh treh kriterijev, kar pa ni mogoče. Največja težava je zagotavljati hitrost transakcij v distribuiranem okolju, kar je izjemno težko izpolnjiva zahteva, saj je decentraliziranost kot princip časovno zahtevna (hitre transakcije »*one-to-many*« in »*many-to-one*« zahtevajo izjemne komunikacijske in procesne zmogljivosti). Če bi želeli transakcijsko zmogljivost rešitev s tehnologijo blokovnih verig približati trenutnim sistemom, bi bilo potrebno zagotoviti visoko zmogljivo komunikacijsko in procesno okolje za nekaj deset tisoč transakcij v sekundi, kar pa bi vplivalo na proizvodno ceno transakcije. Uporabnost v okolju, v katerem je potrebno zagotavljati veliko število transakcij v kratkem času, je torej vprašljiva ali cenovno zahtevna.

V fazi načrtovanja je tako potrebna previdnost. Morebitne oglaševane hitrosti prenosa in zapisa v blokovne verige posameznih platform je potrebno ustrezno oceniti glede na dejanske zmožnosti platform v realnih scenarijih. Kadar je hitrost delovanja tehnologije bistven del poslovnega modela uporabe, je z veliko verjetnostjo pričakovati, da bo to največja težava pri izvedbi, saj tehnologija danes zanjo morda še ni zrela. Temu se kaže izogniti, tako da predvidimo uporabniške scenarije, ki niso popolnoma odvisni od pogostega in hitrega zapisovanja v bazo podatkov. Večino nazivnih hitrosti pri

novejših platformah so namreč dosegli pri testiranju v zaprtih omrežjih, z izredno majhnim številom vozlišč, z optimiziranimi zapisi z večjim številom transakcij na blok in neposredno povezavo.

Tehnologija je neučinkovita na področju podpore uporabnikom. Enostavni problemi, s katerimi se srečujemo ob uporabi programskih rešitev, kot je npr. izguba ali pozaba gesla, so trenutno rešljivi v nekaj minutah. Tehnologija blokovnih verig pa predvideva uporabo personaliziranih denarnic, ki so vezane na strojno opremo, npr. na trdi disk, in s tem posledično na uporabnika. Izguba ali pozaba gesla tako pomeni, da nimamo več dostopa do virov. Ni namreč nadrejene ali paralelne centralizirane avtoritete, ki bi zagotavljala podporo uporabnikom. Prav tako še ni izdelanih principov, ki so osnova zaupanja, kakršni so zahtevki po povračilu denarja, obravnava kraj in ukradene identitete ipd. Tehnologija blokovnih verig torej predpostavlja uporabnike, ki ne potrebujejo podpore in so nevtralni do vprašanj zaupanja, kar ni stvarna domneva.

Tehnologija uvaja težave v procese, ki trenutno delujejo brez težav. Internetna prodaja in kartično poslovanje sta skupaj z obstoječimi komunikacijskimi tehnologijami dosegla zadovoljivo raven ugodja uporabnikov in hitrosti, pri čemer lahko s transakcijami, ki trajajo med 2 in 10 sekundami, opravimo praktično vse, kar potrebujemo za operativno poslovanje ali življenje. Nekaj poizkusov, s katerimi so testirali alternativne rešitve na platformi Ethereum, pa je pokazalo, da hitrost in obseg transakcij še zdaleč ne dosegata željenega oziroma obstoječega nivoja, še zlasti zato, ker rastejo sistemske zahteve eksponencialno s številom uporabnikov. Iz tega lahko zaključimo, da iščemo alternativno rešitev za upravljanje s procesi, ki trenutno delujejo dobro. Kako torej najti vzvod za uporabo tehnologije?

Pomanjkanje regulative in nadzora ustvarja tvegano okolje. To okolje je izpostavljeno znatnim možnostim za kibernetske napade. Dokumentirani so številni primeri vdorov in zlorab, katerih posledica so nemalokrat tudi nezanemarljive finančne izgube. Prav tako se lahko zgodi, da so zakonodajna in vladna telesa prisiljena sprejti ukrepe proti posameznim platformam in aplikacijam veriženja blokov zaradi kibernetskih groženj ali poslovnih modelov, ki niso skladni z regulativo.

Kompleksnost tehnologije lahko povzroči, da uporabniki ne prepoznajo njenih prednosti. Za uporabnike so koncepti enkripcije, decentraliziranosti in porazdeljenosti, ki so temelj tehnologije blokovnih verig, pogosto zapleteni za razumevanje. Posledično ne dojamejo potencialov uporabnosti tehnologije. Marsikdo v modelu porazdeljenih transakcij in pametnih pogodb, ki odpravlja posredniške vloge, kakršne so finančne ustanove ali dobavitelji na energetskem trgu, ne razpozna prednosti. Brez zadostnega sprejemanja tehnologije pa le-ta težko v popolnosti zaživi in se dolgoročno obdrži.

Vse zapisano pomeni, da je potrebno pred uporabo tehnologije blokovnih verig resneje razmisliti o temeljnih problemih, ki jih poskušamo reševati z njo, in o omejitvah tehnologije. Sklepamo lahko, da tehnologija blokovnih verig ni univerzalen odgovor za vse probleme.

6. ZAKLJUČEK

Razvoj pilotske rešitve je pokazal, da je možno tehnologijo veriženja blokov razmeroma koristno uporabiti v različnih domenah, ki niso neposredno vezane na princip plačevanja s kriptovalutami, tudi v energetiki. Pri tem je na voljo zadovoljiva podpora večjega nabora obstoječih platform, ki omogočajo, da aplikacije dokaj hitro in enostavno implementiramo ter namestimo v produkcijsko okolje. Kljub temu je tehnologija zaenkrat še podvržena določenim omejitvam, zaradi katerih nekateri scenariji uporabe niso izvedljivi ali so izvedljivi le pogojno. Pri v popolnosti izvedljivih scenarijih se poraja vprašanje, ali ima njihova informatizacija na osnovi tehnologije veriženja blokov kakršnekoli bistvene konceptualne, tehnološke in razvojne prednosti pred bolj tradicionalnimi spletnimi, podatkovnimi, integracijskimi ter varnostnimi pristopi in tehnologijami. Za uvedbo rešitev z višjo inovativno in uporabno vrednostjo bo zato potrebno odpraviti pomanjkljivosti in omejitve tako na nivoju tehnologije, zlasti v smislu zrelosti in skalabilnosti, kot tudi na regulativno-poslovni ravni ter z vidika splošne sprejetosti. Če se bo v

naslednjih nekaj letih izkazalo, da so problemi predvsem tehnološke narave in je veriženje blokov resnično konceptualno ustrezno za širšo uporabo, za premostitev problemov morda ni večjih ovir. Tako so npr. tudi tehnologije inteligentnih sistemov, ki jih Gartner uvršča med nekaj najbolj aktualnih trendov v tem trenutku [8], v več desetletjih prehodile dolgo, sicer razmeroma uspešno razvojno in raziskovalno pot, pa so šele v zadnjem času zaradi dosežene ravni zrelosti, splošnega napredka informacijske tehnologije in širše sprejetosti dosegle bistven preskok in razcvet v poslovnem svetu.

7. LITERATURA

[1] BARTOLETTI Massimo, POMPIANU Livio »An empirical analysis of smart contracts: Platforms, applications, and design patterns«, Financial Cryptography and Data Security, Springer, 2017.

[2] BONTJE Joris »DApp design patterns: Emerging best practices in the world of Decentralized Applications«, 2015, www.slideshare.net/mids106/dapp-design-patterns, obiskano 18. 5. 2018.

[3] BREGAR Andrej, KAFOL Ciril »Sistematičen pristop h kibernetski varnosti v IT/OT-integriranem elektroenergetskem okolju na osnovi sektorskega varnostnega operativnega centra in tehnologij veriženja blokov«, Dnevi slovenske informatike DSI 2018: Zbornik 25. konference, Slovensko društvo Informatika, Portorož, 2018.

[4] FRISBY Adam »Why blockchain isn't ready for primetime«, venturebeat-com.cdn.ampproject.org/c/s/ venturebeat.com/2018/03/11/why-blockchain-isnt-ready-for-primetime/amp/, obiskano 15. 4. 2018.

[5] MARR Bernard »The 5 big problems with blockchain everyone should be aware of«, Forbes, februar 2018, www.forbes.com/sites/bernardmarr/2018/02/19/the-5-big-problems-with-blockchain-everyone-should-be-aware-of/#5adf0c661670, obiskano 16. 5. 2018.

[6] MEARIAN Lucas »5 ways blockchain is the new business collaboration tool«, Computerworld, februar 2018, www.computerworld.com/article/3197695/blockchain/5-ways-blockchain-is-the-new-business-collaboration-tool.html, obiskano 14. 5. 2018.

[7] MOSES Sam Paul »Deploy smart contracts on Ropsten Testnet through Ethereum Remix«, The Startup, marec 2016, medium.com/swlh/deploy-smart-contracts-on-ropsten-testnet-through-ethereum-remix-233cd1494b4b, obiskano 18. 5. 2018.

[8] PANETTA Kasey »Gartner's top 10 strategic technology trends for 2017«, Gartner, oktober 2018, www.gartner.com/smarterwithgartner/gartners-top-10-technology-trends-2017, obiskano 17. 5. 2018.

[9] PEČJAK Vid, Poti do idej: tehnike ustvarjalnega mišljenja v podjetjih, šolah in drugje, 1989.

[10] PRUSTY Narayan, Building Blockchain Projects: Building Decentralized Blockchain Applications with Ethereum and Solidity, Pack Publishing, 2017.

[11] SPENCER Donna, Card Sorting: Designing Usable Categories, Rosenfeld Media, 2009.

[12] SWAN Melanie, Blockchain: Blueprint for a New Economy, O'Reilly, 2015.

[13] »Blockchain – an opportunity for energy producers and consumers?«, PwC global power & utilities, november 2016, www.pwc.com/gx/en/industries/energy-utilities-resources/publications/opportunity-for-energy-producers.html, obiskano 10. 5. 2018.

[14] »Energetski zakon (uradno prečiščeno besedilo)«, Uradni list RS, št. 27/2007, 26. 3. 2007, www.uradni-list.si/glasilo-uradni-list-rs/vsebina?urlid=200727&objava=1351, obiskano 15. 5. 2018.

[15] »Regulativne spremembe za vzpostavitev nove vloge na trgu *aktivni odjemalec*«, Agencija za energijo, oktober 2017, www.agen-rs.si/documents/10926/106759/Regulativne-spremembe-za-vzpostavitev-nove-vloge-na-trgu-Aktivni-odjemalec/6a00e54d-e9c8-419f-99a4-e7e2dcf0c3b9, obiskano 14. 5. 2018.

[16] »Uredba 2016/679/EU: Splošna uredba o varstvu osebnih podatkov«, Evropski parlament in Svet, 27. 4. 2016, eur-lex.europa.eu/legal-content/SL/TXT/?uri=CELEX%3A32016R0679, obiskano 18. 5. 2018.

[17] bitcoinexchangeguide.com/proof-of-work-vs-proof-of-stake-mining, Proof of Work vs Proof of Stake – What is POW & POS mining?, obiskano 13. 5. 2018.

[18] en.wikipedia.org/wiki/Initial_coin_offering, Initial coin offering, obiskano 11. 5. 2018.

[19] en.wikipedia.org/wiki/Lightning_Network, Lightning network, obiskano 11. 5. 2018.

[20] eos.io, EOS.IO, obiskano 12. 5. 2018.

[21] entethalliance.org, Enterprise Ethereum Alliance, obiskano 13. 5. 2018.

[22] expressjs.com, Express framework for Node.js, obiskano 18. 5. 2018.

[23] github.com/ethereum/web3.js, Ethereum JavaScript API, obiskano 13. 5. 2018.

[24] github.com/ethereum/wiki/wiki/Proof-of-Stake-FAQ, Proof of Stake FAQ, obiskano 14. 5. 2018.

[25] github.com/ethereum/wiki/wiki/White-Paper, A Next-Generation Smart Contract and Decentralized Application Platform, Ethereum Whitepaper, obiskano 14. 5. 2018.

[26] hackernoon.com/kyc-aml-and-cryptocurrencies-4e4cf929c151, KYC, AML, and Cryptocurrencies, obiskano 18. 5. 2018.

[27] neo.org, NEO – An open network for smart economy, obiskano 12. 5. 2018.

[28] www.bitcoin.com, Bitcoin, obiskano 12. 5. 2018.

[29] www.cardano.org, Cardano, obiskano 12. 5. 2018.

[30] www.coinmarketcap.com, Top 100 cryptocurrencies by market capitalization, obiskano 11. 5. 2018.

[31] www.cryptocompare.com/coins/guides/what-is-the-gas-in-ethereum/, What is the »Gas« in Ethereum?, obiskano 13. 5. 2018.

[32] www.ebix.org, European forum for energy Business Information eXchange, obiskano 15. 5. 2018.

[33] www.ethereum.org, Ethereum, obiskano 12. 5. 2018.

[34] www.heroku.com, Heroku, obiskano 18. 5. 2018.

[35] www.hyperledger.org/projects/fabric, Hyperledger Fabric, obiskano 12. 5. 2018.

[36] www.iota.org, IOTA, obiskano 12. 5. 2018.

OTS 2018 SODOBNE INFORMACIJSKE TEHNOLOGIJE IN STORITVE
ZBORNIK TRIINDVAJSETE KONFERENCE, MARIBOR, 19. IN 20. JUNIJ 2018
M. HERIČKO IN K. KOUS

Univerzitetna založba Univerze v Mariboru

UPORABA TEHNOLOGIJE VERIŽENJA BLOKOV ZA UPRAVLJANJE EKOSISTEMA PODATKOV O VOZILIH

JAKA JENKO, ANDREJ MEH IN AMBROŽ STROPNIK

Povzetek: Prispevek opisuje eno od možnosti uporabe tehnologij veriženja blokov v okviru lastne rešitve IT Optitech za upravljanje podatkov o vozilih in sicer za potrebe hrambe podatkov o stanju vozila in sicer konceptualne vidika. V same tehnične podrobnosti implementacije ne zahajamo. Poleg uporabe tehnologije veriženja blokov, v prispevku predstavimo še platformo Optitech, katere osnovni namen je pridobivanje, ter obdelava podatkov iz vozil preko OBD II naprav v realnem času in je bistvena za razumevanje umestitve uporabe tehnologije veriženja blokov. Pri tem se dotaknemo še teoretičnega ozadja tehnologije veriženja blokov, vendar le toliko, da podamo vpogled, kaj smo pri implementaciji platforme Optitech uporabili.

Ključne besede: • tehnologija veriženja blokov • ekosistem podatkov o vozilih • register vozil

NASLOV AVTORJEV: Jaka Jenko, Kivi Com d.o.o., Kidričeva 3a, 2380 Slovenj Gradec, Slovenija, e-pošta: jaka.jenko@kivi.si. Andrej Meh, Kivi Com d.o.o., Kidričeva 3a, 2380 Slovenj Gradec, Slovenija, e-pošta: andrej.meh@kivi.si. dr. Ambrož Stropnik, Kivi Com d.o.o., Kidričeva 3a, 2380 Slovenj Gradec, Slovenija, e-pošta: ambroz@kivi.si

DOI https://doi.org/10.18690/978-961-286-162-9.4 ISBN 978-961-286-163-6

Dostopno na: http://press.um.si.

1. UVOD

Digitalizacija procesov, inteligentni sistemi, tehnologije veriženja blokov (angl. blockchain) in poslovanje s kriptovalutami so področja brez katerih si ne znamo več predstavljati sodobnih informacijskih sistemov in informacijskih storitev. Njihovo praktično uporabo zasledimo na vseh področjih, od uporabe v proizvodnih informacijskih sistemih, v zdravstvu in nenazadnje tudi v različnih sistemih namenjenih za obdelavo podatkov iz vozil.

V prispevku predstavljamo naš pogled na uporabo tehnologije veriženja blokov v okviru lastne rešitve za upravljanje ekosistema podatkov o vozilih imenovana Optitech, ki je sposobna pridobivati podatke o statusu vozila različnih proizvajalcev. Bistveni del uporabe tehnologij veriženja blokov pri upravljanju ekosistema podatkov Optitech o vozilih predstavlja potreba po beleženja statusa vozila (npr. opravljen servis vozila, beleženi prevoženi kilometri ob servisnih posegih, status vozila…). V osnovi lahko tehnologijo veriženja blokov označimo kot porazdeljeno podatkovno bazo, ki ima shranjene podatke na različnih strežnikih in onemogoča brisanje ali spreminjanje podatkov. Kot takšna je tehnologija veriženja blokov primerna za vodenje zgodovine prej omenjenih dogodkov vozil (npr. servisni posegi itd.), ki pa so v fazi prodaje rabljenega vozila bistvenega pomena (npr. vidna so odstopanja, če so bili na vozilu nazaj prevrteni kilometri). Preverljiva zgodovina vozila prodajalcu omogoča prodajo vozila po višji ceni, na drugi strani pa kupec točno ve kakšno vozilo kupuje, v kakšnem stanju je in kaj lahko od vozila pričakuje (npr. kdaj je naslednji veliki servis vozila).

Omenjeni način uporabe tehnologije veriženja blokov za potrebe beleženja stanja vozil ima tudi širši vpliv, saj lahko močno vpliva na sam trg rabljenih vozil. Konkretno, v Sloveniji namreč ni zakonsko obvezno beleženje statusa vozil pri servisnih posegih v centralni register (npr. stanja števca kilometrov – medtem, ko je v nekaterih državah Evropske unije zakonsko obvezno) in posledično tudi ni možno na enostaven način preveriti stanje rabljenega vozila. To omogoča prodajalcem rabljenih vozil prodajo vozil vprašljive kvalitete oz. stanja vozila (npr. prevrteni kilometri vozila itd.).

Uporaba tehnologije veriženja blokov v okviru ekosistema podatkov o vozilih Optitech za potrebe beleženja stanja vozil, vsekakor predstavlja novost na trgu tovrstnih IT rešitev in je tudi bistvo tega prispevka. V poglavju 2 najprej predstavimo stanje sorodnih IT rešitev. Poglavje 3 je namenjeno predstavitvi tehnologij veriženja blokov, ki so uporabljene kot del naše IT rešitve Optitech. Jedro prispevka predstavlja poglavje 4, kjer je predstavljena IT rešitev Optitech.

2. SORODNE IT REŠITVE

Področje sistemov za obdelavo podatkov iz vozil je v zadnjih nekaj letih naredilo ogromen korak naprej, saj dandanes že težko najdemo vozilo, ki ne komunicira s podatkovnim centrom proizvajalca vozila ali v nekaterih primerih tudi z drugimi vozili. Posledično se uporabniki sodobnih vozil srečujejo s storitvami kot so: elektronska servisna knjiga, elektronsko spremljanje servisnih intervalov, sledenje lokacije vozila, upravljanje vozila preko mobilnega telefona (npr. aktivacija gretja, odpiranje/zapiranje oken vozila…), delna diagnostika vozila – napake/opozorila (nepripet pas), naročilo na servis itd… [1][2][3] Pri tem je potrebno opozoriti, da so vse omenjene funkcionalnosti v različnem obsegu na voljo v odvisnosti od proizvajalca vozila, ki so znana pod različnimi znamkami, kot so: BMW ConnectedDrive [1], Škoda Connect [2], Volkswagen Connect [3]… Še več, nekateri proizvajalci vozil so šli tako daleč, da za dostop do podatkov ponujajo tudi integracijske API-je (angl. Application Programming Interface) preko katerih je omogočen dostop oz. izmenjava podatkov o vozilih z drugimi aplikacijami [4][5].

Obstajajo pa tudi IT rešitve za zajemanje podatkov iz vozil (in kasneje tudi njihovo obdelavo), ki so neodvisne od proizvajalca vozil kot tudi letnika izdelave vozila. Ena najbolj znanih in najbolj popularnih aplikacij je Munic.io [6], ki omogoča spremljanje statusa vozila neodvisno od proizvajalca vozila, pri čemer je za uspešno delovanje aplikacije potrebno uporabljati njihovo napravo za branje podatkov o vozilu preko OBD II vtičnika vozila (angl. OBD II Reader). Tako naprava nameščena v vozilu preko

mobilnega omrežja pošilja podatke o vozilu na strežnik, tam pa se potem ti podatki ustrezno obdelajo in na skladen način tudi predstavijo uporabniku [6].

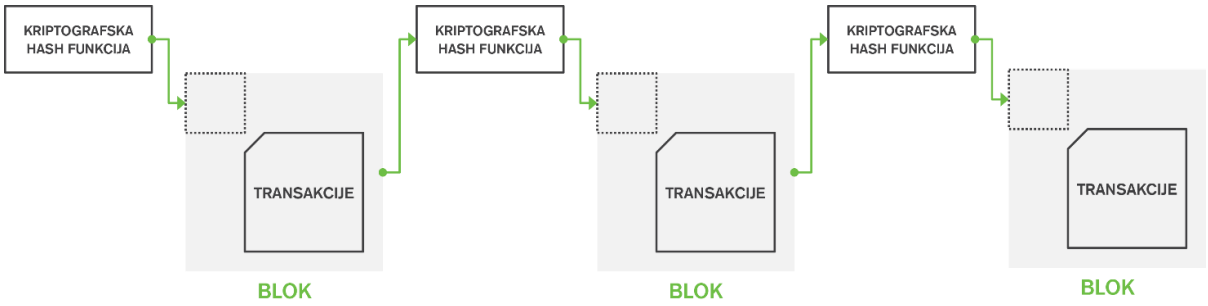
V zadnjih dveh do treh letih je moč zaznati velik porast zagonskih projektov (angl. Start-up), ki sredstva za izvedbo projekta pridobivajo z izdajo lastne kriptovalute (angl. ICO - Initial Coin Offering). V tem segmentu je kar nekaj projektov s področja upravljanja podatkov o vozilih, ki so še na idejni ravni oz. še nerealizirani (npr. Gluon [7]).

Opažamo pa, da na trgu manjkajo IT rešitve, ki bi omogočale pridobivanje telemetričnih podatkov iz različnih vozil in omogočale raznovrstne obdelave teh podatkov, ki bi nudile uporabnikom in lastnikom vozil dodano vrednost (npr. celotno zgodovino vozila, vključno z dogodki).

3. TEHNOLOGIJA VERIŽENJA BLOKOV

Tehnologija veriženja blokov (ang. blockchain) je javen zapisnik transakcij. Transakcije se dodajajo v bloke (angl. block), ki se povezujejo med seboj, in tako tvorijo verigo urejeno po času nastanka posameznega bloka oz. transakcije [8]. Tehnologija uporablja kriptografijo za zagotavljanje nespremenljivosti dodanih blokov.

Vsak blok je s svojim predhodnikom povezan z uporabo kriptografskega »hash-a« predhodnega bloka. Tako povezana, neprekinjena veriga blokov zgotavlja, da bloka, po tem, ko je enkrat dodan v verigo, ni več moč spremeniti. Vsaka sprememba bi namreč spremenila kriptografski hash (angl. cryptographic hash), in na ta način prekinila povezavo z naslednikom in vsemi sledečimi bloki [8].



Slika 1: Shema veriženja blokov

Tehnologija veriženja blokov je načrtovana z namenom zagotavljanja visoke varnosti in je kot taka idealna za vodenje registra transakcij. Omogoča izvajanje transakcij med anonimnimi strankami, brez potrebe po zaupanju oz. po zaupanja vrednemu mediatorju. Vsak računalnik v P2P povezanem omrežju (vozlišče, angl. node) k sebi prenese kopijo verige blokov. Nobeno od vozlišč nima osrednje vloge nadzora ali avtoritete [8]. Vsako od sodelujočih vozlišč neodvisno od drugih potrdi novo dodane bloke, zato ni potrebe po zaupanju.

Omenjene lastnosti tehnologije veriženja blokov delajo tehnologijo idealno za uporabo v tehnoloških rešitvah, ki morajo zagotavljati sledljivost in zasebnost, kot so obdelava transakcij, medicinske kartoteke ali dokumentacija procesov, saj omogočajo delovanje brez centralizirane avtoritete, kateri je potrebno zaupati. Delovanje na globalnem trgu, brez potrebe po centralizirani avtoriteti, ter z omogočanjem transakcij in izvajanjem pogodb je velika prednost.

3.1. Platforma Ethereum

Ethereum je ena od najbolj popularnih platform v kripto-svetu. Gre za odprtokodno, javno porazdeljeno platformo. Ima močno podporo s strani razvijalcev, med drugim tudi za projekte zbiranja sredstev.

Na Ethereum platformi je osnovni žeton (angl. token) za izmenjavo vrednosti imenovan »ether« oz. krajše ETH. Z ETH se trguje na večini kripto-borz, kot so Coinbase, BitStamp, Coinspot in številne druge. Na borzah se lahko ether kupi s FIAT valutami ali zamenja za druge kripto-valute. Etherum žeton je drugi po vrsti po skupni vrednosti trga.

Najpomembnejša razlika platforme Ethereum od ostalih kriptovalut oz. platform je v omogočanju pametnih pogodb (angl. smart contracts) [9]. Druga pomembna razlika je krajši interval dodajanja novih blokov transakcij (ang. block time) kar omogoča hitrejše potrjevanje opravljenih transakcij. Ethereum platforma je bila prva oz. ena od prvih z idejo izvajanja uporabniške kode na porazdeljenem omrežju in kot taka predstavlja globalno porazdeljen računalnik ali "the world computer". Možnosti, ki se odpirajo s takšnim konceptom niti ni potrebno naštevati.

Hkrati pa se pojavlja tudi učinek snežne kepe, saj veliko število razvijalcev in posledično tudi vse več virov za pomoč olajša vstop novim razvijalcem, in jih tako pritegne še več.

3.2. Pametne pogodbe in tehnologija veriženja blokov

Pametne pogodbe so v programsko kodo zapisane pogodbe, ki natančno določajo pogoje po katerih si stranki določita sodelovanje oz. medsebojni odnos. Ko je pametna pogodba potrjena s strani vseh udeležencev, je ni možno več spreminjati – to je zagotovljeno s strani tehnologije veriženja blokov. Pogodba se izvrši samodejno, ko so vsi v pogodbi določeni pogoji tudi izpolnjeni. Koda pametne pogodbe se izvaja na decentraliziranem Ethereum omrežju. Transakcije, ki jih izvede pametna pogodba so varovane in zaščitene brez potrebe po neodvisni zunanji entiteti, ki bi posegala v delovanje. Vse izvedene transakcije so nepovratne in povsem transparentne.

3.3. Standard ERC20

ERC 20 je široko sprejet in podprt standard, z uporabo katerega zagotovimo, da je izdan žeton kompatibilen z veliko večino obstoječih denarnic, borz, in aplikacij. Standard definira pravila katerih se mora držati žeton na Ethereum omrežju.

Pravila definirajo šest standardnih metod, ki jih mora podpirati žeton za uspešno delovanje v Ethereum ekosistemu. Te metode predstavljajo vmesnik preko katerega okolica izvaja operacije z žetoni [10]. Trenutno je to najbolj razširjen standard, in kot tak skoraj obvezen za uspešno delovanje znotraj omrežja.

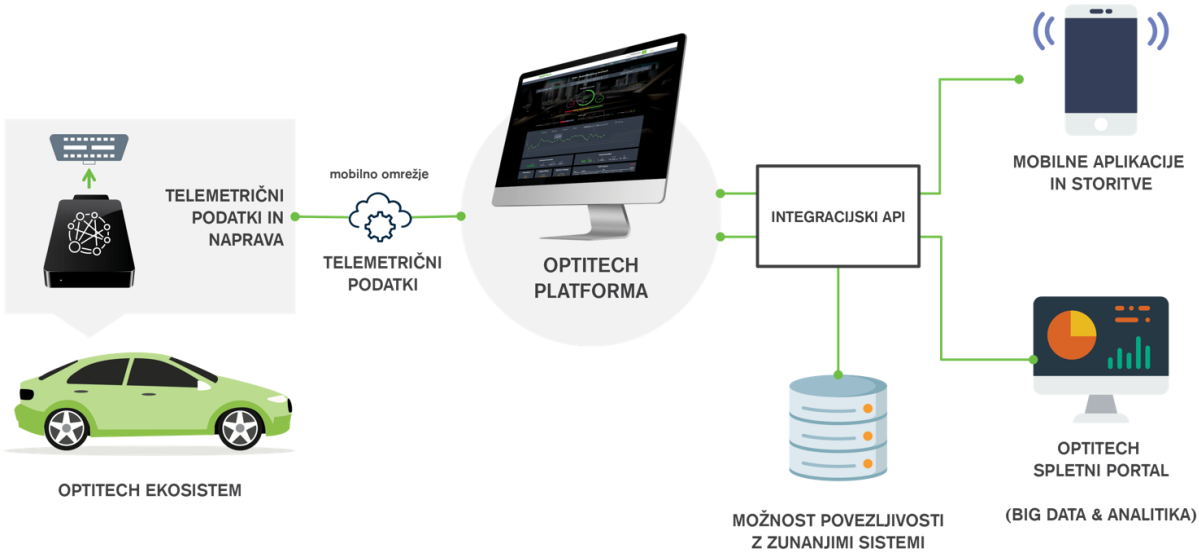
4. EKOSISTEM UPRAVLJANJA S PODATKI O VOZILIH – OPTITECH

Platforma Optitech je lastni produkt, ki je namenjen pridobivanju in obdelavi podatkov iz vozil različnih proizvajalcev preko OBD II naprav različnih ponudnikov, ki temelji na tehnologiji veriženja blokov. Razviti produkt je v celoti plod lastnega več letnega razvoja in je trenutno v fazi obširnega testiranja in verificiranja delovanja v povezavi z našimi poslovnimi partnerji.

Platforma Optitech sestoji iz naslednjih komponent:

- **OBD II "plug & play" naprava v vozilu (Onboard Diagnostic connector)**, ki je povezana s platformo Optitech preko mobilnega omrežja in sproti zajema podatke o vožnji in vozilu ter jih posreduje v procesni center Optitech,
- **analitični sistem**, ki obdeluje in pripravlja podatke za potrebe različnih skupin uporabnikov ekosistema in
- **portal** Optitech, ki omogoča preglede podatkov in upravljanje vseh delov sistema,

- **integracijski API**, ki omogoča povezljivost platforme Optitech z drugimi aplikacijami in zunanjimi sistemi (npr. mobilna aplikacija…) .

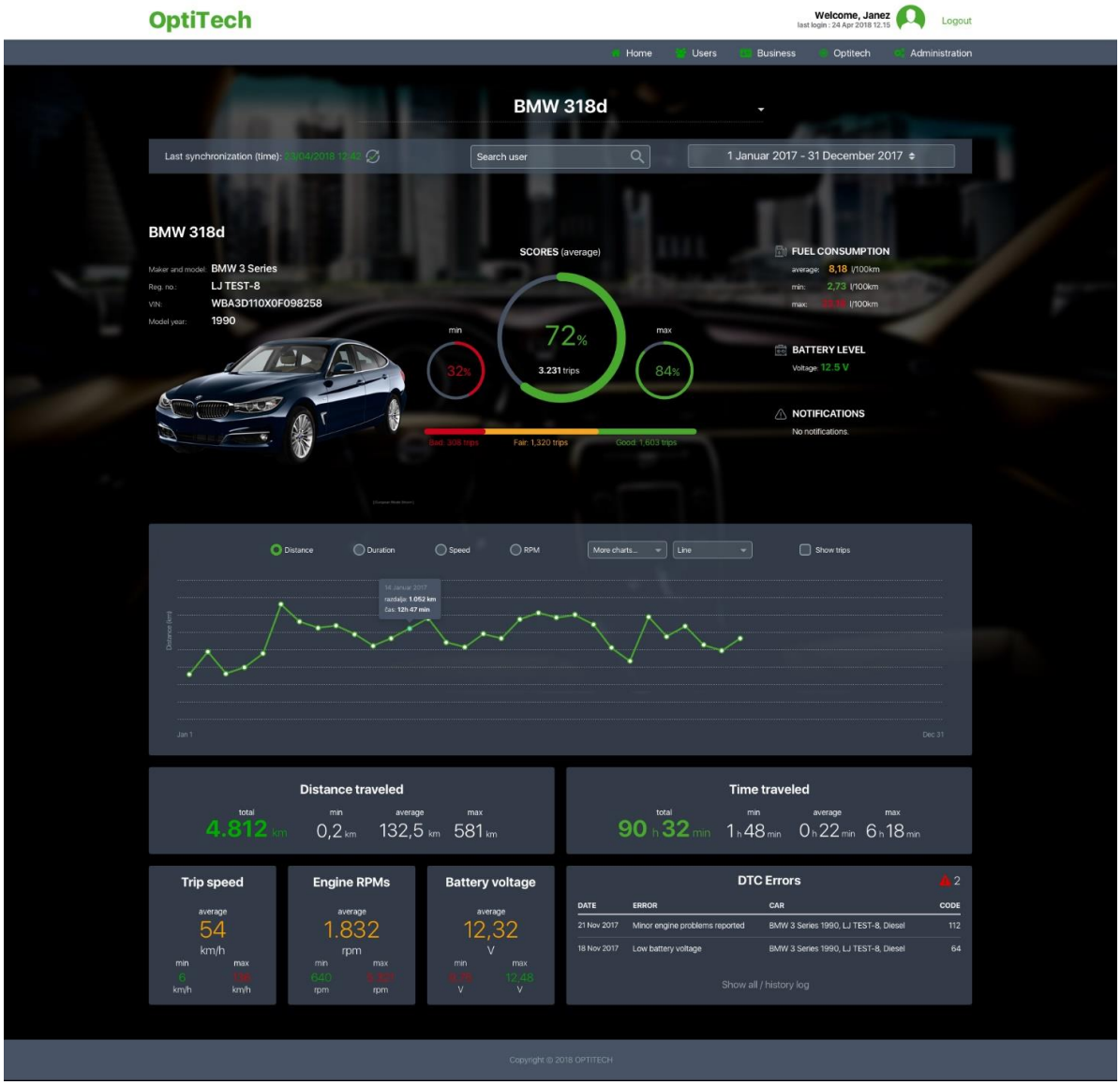


Slika 2: Ekosistem OPTITECH

Platforma Optitech je namenjena različnim ciljnim skupinam uporabnikom in sicer:
- poslovnim subjektom za upravljanje in nadzor lastnega voznega parka,
- fizičnim osebam za nadzor nad svojim(i) vozilom,
- servisnim hišam in serviserjem vozil za lažjo diagnostiko vozil in spremljanje statusa vozila (ustrezno predvidevanje in načrtovanje vzdrževanja vozila, beleženje prevoženih kilometrov, opravljen servis vozila…),
- drugi potencialne uporabniške skupine, ki pri svojem delu operirajo s podatki vozil (npr. zavarovalnice, prodajalci vozil itd…).

Na spodnji sliki (slika 3) je prikazan primer zaslonske maske portala namenjenega fizičnim osebam, ki zajema prikaz vseh podatkov, ki jih platforma Optitech zajema iz vozila in na ustrezni način obdela, ter prikaže uporabniku na zaslon. Podatki, ki jih ima uporabnik na zaslonu so sledeči:
- dnevnik vožnje,
- ocenjevanje voznika glede na podatke o vožnji in uporabi vozila (angl. Drive Score),
- učinkovitost vožnje in porabe
- diagnostika avtomobila (kode DTC – angl. Diagnostic Trouble Code),
- predvidevanje in načrtovanje vzdrževanja vozila.

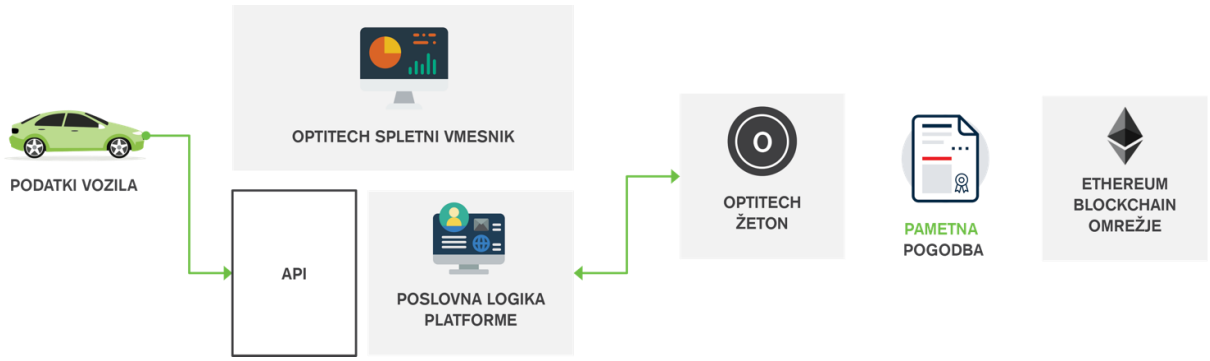
Dostop do omenjenih podatkov imajo vse omenjene ciljne skupine uporabnikov. Razlika je samo v načinu prikaza teh podatkov, ki je odvisen od uporabniških pravic in načina dostopa (npr. poročila, agregirani podatki v primeru voznih parkov…).



Slika 3: Uporabniški vmesnik portala Optitech

4.1. Arhitektura IT rešitve

Arhitektura ekosistema Optitech je klasično tri slojna. Slika spodaj (slika 4) prikazuje sistemsko shemo arhitekture ekosistema Optitech.



Slika 4: Arhitektura IT rešitve

Predstavitvena logika ekosistema Optitech je implementirana kot spletna aplikacija, temelječa na tehnologiji Angular in BootStrap, ter je preko ustreznih API-jev, temelječih na JSON tehnologiji (angl. JavaScript Object Notation), integrirana z zalednim sistemom (angl back-end).

Zaledni sistem, natančneje poslovna logika sistema (angl. Business Logic) temelji na Microsoftovih tehnologijah. Celotni zaledni sistem je implementiran s pomočjo programskega jezika C#. Za potrebe integracije z Ethereum blockchain omrežjem, je uporabljena programska knjižnica Nethereum.

Podatkovni nivo (angl. Data Logic) tvorijo trije podatkovni sistemi in sicer primarni podatkovni sistem predstavlja Microsoft SQL Server 2017, v katerem so shranjeni bistveni podatki za delovanje sistema Optitech (npr. uporabniki, razni šifranti itd.). Drugi podatkovni sistem predstavlja Ethereum blockchain omrežje, ki predstavlja bistveni del sistema Optitech, saj se vanj shranjujejo podatki o statusu vozil. Tretji podatkovni sistem pa predstavlja datotečni sistem strežnika v okviru katerega so shranjeni različni dokumenti v fizični obliki (npr. pdf).

Zelo pomemben del arhitekture sistema Optitech predstavlja integracijski vmesnik (API), ki je namenjen integraciji z drugi aplikacijami in sistemi (npr. mobilna aplikacija, analitični sistem, diagnostični sistem itd…). Slednji omogoča izmenjavo podatkov tako z uporabno JSON tehnologije, kot z uporabo klasičnih spletnih storitev (angl. web service). V okviru integracijskega API-ja je implementiran tudi integracijski vmesnik za izmenjavo podatkov med sistemom Optitech in OBD II napravo, ki je nameščena v vozilu. Pri tem se za potrebe integracije uporablja tehnologija omrežnih vtičnikov (angl. network socket).

4.2. Uporaba tehnologije veriženja blokov v projektu Optitech

Tehnologija veriženja blokov poleg povezljivosti z različnimi OBD II napravami za zajem telemetričnih podatkov iz vozil, predstavlja ključno tehnologijo platforme Optitech. Vsi podatki vezani na vozilo se shranijo v Ethereum BlockChain omrežje, ki zagotavljala decentralizirano in porazdeljeno hranjenje podatkov o vozilu. V Ethereum BlockChain omrežje se ne shranijo nobeni osebni podatki, ampak izključno podatki vezani na vozilo in sicer:

- prevoženi kilometri vozila (t.i. odometer),
- opravljeni posegi na vozilu (npr. opravljen redni servis – kaj je bilo opravljeno, kateri deli so bili zamenjani, katero motorno olje uporabljeno itd. - t.i. elektronska servisna knjiga),
- dogodki vezani na vozilo (katere kode DTC so bile aktivne in kdaj je bila napaka odpravljena),
- ocena voznika (kako je bil avto vožen),
- dnevnik vožnje (na kakšne razdalje je bil avto vožen).

Na omenjen način dobimo celostni pogled na stanje vozila, ki je izjemnega pomena ob prodaji vozila ali menjave ponudnika servisnih storitev. Iz tehnološkega vidika, tehnologije veriženja blokov poleg že znane porazdeljene hrambe podatkov onemogočajo brisanje ali spreminjanje že shranjenih podatkov, kar pa pri zagotavljanju zgodovine vozila predstavlja bistveno funkcionalnost.

5. ZAKLJUČEK

V prispevku smo predstavili našo IT rešitev za upravljanje ekosistema podatkov o vozilih imenovano Optitech, ki temelji na tehnologiji veriženja blokov. Predstavili smo predvsem zastavljeno širino IT rešitve in eno od možnosti uporabe tehnologije veriženja blokov v okviru ekosistema upravljanja s podatki o vozilih, ki lahko bistveno vpliva na trg rabljenih vozil. Nadaljnje delo bo usmerjeno predvsem v širjenje različnih storitev v okviru v okviru IT rešitve Optitech (npr. storitve za servisne hiše, diagnostika vozil itd.), s katerimi je cilj doseči množično uporabo rešitve.

6. LITERATURA

[1] https://www.bmw-connecteddrive.at/app/index.html#/portal, BMW ConnectedDrive, obiskano 13.5.2018.

[2] https://www.skoda.si/skoda-connect/paketi-skoda-connect, Škoda Connect, obiskano 13.5.2018.

[3] https://www.vwconnect.com/, Volkswagen Connect, obiskano 13.5.2018.

[4] https://www.w3.org/Submission/2016/01/, Volkswagen Infotainment Web Interface, obiskano 3.1.2018.

[5] https://www.w3.org/Submission/viwi-protocol/, Volkswagen Infotainment Web Interface protocol definition, obiskano 3.1.2018.

[6] https://www.munic.io/, Munic.io, obiskano: 3.1.2018.

[7] https://www.gluon.com/, Gluon, obiskano: 18.3.2018.

[8] Satoshi Nakamoto, Bitcoin: A Peer-to-Peer Electronic Cash System, 2008

[9] Vitalik Buterin, Ethereum white paper - a next generation smart contract and decentralized application platform, 2013.

[10] Fabian Vogelsteller, Vitalik Buterin, ERC-20 Standard, 2015.

OTS 2018 SODOBNE INFORMACIJSKE TEHNOLOGIJE IN STORITVE
ZBORNIK TRIINDVAJSETE KONFERENCE, MARIBOR, 19. IN 20. JUNIJ 2018
M. HERIČKO IN K. KOUS

Univerzitetna založba
Univerze v Mariboru

BUILDING AN OPEN-SOURCE BLOCKCHAIN ECOSYSTEM WITH ARK

KRISTJAN KOŠIČ, ROK ČERNEC, ALEX BARNSLEY AND FRANCOIS-XAVIRER THOORENS

Abstract: Ark aims to create an entire ecosystem of linked chains by providing easy to use tools to deploy your own blockchain. Being highly flexible and adaptable, it allows products to be adopted by the general public much quicker and smoother. By having open source code, it is available to anyone who wants to contribute, or to build their own blockchain based on Ark technology stack. With a new block created every 8 seconds, its transactions are incredibly fast! In this paper we will present the building foundation of ARK Ecosystem, the decentralized organization behind ARK token, what are the challenges we are currently facing in general with blockchain technology and what is the role of the open-source community (ARK community fund, GitHub bounty program). We will also cover basics about the new ARK Core v2, which is a complete rewrite of the core blockchain code, addressed to deliver our vision of blockchain technology.

Key words: • ark • blockchain • open-source software • crypto • programming • javascript • architecture

AUTHORS: Kristjan Košič, core engineer at ARK Ecosystem, e-mail: kristjan@ark.io. Rok Černec, COO and board member at ARK Ecosystem, e-mail: rok@ark.io. Alex Barnsley, full-stack developer at ARK Ecosystem, e-mail: alex@ark.io. Francois-Xavier Thoorens CTO at ARK Ecosystem, e-mail: fx@ark.io.

DOI https://doi.org/10.18690/978-961-286-162-9.5 ISBN 978-961-286-163-6

Dostopno na: http://press.um.si.

1. INTRODUCTION

We are ten years into the decentralisation revolution, since the publication of Satoshi's paper and the introduction of the first decentralised solution to solve the problem of double-spending in digital networks. Currently we are moving through the phase of disillusionment where interest wanes as experiments and implementations fail to deliver. Producers of the technology shake out or fail. Further investment continues only if the surviving providers improve their products to the satisfaction of early adopters[1] [2]. Blockchain project are moving from experimentation around decentralised technologies to complete solutions including consensus mechanisms, identity, data structures, crypto-economic designs and smart contracts. In their convergence report Outlier Ventures[9] stated that: "*Most projects will fail, but the open-source nature of the ecosystem means learnings and code will be available to all*". We can learn and build faster than ever.

There is still a ton of hype around blockchain technology. You can read about how blockchain will erase global hunger, make world corruption free, end poverty - all with seamless technology delivered out of the box. Of course, we believe in such promises in the long run, but in order to deliver this with blockchain technology, we have to make it usable, configurable and seamlessly adjustable. Never before has a "back-end" technology gotten so much attention in media and well - we can basically read about it everywhere [3].

Should a common man have heard about blockchain and blockchain technology? We think not, however the people delivering this technology are still in the so cold "dark-ages" where first tools are still about to be forged in order to deliver the promise that blockchain technology can be harvested to disrupt and engage communities with new trustless solutions and business models that are yet to be discovered [4].

We don't want new blockchain solutions just to replace VISA or Mastercard or other payments systems. Let's be honest, these are optimized and mature solutions doing fastest payments in the traditional economy space. Many speed comparisons in the form of "TPS - transactions per second" between current systems and other blockchain platforms [5] can be found. At first, this looks like a very important piece of information, but in order to deliver the trustless promise, via transactions based on P2P (peer-to-peer network communication)[6], we know that it is possible, but in this case the network itself is not decentralized, but already centralized with fast central servers (nodes) delivering and confirming transactions. These are the models that big corporations will use and we already see them going in this direction with R3 Corda [7], IBM HyperLedger and similar technology [8].

The most common question we hear is "What can blockchain do that other technologies cannot do?" Is it fair to expect a 9 year old technology to outperform all other technologies in the world? [9] And yet, it already does on so many levels, the challenge we are facing is how to make this technology properly architectured and in the end user-friendly in order to deliver mass adoption of this so called "back-end" technology.

In the next section we present ARK ecosystem and how it contributes to the blockchain landscape today.

1.1 How does ARK add value to the blockchain tech landscape?

ARK[10] is looking to eliminate the barrier to entry in the blockchain space that is caused by the immense complexity of the technology. We want ARK to do for blockchain what WordPress did for website development. We are making that a reality with our focus on push-button blockchain technology and by steering our efforts toward meeting the needs of enterprises, start-ups, and developers as our prime customer base first and foremost. ARK will enable simplified deployment of custom blockchains while allowing the deployment of a wide array of services.

[9] Convergence Ecosystem report by Outlier ventures https://outlierventures.io/wp-content/uploads/2018/03/The_Convergence_Ecosystem_Report_Outlier_Ventures_2018.pdf

ARK's value proposition lies within that core platform and the utility of the services we are building into the ecosystem. Services such as the capability of connecting to different blockchains via our SmartBridge technology (see Figure 1). We have successfully connected to Ethereum, Bitcoin, and Litecoin blockchains with more in development. SmartBridge allows ARK to move data from one blockchain to another with the use of special Encoded Listener Nodes that can interpret and process data back and forth between different chains. With the integration of one of our next milestones, ArkVM, all ARK blockchains will be capable of utilizing smart-contracts as well and we are concurrently working on an IPFS based blockchain storage solution.

The ARK platform will be a great place for future blockchain developers to get their feet wet in this budding industry. ARK will allow them to customize a blockchain for their businesses with the features their companies need right at their fingertips, with an all-in-one ARK blockchain sandbox.

Figure 1: ARK SmartBridge concept Sending BTC to an ARK Address

In the next section we will briefly cover Open-source software (OSS)[11], it's concepts and challenges of developing software solutions out in the open, and how do we use the OSS model to deliver one of the best blockchain platforms out there.

2. DEVELOPING OPEN-SOURCE SOLUTIONS BASED ON ARK ECOSYSTEM

If we look at the definition of OSS: "***Open-source software*** *(**OSS**) is a type of computer software whose source code is released under a license in which the copyright holder grants users the rights to study, change, and distribute the software to anyone and for any purpose*" [12] . With open-source software, anyone is allowed to create modifications of it, port it to new operating systems and instruction set architectures, share it with others or, in some cases, market it.

Free and open source software developing models have made it possible to form new virtual teams, and enable team work between people, who may have not even been acquainted before, to help each other and to follow a common goal.

Figure 2. Why Open-source software [13]

According to a study by InfoSys [13] more than 78 percent of enterprises run on open source and fewer than 3 percent indicate they don't rely on open source software in any way. IT companies are in the phase of a big shift and transformation in the OSS arena with a healthy and positive mindset that *Open Source Software Gains in Strategic Value.* Big companies like Walmart, GE, Merck, Goldman Sachs, Google and Facebook are also analyzing and moving towards open software [14]. OSS is the core engine of innovation and can be seen as a first approach for enterprise architecture.

Of course, there are ups and downs of OSS, and it depends if you are a developer of OSS or a consumer, as most of the big companies mentioned above. Looking from the consumer side you have more to gain. Where we see the magic happening and how the value is added by observing and helping OSS teams to follow a common goal. Of the reasons why ARK is an "ecosystem" is the fact that **ARK is a living and breathing organism.** If we look at the definition of an ecosystem: *"An ecosystem is a community made up of living organisms and nonliving components such as air, water, and mineral soil*"[15], we can mirror the definition to **ARK - where the living organism is our community with more that 12,000 active users**. Non-living components are presented in the form of tools we use and the products we deliver (ARK blockchain and ARK SDKs in more than 20 programming languages). Our work process is built around distributed best practices, which are promoted and used by GitHub and other big opensource projects. All our code[10] is free, issued under the MIT OSS license.

ARK will continue to support community outreach initiatives through partnerships with college hackathons, local meetups, and by sponsoring major conferences. We are doing anything we can to preach the gospel of blockchain to developers, hobbyists, or really anyone who will listen. We will also be advising and helping new blockchain projects that want to develop their use cases and be powered by ARK technology. One of the good examples of how the OSS model connects and motivates people is ARK Community fund.

As mentioned by Richard Burton: "The hardest mental leap for people when they join crypto is the move from closed to open. Code is worth almost nothing. Community is worth everything. Anyone can fork the code. Very few people can fork a community. Internalising that reality is just impossible for some people ."

ARK realized that from the start and defined the building blocks of the ecosystem, together with the community. The result is one of the strongest communities in crypto space. With more than 12,000 daily active users you get all the benefits (see Figure 1) of the OSS model delivered to you. Cost reduction, quality improvement and quicker time to market are just some of the gains you get by joining our

[10] GitHub Ark Ecosystem - https://github.com/arkecosystem

community. If you want to develop solutions on top of ARK, there is always someone available and willing to help you out. It's like a friendly call centre, where everyone helps you and you all work together, of course help is for free. On the OSS developer side, the code (product) is more secure, as it is constantly exposed and under eyes of the experts or hackers that want to gain, by harming the network.

2.1 ARK community fund - ACF

The ACF[11] is an entirely community owned and operated vehicle for matching developers and entrepreneurs with potential seed funding through the use of the ARK ecosystem. The ARK Community Fund (ACF) is a fund that is run by the ARK community members[16]. It can be seen as supplemental to the core development of ARK ecosystem and shall support the ideas and projects of ARK community members to help promote the advancement of the ARK Ecosystem. The ACF has two main purposes[16]:

- to provide a possibility to those ARK community members, that would like to support the ARK Ecosystem with contributions of ARK tokens for community purposes.
- to provide a place for community developers to request funding, for projects to develop tools, software or hardware which supports the ARK Ecosystem.

Potential projects are able to submit applications to the team running the ACF to be considered for funding which can range from simple apps to more complex an custom solutions, as well as marketing efforts to help with recognition of ARK brand across the globe.

3. INTEROPERABILITY AS ONE OF THE BIGGEST BLOCKCHAIN CHALLENGES

Blockchain is HERE! In order for the blockchain technology to be mass adopted we need to deliver seamless communication between different blockchains. Richard Brown, CTO at R3 Corda, defined this as: *Universal interoperability - business needs the universal interoperability of public networks with the privacy and power of private networks [17]*. Interoperability simply means inter- and cross-blockchain communication.

ARK is at its core defined to address the challenge of interoperability with so called SmartBridge field. ARK ACES - Aces Contract Execution Services[18] is a perfect example of a solution addressing interoperability issues between different blockchains and is developed as a community project for anyone to join in and help improve, add new features or report issues faced when using ACES (see section 3.2).

Interoperability should be addressed at different levels. Looking at ARK technology stack we are addressing this with ARK ACES for cross-chain communication. As for interchain communication (two different blockchains with the same technology stack should be able to "talk" with each other - i.e. enabling atomic swaps, sharing data, using the same consensus mechanism). We don't want to bloat the main network (the famous CryptoKittes [19] example) by pushing all the stakeholders on the same level of technology - there is no blockchain to rule them all. To address this we deliver options to clone our technology (by using ARK Deployer) and enable users, local communities, and other value chains to start their own network where the most of the workload will be done, and when needed the communication with master chain will be used. By using the master chain, all the bridgechains gain the benefits, such as being already listed on exchanges, established trust and presence, all the technology in the back-end can be reused.

3.1 ARK deployer

One of the key tenants of ARK's business model is the Push-Button blockchain deployment. Our goal for ARK is to create the core experience and modular ecosystem that does for blockchain what

[11] ARK community fund - https://arkcommunity.fund/

WordPress has done for websites and blogging. ARK Deployer is a lightweight deployment script for creating your own ARK based blockchains - bridgechain. By utilizing the ARK Deployer, developers can create their own blockchain in a matter of minutes. ARK Deployer is just the first step in building a more robust ecosystem that will be user friendly, customizable, and will feature the same caliber of user experience you have all come to expect from an ARK project.

The ARK Deployer script is being published for developers, hackers and tech enthusiast to launch their own blockchain based on ARK technology, start learning how the ARK ecosystem works, and get accustomed to the code. Installation instructions and code can be found at: https://github.com/ArkEcosystem/ark-deployer.

ARK Chain is launched with preconfigured ARK blockchain parameters and can also be adjusted to fit your custom needs. ARK Deployer configures, deploys and integrates the following:

1. Deploys ARK node in auto-forging mode on a single computer / server, with 51 forging genesis delegates (clones the ARK node and sets their custom parameters, creates their own genesis file with auto-forging delegates).
2. Deploys ARK explorer that is already configured and talking with installed ARK node (clones, configures, installs and integrates ARK explorer with the ARK node).
3. Configures ARK API for the developer to start exploring, hacking and developing solutions based on ARK Ecosystem technology.

A Vagrant[20] script is also available, where we even automated the deployer steps (read instructions in GitHub repository).

There are also other blockchain solutions targeting this challenges, blockchain suck as Polkadot, Aion Network, MultiChain, BlockNet, Stratis and some others. All this blockchain solutions recognized that in order for a technology to succeed and grow, a *strong network needs to be built and value must be delivered by using supporting protocols talking to each other*. While some hybrid blockchain networks address this challenge successfully, they also take into account that 99% traffic is made off-chain and 1% on-chain for state management only [21].

3.2 ARK contract execution services - ACES

ACES is a community based project solving the problem of interchain transactions and communication. ACES enables the blockchain economy by allowing micro-chain projects to connect to the liquidity and functionality of the entire blockchain ecosystem. We cannot achieve this with protocol-specific interoperability, but protocol-specific interoperability is almost certainly cheaper, faster, and possibly more trustless than a protocol-agnostic interoperability. ACES focuses on protocol-agnostic interoperability because it allows us to integrate all potential future developments in the space. Protocol-agnostic interoperability combined with protocol-specific interoperability is a powerful concept that adds massive efficiencies to the entire space [18].

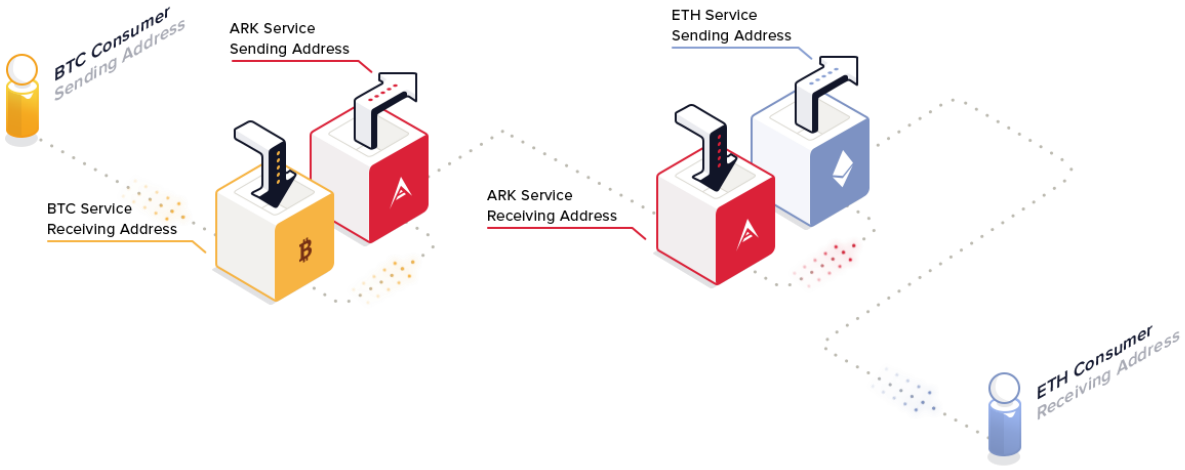


Figure 3. ARK ACES concept

Interoperability should be cheap and empower users. That's where ACES comes in. There are many possible avenues for implementing trustless and decentralized mechanism into the ACES ecosystem in the near future. As such, our focus will remain on making tools easy to launch and develop so that even when decentralized tools are ubiquitous, users and developers can continue to customize software to their needs and contribute to the success of this ecosystem.

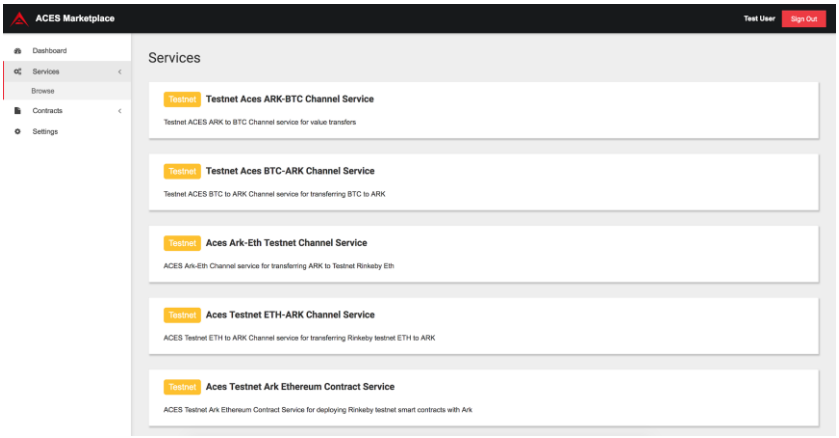


Figure 4. ARK ACES services overview

ACES emphasis is to deliver a marketplace12 directory and customized services which will enable participants in the blockchain economy to launch interoperability businesses. These businesses could be viewed similarly to the type of businesses that operate on Amazon. Some may be small and unprofitable, just experiments or hobbies. Others may be massive multi-million dollar operations squeezing out small margins on high volume[18].

Building solutions on top of OSS concepts and make a living organism is our promise and our long term vision. To deliver a sustainable ecosystem, where all stakeholders can flourish and grow - is our common path to help spread mass adoption of blockchain technology.

Next section will briefly mention the new ARK core v2 and how we used what we learned in the last year to improve, iterate and deliver a completely rewritten core - that will help all of us to deliver the vision of ARK, Satoshi and other decentralised solutions out there.

[12]ARK ACES Marketplace - http://marketplace.arkaces.com

4. ARK CORE V2 - THE NEXT FRONTIER

In order to prepare for the next generation of deployable DPoS blockchains, we understood very quickly that we had to thoroughly rethink the legacy code and to move to our own, completely rewritten from scratch. Throughout the current time period of version v1, and with experience gained during the development of ARK, we have been able to identify several key elements in the core design that could be optimized. Now, we embark on a new voyage to completely overhaul the ARK core code as well as the protocol as a foundation of our ambitious roadmap.

The new architecture (see Figure 5) has been completely rethought to decouple delegate forging activity, transaction pool management, and API interface on separate threads. Transactions will need to pass complete SPV (Simple Payment Verification)[22] on a separate process or server **before hitting the mempool,** completely sandboxing the activity of the node against attacks.

ARK-core v2 is completely **configurable**, meaning you can adjust the blockchain mechanics to your preferred setup. Users are able to adjust block times, number of delegates, block size, customize fees and set block rewards. This configuration is a living setup, meaning it can be adjusted when you arkchain is already running, all you need to provide is the block height parameter - from where the new configuration will be valid and in use.

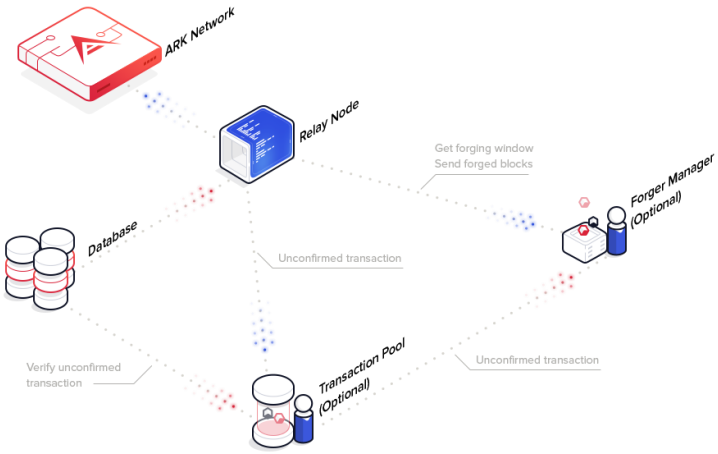


Figure 5. ARK core v2 Architecture

The database communication will be decoupled with an interface layer so it will be possible to use your own favorite technology like MySQL, SQLite, PostgreSQL, or MsSQL (so far). It will also be possible to override your own database interface and pass it to the node. Initial internal tests are already showing impressive results with lightning fast rebuilds using SQLite as backend, leveraging the multi-core capabilities on our test servers. The security will be hyper-enhanced using a completely independent transaction pool, responsible for keeping a list of unconfirmed transactions that could then be broadcasted or passed along to the forger, to forge a block. Finally, the forging process will be completely independent and self contained. It will be architected in a way to switch communications with other relay nodes if the original relay node is getting attacked or forked.

We have designed the relay nodes to be as light and stable as possible and we recommend these relay nodes if you are building a business over ARK blockchain (exchanges, smartbridge, etc…) instead of a light client, since it will allow for more powerful computations such as Complex SPV Proof of Balance, or Batch Verifications.

4.1 New plugin system

ARK Core v2 will be split into multiple packages using Lerna to manage development and publishing of those packages. The benefit of this approach is that it is easier to focus on smaller parts of the whole

system in isolation. Each part of the core can be easily replaced by custom implementations, imagine a core-logger-logstash package that replaces the default logger. All of those plugins are interconnected via the *core-plugin-manager package*, which functions as a container to hold all the instances that are shared across plugins.

The plugin manager allows us to provide different Bootstrap processes for things like starting a relay node or a forging node. The plugin manager accepts two (2) parameters, the path to a folder that contains a plugins.json file and an optional parameter that can contain options like including and excluding plugins from the Bootstrap process or plugin specific options that are not available from the configuration file.

Figure 6. Plugin registration left and plugin lifecycle hooks on the right

4.2 Listening to blockchain events with webhooks

Webhooks implementation is a realization of Ark Improvement Proposal—AIP15 [23]. Webhooks enable ARK blockchain app developers to listen to blockchain events in a simple manner (by subscribing to events and waiting for callbacks). Polling is just wasteful and inefficient for both the client and server side. On average, 98.5% of polls are wasted and increase the workload on the server, making Webhooks much more efficient.

Best practices and happy developers with efficient tools are all part of our motivation to deliver ARK Ecosystem as a platform—while providing a stable and efficient base to build blockchain apps.

With ARK's new API v2 you will also get Webhooks capability, that will call and deliver data based on event conditions.

4.3 ARK testsuite based on jest technology stack

Test suites are an important part of any serious development environment, more so with blockchain technologies where each mistake can end up being very costly. As most of software developers are already aware on how hard it is to get full test coverage over different testing phases (functional tests, integration tests...) that must take multiple stakeholders into account. Now add blockchain mechanics to that recipe and think about how to test distributed systems, their mechanics, security, block propagation, transaction management, transaction pool handling, fork management, client api... - where to start.

With this challenges in mind we had to pick the right tool, that is flexible and enough powerful. We have chosen Jest Framework[24] as our base testing framework. Jest was developed and is used by Facebook to test all of their JavaScript code including React applications. Jest is also used by Airbnb, Twitter, Pinterest, Instagram, and Oculus.

With the growth of our team we are establishing a common base (see Figure 7) for developers, delivering the best possible tools (powerful mocking, snapshot testing, built-in code coverage, zero configuration) and making cross team collaboration smooth with testing appearing uniform across different sections of code. By doing so we deliver priceless implementation examples enabling newcomers to learn and understand the existing code [25].

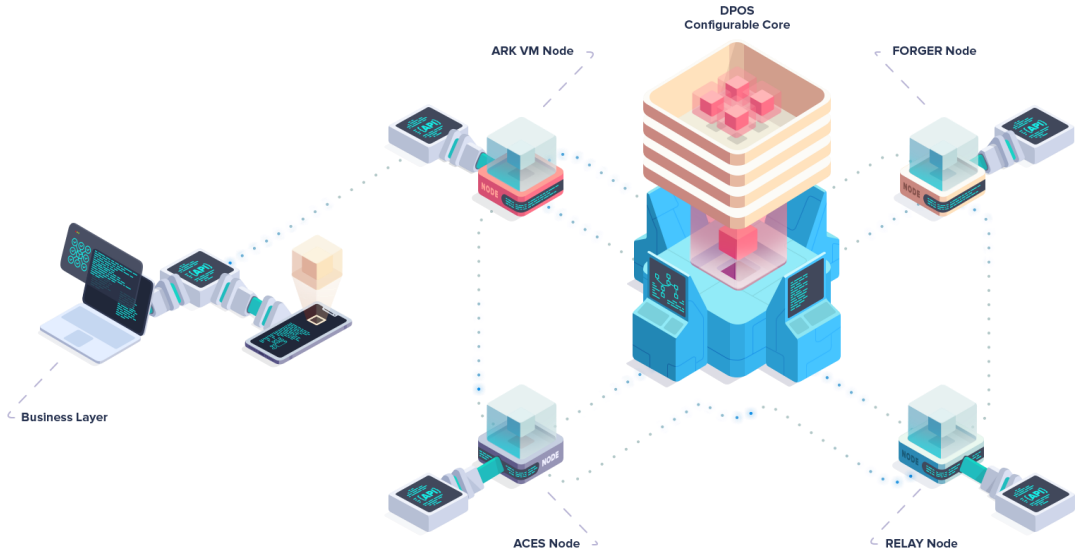


Figure 7. Parts of the ARK Technology stack

5. CLOSING THOUGHTS

A quick overview of blockchain technology and its top challenges where presented. Inter- and cross-chain interoperability is the key issue that needs to be addressed for the mass adoption of blockchain technology, thus enabling business and organization to deliver new business models, delivering true decentralized nature via trustless networks. Across the plethora of different altcoins/products targeting specific solutions, there are a few building entire ecosystems or platforms that promise to deliver the toolset to establish your own blockchain with already existing and tested ARK Technology stack (see Figure 7). ARK Ecosystem is one of them.

Understanding the power of OSS development model and adjusting it in to a collaborative effort, together with the community, will enable us to develop trustworthy applications and deliver them faster to market. We are here to solve a problem and help educate the world on the possible solutions that are available beyond banks and beyond traditional thinking. We believe a blockchain project should aim to achieve easier and much more user friendly solutions.

The end goal for any blockchain project should be that consumers don't even need to know that there is a blockchain under the hood - it should all be intuitive and easy to use, like email, but just like in email, there will be a slight learning curve that must be dealt with through education and time and we are doing our best to play an important role in the future.

6. REFERENCES

[1] Z. Zheng, S. Xie, H. Dai, X. Chen, and H. Wang, "An Overview of Blockchain Technology: Architecture, Consensus, and Future Trends," in *2017 IEEE International Congress on Big Data (BigData Congress)*, 2017, pp. 557–564.

[2] Wikipedia contributors, "Hype cycle," *Wikipedia, The Free Encyclopedia*, 06-May-2018. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Hype_cycle&oldid=839918558. [Accessed: 04-Jun-2018].
[3] M. Pisa and M. Juden, "Blockchain and Economic Development: Hype vs. Reality," *Center for Global Development Policy Paper*, vol. 107, 2017.
[4] W. Nowiński and M. Kozma, "How Can Blockchain Technology Disrupt the Existing Business Models?," *Entrepreneurial Business and Economics Review*, vol. 5, no. 3, pp. 173–188, Sep. 2017.
[5] M. Demary and V. Demary, "Blockchain: Down to earth," IW-Kurzberichte, 2017.
[6] G. F. Coulouris, J. Dollimore, and T. Kindberg, *Distributed Systems: Concepts and Design*. Pearson Education, 2005.
[7] R. G. Brown, "Introducing R3 Corda: A Distributed Ledger for Financial Services," *R3, April*, vol. 5, 2016.
[8] V. Morris *et al.*, *Developing a Blockchain Business Network with Hyperledger Composer using the IBM Blockchain Platform Starter Plan*. IBM Redbooks, 2018.
[9] R. Nagpal, "Blockchain — hype v/s reality – Blockchain Blog – Medium," *Medium*, 15-Apr-2017. [Online]. Available: https://medium.com/blockchain-blog/blockchain-hype-v-s-reality-c33fc1410890. [Accessed: 03-Jun-2018].
[10] "ARK | All-in-One Blockchain Solutions." [Online]. Available: http://www.ark.io. [Accessed: 03-Jun-2018].
[11] Wikipedia contributors, "Open-source software," *Wikipedia, The Free Encyclopedia*, 02-Jun-2018. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Open-source_software&oldid=844023877. [Accessed: 03-Jun-2018].
[12] A. M. St. Laurent, *Understanding Open Source and Free Software Licensing: Guide to Navigating Licensing Issues in Existing & New Software*. "O'Reilly Media, Inc.," 2004.
[13] "Open Source Software(OSS) : Wave of the Future." [Online]. Available: http://www.infosysblogs.com/infosysdigital/2016/06/open_source_software_wave_of_future.html. [Accessed: 03-Jun-2018].
[14] H. Pickavet, "The next wave in software is open adoption software," *TechCrunch*, 20-Jun-2016.
[15] Wikipedia contributors, "Ecosystem," *Wikipedia, The Free Encyclopedia*, 02-Jun-2018. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Ecosystem&oldid=844035609. [Accessed: 03-Jun-2018].
[16] "ARK Community Fund." [Online]. Available: https://arkcommunity.fund/. [Accessed: 04-Jun-2018].
[17] R. Brown, "Universal Interoperability: Why Enterprise Blockchain Applications Should be Deployed to Shared…," *Medium*, 05-Apr-2018. [Online]. Available: https://medium.com/corda/universal-interoperability-why-enterprise-blockchain-applications-should-be-deployed-to-shared-3d4daff97754. [Accessed: 03-Jun-2018].
[18] A. Aces, "Satoshi Supported the Idea of Alt Coins in 2010 and How That Relates to the ARK Ecosystem," *Medium*, 23-Mar-2018. [Online]. Available: https://medium.com/@arkaces/satoshi-supported-the-idea-of-alt-coins-and-the-ark-ecosystem-in-2010-827866df2972. [Accessed: 03-Jun-2018].
[19] A. Ovechkin, *Blockchain: The Ultimate Beginners Guide to Understanding Blockchain Technology*. CreateSpace Independent Publishing Platform, 2018.
[20] M. Peacock, *Creating Development Environments with Vagrant*. Packt Publishing Ltd, 2013.
[21] S. Lessin, "The Future of Hybrid Centralized-Decentralized Apps," *The Information*. [Online]. Available: https://www.theinformation.com/articles/the-future-of-hybrid-centralized-decentralized-apps?shared=5b152f291c7c6ec1. [Accessed: 04-Jun-2018].
[22] M. Swan, *Blockchain: Blueprint for a New Economy*. "O'Reilly Media, Inc.," 2015.
[23] *AIPs*. Github.
[24] A. Fedosejev and A. Boduch, *React 16 Essentials: A fast-paced, hands-on guide to designing and building scalable and maintainable web apps with React 16*. Packt Publishing Ltd, 2017.
[25] BoldNinja, "ARK Core v2 Technical Update Series — New Testing Suite," *ARK.io | Blog*, 13-Feb-2018. [Online]. Available: https://blog.ark.io/ark-core-v2-technical-update-series-new-testing-suite-a1087cfc8094. [Accessed: 04-Jun-2018].

Univerzitetna založba Univerze v Mariboru

KAKO NAČRTOVATI SODOBNO ARHITEKTURNO REŠITEV ZA KOMPLEKSNE INFORMACIJSKE SISTEME

ERVIN LEMARK, TJAŠA ŠOSTER, DAMIJAN KAVS IN VID BEVČAR

Povzetek: Kot razvijalci in vzdrževalci informacijskih rešitev se vedno znova vprašujemo, kako in s čim bi se lotili razvoja kompleksnega sistema, ki bi bil sposoben zadostiti hitro rastočim potrebam, bi se ga dalo enostavno razširjati z novimi zmožnostmi, bi bil dostopen na različnih napravah, bi se ga dalo celo klonirati in uporabiti drugje, hkrati pa bi bil varen, bi zadoščal standardom in zakonskim zahtevam ter še prijazen do uporabnika in enostaven za vzdrževanje?

V članku vam bomo predstavili arhitekturni model, zasnovan na mikro storitvah. Hkrati bomo prikazali, s katerimi tehnologijami menimo, da najbolje izvedemo posamezne dele sestavljanke. Prikazali bomo še primere iz prakse, kjer mikro storitve s pridom uporabljamo.

Ključne besede: • mikro storitve • modularnost • oblačne storitve • razširljivost • enostavno vzdrževanje

NASLOV AVTORJEV: Ervin Lemark, tech lead, Comland d.o.o., Litostrojska 58c, 1000 Ljubljana, Slovenija, e-pošta: jaka.jenko@kivi.si. Tjaša Šoster, senior IT solutions architect programskih rešitev, Comland d.o.o., Litostrojska 58c, 1000 Ljubljana, Slovenija, e-pošta: tjasa.soster@comland.si. Damijan Kavs, senior backend developer, Comland d.o.o., Litostrojska 58c, 1000 Ljubljana, Slovenija, e-pošta: damijan.kavs@comland.si. Vid Bevčar, senior full stack developer, Comland d.o.o., Litostrojska 58c, 1000 Ljubljana, Slovenija, e-pošta: vid.bevcar@comland.si.

DOI https://doi.org/10.18690/978-961-286-162-9.6 ISBN 978-961-286-163-6

Dostopno na: http://press.um.si.

1. UVOD

Kot razvijalci in vzdrževalci informacijskih rešitev se vedno znova vprašujemo naslednje:

Kako in s čim bi se lotili razvoja kompleksnega sistema, ki bi bil sposoben zadostiti hitro rastočim potrebam, bi se ga dalo enostavno razširjati z novimi zmožnostmi, bi bil dostopen na različnih napravah, bi se ga dalo celo klonirati in uporabiti drugje, hkrati pa bi bil varen, bi zadoščal standardom in zakonskim zahtevam ter še prijazen do uporabnika in enostaven za vzdrževanje?

Seveda, večno Million Dollar vprašanje izdelovalcev informacijskih sistemov.

Zgodi se tudi, da nam to vprašanje zastavi naročnik. Za kar smo mu globoko hvaležni, tudi če se na koncu odloči za manj sodobno rešitev.

To vprašanje smo si zadnje čase pogosto zastavljali. Z veseljem vam povemo, da smo odgovor našli in ga zapisali.

Naš odgovor je …

2. ARHITEKTURA, ZASNOVANA NA MIKRO STORITVAH

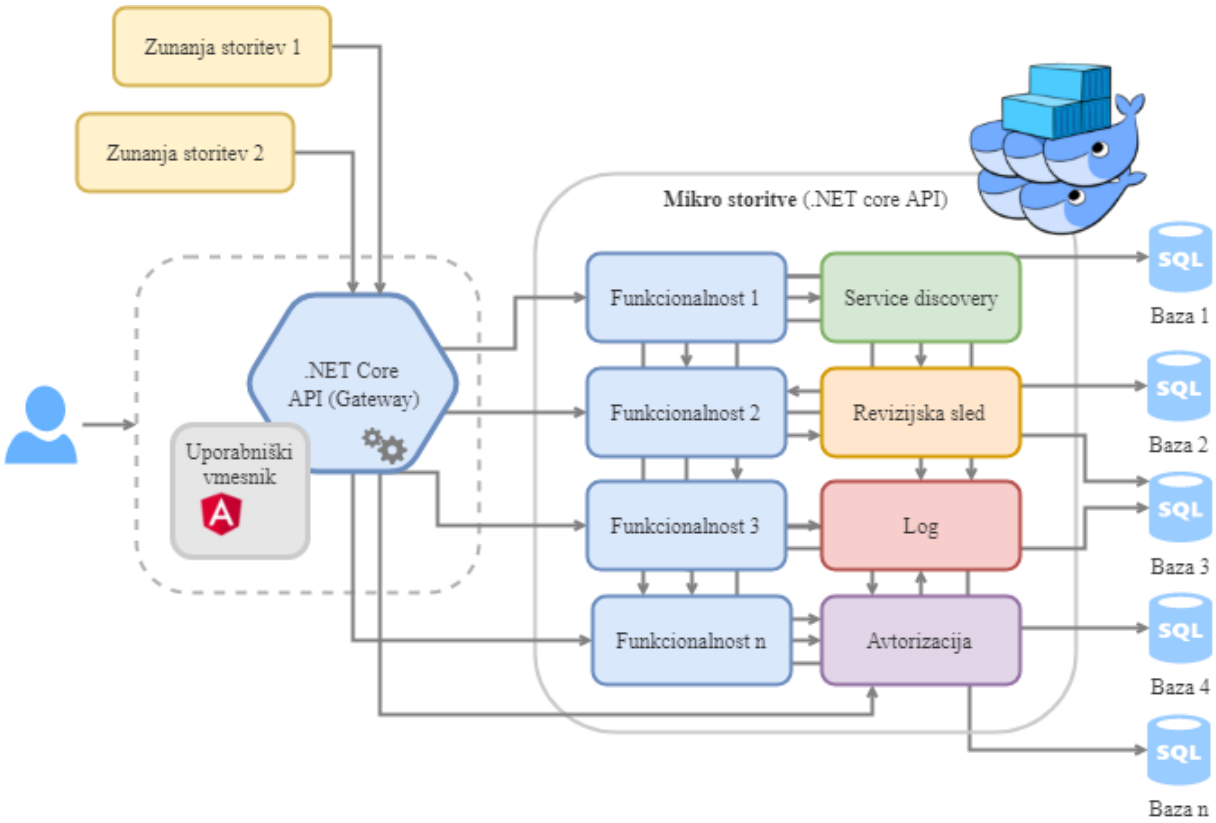
Nič novega, boste rekli. Mikro storitve [1] so buzzword že nekaj časa. Vsi res veliki informacijski sistemi jih uporabljajo. Ko skočite na katerokoli globalno spletno aplikacijo (Amazon, Facebook, Twitter, eBay, …) uporabljate arhitekturo z mikro storitvami.

Kaj pa uporaba mikro storitev v poslovnih informacijskih sistemih, aplikacijah za naročnika, naših lastnih izdelkih? Tu se še kar oklepamo starejših, z leti preizkušenih pristopov.

Čas je, da gremo naprej. Kako? Tu je recept.

2.1. Model arhitekture (in tehnologije) sodobnega informacijskega sistema

Predstavljamo vam naš arhitekturni recept. Hkrati odgovarjamo tudi, s katerimi tehnologijami menimo, da najbolje izvedemo posamezne dele sestavljanke.



Slika 1. Arhitektura informacijskega sistema z uporabljenimi mikro storitvami

Poslovno logiko delovanja informacijskega sistema razdelimo v *več mikro storitev* glede na pričakovane funkcionalnosti. S tem omogočimo zmožnost prilagajanja *zmogljivosti* ter lažje *vzdrževanje* in *nadgradnje*.

Mikro storitve med seboj *povežemo*, tako da si (po potrebi) *izmenjujejo podatke*.

Poleg mikro storitev, ki vsebujejo poslovno logiko, dodamo še *namenske mikro storitve*, kot so *beleženje dogodkov* (log), samodejno *odkrivanje storitev* (service discovery [2]), *revizijska sled*, *avtorizacija*, ...

Komunikacijo uporabnikov in zunanjih storitev z mikro storitvami uredimo preko *REST API*-ja [3] (Gateway). Podatke izmenjujemo preko JSON datotek. Za potrebe uporabnikov po dostopu do obličja informacijskega sistema razvijemo *uporabniški vmesnik* v tehnologiji *Angular* [4].

Vsako mikro storitev posebej (in Gateway API) pripravimo v obliki lastnega *Docker vsebnika* [5] (container-ja). S tem omogočimo enostavno *nameščanje, vzdrževanje* in *širjenje* zmogljivosti.

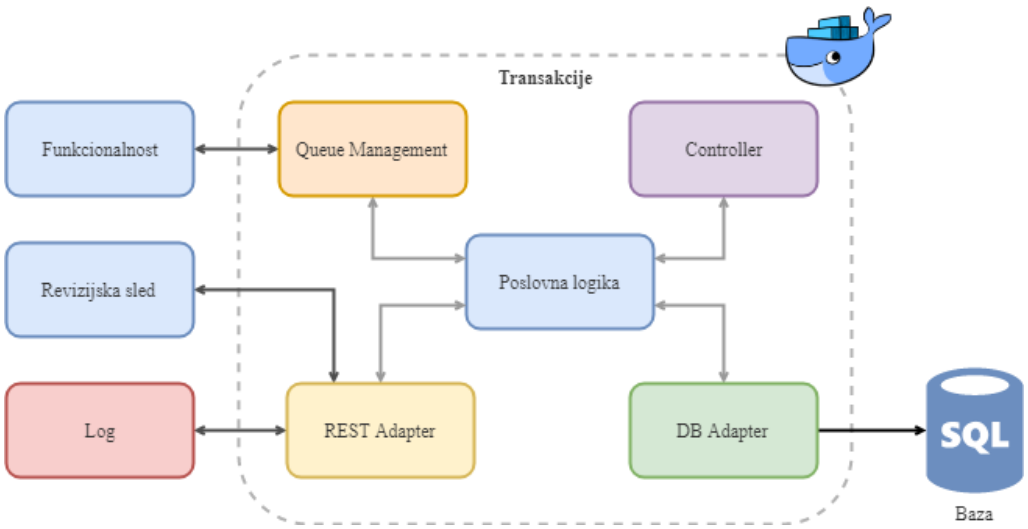
Za upravljanje z viri in skalabilnost uporabimo rešitev *Docker Swarm* [6]. S tem omogočimo namestitev tako na Linux kot na Windows strežnike.

Ob namestitvi v oblak priporočamo uporabo *Kubernetes orkestracije* [7] za samodejno prilagajanje obremenitvam ter samodejno zaznavanje in premeščanje izpadlih strežnikov.

2.2. Zgradba posamezne mikro storitve

V arhitekturnem modelu, kot smo ga opisali zgoraj, je posamezna mikro storitev zgrajena iz naslednjih sestavin:

- poslovna logika,
- controller – predstavitveni sloj, ki sprejema REST klice,
- DB Adapter – uporablja se za povezovanje s podatkovno bazo,
- REST Adapter – uporablja se za povezovanje z drugimi storitvami,
- Queue Management – upravljanje čakalne vrste transakcij, kjer je to potrebno.



Slika 2. Arhitektura posamezne mikro storitve

2.3. Uporaba podatkovnih baz

Mikro storitve se povezujejo s *podatkovnimi bazami*. Glede na svoj namen (funkcionalnost) v baze podatke pišejo, jih spreminjajo in jih iz njih berejo. Baze so tehnološko gledano lahko različne. SQL, NoSQL, … Pač izberemo tehnologijo za optimalno hranjenje in posredovanje podatkov glede na konkretne potrebe mikro storitve in funkcionalnosti, ki jih le-ta podpira.

Dejansko fizično izvedbo razdelitve baz in njihove vrste tako lahko prilagajamo glede na obremenitev sistema in na ta način dosežemo popolno prilagodljivost. V zgodbo smiselno vključimo tudi življenjski tok in namen hranjenih podatkov (produkcijske baze, podatkovna skladišča, revizijska sled, zapisi delovanja in dostopov, arhiv, …).

2.4. Zagotavljanje varnosti

Za *zagotavljanje varnosti* podatke, ki se posredujejo med zunanjim svetom in našim informacijskim sistemom, ustrezno zavarujemo z zgoščevalno funkcijo. S tem preverjamo istovetnost in nespremenljivost podatkov. Vse povezave potekajo preko HTTPS protokola, s čimer zaščitimo vsebino prometa in podatke pred vmesnimi opazovalci.

2.5. Avtentikacija in avtorizacija

Za avtentikacijo in avtorizacijo pripravimo posebne mikro storitve, ki imajo skupne vhodne in izhodne lastnosti, znajo pa se pogovarjati z različnimi zunanjimi storitvami. Tako lahko v istem sistemu omogočimo več različnih načinov avtentikacije ter tudi več nivojev avtentikacije [8]. V praksi to pomeni podporo domačim storitvam, kot so Varnostna shema MJU [9], Varnostna shema MDDSZ [10], SI-CAS [11], in generičnim rešitvam, kot sta Google Authenticator [12] in Google reCAPTCHA [13], ter naprednim možnostim, kot so biometrične značilke [14] (prstni odtis, obrazne značilnosti, …).

2.6. Decentralizacija na več sistemov in več lokacij

Predstavljeni model lahko nadgradimo in razširimo na dva načina:

- horizontalno – več vzporednih informacijskih sistemov,
- vertikalno – vzpostavimo vrhnje entitete, ki hierarhično vsebujejo enega ali več informacijskih sistemov.

Ob tem ohranimo obstoječo zgradbo in funkcionalnost ter dodamo višji nivo, ki ga podpremo z imenikom sistemov. Dodamo storitev za določanje naslovov in usmerjanje prometa. Dodamo poslovno logiko za izvajanje funkcionalnosti v razširjenem ekosistemu.

S tem pristopom informacijski sistem lahko tudi decentraliziramo.

Pri bazah lahko izberemo tudi verigo blokov in tako decentraliziramo delo s podatki. Seveda verige blokov prispevajo še dodatne zmožnosti, ki so značilne za njih.

3. KATERE TEHNOLOGIJE PRIPOROČAMO OB DELU Z MIKROSTORITVAMI

Ob sliki 1 smo že zapisali, s katerimi tehnologijami se lotevamo izgradnje nekaterih delov celote. Na kratko jih ponovimo in dodajmo manjkajoče. Ob tem naj opomnimo, da tak izbor ni dokončen ali edini. Glede na primer in potrebe ter želje naročnika se odločimo tudi drugače.

Razvojno okolje in hkrati okolje za vodenje postopka razvoja: MS Visual Studio [15] in .NET Core [16], ker ima to okolje že vsebovano vse, kar potrebujemo za razvoj in ker je rezultat dela neodvisen od operacijskega sistema.

Programski jezik mikro storitev: C#, razen kjer je zaradi specifičnih potreb bolj primerno kaj drugega. Ogrodje s povezovalnimi knjižnicami pripravljeno z ASP.NET Core [17].

Uporabniški vmesnik: Angular.
Podatkovne baze: tu pa res ni omejitev – SQL (Oracle [18] in Microsoft [19]), NoSQL[20] (običajno MongoDB [21]), …)
Tehnologija vsebnikov: Docker.
Tehnologija orkestracije: Kubernetes.

V primerih, ki sledita, poleg naštetih najdemo še kako orodje in tehnologijo, kar bomo posebej poudarili.

4. PRIMERI UPORABE ARHITEKTURNIH MODELOV, ZASNOVANIH Z MIKRO STORITVAMI

Prikazali bomo dva primera uporabe predstavljenega arhitekturnega modela z uporabo mikro storitev. Prvi primer prikazuje povezovanje več sistemov preko vsebnikov, ki so nameščeni v oblačne storitve. Drugi primer pa je namenska aplikacija, ki uporablja tehnologijo računalniškega vida in s pridom uporablja vse prednosti mikro storitev.

4.1. Zajem podatkov – povezava treh sistemov

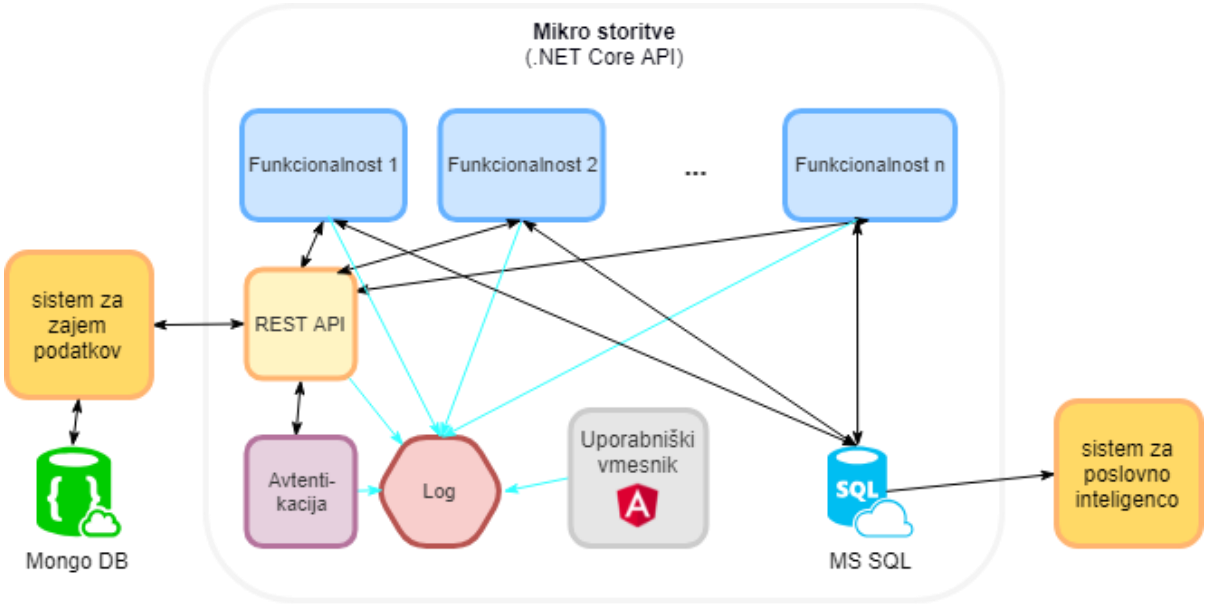
Naš prvi primer prikazuje povezavo treh sistemov, kjer je na eni strani sistem za zajem podatkov, na drugi strani sistem za poslovno inteligenco, kot vmesnik pa smo pripravili sklop mikro storitev, ki ta sistema na smiseln način povezujejo, in med njima prenašajo podatke.

Povezovanje sistemov predstavlja podoben izziv, kot je povezovanje (mikro) storitev. Dodaten napor lahko pomeni prilagajanje delov, ki niso neposredno pod našim nadzorom. V lastnem sistemu, ki ga sestavljajo mikro storitve, imamo sami nadzor nad vsemi deli, ne glede nato, kako kompleksni so. Ob

povezovanju z zunanjimi sistemi, ki so izven našega nadzora, pa se v praksi srečujemo z množico nepredvidenih izzivov, ki lahko pomembno podaljšajo čas izvedbe.

V naslednjem primeru smo povezali dva obstoječa sistema – lastno aplikacijo za zajem podatkov na terenu in sistem za poslovno inteligenco. Dodali smo namensko razvit modul, ki je iz zalednega sistema pripravil podatke. Te podatke je nato prevzel sistem za zajem na terenu, ki je dopolnjene podatke vrnil modulu. Ta pa je zajete podatke prilagodil, da so bili primerni za obdelavo v sistemu za poslovno inteligenco.

Modul smo pripravili z arhitekturo mikro storitev, kjer smo posamezne dele poslovnega procesa podprli z namenskimi mikro storitvami. Tako smo dosegli njihovo neodvisnost glede obremenitve iz vidika časa izvajanja in količine podatkov. Umestitev modula v oblak in orkestracija omogočata prilagodljivost glede na potrebe in skrbita za stalno dostopnost.



Slika 3. Arhitektura sistema za zajem podatkov z uporabo povezovalnega modula z mikro storitvami

Tehnologije, uporabljene za modul z mikro storitvami: .NET Core, Docker, Kubernetes, Microsoft SQL baza in Angular uporabniški vmesnik, REST povezave.

4.2. Računalniški vid v akciji

Drugi primer poleg naštetih tehnologij uporablja še tehnologijo računalniškega vida. Mikro storitev, ki prepoznava video zapis, je napisana v jeziku C++, ker je pri njej izrednega pomena hitrost izvajanja.

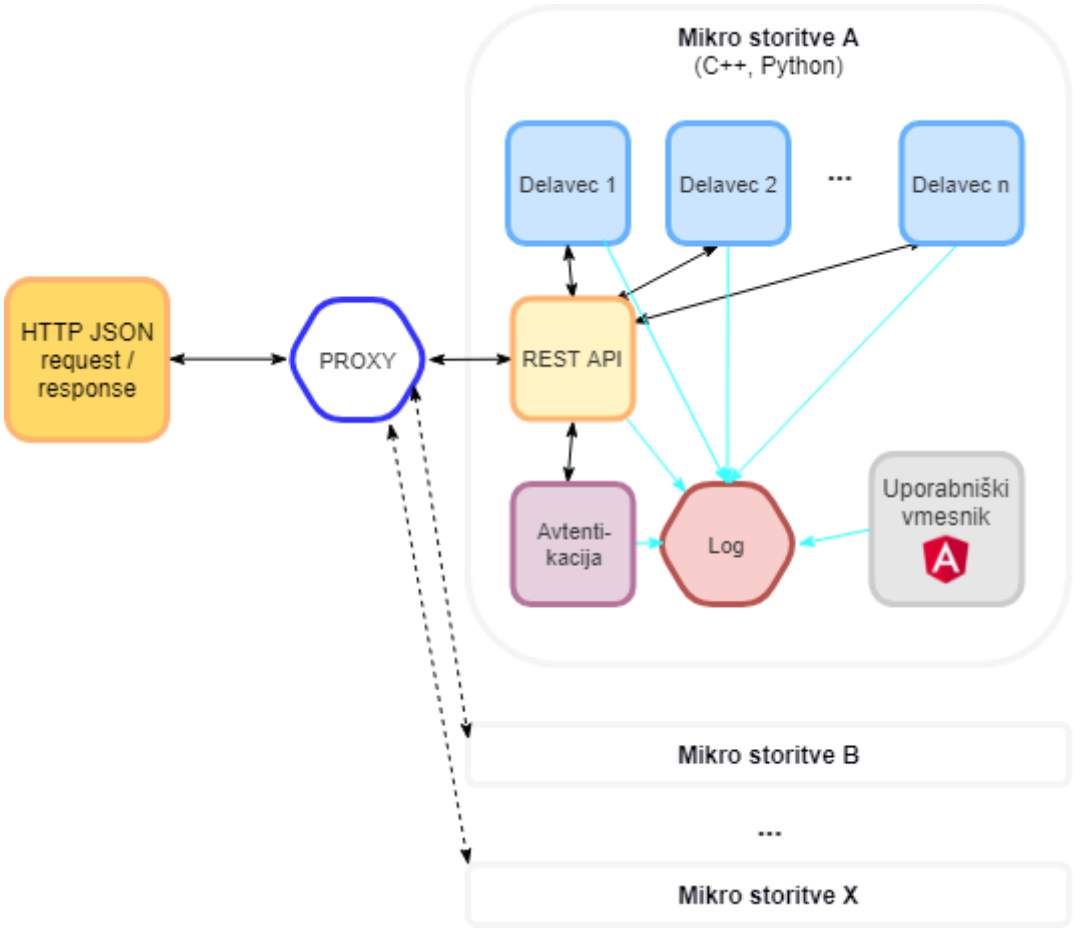
Primer prikazuje zahtevno okolje prepoznavanja vzorcev v živo v video zapisu. Število in sočasnost video zapisov nista znana v najprej. Izziv je torej priprava modularnega sistema, ki je sposoben *hitro pregledati* in *prepoznati* v naprej *neznano količino* in *pogostost* video zapisov in *vedno pravilno ugotoviti vzorec*. V tem primeru je izvedba z mikro storitvami kar sam po sebi umevna.

Pripravili smo arhitekturo v kateri je vhodno izhodni člen vmesnik (proxy), ki skrbi za posredovanje zahtev za pregled videa ter vrača rezultate. Kot mikro storitve v sistemu nastopajo še moduli za avtentikacijo, povezovanje (REST API) in administracijo (spletni vmesnik). Najpomembnejši del pa so mikro storitve »delavci«, na katerih je breme posamezne obdelave in prepoznavanja videa. Tu smo se odločili za neobičajen pristop, ki je najbolj ustrezal zahtevnosti naloge.

V teoriji mikro storitev velja, da se število sočasno delujočih storitev ene funkcionalnosti prilagaja potrebam. Delo prične N storitev. Po doseženi obremenitvi se jim pridružijo nove. Ko obremenitev pade, se njihovo število (morda) zmanjša.

V našem primeru smo se odločili za pristop »ena zahteva – en delavec«. Ko prispe zahteva za obdelavo videa, orkestracija *dvigne* Docker vsebnik z *delavcem*, ta video obdela (dejansko v živo *gleda* video stream), prepozna vzorec, vrne rezultat in se ugasne. Sočasno torej deluje toliko pojavitev ene mikro storitve, kolikor je zahtev za obdelavo videa. Ne več in ne manj. Izkoriščenost sistema je tako optimalna.

Arhitekturo takega sistema prikazujemo na naslednji sliki.



Slika 4. Arhitektura mikro storitve pri prepoznavanju videa

Po potrebi lahko predstavljeni sistem vodoravno razmnožimo (v gornji sliki mikro storitve od B do X) in ga s tem prilagodimo ter zadovoljimo večjim zahtevam. V praksi velja, da vedno sočasno delata vsaj dva sistema.

Uporabljene tehnologije: C++ [22] za mikro storitve za prepoznavo videa, NodeJS [23] in Angular za vse druge mikro storitve razen delavcev, Python [24] za povezovanje med storitvami, elasticsearch [25] za analitiko delovanja sistema.

Prilagamo nekaj številk za ilustracijo hitrosti vzpostavitve Docker vsebnika za prepoznavo videa:

- 0.00 sekunde za zagon vsebnika,
- 1,39 s za inicializacijo procesa,
- 0.23 sekunde za inicializacijo video vira.

Skupaj torej 1.61 sekunde, da lahko mikro storitev prične s svojim osnovnim delom – prepoznavno videa. Ker je v video signalu prvih 10 sekund privzeto nepomembnih, je to več kot dovolj hitro, da lahko storitev uspešno opravi svoje delo.

Orkestracijo sistema v namestitvi pri ponudniku storitev v oblaku upravljamo s pomočjo Kubernetesa. Enako postavitev lahko izvedemo tudi na fizičnem strežniku, kjer orkestracijo izvedemo s skriptami s pomočjo Docker compose [26] ukazov.

5. ZAKLJUČEK

Priznajmo, da *mikro storitve* kot osnovni pristop k izgradnji informacijskih sistemov niso nič novega. Razvijalci smo vedno stremeli k deljenju izziva na dele in modeliranju teh delov kot čim bolj samostojne celote. Modularni pristop prinaša veliko prednosti, od katerih bi omenil le dve – razbijanje kompleksnosti in ponovna uporaba funkcionalnosti.

Ekosistem mikro storitev pa igro dviga na popolnoma nov nivo. Strojna in programska podpora je končno dozorela in je dovolj zmogljiva, da prevzame delitev in povezovanje modulov ter jih izvaja namesto programerja in administratorja. Pravzaprav je evolucija informacijskih sistemov tu dohitela *neuradno* teorijo in ji ponudila okolje, kjer lahko preide v prakso.

Priča smo prehodu od fizičnih strežnikov preko navideznih strežnikov do oblačnih namestitev in od monolitnih sistemov preko vodoravno in navpično deljenih sistemov do popolnoma prožnih rešitev.

Na eni strani imamo torej ponudbo. Ponudbo, ki jo pravzaprav potrebuje in jo zahteva povpraševanje. Vedno več podatkov v kakršnikoli obliki, big data, deep learning, computer vision, IoT, spletna omrežja, … Brez uporabe arhitekture mikro storitev velika sodobna spletišča, in celotni informacijski sistemi za njimi, sploh ne bi obstajala.

Ne pozabimo pa, da se orodja in naša uporaba le-teh sproti razvijajo, da standardi sploh še niso dorečeni, da se učimo in rastemo, da prepoznavamo ozka grla in jih nadomeščamo z boljšimi rešitvami.

Prihodnost mikro storitev je v … še bolj učinkovitih mikro storitvah. Naša dolžnost v vlogi ponudnikov odličnih informacijskih rešitev je, da jih čim bolje uporabljamo in razvijamo.

V našem prispevku smo z malce teorije in na dveh dokaj enostavnih primerih prikazali, kako se lahko lotimo rešitev s pomočjo mikro storitev. Čar pristopa je v tem, da z rastjo sistema njegova kompleksnost ne raste – ker stoji na trdnem temelju mikro storitev.

6. LITERATURA

[1] https://en.wikipedia.org/wiki/Microservices, Microservices (From Wikipedia, the free encyclopedia), obiskano 18.05.2018

[2] https://www.nginx.com/blog/service-discovery-in-a-microservices-architecture/, Service Discovery in a Microservices Architecture, obiskano 18.05.2018

[3] https://en.wikipedia.org/wiki/Representational_state_transfer, Representational state transfer (From Wikipedia, the free encyclopedia), obiskano 18.05.2018

[4] https://angular.io/, Angular - One framework. Mobile & desktop., obiskano 18.05.2018

[5] https://www.docker.com/, Docker - Build, Ship, and Run Any App, Anywhere, obiskano 18.05.2018

[6] https://docs.docker.com/engine/swarm/key-concepts/, Swarm mode key concepts, obiskano 18.05.2018

[7] https://kubernetes.io/, Production-Grade Container Orchestration, obiskano 18.05.2018

[8] https://en.wikipedia.org/wiki/Multi-factor_authentication, Multi-factor authentication (From Wikipedia, the free encyclopedia), obiskano 18.05.2018

[9] https://vs.gov.si/VS.web/, Varnostna Shema, obiskano 18.05.2018

[10] https://vs.gov.si/VS.web/, Ministrstvo predstavilo informacijski sistem za odločanje o pravicah iz javnih sredstev, obiskano 18.05.2018

[11] http://nio.gov.si/nio/asset/centralni+avtentikacijski+sistem+sicas, Centralni avtentikacijski sistem SI-CAS, obiskano 18.05.2018

[12] https://en.wikipedia.org/wiki/Google_Authenticator, Google Authenticator (From Wikipedia, the free encyclopedia), obiskano 18.05.2018

[13] https://www.google.com/recaptcha/intro/v3beta.html, reCAPTCHA: Easy on Humans, Hard on Bots, obiskano 18.05.2018

[14] https://en.wikipedia.org/wiki/Biometrics, Biometrics (From Wikipedia, the free encyclopedia), obiskano 18.05.2018

[15] https://www.visualstudio.com/, Visual Studio IDE, Code Editor, VSTS, & App Center, obiskano 18.05.2018

[16] https://www.microsoft.com/net/learn/what-is-dotnet, What is .NET?, obiskano 18.05.2018

[17] https://docs.microsoft.com/en-us/aspnet/core/web-api/?view=aspnetcore-2.0, Build web APIs with ASP.NET Core , obiskano 18.05.2018

[18] https://www.oracle.com/database/index.html, Database | Cloud Database | Oracle, obiskano 18.05.2018

[19] https://www.microsoft.com/en-us/sql-server/sql-server-2017, SQL Server 2017 on Windows and Linux | Microsoft, obiskano 18.05.2018

[20] https://en.wikipedia.org/wiki/NoSQL, NoSQL (From Wikipedia, the free encyclopedia),, obiskano 18.05.2018

[21] https://www.mongodb.com/, MongoDB for GIANT Ideas, obiskano 18.05.2018

[22] https://sl.wikipedia.org/wiki/C%2B%2B, C++ (Iz Wikipedije, proste enciklopedije) , obiskano 18.05.2018

[23] https://nodejs.org/en/, Node.js, obiskano 18.05.2018

[24] https://www.python.org/, Welcome to Python.org, obiskano 18.05.2018

[25] https://www.elastic.co/, Open Source Search & Analytics · Elasticsearch | Elastic, obiskano 18.05.2018

[26] https://docs.docker.com/compose/, Docker Compose, obiskano 18.05.2018

Univerzitetna založba Univerze v Mariboru

IT INTEGRACIJE V SODOBNIH ELEKTROENERGETSKIH OMREŽJIH Z UPORABO MODELA CIM

NIKOLA RISTESKI

Povzetek: Na področju proizvodnje in dobave elektrike se dogajajo korenite spremembe – zavezani smo k zniževanju ogljičnega odtisa, povečanju deleža elektrike iz obnovljivih virov, uvajamo električne avte, prehajamo na distribuirano in manj predvidljivo proizvodnjo elektrike (sončne, veterne elektrarne). Rešitev za prej omenjene izzive elektroenergetskih omrežij predstavlja razvoj tako imenovanega pametnega omrežja (angl. "Smart Grid"). Večina definicij pametnega omrežja izpostavlja uporabo digitalne izmenjave informacij z namenom zagotavljanja stabilnosti, varnosti, zanesljivosti in učinkovitosti, kot tudi interoperabilnosti med različnimi udeleženci znotraj omrežja, kot tudi med različnimi omrežji. Iz tega je razvidno, da je razvoj pametnega elektroenergetskega omrežje nepredstavljiv brez integracij nekoč popolnoma nepovezanih sistemov. V okviru prizadevanj za implementacijo pametnega omrežja, slovenski sistemski operater prenosnega omrežja ELES izvaja projekt pilotnega razvoja pametnega omrežja v Sloveniji v sodelovanju z japonsko agencijo NEDO. V okviru navedenega projekta sodelujemo tudi podjetje Bintegra d.o.o., kjer smo se osredotočili na zagotavljanje integracij oz. interoperabilnosti z implementacijo arhitekture storitvenega vodila (angl. Enterprise Service Bus), oziroma natančneje arhitekturo mikrostoritev. Tehnologija, ki smo jo izbrali za implementacijo rešitve je Apache ServiceMix, kar predstavlja skupek različnih odprtokodnih tehnologij za implementacijo integracijskih projektov. Potreba po interoperabilnosti med razlinimi operaterji napeljuje k implementaciji splošnih standardov, ki veljajo za vse operaterje. V tem primeru smo se odločili za uporabo modela Common Information Model (CIM), ki je namenjen kot osnova za standarde za izmenjavo komunikacij na področju energetike. Pri integraciji smo uporabili nabor standardov IEC (International Electrotechnical Commision) za izmenjavo podatkov med udeleženci elektroenergetskega trga.

Ključne besede: • integracije • mikrostoritve • CIM • Apache ServiceMix • elektroenergetika

NASLOV AVTORJA: Nikola Risteski, Bintegra d.o.o., Železnikova ulica 4, 2000 Maribor, Slovenija, e-pošta: nikola.risteski@bintegra.com

DOI https://doi.org/10.18690/978-961-286-162-9.7 ISBN 978-961-286-163-6

Dostopno na: http://press.um.si.

1. UVOD

Na področju proizvodnje in dobave elektrike se dogajajo korenite spremembe - zavezani smo k zniževanju ogljičnega odtisa, povečanju deleža elektrike iz obnovljivih virov, uvajamo električne avte, prehajamo na distribuirano in manj predvidljivo proizvodnjo elektrike (sončne, veterne elektrarne). Rešitev za prej omenjene izzive elektroenergetskih omrežij predstavlja razvoj tako imenovanega pametnega omrežja (angl. "Smart Grid"). Večina definicij pametnega omrežja izpostavlja uporabo digitalne izmenjave informacij z namenom zagotavljanja stabilnosti, varnosti, zanesljivosti in učinkovitosti, kot tudi zagotavljanja interoperabilnosti med različnimi udeleženci znotraj omrežja, kot tudi med različnimi omrežji. Iz tega je razvidno, da je razvoj pametnega elektroenergetskega omrežje nepredstavljiv brez razvoja informacijskih tehnologij, ki so temelj za komunikacijo in upravljanje le-tega. Eden izmed ključnih pogojev za doseganje omenjenih ciljev je sodelovanje med različnimi sistemi, ki so doslej popolnoma neodvisno opravljali svoje naloge.

V okviru prizadevanj za implementacijo pametnega omrežja, slovenski sistemski operater prenosnega omrežja ELES izvaja projekt pilotnega razvoja pametnega omrežja v Sloveniji v sodelovanju z japonsko agencijo NEDO. V okviru navedenega projekta sodelujemo tudi podjetje Bintegra d.o.o.

V okviru projekta NEDO smo sodelovali pri implementaciji storitvenega vodila oziroma arhitekture mikrostoritev, s čimer smo želeli postaviti temelje za nadaljnje integracije na primeru sistemskega operaterja distribucijskega omrežja Elektro Celje. Pri implementaciji teh nalog smo sodelovali z Elektroinštitutom Milan Vidmar in podjetjem Iskratel d.d.

2. CIM (COMMON INFORMATION MODEL)

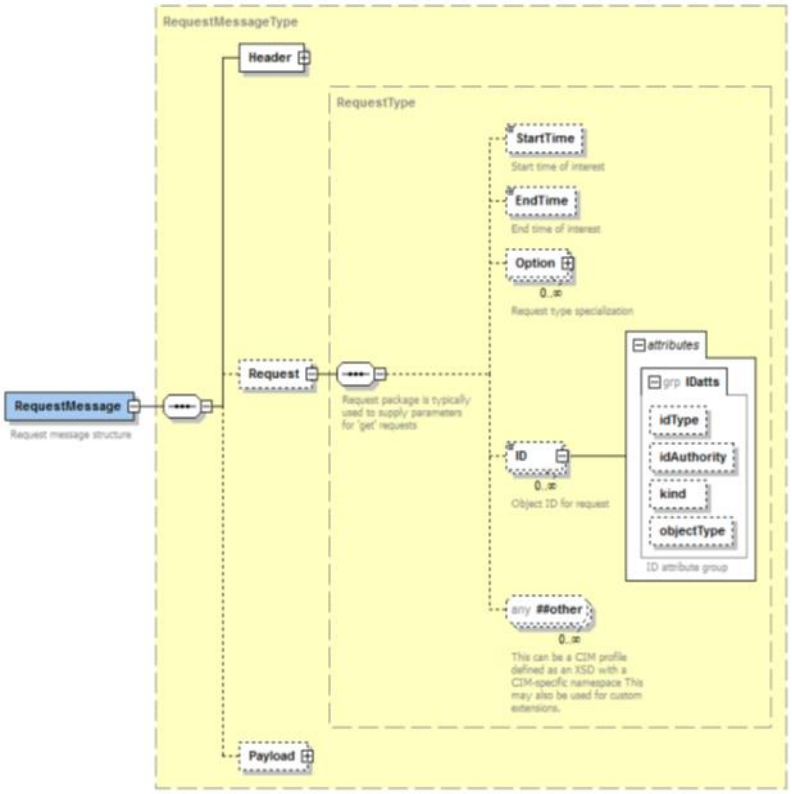
Common Information Model oziroma CIM je standard na področju energetike, ki je sprejet s strani Mednarodne Komisijje za Elektrotehniko (angl. International Eceltrotechnical Comission oz. IEC), z namenom izmenjave informacij o energetskem omrežju med različnimi aplikacijami. Potrebno ga je ločiti od istoimenskega standarda, ki obstaja na področju informacijskih tehnologii, je pa ekvivalenten glede namena, saj standard, ki je namenjen informacijskim tehnologijam se uporablja za opis informacijske arhitekture oz. infrastrukture podjetij.

CIM model je predstavljen kot UML (Unified Modelling Language) model. Standard definira skupni slovar in osnovno ontologijo za posamezne aspekte s področja energetike. CIM model modelira omrežje z uporabo t. i. "modela žic". Le-ta opisuje osnovne komponente, ki so v uporabi z namenom transporta elektrike. [1]

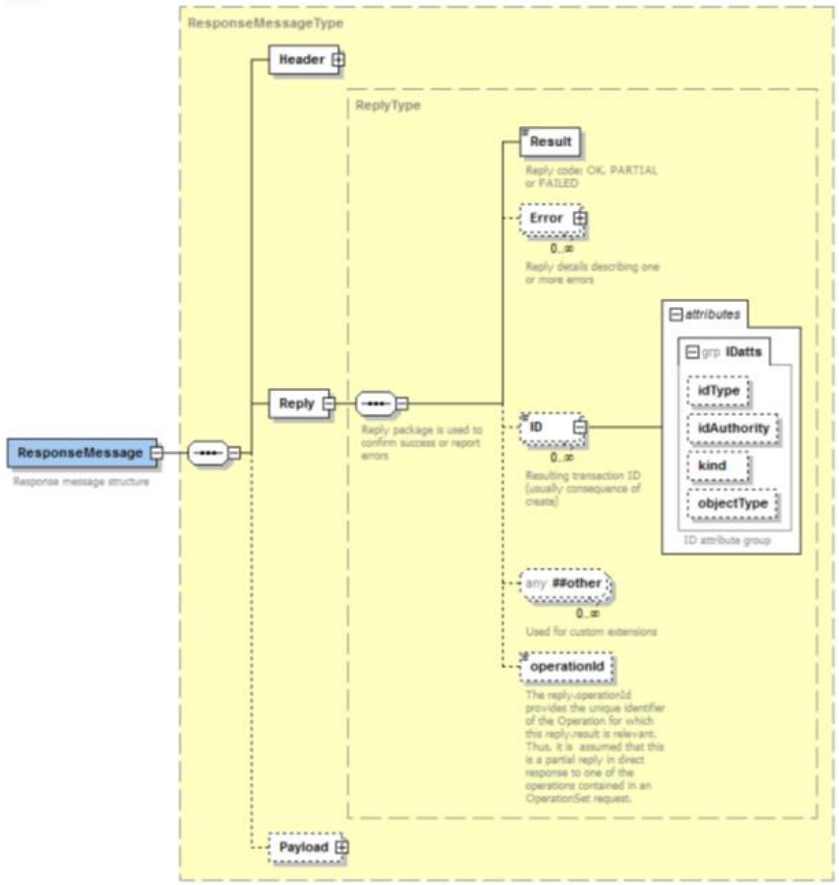
IEC definira različne standarde, ki so namenjeni različnim področjem elektrotehnike. Za nas je bil relevanten standard IEC-61968 (Application integration at electric utilities – System interfaces for distribution management), ki je namenjen integraciji aplikacij pri elektropodjetjih in tudi definira vmesnike za elektrodistribucijska omrežja. [2]

Standard IEC-61968 je dejansko sestavljen iz večih delov, ki spadajo pod to oznako in definirajo strukturo podatkov za predstavitev različnih segmentov električnih omrežij oziroma področij poslovanja. Na primer, standard IEC 61968-3 je namenjen mrežnim operacijam (angl. Network Operations), standard IEC 61968-8 je namenjen podpori uporabnikom (angl. Customer Support), IEC 61968-9 pa je namenjen meritvenim podatkom. [2]

Grafični prikaz zahtevka in odgovora po CIM standardu sta predstavljena spodaj.



Slika 1. Prikaz zahtevka po CIM standardu [3]



Slika 2. Prikaz odgovora po CIM standardu [3]

3. PROJEKT NEDO

Pri podjetju Bintegra smo se v okviru projekta NEDO osredotočili na zagotavljanje interoperabilnosti z implementacijo arhitekture storitvenega vodila (angl. Enterprise Service Bus), oziroma natančneje arhitekturo mikrostoritev. Tehnologije, ki smo jih izbrali za implementacijo rešitve so Apache ServiceMix, ki je skupek različnih odprtokodnih tehnologij za implementacijo integracijskih projektov. Pri integraciji smo uporabili standarde IEC (International Electrotechnical Commision) za izmenjavo podatkov med udeleženci elektroenergetskega trga.

V nadaljevanju so opisane tehnologije in postopke, ki smo jih raziskovali in razvijali tekom projekta.

3.1. Arhitektura mikrostoritev

Pod pojmom "arhitektura mikrostoritev" razumemo različico oziroma nadgradnjo že dobro uveljavljene storitveno usmerjene arhitekture (angl. Service-oriented architecture, v nadaljevanju SOA). To, kar loči arhitekturo mirkostoritev od SOA arhitekture, so lastnosti, po katerih je arhitektura pridobila ime: implementirane storitve so manjše oziroma bolj fino granulirane, protokoli, ki se uporabljajo, pa so prav tako bolj enostavni (angl. lightweight). Prednost takšne implementacije je večja stopnja modularnosti rešitve, kar olajša razvoj z razdelitvijo razvoja na več razvojnih ekip, bolj pogosta ponovna uporaba že implementiranih storitev, lažje lestvičenje (angl. scaling), lažje testiranje in druge prednosti, ki so posledica fine granulacije storitev. [4]

Pri odločitvi glede orodja, ki bi ga uporabili za implementacijo informacijske arhitekture, ki smo jo načrtovali, smo obravnavali več popularnih orodij za implementacijo arhitekture mikrostoritev. Glede na občutljivost rešitve za energetska omrežja in dosedanje navade elektrodistributerjev, da praviloma imajo vso informacijsko infrastrukturo praviloma lokalno na svojih strežnikih, smo izločili oblačne rešitve za implementacijo arhitekture mikroservisov. Dodatne želje oziroma zahteve glede splošnosti in neodvisnosti rešitve od operacijskega sistema so nas pripeljale do ožje izbire treh programskih paketov namenjene integracijam oziroma implementaciji arhitekture mikrostoritev. Vsi trije paketi temelijo na programskem jeziku Java: Apache ServiceMix, Mule ESB in Talend ESB.

Po analizi lastnosti vseh treh programskih paketov, smo se odločili za edino popolnoma odprtokodno rešitev od naštetih: Apache ServiceMix.

3.2. Apache ServiceMix

Apache ServiceMix je odprtokodno storitveno vodilo, ki vključuje različne projekte pod okriljem organizaije Apache, temelji pa na OSGi specifikaciji. OSGi specifikacija je namenjena modularnosti in definira standarde, kako se naj implementira modularno delovanje komponent.

Apache ServiceMix je v svojem bistvu skupek Apache projektov, med najbolj pomembnimi pa lahko omenimo [5]:

- Apache Karaf (vsebnik za izvedbo OSGi komponent, ki temelji na projektu Apache Felix)
- Apache Camel (ogrodje za implementacijo integracijskih vzorcev)
- ActiveMQ (sporočilno-usmerjeno povezovalno programje, ki implementira številne protokole in standarde, vključno z Java Messaging Service (JMS))
- Apache CXF (ogrodje za spletne storitve)

Poleg tega pa ServiceMix okolje ponuja možnosti vključitve tudi za številne druge projekte, kot je npr. podpora za BPM procese z uporabo orodja Activiti [5].

Za implementacijo integracijske logike smo izbrali Apache Camel kot očitno izbiro. Obstajata dva glavna pristopa pri implementaciji integracijskih "poti" (angl. routes) v Apache Camel: Z uporabo domensko-specifičnega jezika (angl. domain-specific language, DSL), ali z uporabo tehnologije Blueprint XML, ki predstavlja izgradnjo integracijske poti s pomočjo namensko oblikovane XML datoteke.

Primer Blueprint XML implementacije integracijske poti, ki premakne datoteko in objavi JMS sporočilo na ActiveMQ je spodaj.

```
<?xml version="1.0" encoding="UTF-8"?>
<blueprint
   xmlns="http://www.osgi.org/xmlns/blueprint/v1.0.0"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://www.osgi.org/xmlns/blueprint/v1.0.0
    http://www.osgi.org/xmlns/blueprint/v1.0.0/blueprint.xsd">
   <camelContext xmlns="http://camel.apache.org/schema/blueprint">
     <route>
       <from uri="file:activemq/input"/>
       <to uri="file:activemq/output"/>
       <setBody>
         <simple>
          FileMovedEvent(file: ${file:name}, timestamp: ${date:now:hh:MM:ss.SSS})
         </simple>
       </setBody>
       <to uri="activemq://events" />
     </route>
   </camelContext>
</blueprint>
```

Izvorna koda 1. Prikaz odgovora po CIM standardu

Pri implementaciji integracij za projekt NEDO smo se odločili za uporabo Blueprint XML pristopa zaradi enostavnosti in preglednosti implementacije integracijskih poti.

3.3. Implementacija za elektroenergetiko

Pri implementaciji arhitekture mikrostoritev, oziroma storitvenega vodila za področje elektroenergetike smo se soočili z izzivi, s katerimi smo se v preteklosti soočali pri implementaciji podobnih arhitektur na drugih področjih:

- Kako pridobiti informacije, ki so potrebne za ustrezno oblikovanje oz. preoblikovanje podatkov?
- Najti ustrezne sogovornike, ki so odgovorni za posamezne zaledne sisteme;
- Poskrbeti za ustrezno monitoriranje in beleženje dogodkov oziroma klicev, ki se izvedejo preko posameznih storitev;
- Poskrbeti za varnostne vidike, saj včasih gre za občutljive podatke in sisteme, saj zato doslej niso bili dostopni od zunaj.

Z implementacijo sodobnih uporabniških scenarijev na področju pametnih omrežij zanesljivost delovanja je ključnega pomena. Za zagotavljanje zanesljivosti pa je potrebno ustrezno nadzirati in beležiti dogajanja. Za infrastrukturno okolje, ki smo ga izbrali obstajajo številna orodja za monitoriranje in logiranje. Mi smo izbrali orodje Hawtio, ki je namenjeno nadzoru vseh javanskih aplikacij.

V skladu s prej omenjenim standardom IEC-61968 smo pripravili SOAP spletno storitev, ki ima vlogo adapterja za komunikacijo za sistem za upravljanje z meritvenimi podatki (angl. Meter Data Management System, MDMS). Spletna storitev ima vlogo adapterja, saj sprejme sporočilo, ki je v skladu s CIM standardi in na ta način zakrije kompleksnost oziroma specifike zalednega sistema MDMS.

V nadaljevanju je prikazan primer klica SOAP spletne storitve za pridobivanje merilnih podatkov, ki smo ga pripravili.

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:mes="http://www.iec.ch/TC57/2008/schema/message">
  <soapenv:Header/>
  <soapenv:Body>
    <mes:RequestMessage>
      <Header>
        <Verb>get</Verb>
        <Noun>MeterReadings</Noun>
      </Header>
      <Request>
        <StartTime>2017-10-11T09:00:00</StartTime>
        <EndTime>2017-10-11T12:00:00</EndTime>
        <ID>skdfgsdiopfzgfseuigipdfzgiauezf8oa7strlaweugfo8a7w</ID>
      </Request>
    </mes:RequestMessage>
  </soapenv:Body>
</soapenv:Envelope>
```

Izvorna koda 2. SOAP sporočilo za zahtevo za merilne podatke

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <ns2:ResponseMessage xmlns:ns2="http://www.iec.ch/TC57/2008/schema/message" xmlns:ns3="http://www.eimv.net/2017/cim">
      <Header>
        <Verb>reply</Verb>
        <Noun>MeterReadings</Noun>
      </Header>
      <Reply>
        <ReplyCode>SUCCESS</ReplyCode>
      </Reply>
      <Payload>
        <ns3:MeterReadings>
          <ns3:MeterReading>
            <ns3:Meter/>
            <ns3:Readings>
              <ns3:value>0</ns3:value>
              <ns3:timeStamp>2017-10-11T09:15:00.000+02:00</ns3:timeStamp>
              <ns3:ReadingType ref="0.0.2.4.1.2.37.0.0.0.0.0.0.0.0.3.38.0"/>
            </ns3:Readings>
            <ns3:Readings>
              <ns3:value>0</ns3:value>
              <ns3:timeStamp>2017-10-11T09:30:00.000+02:00</ns3:timeStamp>
              <ns3:ReadingType ref="0.0.2.4.1.2.37.0.0.0.0.0.0.0.0.3.38.0"/>
            </ns3:Readings>
```

```
    ...
  </ns3:MeterReading>
        </ns3:MeterReadings>
        <Format>XML</Format>
      </Payload>
    </ns2:ResponseMessage>
  </soap:Body>
</soap:Envelope>
```

Izvorna koda 3. SOAP sporočilo za odgovor za merilne podatke

4. ZAKLJUČEK

Področje elektroenergetike doživlja korenite spremembe, s tem pa se odpira pot za informacijski preporod tega področja. Spremembe, kot so uvedba električnih avtov, povečanje deleža distribuiranih in obnovljivih virov energija, pojav »prosumerjev« in znižanje ogljičnega odtisa silijo področje v medsebojno povezovanje med igralci na trgu.

V okviru projekta NEDO smo imeli priložnost odgovoriti tem potrebam in smo v praksi uporabili izkušnje, ki jih imamo z integracijami na drugih domenskih področjih in s tem pospešili oziroma lažje vodili pot k vzpostavitvi arhitekture mikrostoritev v okolju, kjer so sistemi dolga leta živeli kot informacijski monoliti.

5. LITERATURA

[1] https://en.wikipedia.org/wiki/Common_Information_Model_(electricity) Common Information Model (electricity), nazadnje obiskano: 29. 5. 2018

[2] Application integration at electric utilities - System interfaces for distribution management - Part 1: Interface architecture and general recommendations, International Electrotechnical Commision, 2012.

[3] Report on definition of interfaces and integration patterns, Elektroinštitut Milan Vidmar, Iskratel d.d., 2017 (interno gradivo projekta NEDO)

[4] https://en.wikipedia.org/wiki/Microservices, Microservices, nazadnje obiskano: 28. 5. 2018

[5] http://servicemix.apache.org/docs/7.x/user/index.html , Apache ServiceMix documentation, obiskano 24. 5. 2018.

Univerzitetna založba Univerze v Mariboru

INFORMACIJSKA REŠITEV ZA UPRAVLJANJE ODJEMA ELEKTRIČNE ENERGIJE GOSPODINJSKIH ODJEMALCEV

GAŠPER LAKOTA IN TOMAŽ BUH

Povzetek: V okviru slovensko-japonskega sodelovanja na področju pametnih omrežij, t.i. NEDO projekta, se izvaja pilotni projekt upravljanja odjema električne energije gospodinjstev. Gre za enega izmed prvih projektov upravljanja odjema električne energije gospodinjstev v Sloveniji in je namenjen pridobivanju znanja in izkušenj pri gospodinjskih uporabnikih in njihovem obnašanju v primeru različnih načinov upravljanja odjema, prepoznati zahteve in načine integracije različnih informacijskih sistemov za potrebe vodenja elektrodistribucijskega omrežja ter preveriti, kako lahko različne vrste naprav za upravljanje odjema povežemo v celoten sistem upravljanja z energijo v elektrodistribucijskem podjetju.

Ključne besede: • upravljanje odjema • CIM – Common Information Model • napredni merilni sistem • Demand Response • DRCS – Demand Response Control System • električna energija • gospodinjski odjem • aktivni odjemalec

NASLOV AVTORJEV: Gašper Lakota, produktni vodja, Solvera Lynx d.o.o., Stegne 23a, 1000 Ljubljana, Slovenija, e-pošta: gasper.lakota@solvera-lynx.com. Tomaž Buh, produktni vodja, Solvera Lynx d.o.o., Stegne 23a, 1000 Ljubljana, Slovenija, e-pošta: tomaz.buh@solvera-lynx.com.

DOI https://doi.org/10.18690/978-961-286-162-9.8 ISBN 978-961-286-163-6

Dostopno na: http://press.um.si.

1. UVOD

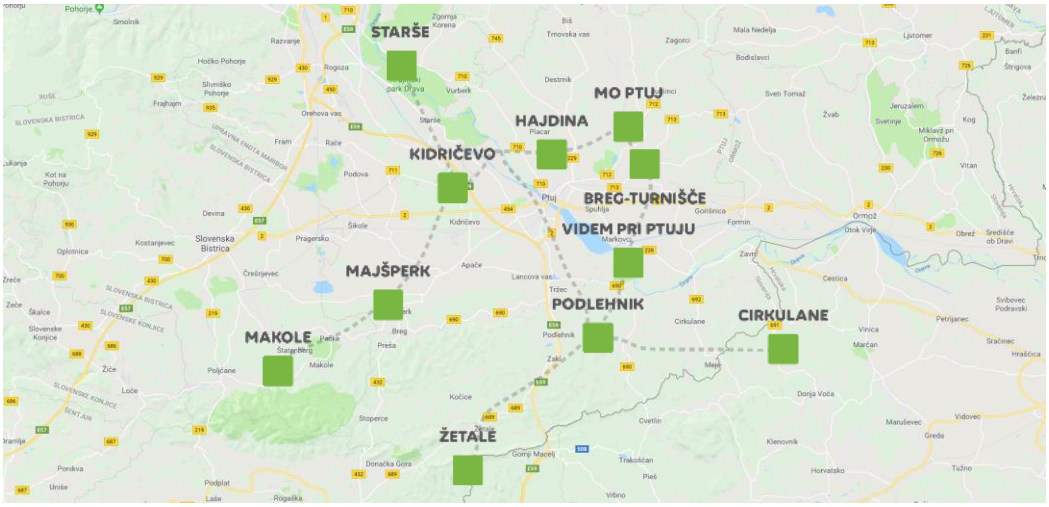
V okviru slovensko-japonskega sodelovanja na področju pametnih omrežij, t.i. NEDO projekta, se izvaja triletni pilotni projekt pametnih omrežij, v katerem so osnovni partnerji Japonska razvojna agencija NEDO in njegov pooblaščeni izvajalec Hitachi ter družba ELES. Poleg družbe ELES bo je slovenski strani v projekt vključenih še veliko število deležnikov, zato ga upravičeno imenujemo nacionalni projekt, zaradi česar je to tudi edinstven tovrsten projekt v Evropi. Sorodni projekti na območju celotne Evropi so osredotočeni na ožje področje ali skupnosti, v našem primeru pa lahko z integriranimi in centralno vodenimi rešitvami v oblaku dejansko govorimo o uvajanju pametnega omrežja na ravni celotne države [1].

Solvera Lynx koordinira enega od 4 sklopov projekta NEDO, in sicer WG3, v katerem sodelujemo z elektrodistribucijskim podjetjem Elektro Maribor, EIMVjem, TECESom, Lokalno energetsko agencijo Ptuj, Kreativnim laboratorijem ter več kot 800 končnimi uporabniki omrežja (, ki so pristopili k projektu Premakni porabo. Gre za enega izmed prvih projektov upravljanja odjema električne energije gospodinjstev v Sloveniji in je namenjen pridobivanju znanja in izkušenj pri gospodinjskih uporabnikih in njihovem obnašanju v primeru različnih načinov upravljanja odjema (angl.: *»Demand Response«*) v času, ko je v omrežju obdobje visoke porabe energije. Prav tako je cilj prepoznati zahteve in načine integracije najrazličnejših informacijskih sistemov za potrebe vodenja elektrodistribucijskega omrežja ter preveriti, kako lahko različne vrste naprav za upravljanje odjema povežemo v celoten sistem upravljanja z energijo v elektrodistribucijskem podjetju.

V okviru projekta je Solvera Lynx zagotovila tudi razvoj programske opreme, sistema za upravljanje vodenega odjema (Deman Response Control System) ter koordinirala namestitev ComBox.L DLC naprav, ki z DRCS sistemom komunicirajo preko LoRaWAN brezžičnega omrežja in namestitev opreme za upravljanje rabe z energijo v gospodinjstvih (Home energy Management System) dveh slovenskih ponudnikov, in sicer podjetja Entia in Robotina. Omenjene naprave služijo za neposredni odklop električnih porabnikov. V projekt Premakni porabo pa je vključenih tudi nekaj več, kot 800 končnih uporabnikov omrežja, ki imajo nameščen zgolj obračunski števec za merjenje pretokov električne energije (v nadaljevanju sistemski števec) in so vključeni v sistem naprednega merilnega sistema. S spodbudo, ki je bila pripravljena s strani Agencije za energijo RS za t.i. aktivnega odjemalca je bila pripravljena shema prilagajanja odjema. Doseženo je bilo, da se končni uporabniki omrežja lahko aktivno vključijo v programe prilagajanja odjema ter tako prispevajo k razvoju naprednih storitev elektroenergetskega sistema [2].

Z vstopom v shemo prilagajanja odjema, odobreno s strani Agencije za energijo RS, so končni uporabniki omrežja za obdobje enega leta postali del skupine aktivnih odjemalcev na območju 10 občin na desni strani Drave, ki jih pokriva razdelilna transformatorska postaja (RTP) Breg. Obenem pa so navedeni aktivni odjemalci postali tudi del pomembnega mednarodnega projekta, ki zajema tako slovenska kot japonska podjetja in bo podal usmeritve za razvoj naprednih tarif za celotno Slovenijo. Projekt pokriva občine Cirkulane, Hajdina, Kidričevo, Majšperk, Makole, Podlehnik, Ptuj, Starše, Videm in Žetale, kot to prikazuje Slika 1.

Slovensko-japonski projekt omogoča zanesljivejšo oskrbo, predstavlja priložnost za domačo industrijo, na podlagi pridobljenih izkušenj pa bo podal usmeritve za celotno Slovenijo.



Slika 1: Območje projekta [1]

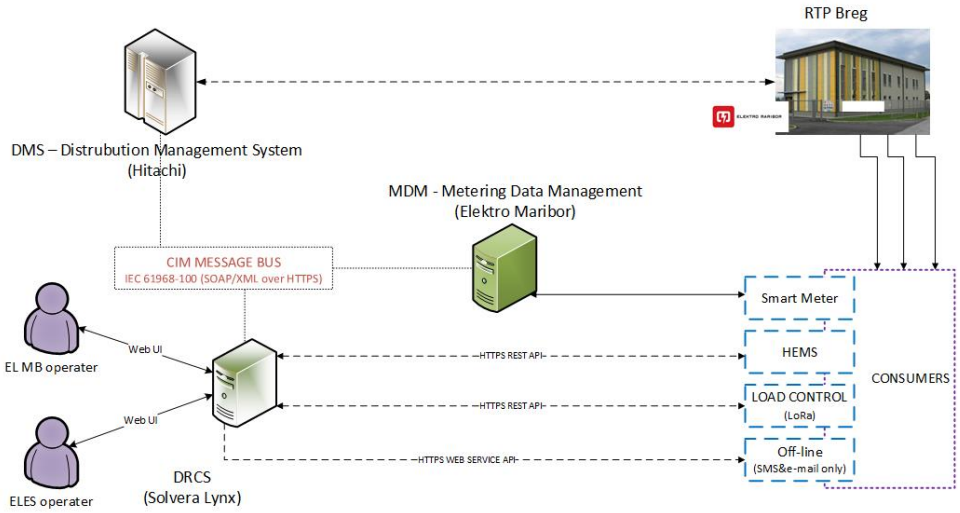
2. ZASNOVA REŠITVE DRCS

2.1. Izhodišča

V sklopu projekta je bila razvita rešitev Demand Response Control System, ki je zmožna sprejemati in izvajati zahtevke za prilagajanje odjema vseh gospodinjstev in malih poslovnih odjemalcev (priključne električne moči <= 43 kW), ki so vključeni v shemo prilagajanja odjema. Prilagajanje odjema se izvaja na avtomatsko zahtevo iz iDMS sistema ali ročno preko uporabniškega vmesnika. Rešitev DRCS izvaja izračun napovedi odjema aktivnih odjemalcev in izračun napoved t.i. potencial, ki pove kolikšno količino odjema je v določenem času je možno prilagodit (zmanjšati) v nizkonapetostnem distribucijskem omrežju. Aktivni odjemalci so obveščeni o želeni prilagoditvi odjema preko SMS in e-mail sporočil, kjer prejmejo informacijo o času in trajanju želene prilagoditve odjema. Del aktivnih odjemalcev je opremljen z napravami za avtomatsko prilagajanje odjema preostali del aktivnih odjemalcev pa mora odjem prilagoditi sam v kolikor le-to lahko stori.

2.2. Zasnova

Rešitev je bila zasnovana, kot sistem za centralno upravljanje odjema (ang. Demand Response Control System - DRCS) gospodinjstev in malih poslovnih odjemalcev, katerih priključna moč ne presega ≤ 43 kW. Zasnovo sistema DRCS prikazuje Slika 2 in koordinira sprejemanje in izvajanje zahtevkov za znižanje odjema električne energije (EE), ki jih DRCS lahko prejme s strani distribucijskega sistema vodenja (angl.: *»Distribution Management System - iDMS«*) preko CIM podatkovnega vodila ali pa preko uporabniškega vmesnika operaterja DRCS na strani distribucijskega ali prenosnega operaterja omrežja.



Slika 2: Zasnova sistem za centralno upravljanje odjema (DRCS)

DRCS je zasnovan na način, da zajema števčne podatke, katere na podlagi analitičnih modelov obdeluje in pripravlja napoved odjema električne energije vseh aktivnih odjemalcev ter hkrati ocenjuje potencial za znižanje odjema. Tako imenovane kazalnike napovedi odjema in potenciala za znižanje odjema DRCS, sproti preko CIM podatkovnega vodila, izmenjuje z iDMS. iDMS tako lahko na podlagi trenutnih razmer v distribucijskem omrežju in njegovih karakteristik lahko poda zahtevo za znižanje odjema električne energije v trenutku, ko je predvideno, da bo distribucijsko omrežje preobremenjeno. DRCS sistem je avtomatiziran sistem, ki samodejno spremlja razmere v distribucijskem sistemu in skupaj z MDM in iDMS samodejno prilagaja odjem električne energije, kar še kako pripomore k zanesljivejšemu delovanju elektroenergetskega sistema, nižjim obratovalnim stroškom in zanesljivejši oskrbi z električno energijo.

Čas, ko je zaznana potreba po znižanju odjema električne energije v distribucijskem omrežju pravimo najava Aktivacije, trenutku, ko se le-ta dejansko izvede pa zgolj Aktivacija. Po prejetem zahtevku za aktivacijo (iDMS ali operater DRCS) se izvede proces najave aktivacije in posredovanja t.i. urnikov (časa v trenutku, ko odklopi priključenega porabnika) vsem ComBox.L DLC ter HEMS napravam. Proces najave aktivacije DRCS sproži klic funkcije na strani Gospodarske javne službe Elektro Maribor (v nadaljevanju Elektro Maribor) za posredovanje najave (SMS ali e-pošta) aktivacije vsem aktivnim odjemalcem. Klic funkcije se prav tako izvede preko CIM podatkovnega modela.

Znižanje odjema porabe električne energije pri aktivnih odjemalcih dosežemo na način, da vse, ki so vključeni v shemo prilagajanja odjema (tako tiste, ki imajo vgrajen zgolj sistemski števec, kot tudi tiste, ki imajo nameščene ComBox.L DLC ali HEMS naprave), obvestimo preko SMS in e-pošte o:

- tipu t.i. aktivacije, ki je lahko izvedena iz strani:
  - operaterja distribucijskega omrežja ali
  - operaterja prenosnega omrežja (sistemska rezerva),
- datumu aktivacije,
- času aktivacije in
- času trajanja aktivacije.

Pri aktivnih odjemalcih, kjer ni nameščenih ComBox.L DLC in HEMS naprav so tako le-ti zgolj pisno obveščeni in se morajo sami angažirati, da znižajo svoj odjem električne energije v času napovedane aktivacije. Aktivnim odjemalcem, ki pa imajo nameščene ComBox.L DLC naprave in HEMS naprave pa se priključeni porabniki v času aktivacije samodejno izključijo in nato po zaključku aktivacije zopet priključijo. Slika 3 prikazuje omenjene naprave za upravljanje odjema.

Slika 3: Naprava za upravljanje odjema (ComBox.L DLC, HEMS tip 1, HEMS tip 2)

3. IZVEDBA

3.1. DRCS

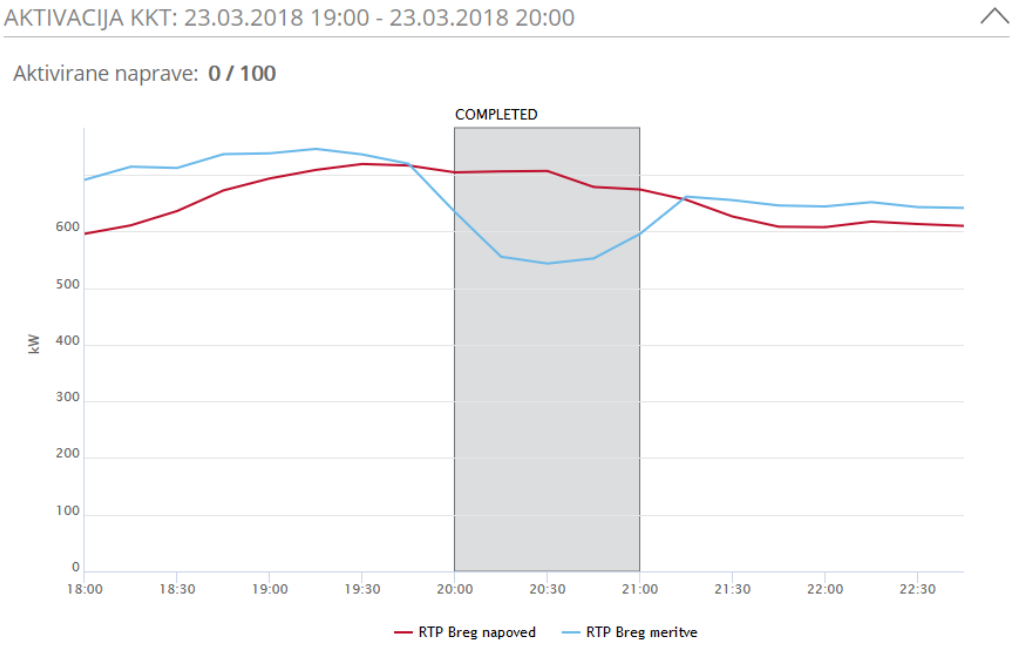
DRCS je namenjen za avtomatsko izvajanje funkcij proženja aktivacij znižanja odjema električne energije aktivnih odjemalcev in periodično izvaja naslednje funkcije:

- pridobivanje merilnih podatkov iz MDM
- izdelava napovedi odjema in napovedi potenciala znižanja odjema za celoten sistem in vsakega odjemalca posebej
- komunikacija z napravami za avtomatsko vodenje odjema
    - Prejemanje statusov
    - Pošiljanje urnikov za prilagajanje odjema

DRCS ComBox.L DLC in HEMS napravam v času najave aktivacije posreduje urnik odklopa priključenega porabnika. Naprave morajo tako v danem trenutku sprejeti zahtevo in le-to tudi potrditi. DRCS tako spremlja odziv naprav in beleži odziv, kot:

- 0#SENT
- 1#SENT_FAILED
- 2#CONFIRMED
- 3#REJECT

DRCS nadalje spremlja odziv vseh aktivnih odjemalcev, kar se odraža v merjenem diagramu katerega meritve zajamemo iz MDM Elektro Maribor. Na podlagi multipla linearne regresije preteklih (obdobje enega leta) merilnih podatkov DRCS izračunava napoved odjema električne energije za posameznega aktivnega odjemalca. Napoved odjema električne energije se izvaja na 15 minutnem merilnem intervalu za nadaljnjih 48 ur. Prav tako številčne (merilne) podatke agregiramo na skupen nivo na podlagi katerih se nato ponovno izvaja skupna multipla linearna regresija preteklih merilnih podatkov, kar nam nadalje poda skupno napoved odjema električne energije vseh vključenih aktivnih odjemalcev. DRCS hkrati izračunava tudi potencial prilagoditve odjema električne energije na podlagi preteklim merilnih podatkov. Primer aktivacije je prikazan na sliki 4, kjer nam sivo okno prikazuje čas napovedane aktivacije, rdeča krivulja nam prikazuje napoved odjema , modra pa merilne podatke vseh odjemalcev, kjer vidimo, da se je dejanski odjem električne energije v času aktivacije znižal za približno 20%. Hkrati ocenjujemo, da se je v danem trenutku poraba električne znižala za približno 150kWh od predvidene napovedi odjema.



Slika 4: Primer pregleda aktivacije

Aktivni odjemalci, ki so vključeni v DRCS sicer delimo v tri skupine:

- skupina A – brez avtomatizacije - informacije o sodelovanju pri izvajanju sistemskih storitev so aktivnim odjemalcem sporočene preko mobilnega telefona (SMS), e-pošte in družbenih omrežij;
- skupina B - uporaba Nadzornega sistema za prilagajanje odjema za pošiljanje informacij odjemalcem. Tistim odjemalcem, ki imajo doma nameščeno hišno avtomatizacijo (Home Energy Management System - HEMS) se avtomatizacija upravljanja odjema izvaja s pošiljanjem signala direktno napravi HEMS;
- skupina C - uporaba Nadzornega sistema za prilagajanje odjema za namen direktnega upravljanja naprav pri odjemalcu (grelniki sanitarne vode , toplotne črpalke , hladilne naprave, ...). Izkoristilo se bo obstoječe sisteme in iskalo sinergije s projektom NEDO.

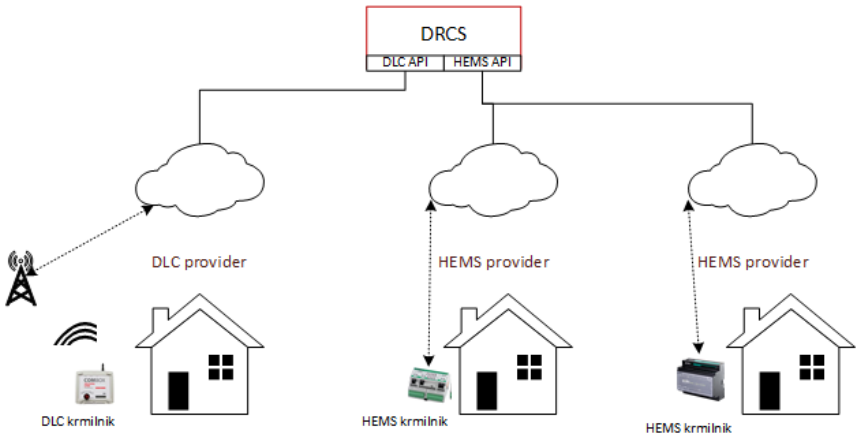
3.2. Programski vmesniki

Za potrebe komunikacije z drugimi sistemi so bili implementirani vmesniki

- CIM 61968-100 za
- Komunikacijo z MDM
- Komunikacijo z iDMS
- HEMS API – RESTful, JSON
- Notification API – SOAP, XML

Na nivoju komunikacije med napravami za upravljanje odjema in DRCS sistemom ločimo dva nivoja komunikacijske poti, kot tudi prikazuje Slika 5. Na prvem spodnjem nivoju med napravami za upravljanje odjema in sistemom za zajem in obdelavo podatkov navedenih naprav poteka komunikacija preko brezžičnega LoRaWAN protokola ali pa omrežne Ethernet komunikacije lokalnega TK ponudnika. Proizvajalci naprav so sistem za zajem in obdelavo podatkov postavili v t.i. oblačno storitev (angl.: *»Cloud services«)* do katere pa nato na drugem nivoju dostopa DRCS. Komunikacija na nivoju med DRCS in t.i. oblakom proizvajalca (v nadaljevanju oblak) poteka preko spletnih storitev (angl.: *»–*

Web Services - WS«), kot so REST/JSON ali SOAP/XML. DRCS za potrebe vodenja naprav za upravljanje odjema podatke zajema in posreduje torej preko spletnih storitev v oblak, kot to tudi prikazuje primer na Sliki 5. Naprave svoje podatke v oblaku zajamemo in obdelajo ter nadalje posredujejo do končne naprave za upravljanje odjema. Za potrebe zajema in obdelave podatkov je v DRCS implementiran t.i. DLC API in HEMS API. Frekvenca osveževanja podatkov na nivoju ComBox.L DLC naprav je 15 minut ter na nivoju HEMS naprav 1 minuta. CIM integracijsko vodilo, ki skrbi za komunikacijo z MDM in iDMS pa je je implementirano na ESB platformi, ki za komunikacijo z integriranimi sistemi uporablja SOAP/XML komunikacijo. Za vsakega aktivnega odjemalca je dnevno na voljo 96 novih 15 minutnih števčnih (merilnih) podatkov.



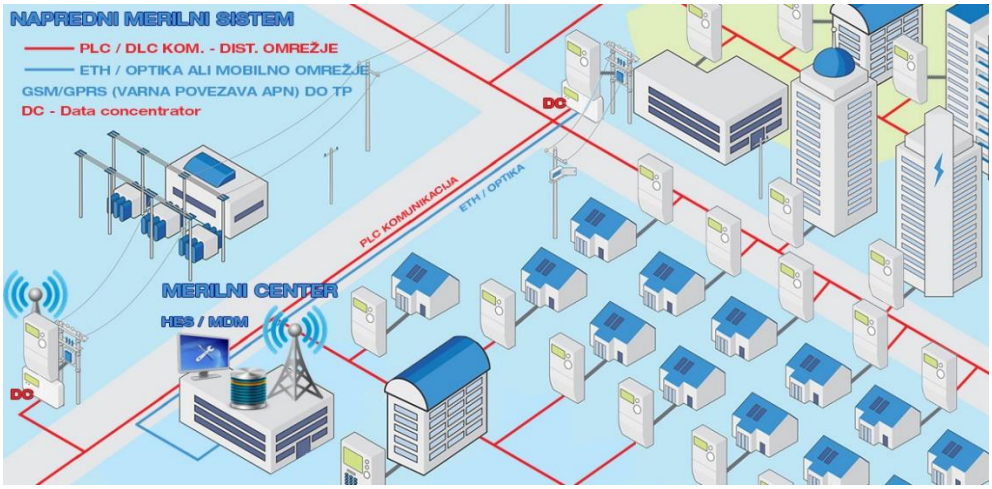
Slika 5: Vmesnik za zajem in obdelavo podatkov naprav za upravljanje odjema

3.3. Meter Data Management - MDM

Gospodarsko javna služba Elektro Maribor (v nadaljevanju Elektro Maribor) ima pri svojih končnih uporabnikih nameščene obračunske števce, ki omogočajo daljinski zajem števčnih podatkov. Za potrebe zajema števčnih podatkov končnih uporabnikov omrežja smo tako v sklopu projekta vzpostavili povezavo med DRCS in obstoječim informacijskim sistem t.i. Meter-Data-Management (v nadaljevanju MDM), ki se nahaja v merilnem centru podjetja Elektro Maribor. MDM združuje množico funkcij, kot je:

- VEE (validacija, ocenjevanje in urejanje),
- povezava z eIS (»MetaData«),
- priprava podatkov za obračun (»Billing«),
- izdelava poročil in
- izmenjava podatkov z drugimi informacijskimi sistemi npr. DRCS.

Del MDM je tudi t.i. Head-End-System (v nadaljevanju HES), ki pa pravzaprav skrbi za potrebe zajema števčnih podatkov na fizičnem nivoju. Za zajem števčnih podatkov je tako uporabljena Power-Line-Carrier (v nadaljevanju PLC) komunikacija s frekvenčno S-FSK modulacijo, ki poteka preko energetskih vodov in omogoča komunikacijo med obračunskimi števci in podatkovnim zbiralnikom (koncentratorjem) v transformatorski postaji. Od podatkovnega zbiralnika do HES pa je uporabljena optična komunikacija. Zajem števčnih podatkov se izvaja tekom celega dneva, HES pa zajem števčnih podatkov iz podatkovnega zbiralnika izvede enkrat dnevno. Celotnemu sklopu pa bi lahko rekli tudi napredni merilni sistem, ki je prikazan na Sliki 6.

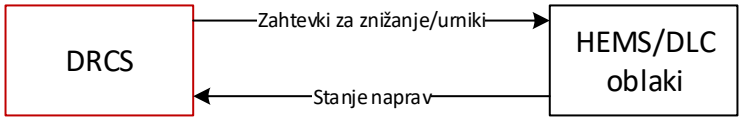


Slika 6: Primer naprednega merilnega sistema

Za potrebe zajema števčnih podatkov na nivoju med DRCS in MDM pa je bil uporabljen CIM podatkovni model, kjer prav tako enkrat dnevno izvajamo zajem števčnih podatkov.

3.4. LoRa DLC

DLC oziroma Direct Load Control je ime za naprave, ki električne porabnike izključijo ali vključijo po prejemu signala ali urnika iz DRCS. V projektu je bila pri 50 odjemalcih nameščena oprema, ki je za komunikacijo z nadzornim sistemom uporabila brezžični LoRaWAN (Long Range, low power Radio) protokol. Končne naprave DLC se brezžično povežejo preko bazne postaje (slika 5), nato pa preko oblačnega Network Management Sistema preko DLC API le-te podatke izmenjujejo z DRCS.



Slika 7: Potek komunikacij med DRCS in HEMS/DLC oblak

DRCS je torej standardno povezan preko vmesnika DLC API in zajema naslednje statusne podatke:

- daljinsko vodenje [vključeno/izključeno],
- stanje aktivacije releja LoRa [DA/NE],
- stanje aktivacije releja [DA/NE],
- stanje potrjenosti urnika [POSLANO/POTRJEN/NEPOTRJEN].

Na podlagi prejeti podatkov nato DRCS obdela statuse in le-te zapiše v enoten zapis, kot le-te tudi obdelujemo na nivoju HEMS API.

Aktivni odjemalec ima možnost izbire ali v trenutku najave želi sodelovati v aktivaciji s premikom stikala Daljinsko vodenje v položaj vključeno ali izključeno. Na ta način se njegov priključeni porabnik bo ali pa ne bo izključil, kar je torej pogojeno z njegovo izbiro.

3.5. HEMS

HEMS oziroma Home Energy Management System so sistemi za upravljanje z energijo v domu. Njihova primarna naloga je omogočiti udobje in pametno porabo energije. Pomemben del HEMS sistema je PLC krmilnik, ki omogoča krmiljenje stikal in naprav, ter zajem meritev in signalov iz merilnih naprav ter senzorike. V projektu so bili pri 50 odjemalcih nameščeni sistemi HEMS, ki so bili

za potrebe projekta razširjeni s funkcionalnostjo sprejemanja signalov in urnikov iz nadzornega sistema. Naprave HEMS so preko komunikacijskega kanala (internet) povezane v ponudnikov HEMS cloud – od tam naprej pa preko HEMS API v DRCS. HEMS API je implementiran z REST/JSON. (slika 5). DRCS torej preko vmesnika HEMS API zajema en statusni podatek, ki je pravzaprav spremenljiv podatek in tako podaja naslednje statuse:

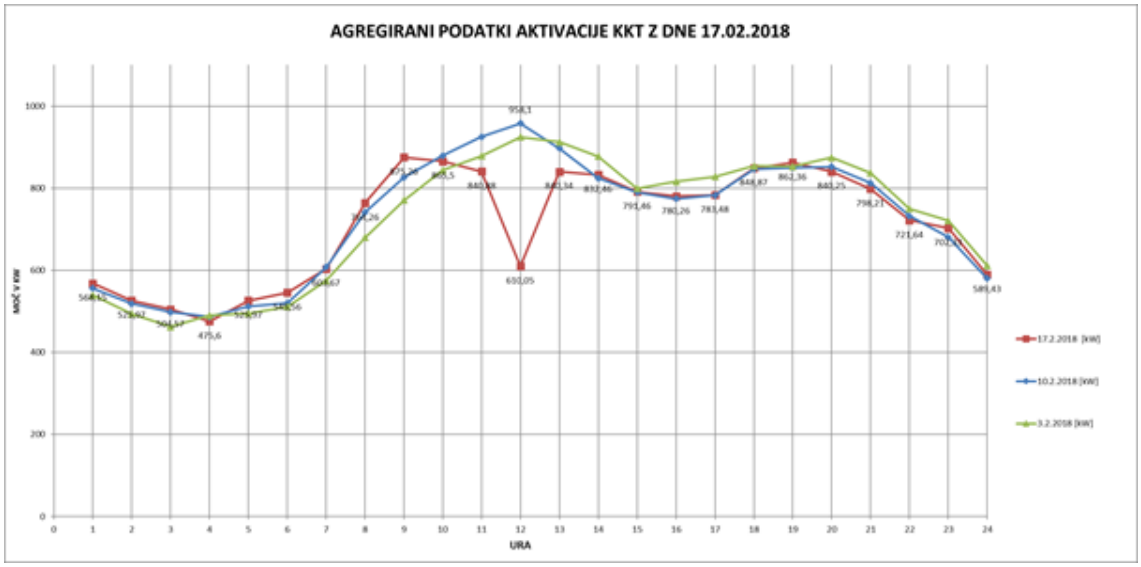
- 0#READY
- 1#NOT_READY
- 2#USER_DISABLED
- 3#ACTIVATION_PENDING
- 4#ACTIVATION_RUNNING
- 5#ERROR

Aktivni odjemalec ima možnost izbire ali v trenutku najave želi sodelovati v aktivaciji s premikom stikala Daljinsko vodenje v položaj vključeno ali izključeno. Na ta način se njegov priključeni porabnik bo ali pa ne bo izključil, kar je torej pogojeno z njegovo izbiro.

3.6. Odziv na prilagajanje odjema

Iz dosedanjih odzivov uporabnikov na nastop kritično konično tarifo (v nadaljevanju KKT) je razvidno, da se obremenitev v času KKT zniža za 30 % in več. V soboto, 17.2.2018, je bilo tako ob napovedi KKT med 11:00 in 12:00 uro doseženo znižanje obremenitve za celo 36%, kar je bilo največ do sedaj. Spodnji graf prikazuje potek dnevne obremenitve na dan aktivacije v primerjavi s kontrolnima dnevoma brez aktivacije. Gre za agregirane podatke uporabnikov, vključenih v projekt (828). Kot primer je na Sliki 8 prikazan prilagojen odjem v času aktivacije kritične konične tarife z dne 17.2. 2018 med 11:00 in 12:00 [1]. Ob izvajanju aktivacije kritične konične tarife so vsi odjemalci skupaj dosegli več kot 30% znižanje odjema, kar je trikratna vrednost pričakovanega odziva oziroma znižanja odjema.

Skupna poraba gospodinjskih odjemalcev vključenih v projekt je v januarju 2018 znašala 338.456 kWh, od tega 0,1 % v času kritične konične tarife (KKT), 48 % v času večje (VT) in 52 % v času manjše tarife (MT). Skupna poraba malih poslovnih odjemalcev vključenih v projekt je v januarju 2018 znašala 139.159 kWh, od tega 0,1 % v času KKT, 48 % v času VT in 52 % v času MT.



Slika 8: Odziv na prilagajanje odjema [1]

4. ZAKLJUČEK

Projekt Premakni porabo traja še nadaljnjo leto, kar pomeni, da bodo še naprej potekalo izvajanje aktivacij testiranje odziva aktivnih odjemalcev. Od začetka projekta in vse do danes je bilo skupno izvedenih že 25 aktivacij. Pilotni projekt je tako odlična priložnost, da tako distribucijski, kot tudi sistemski operaterji pridobijo informacijo kako sistem deluje v praksi in hkrati vidijo ali je zasnova shem in produktov zasnovana na pravilen način ter kaj bi bilo potrebno prilagoditi v prihodnje, da bi lahko zaživel razvoj novih tržnih produktov.

Analiza odziva prilagajanja odjema je pokazala, da se v času aktivacije število angažiranih aktivnih odjemalcev presenetljivo visoko, kar daje velik potencial k razvoju in uporabi tovrstnih storitev. Prav tako je bilo ugotovljeno, da največji izziv predstavlja TK infrastruktura ComBox.L DLC in HEMS naprav, kjer na strani LoRaWAN omrežja izziv predstavlja čim boljšo pokritost omrežja, HEMS naprav pa zanesljivost stalne povezave preko javne TK infrastrukture najrazličnejših TK ponudnikov storitev.

V tujini se opaža porast projektov, ki se nanašajo na upravljanje s porabo z uvedbo tarif, podobno kot je v Sloveniji Agencija za energijo pripravila kritično konično tarifo za pilotne projekte. Iz tega lahko sklepamo, da bodo zniževanje konične obremenitve imelo pomembno vlogo pri zniževanju stroškov v omrežjih v prihodnosti, kar je tudi eden od pričakovanih rezultatov celotnega projekta NEDO. Upravljanje s porabo je že precej razvito pri porabnikih, ki imajo visoke moči odjema in lahko znatno prispevajo k obremenitvi omrežja, vendar je njihovo število majhno. Prav tako pa je na strani gospodinjstev veliko število odjemalcev, ki lahko ponudijo nizke moči, vendar so izredno številčni. Dinamika obratovanja posameznih naprav bo odločilnega pomena pri določitvi, kateri porabniki bomo smiselni in primerni za obravnavo in aktivno vključenost, z uporabo različnih tehnologij pa bo mogoče primerjati uporabnost različnih rešitev.

5. LITERATURA

[1] Spletna stran projekta NEDO, Dosegljivo: https://www.eles.si/projekt-nedo. Dostopano 28. 5. 2018.

[2] FATUR, Tomaž, LAKOTA, Gašper, »Upravljanje s porabo v gospodinjstvih«, 13. konferenca slovenskih elektroenergetikov CIGRE-CIRED, Maribor 2017

[3] Spletna stran Premakni porabo, Dosegljivo: https://premakni-porabo.si/. Dostopano: 28. 5. 2018.

OTS 2018 SODOBNE INFORMACIJSKE TEHNOLOGIJE IN STORITVE
ZBORNIK TRIINDVAJSETE KONFERENCE, MARIBOR, 19. IN 20. JUNIJ 2018
M. HERIČKO IN K. KOUS

Univerzitetna založba
Univerze v Mariboru

MICRO-AMNESIA – HOW MICROSERVICES IN A MESSAGE-ORIENTED ARCHITECTURE SOLVE THE GDPR CHALLENGE

TOMAŽ LUKMAN AND NORMAN SEIBERT

Abstract: The General Data Protection Regulation (GDPR) aims to ensure the protection of EU residents' personal data and thus indirectly introduces requirements on IT systems processing this kind of data. As a development partner for complex projects we were brought in to end the GDPR nightmare for a German automotive OEM. The aim was to develop an IT solution for GDPR in the context of a personalized customer web portal and corresponding mobile apps. We depict the microservice architecture of the minimal viable product (MVP) which uses asynchronous messaging via Apache Kafka, a well-established message broker. This approach allowed us to broaden the scope of our solution in a short period of time and cover a growing number of GDPR-related requirements.

Key words: • Microservices • General Data Protection Regulation (GDPR) • Message-Oriented Architecture • Apache Kafka • Web Portal

AUTHORS: Dr. Tomaž Lukman, iteratec GmbH, St.-Martin-Straße 114, 81669 München, Germany, e-mail: tomaz.lukman@iteratec.de. Norman Seibert, iteratec GmbH, St.-Martin-Straße 114, 81669 München, Germany, e-mail: norman.seibert@iteratec.de.

DOI https://doi.org/10.18690/978-961-286-162-9.9 ISBN 978-961-286-163-6

Dostopno na: http://press.um.si.

1. INTRODUCTION

As of 25 May 2018, the General Data Protection Regulation (GDPR) [1] applies to all IT systems that process the personal data of EU residents. GDPR is a massive challenge for most customer-facing companies and for those burdened with a multitude of legacy systems it is close to a nightmare. In this paper we describe the development of an IT solution for GDPR in the context of a personalized customer web portal and corresponding mobile apps. In this context personal data is processed in over seventy distinct systems/services.

First, we will take a look at the requirements that stem from GDPR and then describe the initial IT ecosystem of the aforementioned portal. We will describe how we addressed the most important requirements with a minimal viable product (MVP) created to erase personal data. We will depict the microservice architecture of the MVP which is based on message-oriented architecture.

We will also show how the architecture of our solution has evolved to cover more and more GDPR-related requirements. One main architecture requirement was horizontal scalability since it was initially not clear how many systems/services in the portal ecosystem persist personal data. At the end we describe how the chosen architecture allowed us to broaden the scope of our GDPR solution in a short period of time. As a result, all sales-related systems are covered today.

2. REQUIREMENTS

In this section we will first look at GDPR and then look at the desired software requirements derived from our client and the principles governing GDPR.

2.1. General Data Protection Regulation

GDPR is the most important change in data privacy regulation in 20 years. It regulates many aspects of handling EU residents' data and therefore has a vast impact on the processes of customer-facing organizations and their IT systems. The organizations cannot ignore GDPR because failure to comply could result in substantial fines of up to 20 million euros or 4% of the worldwide annual revenue of the prior financial year - whichever is higher.

It is driven by a philosophical approach to data protection, based on the concept of privacy as a fundamental human right [2]. The following important principles are defined by GDPR [3]:

- Art. 13 and Art. 14 "Right to be informed" – individuals have the right to be informed about the collection and use of their personal data.
- Art. 15 "Right of access" – the right for individuals to be provided with a copy of their processed personal data upon request.
- Art. 16 "Right to rectification" – the right for individuals to have inaccurate personal data rectified or completed if it is incomplete.
- Art. 17 "Right to erasure" or the "right to be forgotten" – the right for individuals to have personal data erased upon request. This right is not absolute and applies only in certain circumstances. Data that has to be kept to comply with a legal obligation must not be erased.
- Art. 18 "Right to restrict processing" – right to request the restriction or suppression of their personal data.
- Art. 20 "Right to data portability" – allows individuals to obtain and reuse their personal data for their own purposes across different services.
- Art. 21 "Right to object" – right to object to the processing of their personal data in certain circumstances.
- Art. 22 "Rights in relation to automated decision making and profiling" – additional rules to protect individuals if you are carrying out solely automated decision-making that has legal or similarly significant effects on them.

Together with the client's team consisting of members from the sales department responsible for the web portal and the legal department, we considered the relevance of these principles for our project. In our analysis we generated a prioritized list of relevant requirements. The "right to be informed" had already been implemented in the frontend of the portal and demanded only minor text changes. The right to object (Art. 21) had been taken into account in the right to erasure (Art. 17) so that exercising it would lead to data erasure. Art. 22 does not apply to the web portal. The following prioritized list was compiled:

1. "Right to erasure" or the "right to be forgotten" – is the first requirement that was implemented and is the main focus of this paper.
2. "Right of access" and "right to data portability" – was implemented in the second step and is briefly described in the second part of the article. The gathered personal data is used to comply with both requirements. The later requirement is being met with a JSON file and the former with a PDF file that contains a spreadsheet and is derived from the JSON file.
3. "Right to restrict processing" – this is solved in a similar way to data erasure but instead of erasing the data it is being locked in order to avoid further processing. The personal data of a user can also be unlocked. This particular point is beyond the scope of this paper.
4. "Right to rectification" – has been initially solved in a manual way via a business process in so far that tickets are assigned to the support team of a specific system containing the inaccurate data. We will design and develop an automated IT solution for this requirement soon. It will extend the solution presented in this paper.

Some IT professionals will be looking at these requirements and saying to themselves "Jeez, that's CRUD (Create, Retrieve, Update and Delete) [4] without the Create part, so it is easy to implement". This is true if one ignores the various distributed and heterogenous systems that have to be included in the solution. We will look at the portal's IT ecosystem in the next section.

2.2. Requirements for the GDPR Solution

In addition to the basic requirements stemming from GDPR, the client and our team defined the following requirements that had to be met by the GDPR solution for the portal:

- Each user has to be able to trigger a GDPR compliant erasure job in the portal by clicking on the erase account button and also be informed that the erasure process has begun.
- All personal data of a user has to be erased within 1 month after the erasure of the account was requested.
- Each erasure job has to be delivered to all relevant systems/services in a reliable and a comprehensible way.
- Success or failure of processing an erasure job has to be trackable for each system (via acknowledgments).
- A job is successful after all relevant systems send an acknowledgment with success within a configured time.
- After the configured time has passed without success or failure acknowledgement, the job has to run into an auto fail (timeout in one or more systems).
- A small group of employees on the client side has to be alerted about each failed job by e-mail.
- A small group of employees can review the status of each job using a graphical user interface (GUI) and see which systems have succeeded and which have failed to process the job.
- A small group of employees can create an erasure job for a user over a GUI.
- If the personal data that has already been erased is being recovered at one or more of the relevant systems through restoring a backup it has to be possible to immediately trigger the erasure of this data again.

3. INITIAL IT ECOSYSTEM

The personalized web portal is built out of more than hundred features for users that own one or more vehicles and users that are configuring their new vehicle(s). Technically the web portal consists of a

many separate, loosely-coupled systems/services. A fictional example derived from this architecture is depicted in Figure 1. For our GDPR context, the most important information is that roughly seventy distinct components i.e. systems/services process and/or are involved in the persisting of personal data. This fact presents a major challenge for complying with the above mentioned GDPR requirements. We'll look at some possible approaches on how to solve this in the next section.

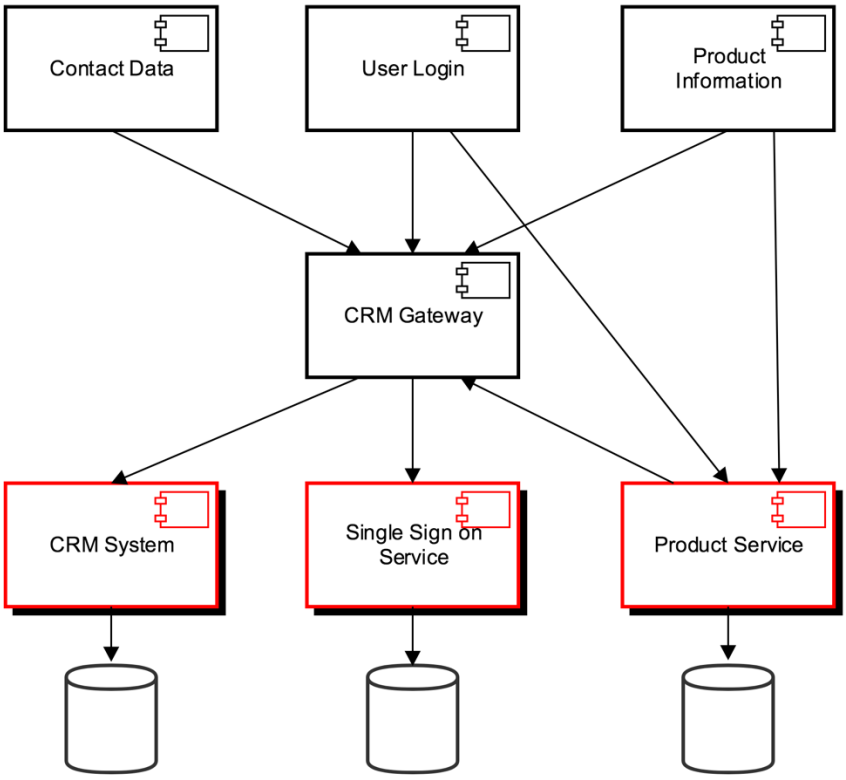


Figure 1. Fictional example of the portal components (highlighted are the ones that persist personal data).

4. ENABLING DIGITAL AMNESIA

In this section we will first take a look at the initial options to develop our GDPR solution and then take a look at the architecture of our minimal viable product (MVP).

4.1. Design decisions

The classical approach would have been a central solution that has access to all databases and implements all the logic to erase all connected systems' personal data in one place. This approach was appealing to some "old-school" IT architects at our client's central IT department. However, a detailed knowledge of all personal data across all of the systems would have been required which would have taken years to develop and cost a small fortune. Even these architects agreed that such an approach was not suitable since access to various persistence mechanisms would have had to have been made available or all existing systems would have had to be refactored to share one database. The latter is a problem for many of the legacy systems and more importantly is the opposite to the microservice pattern "Database per service" [5]. Currently a number of microservices, which we introduced to the ecosystem, are already available and according to the IT strategy of the corporation such systems are preferred in the future.

Despite the decentralized persistence, one of our client's requirements was a central management module to create erasure jobs[17] and monitor their status. In essence, there were two options available [6]:

- **Point-to-point communication**. The management module has to know about all the systems/services with personal data in detail and depends on their erasure interfaces. Additionally, all the systems/services are dependent on its acknowledgment interface. Currently this is the only type of working integration in the portal and it is predominately implemented in a synchronous way.
- **Messaging**. The management module does not need knowledge about the other systems' implementation or data design since they are only coupled loosely over messages. This is the idea of a message-oriented architecture also known as message-oriented middleware [7] and follows the publish/subscribe pattern [6]. It is based on asynchronous communication through an additional massaging component. Many years ago, in our client's portal ecosystem, an Enterprise Service Bus (ESB) was introduced but this approach has failed like many others [8]. The main reason was that business logic was introduced into the ESB and configuration and integration became complex and expensive. Today alternatives to this heavy messaging approach exist and provide good results.

For our project, the list of the systems/services was long and it was not clear if these were all of the systems and how long it would take for a system to process an erasure job. Asynchronous communication and loose coupling was the obvious choice.

4.2. The MVP – Minimal Viable Product

Our MVP covers the "right to be forgotten" and addresses the above-mentioned requirements. We decided to develop a small specialized service, commonly known as microservice [9], that was concerned with generating erasure jobs and gathering erasure acknowledgements. A common technical solution for communication between microservices is to use a message broker.

Decoupling services via a lightweight message broker and moving to asynchronous, parallel processing instead of failure-prone chains of synchronous requests, was a big paradigm shift. With our MVP a first step has been made but it will take more time and effort to adapt this pattern on a larger scale.

The architecture of our MVP is depicted in Figure 2 and comprises the following components:

- a message broker,
- a microservice for data erasure, called GDPR Service, and
- an arbitrary set of systems/services that process or persist personal data.

To check our concept and documentation we integrated one of our own services first. This microservice enables the user to request or book service appointments for his vehicle(s). To ease the integration for other solution providers, we even developed and published a mock subscriber.

[17] and later other GDPR jobs

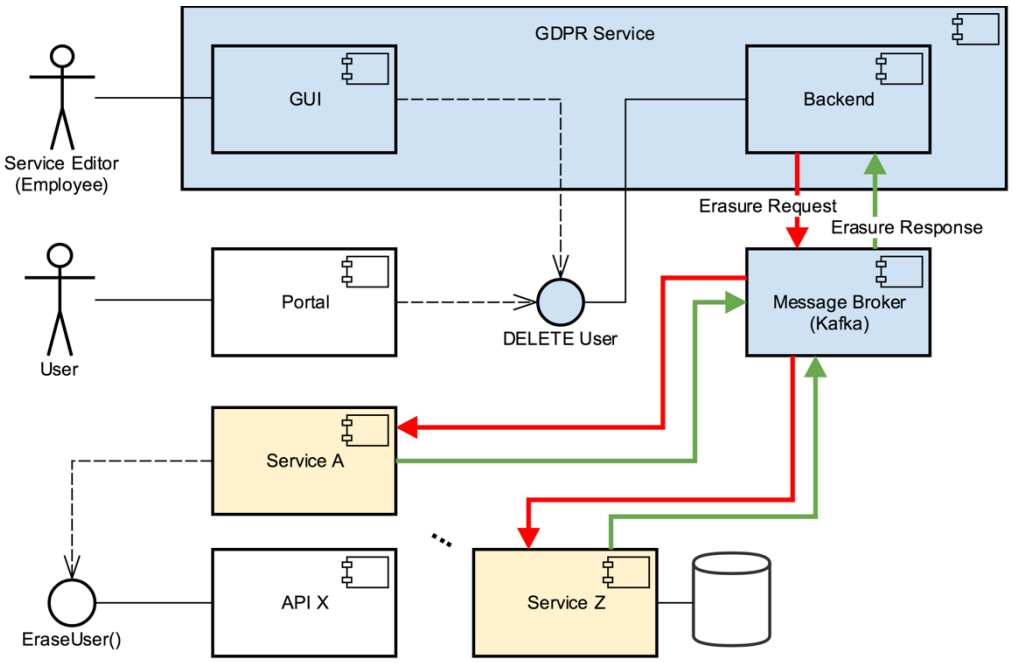


Figure 2. MVP of the GDPR solution.

4.3. Message Broker

The introduction of a message broker was the greatest paradigm shift for our client and therefore had to be addressed as quickly as possible. The main question that had to be answered was "Which message broker to choose?"

We had evaluated various message brokers even before we started the GDPR project:

- HiveMQ [10] – is based on the MQTT (Message Queuing Telemetry Transport) protocol used mainly for IoT devices and simple use cases. E.g. the protocol does not address message caching. Although HiveMQ offers additional features that go beyond MQTT we dismissed it because it was designed for devices.
- RabbitMQ [11] – is a classical message broker with a wide feature set. It is a solid and proven solution that would have been suitable for the current portal which is hosted in a computing center. It is less suitable for cloud-based services since it only supports vertical scalability. Additionally, it is known to be difficult to configure.
- Apache Kafka [12] – a performant message broker often used in a cloud environment. One major advantage is the focus on a small set of powerful yet simple concepts. It is horizontally scalable, provides a distributed commit-log and is therefore able to optimally cache messages. The final and most important reason for choosing Kafka is specific to the GDPR erasure project: Kafka can resend messages via resting its pointer. This addresses our requirement: If the previously erased personal data is being recovered at one or more of the relevant systems through restoring a backup it has to be possible to immediately trigger the erasure of this data.

So, what are the most important concepts of Apache Kafka in a nutshell? They are [13]:

- Messages – a simple key/value pair. The key and the value can be anything serializable. It can be sensors measurements, meter readings, user actions, etc.
- Topics – is a logical grouping of related messages, similar to a channel. A topic can be "water meter readings" or "user clicks".
- Partitions – topics are physically split into partitions. A partition lives on a physical node and persists the messages it receives. A partition can be replicated onto other nodes in a master/slave relationship.
- Producers – publish their messages into the chosen topic and partition.

- Consumers – receive messages and are organized into consumer groups. Every consumer inside a group is assigned one (or more) partitions of the topics it subscribes to. These subscribers consume messages only from their assigned partitions.

4.4. GDPR Service

The GDPR Service itself covers the following concerns:

- **Creating an erasure job**. Then persisting it and sending out the corresponding erasure message.
- **Managing a service registry per topic**. The service registry is basically a list of consumers subscribed to the topic since Kafka does not provide an API for acquiring this information. The list is needed since we need to know how many and which systems are expected to send a successful acknowledgment to mark the job as successfully processed and thereby confirming that all personal data for a user has been erased. This can be seen in Figure 3.
- **Receiving acknowledgments and updating the job state**. All acknowledgments received for a job are persisted and evaluated to compute their impact on the state of the job. If one or more systems from the registry send a failure acknowledgment (see Figure 4) or does not send an acknowledgment at all in the configured timespan (see Figure 5) the job is marked as failed.
- **Alerting via Emails**. Any termination of a job triggers an email with the corresponding alert.

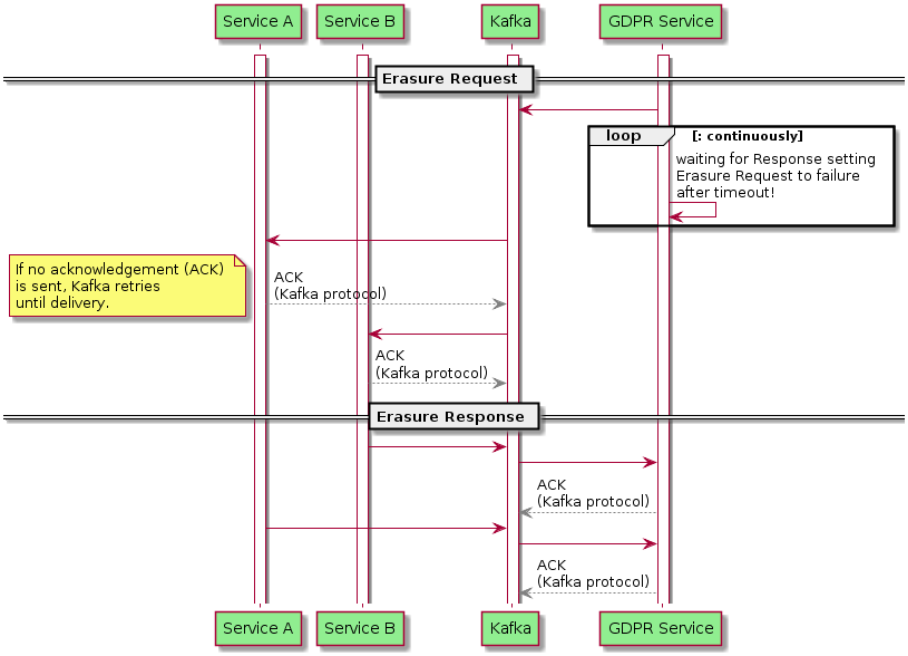


Figure 3. Successful erasure.

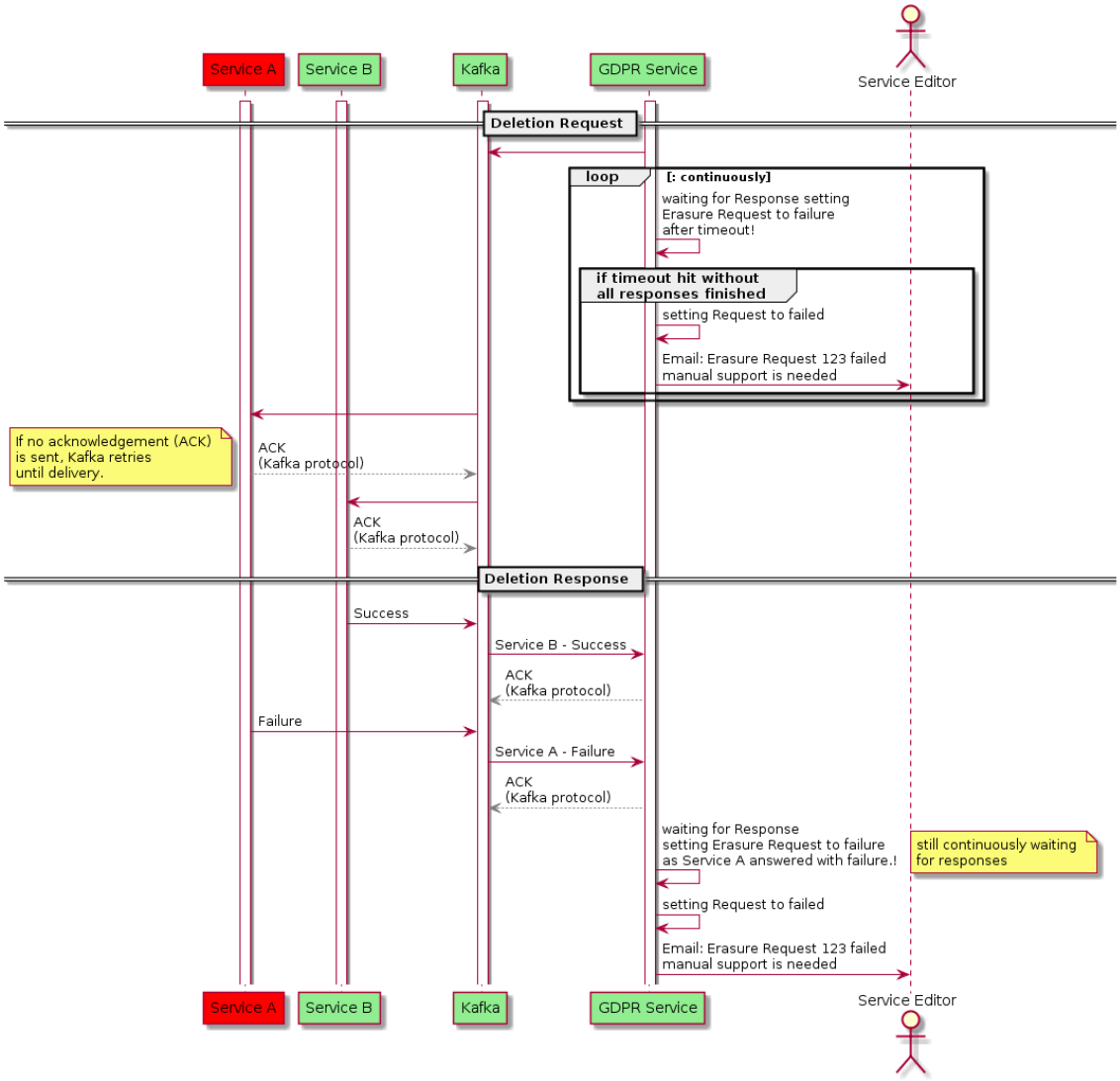


Figure 4. Failed erasure due to an acknowledgment with a failure.

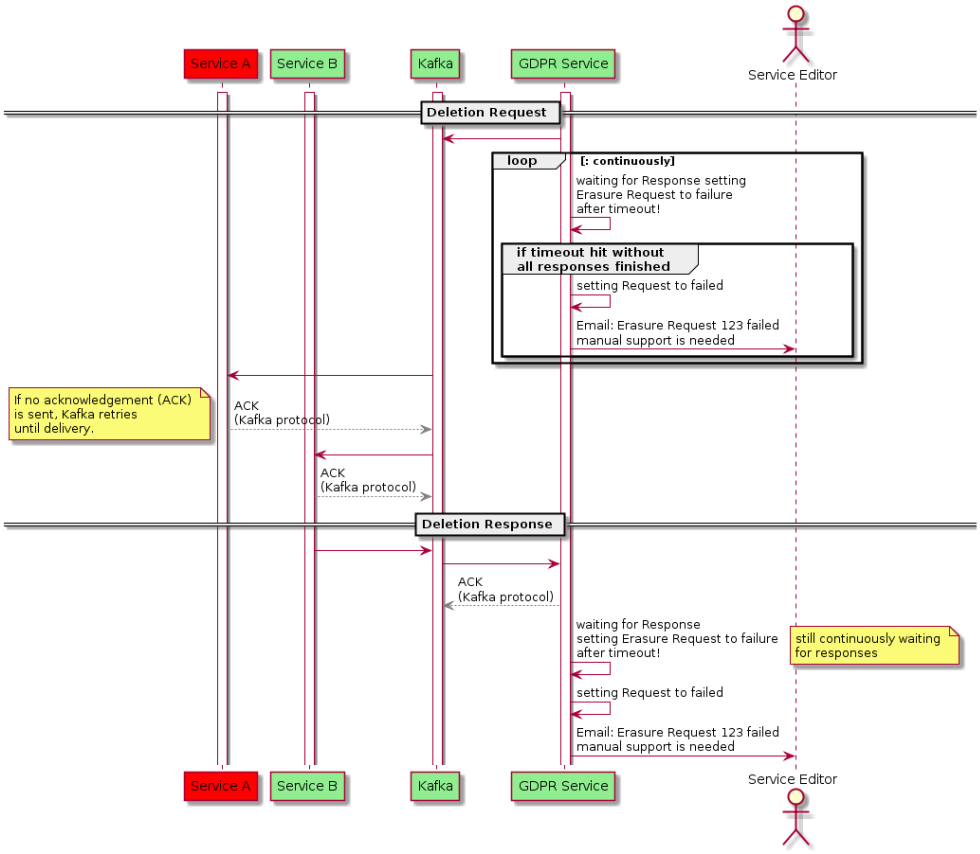


Figure 5. Failed erasure due to a missing acknowledgment (time-out).

The GDPR Service is a microservice based on a typical microservice technology stack with Spring Boot [14] and Angular [15]. Notable further components are the RESTful interface with Swagger Documentation [16] and a Spring Kafka adapter [17]:

- The **RESTful interface** provides a DELETE Operation that is called by two consumers at the moment:
    - A GUI for a small set of employees at our client and
    - The web portal in which the users can erase their account.
- The **Kafka adapter** is connected to two topics. The GDPR Service acts as a producer on the "gdpr_erasure_request" topic and as a consumer on the "gdpr_erasure_response" topic. In this case the contract or more specifically the interface are the messages that are being exchanged through these topics: the " gdpr_erasure_request_event" and the " gdpr_erasure_response" message. These can be seen in Listing 1 and Listing 2.

```
{
  "messageId": "e8672a08-473e-4047-9881-fa92218919bc",
  "messageTime": "2018-04-23T15:47:30.334Z",
  "messageSource": "GDPRS",
  "useCase": "deletion",
  "requestId": null,
  "payload": {
    "recordId": "227ca2e9-4cf3-4dc6-b626-e34078242f04",
    "userId": "W424776885",
    "email": "bilbo.baggins@example.org"
  }
}
```

Listing 1. Erasure request example.

```
{
  "messageId": "ad3c9cd9-c99a-4871-a11c-74f3566119df",
  "messageTime": "2018-04-23T15:47:45.334Z",
  "messageSource": "SAS",
  "useCase": "deletion",
  "requestId": null,
  "payload": {
    "recordId": "227ca2e9-4cf3-4dc6-b626-e34078242f04",
    "status": "SUCCESS"
  }
}
```

Listing 2. Erasure response example.

4.5. Services/Systems that process personal data

Erasing the actual personal data is the concern of the systems that persist this data and/or relay this data to other systems/services that cannot be connected to Kafka. Each system owner has to know what kind of personal data is processed by his system and has to take care of either invoking the erasure operation (synchronous) of the other systems or informing the personnel responsible for them.
To minimize the probability of erroneous implementations or of missing important aspects we wrote a guide on how to implement the erasure operation. It covers aspects like backups, caching and logging.
We also wrote a guide on how to implement the Kafka adapter. It contains best practices and code snippets. The guide introduces two important requirements for every system triggered by the GDPR Service:

- Implement the tolerant reader pattern [18].
- The deletion operation has to be idempotent [19]. This means that an operation can be applied multiple times without changing the result beyond the initial application. In our case the deletion operation must handle more than one deletion message for the same user ID and must produce the same result.
    - One of the reasons for this requirement is that the “at least once” delivery strategy had to be chosen in Kafka for it to be high performant. This means that in very rare cases the same message can be delivered more than once.
    - If one of the systems with personal data sends an acknowledgement with a failure or no acknowledgement the whole job is marked as failed. In this case another job for the same user must be created and a message for the same user will be sent at a later time to all the consumers. Even the systems that cannot erase this user anymore because no data is available anymore have to send an acknowledgment with success. This is one of the cornerstones of our GDPR solution.

5. ADDITONAL REQUIREMENTS AND A NEW MICROSERVICE

During implementation, an additional requirement emerged since it became evident that many systems needed to send emails to third parties for each erasure job. It was required to send an encrypted email to every address in a list, informing them about a deletion request from a portal user.

Instead of extending the GDPR Service we created a new microservice specifically covering this independent requirement. The new microservice consumes erasure messages and upon receiving one sends out emails to all addresses in a list and then sends an acknowledgment over Kafka. This GDPR2Mail Service is depicted in Figure 6.

A nice extra benefit to this was that we could also inform all the system owners who did not manage to connect their systems to Kafka in time about every erasure job. All they had to do was agree to monitor these email inboxes and trigger the erasure operation manually. This also motivated them to increase the pressure and velocity of their implementation projects for their Kafka adapters.

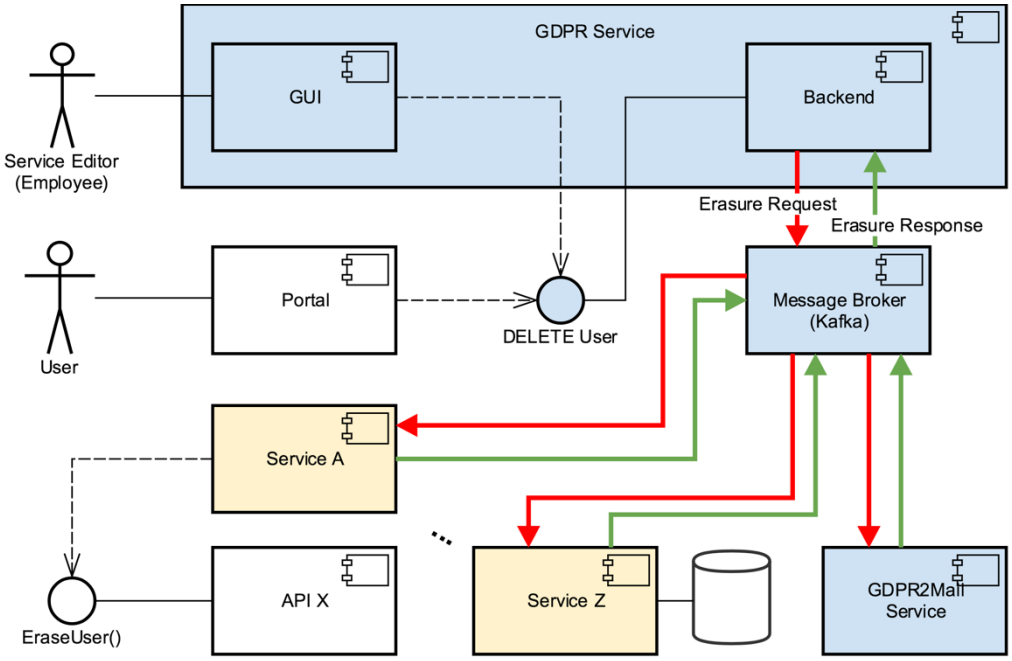


Figure 6. GDPR solution with an additional microservice (GDPR2Mail Service).

6. EXPANDABILITY? NO PROBLEM!

The "right to be forgotten" was a suitable starting point due to the straight-forward requirements and necessary processes. The portal user can trigger the erasure himself and the failed jobs are then handled by a few employees. No communication towards the user is necessary since the data has to be erased regardless of any circumstances within one month of the erasure request.

The next step was to implement the "right of access" and the "right to data portability" for the portal. For inquiry of the saved personal data and a data export, our project team had to also think about a new business process, since these results have to be delivered to a user after the end of processing. Interestingly enough, at that time a GDPR taskforce for all sales departments was created and was focusing on defining the necessary business processes to implement GDPR. The taskforce became aware of our solution for the portal and invited us to join forces with them and to develop an integrated solution consisting of business processes and an automated IT system for GDPR. In this section we will present the technical side of this solution.

We shall begin by taking a look at what the taskforce had produced parallel to us. They introduced a process in which call center agents receive all GDPR requests (except the ones triggered in the portal) and create a ticket for each of them in a ticketing system, more specifically Jira. The coordination of the following activities is done via the Jira ticket and their initial idea was that most of the activities were to be executed "manually". This would be very labor intensive.

The idea was to integrate the results of the taskforce and our GDPR solution in two steps:

- **Extend data erasure to all sales systems** – integrate Jira and our GDPR solution to enable erasure in all of the roughly hundred and fifty sales related systems.
- **Implement data inquiry and data exchange for all sale systems** – implement an inquiry that generates a PDF with all of the personal data and produces a JSON with all of this data and thus address the "right of access" and "right to data portability" for all sales related systems.

6.1. Extend data erasure to all sales systems

This step was not a big challenge, because of the modern and scalable architecture of our MVP. The MVP was integrated with the taskforce results to produce the architecture in Figure 7. The following changes had to be executed:

- **Add data fields to request**. Since the additional sales systems did not all have the same key to identify a user, additional data fields like name and address had to be added to the whole chain.
- **Extend payload of the REST operation**. Jira could generate an erasure job over the existing REST operation. We just had to extend it with some optional fields. The GUI for GDPR Service and the implementation in the portal remained the same.
- **Add data fields to GDPR Service**. The erasure request message had to be extended with additional fields. This meant a change to the Backend. However, the systems already connected to Kafka remained the same since their adapters had implemented the tolerant reader pattern.
- **Connect new systems to Kafka**. The additional sales systems had implemented a Kafka adapter.
- **Inform Jira**. The GDPR Service had to add a REST call to Jira into its state machine. This way it could inform Jira if the job was successful or if it had failed.
- **Extend the cleanup job**. In the GDPR Service, all the additional personal data had to be erased after a job was terminated.

Notice that no changes to Kafka had to be made. Due to its scalability it could handle the additional eighty systems with ease. Most of the changes had to be made because new data fields had to be added to the request to enable the new sales system to find the users in their database.

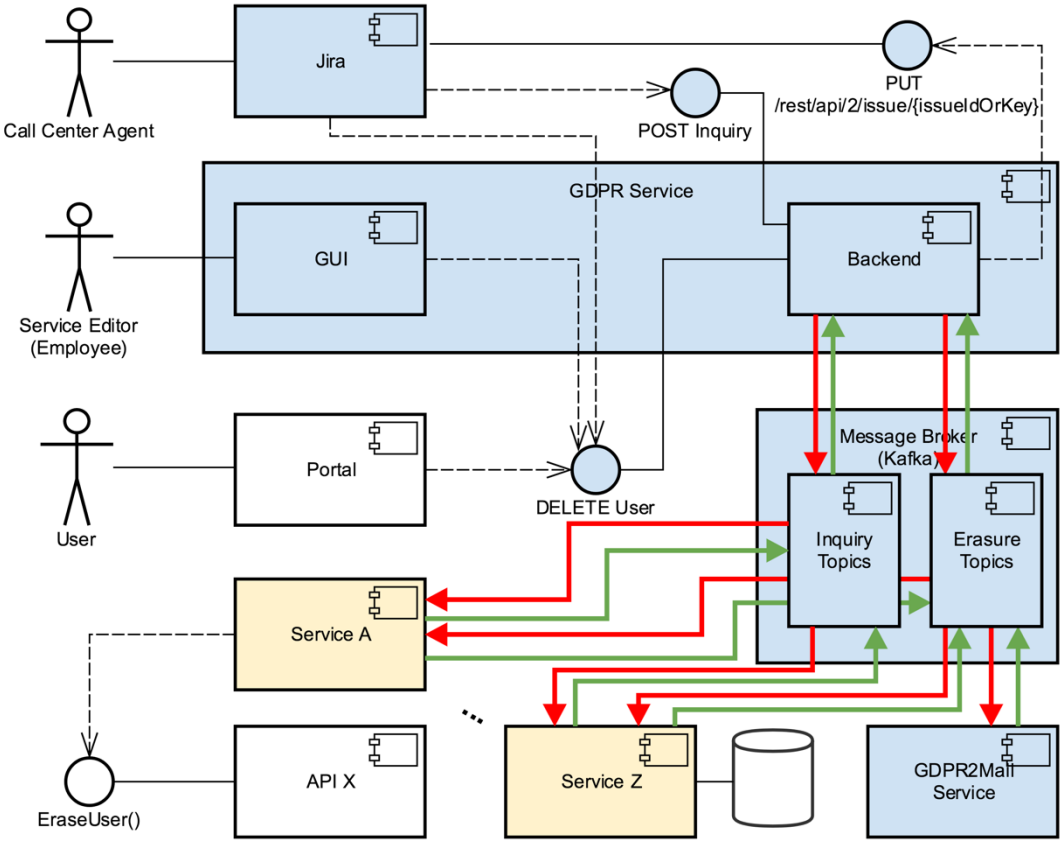


Figure 7. Expanded GDPR solution (erasure for all sales systems and also data inquiry).

6.2. Implement data inquiry and data exchange for all sales systems

Implementing this was more demanding, but due to the architecture of our MVP, many aspects could be solved similarly to erasure (see Figure 7). The following changes had to be made:

- **Define a data exchange format**. A flat JSON was chosen to make the data consolidation from all systems into one file easier. A list of well-known personal data fields was compiled so that duplicated data could be minimized. For all additional data fields each system had to define a name that was understandable and write it into the JSON instead of using cryptic DB field names.
- **Add Topics for inquiry in Kafka**. The "gdpr_ inquiry _request" and the "gdpr_inquiry_response" topics were created.
- **Add the inquiry channel in the GDPR Service**:
    - Additional REST Operation for inquiry.
    - Addition type of job.
    - Extension of the service registry for the new topic.
    - Consolidation of JSON files into one JSON file.
    - Extension of the call to Jira with the consolidated JSON.
- **Implement connection to the new topics in all sales systems/services**.
- **Implement an operation for data inquiry in all the sales systems**.
- **Implement a PDF generator**. A PDF generator was already available in Jira so it could cover the need to generate the PDF from the consolidated JSON file. This way the Jira ticket contained the results for data inquiry and data exchange. These results could be delivered to the user through the following activities of the new business process.

7. CONCLUSION

The experience outlined in this report has shown how an automated IT system that addresses various demanding GDPR requirements was developed based on microservices and a message-oriented architecture.

The microservice approach has enabled us to quickly develop a GDPR Service concerned with creating and distributing erasure jobs and consolidating the results of erasure operations from systems with personal data. We were able to address a new concern that did not fall directly under the GDPR Service remit in the form of a new microservice. This way the handover of GDPR Service to the operations team was not endangered. The new GDPR2Mail Service was rapidly developed and went live within a few weeks.

Choosing a message-oriented architecture that fits the microservice approach or light messaging, as Fowler [20] calls it, in the shape of a modern message broker has resulted in many advantages. It has allowed us to work with an arbitrary amount of systems/services that process personal data without changing the code of GDPR Service. This eliminated the need to ship new releases of GDPR Service when a new service/system was added. Asynchronous communication enabled us to implement a solution where roughly hundred fifty systems/services process GDPR requests and the processing time among them can vary substantially. Although in this paper we are focusing on the "right to erasure" and "right of access" also other GDPR rights like the "right to restrict processing" are solved in similar way via message-oriented architecture.

Choosing Kafka as our message broker was not only necessary but also smart. It was necessary because with its replay functionality it solved the requirement to erase personal data that had already been erased in the productive system but was then being recovered via a system backup that had to be restored. It was smart to use Kafka because it is very performant and scalable. We had no problems connecting twice as many consumers to it as initially planned and transferring a substantially greater amount of data via it after implementing the personal data inquiry functionality.

We have also shown how such state-of-the-art architecture enabled us to quickly integrate our initial GDPR solution with a ticket system for GDPR that emerged from an independent GDPR taskforce. This integration enabled a high degree of automation and considerably reduced the labor cost of processing GDPR requests from users/customers for the German automotive OEM that hired us.

8. LITERATURE

[1] https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=uriserv:OJ.L_.2016.119.01.0001.01.ENG, General Data Protection Regulation (GDPR), visited 11. 5. 2018.

[2] GODDARD Michelle, "Viewpoint", International Journal of Market Research, Vol. 59, Issue 6, 2017.

[3] https://ico.org.uk/for-organisations/guide-to-the-general-data-protection-regulation-gdpr/individual-rights, Guide to the General Data Protection Regulation (GDPR), visited 11. 5. 2018.

[4] https://en.wikipedia.org/wiki/Create,_read,_update_and_delete, Create, read, update and delete (CRUD), visited 11. 5. 2018.

[5] http://microservices.io/patterns/data/database-per-service.html, Pattern “Database per service”, visited 11. 5. 2018.

[6] EUGSTER Patrick Th., et al., "The many faces of publish/subscribe", ACM computing surveys, Vol. 35, Issue 2, 2003, pages 114-131.

[7] https://en.wikipedia.org/wiki/Message-oriented_middleware, Message-oriented middleware, visited 11. 5. 2018.

[8] https://www.infoq.com/presentations/soa-without-esb, Presentation by Martin Fowler and Jim Webber: "Does My Bus Look Big in This?", visited 11. 5. 2018.

[9] THÖNES Johannes, "Microservices", IEEE Software, Vol. 32, Issue 1, 2015, pages 116-116.

[10] https://www.hivemq.com, HiveMQ, visited 11. 5. 2018.

[11] https://www.rabbitmq.com, RabbitMQ, visited 11. 5. 2018.

[12] https://kafka.apache.org, Apache Kafka, visited 11. 5. 2018.

[13] https://www.beyondthelines.net/computing/kafka-patterns, Kafka concepts and common patterns, visited 11. 5. 2018.

[14] https://projects.spring.io/spring-boot, Spring Boot, visited 11. 5. 2018.

[15] https://angular.io, Angular, visited 11. 5. 2018.

[16] https://swagger.io/docs, Swagger Documentation, visited 11. 5. 2018.

[17] http://projects.spring.io/spring-kafka, Spring Kafka adapter, visited 11. 5. 2018.

[18] http://java-design-patterns.com/patterns/tolerant-reader, Tolerant reader pattern, visited 11. 5. 2018.

[19] http://www.restapitutorial.com/lessons/idempotency.html, Idempotence, visited 11. 5. 2018.

[20] https://www.martinfowler.com/articles/microservices.html, Microservices – definition of this new architectural term, visited 11. 5. 2018.

Univerzitetna založba Univerze v Mariboru

VARNOSTNE RANLJIVOSTI PAMETNIH POGODB PLATFORME ETHEREUM

MARKO HÖLBL IN BLAŽ PODGORELEC

Povzetek: Tehnologija veriženja blokov se je uveljavila predvsem preko kriptovalut. Ena izmed najbolj uveljavljenih platform je Ethereum, ki ni samo elektronski plačilni sistem, ampak ponuja decentralizirano računalniško platformo za izvajanje pametnih pogodb, omejenih zgolj s Turingovo polnostjo. Vendar ravno samodejna izvršljivost in dejstvo, da pametne pogodbe pišejo ljudje, povzroča številne varnostne pomanjkljivosti. V prispevku bomo predstavili varnostne vidike in pomanjkljivosti Ethereum pametnih pogodb, tudi s pomočjo konkretnih primerov in orodij, ki omogočajo, da vsaj deloma, zmanjšamo tveganja, povezana z ranljivostmi v pametnih pogodbah. Predstavili bomo tudi priporočila osredotočene na varnostne vidike pametnih pogodb, ki so lahko vodilo za dobre prakse razvoja pametnih pogodb na verigah blokov.

Ključne besede: • tehnologija veriženja blokov • Ethereum • varnost • pametne pogodbe • dobre prakse

NASLOV AVTORJEV: dr. Marko Hölbl, docent, Univerza v Mariboru, Fakulteta za elektrotehniko, računalništvo in informatiko, Koroška cesta 46, 2000 Maribor, Slovenija, e-pošta: marko.holbl@um.si. Blaž Podgorelec, Univerza v Mariboru, Fakulteta za elektrotehniko, računalništvo in informatiko, Koroška cesta 46, 2000 Maribor, Slovenija, e-pošta: blaz.podgorelec@um.si.

DOI https://doi.org/10.18690/978-961-286-162-9.10 ISBN 978-961-286-163-6

Dostopno na: http://press.um.si.

1. UVOD

Implementacija pametnih pogodb (v nadaljevanju PP) na verigah blokov je v omejeni obliki možna že znotraj platforme Bitcoin [2]. Leta 2013 je Vitalik Buterin, predlagal novo platformo verig blokov, ki podpira razvoj PP s Turingovo polnostjo imenovano Ethereum. Za razliko od platforme Bitcoin, Ethereum ni samo elektronski plačilni sistem, ampak ponuja decentralizirano ter hkrati porazdeljeno platformo oz. operacijski sistem za izvajanje PP, omejenih zgolj s Turingovo polnostjo [1]. Platforma Ethereum je odprtokodna in temelji na javno dostopnem omrežju verig blokov [2]. PP nameščene znotraj verige blokov Ethereum prenašajo kovance v skupni vrednosti več milijard dolarjev [3], zato so tovrstne PP na platformi Ethereum nenehna tarča napadalcev z namenom odkritja pomanjkljivosti v implementacijah programske logike PP, ki bi omenjenim akterjem omogočale morebitno finančno okoriščanje s sredstvi, ki so predmet PP [4].

V prispevku bomo predstavili varnost PP. Najprej bomo na kratko predstavili koncept pametnih pogodb na platformi Ethereum, čemur bo sledila obravnava in razvrstitev varnostnih tveganj PP ter predstavitev znanih primerov napadov na PP. S pomočjo primerov bomo prikazali varnostne ranljivosti PP implementiranih v visokonivojskem programskem jeziku Solidity, predstavili orodja, ki lahko pripomorejo k njihovemu odkrivanju in podali smernice dobrih praks razvoja PP na platformi Ethereum.

2. PAMETNE POGODBE PLATFORME ETHEREUM

Platforma Ethereum temelji na t.i. virtualnem stroju (ang. Ethereum Virtual Machine – EVM), ki na zahtevo poganja programe imenovane PP, definirane z nizkonivojskim zlogovnim jezikom. EVM se poganja na vseh vozliščih, ki so del Ethereum omrežja. Pametne pogodbe so zbirka funkcij, pri čemer je vsaka funkcija definirana z določenim zaporedjem osnovnih ukazov predstavljenih z zlogovno kodo [7]. Z uspešno namestitvijo v omrežje verige blokov, se vsaka PP identificira z unikatnim naslovom. Pametna pogodba lahko ima definirano tudi stanje virtualnih kovancev Ether, lastno zasebno shrambo in povezavo z vnaprej definirano določeno izvedljivo kodo [8]. Klici funkcij PP nameščenih v omrežju Ethereum se izvedejo s pomočjo transakcij. V primeru, da klic funkcije PP spreminja stanje v omrežju verige blokov, se ta izvede po vseh vozliščih omrežja. Za izvajanje funkcij, ki spreminjajo stanje, se glede na zahtevnost operacij definiranih v omenjeni funkciji porablja določeno število goriva v domorodni valuti Ether. Gorivo v tem primeru izgoreva in njegova vrednost se nakazuje vozliščem omrežja, ki so zlogovno kodo PP izvedli. Za poizvedovanje stanja omrežja se tovrstno gorivo ne porablja, odgovor na klic funkcije se zagotovi iz najbližjega vozlišča v omrežju [2].

Po zasnovi je PP mehanizem za distribuirano izvajanje programske kode. Za zagotavljanje poštenega nadomestila vozliščem, ki izvajajo PP, je potrebno plačilo pristojbin. Njihova vrednost je enaka sorazmernemu deležu zahtevnosti izvedenih operacij. Vsaka operacija ima tako s protokolom Ethereum vnaprej določeno ceno izvajanja. Med pretvorbo visokonivojskega oblike PP v zlogovno kodo, se hkrati izvede tudi pretvorba v t.i. seznam osnovnih operacij (ang. OPCODE), ki imajo vsak svojo vnaprej določeno ceno. Pri pošiljanju transakcije, ki spreminja stanje verige blokov, mora uporabnik določiti zgornji znesek (ang. gas limit) v valuti Ether, ki ga je pripravljen zagotoviti za izvedbo transakcije. Potrjevalec, ki transakcijo vključuje v blok, naknadno prejeme transakcijsko provizijo. Ta ustreza količini porabljenega goriva pri izvedbi transakcije. V primeru porabe večjega števila goriva pri izvedbi transakcije, kot ga je predhodno definiral uporabnik, se izvršitev transakcije prekine z izjemo, porabljeno gorivo uporabniku ni vrnjeno, stanje transakcije pa se povrne v začetno stanje brez sprememb [8].

V nadaljevanju bomo predstavili določene ranljivosti PP na platformi Ethereum, ki izhajajo iz varnostnih pomanjkljivosti visokonivojskega programskega jezika Solidity in EVM.

3. VARNOST PAMETNIH POGODB

Varnost Ethereum PP je pomembna, saj platforma vključuje tudi kriptovaluto Ether, kar omogoča, da se napadalec materialno okoristi. V primeru, da bi PP uporabljali produkcijskem okolju za izvrševanje

določenih samodejnih aktivnosti, bi lahko prišlo tudi do drugih zlorab. Varnost platforme opredeljuje tudi število varnostnih pomanjkljivosti oz. ranljivosti. Te lahko uvrstimo v tri kategorije [7]:

- ranljivosti visokonivojskega programskega jezika Solidity;
- ranljivost Ethereum virtualnega stroja (EVM) in
- ranljivosti tehnologije veriženja blokov.

Ranljivosti programskega jezika Solidity se nanašajo na nekatere programerske nedoslednosti in pomanjkljivosti, ki ne izvirajo zgolj iz strukture programskega jezika Solidity, kot so težave, povezane s slabim upravljanjem z izjemami ali pretvorba podatkovnih tipov. V primeru klicanja neznanih funkcij (ang. call to the unknown), lahko problem predstavljajo nadomestne funkcije (ang. fallback function), ki se ob tem prožijo. Nadomestne funkcije in težave le teh smo opisali v Poglavju 3.2.1. Primer je funkcija znotraj PP, ki se zaradi določenega razloga ne izvede in se zaradi tega izvede druga nepričakovana funkcija. Ob uporabi funkcije `send` za prenos Etherjev, je mogoč pojav izjeme »out-of-gas«. V programskem jeziku Solidity obstaja več scenarijev, kjer lahko pride do izjem: pri izvajanju zmanjka pristojbin (ang. gas); sklad za shranjevanje klicev funkcij se napolni; uporabimo ukaz `throw`. Tudi pri klicu funkcij, kjer pride do izjeme, je potrebno plačati pristojbine, kar lahko privedete do celotne porabe le-teh. Kljub temu, da prevajalnik programskega jezika Solidity zaznava določene nedoslednosti pri deklariranju spremenljivk in njihovih tipov, ta zaznava ni popolna. V določenih primerih, PP ne javlja napak, tudi če so te prisotne. Tudi atomarnost in sekvenčnost transakcij daje marsikateremu programerju lažno upanje, da ni mogoče klicati ne-rekurzivnih funkcij večkrat (torej preden se en klic zaključi). Vendar to ne drži. Zaradi nadomestne funkcije, lahko pride do večkratnega klicanja le te in na tak način do porabe vseh pristojbin. Težavo povzroča tudi dejstvo, da polja, ki jih deklariramo kot zasebna, niso nujno tudi skrita. Za namen skrivanja podatkov je bolj primerno uporabiti kriptografske mehanizme.

Omenjene ranljivosti programskega jezika Solidity pa niso edine, ki se jih moramo zavedati pri razvoju PP. Ranljivost in napake se lahko pojavijo tudi znotraj EVM. Snovalci PP morajo biti še posebej pazljivi pri pisanju PP, saj je morebitnih napak v implementaciji nameščenih PP ni mogoče naknadno popravljati (ang. patching). Napake v PP so bile že velikokrat izkoriščene, ali za pridobivanje Etherjev ali za preprečitev unovčitve Etherjev [13], [14]. Pri prenosu Etherjev moramo biti pazljivi na naslov prejemnika, saj je veliko naslovov v omrežju osirotelih (ang. orhpan). V primeru, da pošljemo Etherje na osiroteli naslov, so ti za vedno izgubljeni. Pri izvršitvi PP lahko vključujemo tudi funkcije drugih PP, kar lahko privede do dosega omejitve glede števila klicanih funkcij, saj se zapolni sklad klicanja funkcij. Vsak nadaljnji klic funkcije nato proži izjemo, kar je bilo že izkoriščeno tudi za napade. Omejena ranljivost je popravljena od konca leta 2016 [12].

Varnostne pomanjkljivosti, ki smo jih predstavili, lahko za organizacije, ki svoje poslovne procese implementirajo s pomočjo PP na verigah blokov, predstavljajo vzrok za neuspeh izvajanja PP, kar posledično lahko rezultira v poslovni škodi. V nadaljevanju se bomo osredotočili predvsem na kategorijo ranljivosti visokonivojskega programskega jezika Solidity. S pomočjo izsekov kode bomo prikazali varnostno ranljive implementacije PP.

3.1. Znani primeri napadov na pametne pogodbe

O realnosti tveganja priča tudi zgodovina tovrstnih napadov na implementacije PP v omrežju verig blokov Ethereum. Največja in najbolj znana zgodovinska napada na platformi Ethereum sta t.i. Parity in TheDao napad [5] [6]. Napad imenovan Parity je izkoristil ranljivost v knjižnici PP istoimenske večuporabniške denarnice (ang. multisignature wallet). Ranljivost v implementaciji knjižnice, ki je služila kot skupna komponenta vsem ustvarjenim PP, ki so implementirale večuporabniške denarnice, je anonimnemu napadalcu omogočala prevzem lastništva PP implementirane knjižnice in s tem posledično nadzor nad 587 večuporabniškimi denarnicami. Skupna vrednost digitalnih sredstev je bila v času napada ocenjena na okrog 160 milijonov dolarjev. Prevzem lastništva nad knjižnico, ki je bila del implementacije omenjenih PP večuporabniških denarnic, je omogočal anonimnemu uporabniku izvršitev samouničevalne funkcije definirane v PP. Napadalec je izvršil funkcijo, ki je bila del knjižnice

in tako sprožil uničenje PP, ki je bila ključni gradnik večuporabniške denarnice in s tem povzročil neuporabnost in zamrznitev vseh sredstev na 587 večuporabniških denarnicah [5].

TheDao napad se je zgodil na PP, ki je implementirala delovanje decentraliziranega avtonomnega sklada tveganega kapitala [6]. Napadalec je izkoristil pomanjkljivost v implementaciji funkcije PP za opravljanje menjave med žetoni decentralizirane organizacije in kriptovaluto Ether. Z rekurzivnim klicem omenjene funkcije je napadalec lahko en žeton decentralizirane organizacije večkrat zamenjal za njegovo protivrednost v kriptovaluti Ether. Funkcija definirana v pametni pogodbi je namreč, preden je posodobila stanje omenjenih žetonov na računu klicatelja, temu najprej nakazala znesek v Etherjih in šele kasneje posodobila bilanco stanja žetonov. Z rekurzivnim klicem funkcije je tako napadalec zaobšel posodobitev stanja žetonov in na ta način izčrpal PP decentralizirane avtonomne organizacije v skupno ocenjenem znesku več kot 50 milijonov dolarjev [6].

3.2. Primeri varnostnih ranljivosti pametnih pogodb

Razvijalci PP običajno ne zapisujejo z zlogovno kodo, ampak za razvoj uporabljajo visokonivojski programski jezik Solidity. Ta se pred namestitvijo PP v omrežje prevede v zlogovno kodo. Prevajalnik vsako funkcijo označi s podpisom, ki temelji na imenu in tipskih parametrih funkcije.

3.2.1. Nadomestna funkcija

Po klicu funkcije se podpis prenese kot vhod v klicano PP. Če se funkcija ujema s katero funkcijo definirano v klicani PP, se ta tudi izvede. V primeru neujemanja podpisa s katerokoli funkcijo definirano znotraj klicane PP, se proži posebna nadomestna funkcija (ang. fallback function). Nadomestna funkcija ne vsebuje imena in argumentov ter se proži ob neujemanju podpisa klicane funkcije v klicani PP. Izvede pa se tudi v primeru pošiljanja žetona Ether v samo pametno pogodbo [7]. Klici funkcij znotraj programskega jezika Solidity se lahko zgodijo preko treh različno definiranih konstruktov, ki se iz programskega jezika Solidity prevedejo v zlogovno kodo in se sklicujejo na klice funkcij PP znotraj PP ter istočasno dovoljujejo pošiljanje žetonov Ether. Določeni opisani konstrukti ob vključitvi prenosa žetona Ether, istočasno prožijo tudi nadomestno funkcijo njegovega prejemnika [7].

- *CALL* prikliče funkcijo (druge PP ali svojo) in prenese poslan Ether na naslovnika. V primeru neobstoja podpisa klicane funkcije na naslovu klicane PP se proži nadomestna funkcija [9].
- *SEND* se uporablja za prenos žetona Ether. Po prenosu žetona omenjen konstrukt proži nadomestno funkcijo prejemnika [9].
- *DELEGATECALL* je podoben konstrukt kot *CALL*. Razlikuje se v tem, da se prožena funkcija izvede v okolju klicatelja in ne prejemnika [9].

Primer nevarne uporabe nadomestne funkcije je prikazan v Izvorni kodi 1.

```
import './lastnistvo/LastnikPogodba.sol';

contract PrispevkiPogodba is LastnikPogodba {
  mapping(address => uint) private prispevki;

  function prispevaj() public payable {
    require(msg.value < 1 wei);
    prispevki[msg.sender] += msg.value;
    if(prispevki[msg.sender] > prispevki[lastnik]) {
      lastnik = msg.sender;
    }
  }

  function getPrispevki() public view returns (uint) {
```

```
    return prispevki[msg.sender];
  }

  function prevzamiPrispevke() public samoLastnik {
    lastnik.transfer(address(this).balance);
  }

  function() payable public {
    require(msg.value > 0 && prispevki[msg.sender] > 0);
    lastnik = msg.sender;
  }
}
```

Izvorna koda 1. Primer nevarne nadomestne funkcije.

Pametna pogodba *PrispevkiPogodba* iz primera Izvorne kode 1 je sestavljena iz treh funkcij:

- *prispevaj();* s to funkcijo lahko v PP prispevamo določen znesek žetonov Ether. Če prispevamo večji znesek, kot ga je predhodno prispeval lastnik PP, postanemo novi lastnik.
- *getPrispevki();* funkcija, s katero klicatelj preveri stanje njegovih prispevkov v PP.
- *prevzamiPrispevke();* funkcija, s katero lastnik PP prevzame skupno vrednost prispevkov PP.

PP ima prav tako definirano nadomestno funkcijo, ki se izvrši ob prenosu vrednosti v PP. Iz izvorne kode funkcije lahko razberemo, da vsebuje varnostno pomanjkljivo programsko kodo. Napadalec namreč lahko kliče funkcijo *prispevaj()* z simboličnim zneskom 10 wei (0.00000000000000001 Ether) in se na ta način uvrsti na seznam donatorjev. V naslednjem koraku pošlje žeton Ether s simboličnim zneskom 1 wei v PP, pri čemer se proži nadomestna funkcija, ki preverja ali je poslani znesek večji od 1 in ali je pošiljatelj na seznamu donatorjev. Če je pogojem zadoščeno, implementacija nadomestne funkcije klicatelju preda lastništvo PP. Na ta način dobi napadalec možnost klica *prevzamiPrispevke()* in, brez da je prispeval večji znesek od morebitnega lastnika PP, prevzame nadzor in si tako prilasti vse prispevke zbrane znotraj PP.

3.2.2. Pomanjkljivosti računskih operacij

Naslednji opisan primer, prikazan v Izvorni kodi 2 prikazuje nevarnost pomanjkljivosti računskih operacij in posledično prekoračitev. Ob namestitvi PP *Zeton* se s konstruktorjem določi skupno število kovancev v obtoku. Ustvarjeni žetoni se prerazporedijo na račun lastnika. Pri klicu funkcije *posljiZetone*() z ukazom *require* preverjamo, da pošiljatelj ne more poslati večjega števila žetonov kot jih poseduje. Vendar z omenjenim ukazom vedno zadostimo pogojem, saj je za hrambo stanja žetonov definiran podatkovni tip *uint*, pri čemer z odštevanjem ne dosežemo negativnih števil temveč povzročimo prekoračitev (ang. integer overflow) saj omenjeni podatkovni tip ni namenjen hrambi negativnih števil. Tako pošiljatelj žetonov postane lastnik zelo velikega števila žetonov. Zato je pri tovrstnih matematičnih funkcijah zmeraj potrebno preverjati, da ne presežemo maksimalnega števila definiranega podatkovnega tipa.

```
contract Zeton {
  mapping(address => uint) stanjeZetonov;
  uint public skupnaZaloga;

  constructor (uint _zaloga) public {
    stanjeZetonov[msg.sender] = skupnaZaloga = _zaloga;
```

```
  }

  function posljiZetone(address _naslov, uint _vrednost) public returns (bool)
{
    require(stanjeZetonov[msg.sender] - _vrednost >= 0);
    stanjeZetonov[msg.sender] -= _ vrednost;
    stanjeZetonov[_naslov] += _vrednost;
    return true;
  }

  function preveriStanjeRacuna(address _naslov) public view returns (uint) {
    return stanjeZetonov[_naslov];
  }
}
```

Izvorna koda 2. Primer prekoračitve.

3.2.3. Možnost ponovnih vključitev

Izvorna koda 3, prikazuje nevarnost možnosti ponovnih vključitev (ang. reentrency). Takšno opisano pomanjkljivost v implementaciji je vsebovala TheDao pametna pogodba opisana v Poglavju 1.

```
contract PonovnaVkljucitev {
  mapping(address => uint) public stanjeRacuna;

  function doniraj(address _naslov) public payable {
    stanjeRacuna[_naslov] += msg.value;
  }

  function preveriStanje(address _naslov) public view returns (uint) {
    return stanjeRacuna[_naslov];
  }

  function dvigSredstev(uint _znesek) public {
    if(stanjeRacuna[msg.sender] >= _znesek) {
      if(msg.sender.call.value(_znesek)()) {
        _znesek;
      }
      stanjeRacuna[msg.sender] -= _znesek;
    }
  }
  function() public payable {}
}
```

Izvorna koda 3. Primer možnosti ponovnih vključitev.

Pametna pogodba *PonovnaVkljucitev* omogoča uporabnikom donacijo sredstev določenemu računu, preverjanje stanja računa in dvig doniranih sredstev. Napadi na PP se pogosto ne dodajajo s klici funkcij iz uporabniškimi računov ampak s pomočjo drugih PP, katere implementirajo logiko tovrstnih napadov. Izvorna koda 4 prikazuje implementacijo PP, katere namen je izpraznitev vseh doniranih sredstev PP *PonovnaVkljucitev*.

```
import "./PonovnaVkljucitev.sol";

contract PonovnaVkljucitevNapad {
  PonovnaVkljucitev pv;

  function napad(address _naslov) payable {
     pv = PonovnaVkljucitev(_naslov);
     pv.doniraj.value(0.1 ether)(address(this));
     pv.dvigSredstev(0.1 ether);
  }

  function() public payable {
     pv.dvigSredstev(0.1 ether);
  }
}
```

Izvorna koda 4. Primer napada na PP *PonovnaVkljucitev*.

Implementacija funkcije *dvigSredstev*, odbitek dvignjenih sredstev izvede šele po prenosu žetona Ether na naslov računa prožitelja funkcije. Kot smo opisali v Poglavju 3.1.1 prenos žetona Ether na ciljni naslov, ki je lahko tudi PP, kot je vidno našem primeru *PonovnaVkljucitevNapad* v le tej proži nadomestno funkcijo. V tej funkciji prikazani s primerom Izvorne kode 4 je tako v primeru zadostnih sredstev za gorivo, mogoč rekurziven klic funkcije *dvigSredstev*, kar rezultira v izpraznitev celotnega stanja računa *PonovnaVkljucitev*. Funkcija *napad* v *PonovnaVkljucitevNapad* tako na naslov svoje tarče najprej nakaže simbolično število žetona Ether, katerega takoj za tem s funkcijo *dvigSredstev* zahteva nazaj. Kot že opisano funkcija *dvigSredstev* prenese žeton Ether v račun *PonovnaVkljucitevNapad*, pri čemer se proži nadomestna funkcija, ki ponovno zahteva dvig sredstev iz *PonovnaVkljucitev*. Le ta ji to omogoča, saj implementacija funkcije *dvigSredstev* odbitek dvignjenih sredstev izvede šele na koncu prenosa žetona Ether.

3.2.4. Zasebne spremenljivke PP

Deklariranje spremenljivk v PP, kot zasebne (ang. private) preprečuje dostop do njihovih vsebin drugim PP. Spremenljivke stanja (ang. state variables) in lokalne spremenljivke (ang. local variables), deklarirane kot zasebne, so kljub tovrstni deklaraciji še vedno javno dostopne preko odjemalcev protokola Ethereum [17]. PP prikazana v Izvorna koda 5, prikazuje implementacijo PP *Trezor*, ki ob namestitvi v omrežje s konstruktorjem inicializira spremenljivko *zaklep* na vrednost *true* in v spremenljivko *geslo* shrani vrednost podano s parametrom konstruktorja, ki ponazarja geslo. Funkcija *odklepanje* ob pravilno podanem geslu, kot parametru funkcije, omogoča klicatelju spremembo vrednosti spremenljivke *zaklep*, kar prikazuje simbolično operacijo, ki bi lahko ponazarjala kompleksnejše in varnostno občutljivejše operacije ob pravilnem vnosu gesla s strani prožitelja funkcije.

```
contract Trezor {
  bool public zaklep;
  bytes32 public geslo;

  constructor(bytes32 _geslo) public {
    zaklep = true;
    geslo = _geslo;
  }

  function odklepanje(bytes32 _geslo) public {
    if (geslo == _geslo) {
      zaklep = false;
    }
  }
}
```

Izvorna koda 5. Primer PP, ki vsebuje shranjeno geslo.

Do zasebno deklarirane spremenljivke *geslo* znotraj PP *Trezor* lahko dostopamo s pomočjo odjemalca *Geth* [18] in knjižnice *Web3* [19]. Z izvedbo ukaza *web3.toAscii(web3.eth.getStorageAt(»NaslovTrezor«,1))* prožimo funkcijo odjemalca *getStorageAt*, ki kot parametra funkcije sprejeme naslov shrambe PP in indeks elementa v omenjeni shrambi PP. S pomočjo funkcije knjižnice Web3 *toAscii* nato pridobljeno šestnajstiško število pretvorimo v niz znakov. Rezultat izvedbe ukaza je tako vrednost zasebno deklarirane spremenljivke *geslo* iz PP *Trezor*, ki nam tako omogoča podajanje pravilnega parametra v funkcijo *odklepanje* in s tem možnost spremembe vrednosti spremenljivke *zaklep*. V primeru shranjevanja občutljivih vrednosti v spremenljivke znotraj PP je potrebno le te šifrirati, pri čemer ključa za dešifriranje podatkov posledično ne smemo pošiljati znotraj verige blokov [17].

4. ORODJA PREPOZNAVANJA VARNOSTNIH RANLJIVOSTI

Za najučinkovitejši način prepoznavanja morebitnih varnostnih ranljivosti pred nameščanjem PP v omrežje veljajo varnostne revizije izvorne kode PP. V tem primeru izkušeni razvijalci in specializirane ekipe skrbno ročno ter s pomočjo avtomatiziranih orodij preučijo PP z namenom odkrivanja morebitnih ranljivosti v implementaciji in tudi zagotovijo, da implementacija sledi dobrim praksam razvoja PP. Kljub temu lahko razvijalec zmanjša ranljivosti v PP s pomočjo samostojnih varnostnih orodij, ki jih bomo predstavili v tem poglavju. Ti se lahko uporabijo za preventivne preglede implementacije PP z namenom odkrivanja predhodno opisanih varnostnih ranljivosti, tako pred, kot tudi po namestitvi PP v omrežje [10]. Med omenjena orodja sodijo:

- MANTICORE: Prototipno orodje za dinamično binarno analizo PP [10].
- MYTHRILL: Eksperimentalno orodje, ki s pomočjo vmesnika ukazne vrstice ponuja analizo PP. S pomočjo simuliranja izvedbe PP ponuja možnost prepoznavanja različnih varnostnih ranljivosti, kot so nezaščitene funkcije (ang. unprotected functions), ponovne vključitve (ang. reentrency), prekoračitev/podkoračitev pomnilnika celih števil (ang. integer overflow/underflow), časovne odvisnosti (ang. timestamp dependence), odvisnost zaporedja transakcij (ang. transaction-ordering dependence) in izpostavljenost informacij (ang. information exposure) [10], [4].
- SMARTCHECK: Spletne aplikacije, ki je trenutno v Beta različici in omogoča samodejno prepoznavanje varnostnih ranljivostih, podobno kot orodje Mythrill. Odkrite pomanjkljivosti so prikazane po stopnji resnosti. Razpoznava tudi manj resne pomanjkljivosti, kot so različne verzije prevajalnika, redundantne funkcije in omogoča opozarjanje na dobre prakse slogovnega oblikovanja PP [10].

- OYENTE: Prvo in najbolj priljubljeno orodje za varnostno analizo PP, ki temelji na raziskavah Ethereum skupnosti. S pomočjo simuliranja izvedbe PP omogoča odkrivanje podobnih varnostnih ranljivosti, kot orodji Mythrill in SmartCheck. Orodje omogoča analizo zlogovne kode PP, kot tudi analizo PP implementiranih z visokonivojskim programskim jezikom Solidity [8].
- GASPER: Orodje v razvoju, ki še ni na voljo za uporabo, katero se bo s pomočjo analize zlogovne kode PP in simulacijo izvedbe PP fokusiralo na prepoznavanje vzorcev razvoja PP, ki povzročajo prekomerno porabo goriva za izvajanje transakcij znotraj omrežja [11].

Opisana orodja lahko pripomorejo k zmanjšanju tveganja za varnostne ranljivosti, a je zagotovo v primeru, ko bodo PP upravljanje z večjim zneskom žetonov Ether, smiseln tudi najem ustreznih svetovalcev, preverjalcev kode in uporabo že preverjenih odprtokodnih ogrodij za gradnjo varnih PP, ki zmanjšujejo tveganje za ranljivosti v PP z uporabo standardne, preizkušane in revidirane kode, kot je OpenZeppelin [15].

5. DOBRE PRAKSE RAZVOJA PAMETNIH POGODB

Platforma Ethereum in kompleksne PP na verigah blokov predstavljata nove in pogosto eksperimentalne koncepte [16]. Zato je mogoče pričakovati spremembe v smislu odkrivanja novih hroščev in varnostnih tveganj, kar posledični pomeni tudi oblikovanje novih dobrih praks pri razvoju PP. Obramba pred znanimi ranljivostmi ni dovolj, zato je potrebno sprejeti nove konceptualne pristope razvoja tovrstne programske opreme. Pričakujemo lahko, da vsaka ne-trivialna pogodba vsebuje napake, kar lahko privede do neuspešnega izvajanja PP. V ta namen je smiselno, celo nujno imeti razdelane strategije za [16]:

- prekinitev pogodbe v primeru napačnega izvajanja PP (ang. circuit breaker),
- učinkovito upravljanje ogroženega zneska denarja vsebovanega znotraj PP in
- učinkovite poti za nadgradnjo in izboljšavo PP.

Pred produkcijsko uporabo PP je priporočeno za vse funkcionalnosti zasnovati ustrezne teste. Kompleksnost PP povečuje verjetnost napak, zato je potrebno ohranjati enostavnost PP, kar dosežemo z:

- preprosto pogodbeno logiko PP,
- modularizacijo programske kode,
- uporabo že preizkušenih ogrodij in napisanih delov kode ter
- uporabo PP na verigah blokov le za dele poslovnih procesov, ki zahtevajo decentraliziranost.

Za učinkovito in varno izvajanje PP na verigah blokov se je potrebno zavedati lastnosti tehnologije veriženja blokov in specifik, ki jih ta vsebuje. Zato je potrebno pazljivo presoditi, katere poslovne procese je smiselno izvajati s pomočjo PP na verigah blokov in katere poslovne procese je bolje ohraniti na obstoječih sistemih.

6. ZAKLJUČEK

V prispevku smo naslavljali varnost pametnih pogodb na platformi Ethereum. Ker so pametne pogodbe programska koda, ki se izvaja v Ethereum virtualnemu stroju (EVM), je seveda možnosti za izkoriščanje pomanjkljivosti veliko. Določene pomanjkljivosti se nanašajo na programski jezik, v katerem so običajno implementirane PP in na izvajalno okolje EVM. Razvijalci zato morajo dobro poznati ranljivosti in biti zelo dosledni pri razvoju, saj je popravljanje PP, za razliko od klasične programske opreme, precej bolj težavno. V dodatno pomoč so lahko orodja za preverjanje programske kode PP, v primeru večjih projektov pa je priporočljiv najem ustreznih strokovnjakov za pregled.

7. LITERATURA

[1] V. Buterin, "A next-generation smart contract and decentralized application platform," *Etherum*, str. 1–36, 2014.

[2] M. Turkanović, A. Kamišalić, and B. Podgorelec, "DSI 2018 - Pametne pogodbe na verigah blokov," *Dnevi Slovenske Informatike*, april 2018.

[3] www.coinmarketcap.com, Cryptocurrency Market Capitalizations, obiskano 16.5.2018.

[4] I. Nikolic, A. Kolluri, I. Sergey, P. Saxena, and A. Hobor, "Finding The Greedy, Prodigal, and Suicidal Contracts at Scale," 2018.

[5] www.paritytech.io/a-postmortem-on-the-parity-multi-sig-library-self-destruct, "A Postmortem on the Parity Multi-Sig Library Self-Destruct," obiskano 15.5.2018.

[6] www.cryptocompare.com/coins/guides/the-dao-the-hack-the-soft-fork-and-the-hard-fork/, The DAO, "The Hack, The Soft Fork and The Hard Fork," obiskano 16.05.2018.

[7] N. Atzei, M. Bartoletti, and T. Cimoli, "A Survey of Attacks on Ethereum Smart Contracts (SoK)," vol. 10204, Marec 2017, str. 164–186.

[8] L. Luu, D.-H. Chu, H. Olickel, P. Saxena, and A. Hobor, "Making Smart Contracts Smarter," *Proc. 2016 ACM SIGSAC Conf. Comput. Commun. Secur. - CCS'16*, str. 254–269, 2016.

[9] www.solidity.readthedocs.io/en/latest/index.html, Solidity — Solidity 0.4.21 documentation, obiskano 16.5.2018.

[10] A. Dika, "Ethereum Smart Contracts : Security Vulnerabilities and Security Tools", December, 2017.

[11] T. Chen, X. Li, X. Luo, and X. Zhang, "Under-optimized smart contracts devour your money," SANER 2017 - 24th IEEE Int. Conf. Softw. Anal. Evol. Reengineering, str. 442–446, 2017.

[12] www.blog.ethereum.org/2016/10/13/announcement-imminent-hardfork-eip150-gas-cost-changes/, " Announcement of imminent hard fork for EIP150 gas cost changes," obiskano 16.5.2018.

[13] www.governmental.github.io/GovernMental/, GovernMental, obiskano 16.5.2018.

[14] www.bitcointalk.org/index.php?topic=1400536.60, "Bitcointalk: Hi!My name is Rubixi," obiskano 16.5.2018.

[15] www.openzeppelin.org, OpenZeppelin, obiskano 16.5.2018.

[16] www.consensys.github.io, "Consensys, Ethereum Smart Contracts Best Practices," obiskano 16.05.2018

[17] https://ethernaut.zeppelin.solutions/, The Ethernaut, obiskano 16.05.2018

[18] https://github.com/ethereum/go-ethereum/, Official Go implementation of the Ethereum protocol, obiskano 16.05.2018

[19] https://github.com/ethereum/wiki/wiki/JavaScript-API, Web3 JavaScript app API, obiskano 16.05.2018

OTS 2018 SODOBNE INFORMACIJSKE TEHNOLOGIJE IN STORITVE
ZBORNIK TRIINDVAJSETE KONFERENCE, MARIBOR, 19. IN 20. JUNIJ 2018
M. HERIČKO IN K. KOUS

Univerzitetna založba
Univerze v Mariboru

DATA PROTECTION IN A HYPER-CONVERGED WORLD: OUR STORY

DAMIJAN BAČANI

Abstract: In today's world, businesses can't survive and grow without ability to effectively manage their digital information. Hyper-converged infrastructure (HCI) is changing the model of consuming data center resources by enabling end users to provision resources almost instantly and then allow data and applications to distribute over infrastructure. Data protection is a key requirement for every organization and hyper-converged infrastructure requires different backup solution – a backup solution integrated into the HCI environment to eliminate the need for a dedicated backup server and simplify IT operations. HYCU Data Protection is purpose-built backup and recovery for Nutanix hyper-converged infrastructure.

Key words: • Hyper-converged infrastructure • Data protection • HYCU • Snapshot • VM stun

AUTHORS: Damijan Bačani, HYCU Inc., Letališka cesta 29b, 1000 Ljubljana, Slovenija, e-pošta: damijan.bacani@hycu.com.

DOI https://doi.org/10.18690/978-961-286-162-9.11 ISBN 978-961-286-163-6

Dostopno na: http://press.um.si.

1. INTRODUCTION

By 2020, 20% of business-critical applications currently deployed on three-tier IT infrastructure will transition to hyperconverged infrastructure. [1]

There's no doubt cloud architectures and scale-out computing are replacing legacy IT and running more and more applications. This also means that existing data protection solutions will need to be re-designed to fit into data centers.

When we began designing HYCU we told ourselves that we have the opportunity to rethink how to do data protection. We wanted the policy, i.e. the contract between the business and IT administrative staff, to be in a simple language that will be easy to understand and never in doubt as to what the outcome will be.

2. HYPER-CONVERGED INFRASTRUCTURE

2.1. What is it?

The term was first coined in 2012 by Steve Chambers and Forrester Research and there is still a lot of disagreement among experts as to what exactly defines HCI. According to Gartner, it's a category of scale-out software-integrated infrastructure that applies a modular approach to compute, network and storage on standard hardware, leveraging distributed, horizontal building blocks under unified management. [1]

It brings flexibility and simplicity to datacenters and should be viewed as a base block for building an enterprise cloud. This fully software-defined IT infrastructure assembles storage, computing and networking into a single-system solution. This significantly reduces the engineering effort, data center complexity and increases scalability.

2.2. How it differs from traditional IT infrastructure

Traditional IT infrastructure is commonly silos of systems and operations with separate groups and systems for storage, servers and network. Typically, each group handles purchasing, provisioning and support of the infrastructure the group is responsible for. More or less every enterprise looks at software solutions in a way to grow the business. The systematic solution which supports this business process certainly does not lie in the legacy infrastructure model, where computing power, networking, software and storage are standalone blocks which needs an entire army of IT engineers which ultimately do the hard work of integrating and maintaining them. Nor in the more sophisticated legacy model called Converged where the idea is the system which integrates the previously mentioned physical blocks into the physical solution bundle which is pre-configured and ready to connect to electricity. The right answer is hype-converged infrastructure.

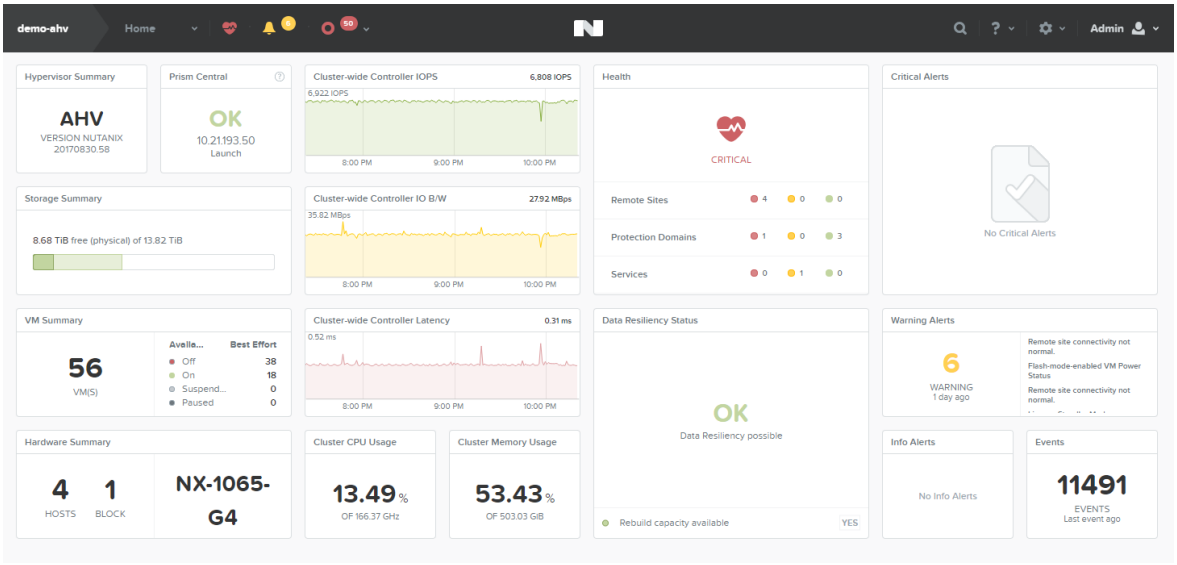
2.3. Components of HCI

Most HCI solutions consist of two base components:

- Distributed data plane

  runs across a cluster of nodes delivering storage, virtualization and networking services for guest applications (VMs or container-based applications).
- Management plane

  allows for the easy administration of all HCI resources from a single view and eliminates the need for separate management solutions for servers, storage networks, storage and virtualization.



Picture 1: Nutanix Prism Management Console

2.4. Hypervisors

As the hypervisor space becomes more competitive and customers are implementing non-VMware products to virtualize workloads, the flexibility to support more than VMware is quickly becoming important. While VMware does still rule the world, there are certain verticals and customers that are adopting KVM or Hyper-V over VMware. Today this likely already plays into the buying decision for some customers when looking at hyper-converged products. Nutanix is already supporting multiple hypervisors, SimpliVity has discussed their desire to in the future, while VMware will always be a 100% vSphere-based solution. Both of these stances can provide benefits for customers as they provide more flexibility or a tightly controlled integration.

2.5. Suppliers

The hyper-converged market has matured over the last few years, with a range of hardware and software offerings. Simplivity, VMware, Dell EMC, Stratoscale, Cisco and Nutanix are some of HCI players with Nutanix being a leader in hype-converged infrastructure segment.

Our company is working with Nutanix to improve data protection for applications running on Nutanix hype-converged infrastructure and has created a product that perfectly fits into this strategy - HYCU Data Protection for Nutanix.

Nutanix solution allows customers to seamlessly select, provision, deploy & manage their Business Apps across all their infrastructure, both private and public cloud. It will eventually support all the components required to manage a complete Software Defined Data Center (SDDC).

2.6. Benefits for Dev/Test and DevOps teams

Whether your development team is developing new IT applications or producing new in-house or commercial software, an efficient, high-performance development and test environment can help drive productivity, improve time to market, and have a direct impact on revenue.

What does that mean? “MS Azure or AWS like” self-service access to infrastructure, API-driven infrastructure resources that can be consumed within your application, ability to quickly create and clone dev/test environments are just a few goodies for developers.

The goal of a hyperconverged infrastructure (HCI) is to simplify how to apply compute, network and storage resources to applications. Ideally, the data center’s IT needs are consolidated down to a single

architecture that automatically scales as the organization needs to deploy more applications or expand existing ones.

3. DATA PROTECTION

3.1. Is traditional backup appropriate for HCI committed datacenter?

Let's look at how exactly things used to be before server virtualization. Applications were deployed on individual servers and each host was provided with a lot of network bandwidth. Some enterprises deployed dedicated networks to backup and restore data. Each server had an agent installed and the backup platform (metadata and media servers) was in the center of it.

Backup creates several separate architectures outside of the HCI architecture. Each of these architectures need independent management. First, the backup process will often require a dedicated backup server. That server will run on a stand-alone system and then connect to the HCI solution to perform a backup. Second, the dedicated backup server will almost always have its own storage system to store data backed up from the HCI. Third, there are some features, like instant recovery and off-site replication, that require production quality storage to function effectively.

The answer for IT is to find a backup solution that fully integrates with the HCI solution, eliminating the need to create these additional silos.

3.2. Benefits of HCI-Focused backup

What if backup was designed specifically to work with HCI? To achieve this, the backup software would need tight integration with the HCI platform itself. This means:

- Integration with platform-specific APIs to extract and backup data. Integration also needs to manage snapshots and pausing applications for consistent backups.
- Scalability within the backup application/instance to cater for increased load as deployments increase.
- Automatic identification and protection of virtual machines/instances, once given credentials.
- Licensing that mirrors the platform deployment model, for example, per-VM or instance charging.
- Self-protection of the backup VM, to be able to restore the entire environment in case of hardware failure.
- Multi-tenancy – allowing multiple backup environments to back up segments of workload on demand.
- Scripted or automatic deployment of the backup solution onto the HCI platform.
- HCI allows a high degree of application deployment automation via API and CLI. The platform is there to be consumed, which can be achieved without the intervention of an administrator. Why shouldn't backup be the same? Imagine a new project that will run a few dozen virtual instances for a few months. If this is a development project, it makes sense to keep backups separate from production, write the backup data to a dedicated target and isolate the backups for future use.
- The last point – automation – speaks exactly to this requirement. Backup becomes another service that can be deployed and configured on-demand to meet the needs of the end user. The most logical conclusion of this being the addition of backup to a service catalogue accessible by end users.

4. HYCU DATA PROTECTION FOR NUTANIX

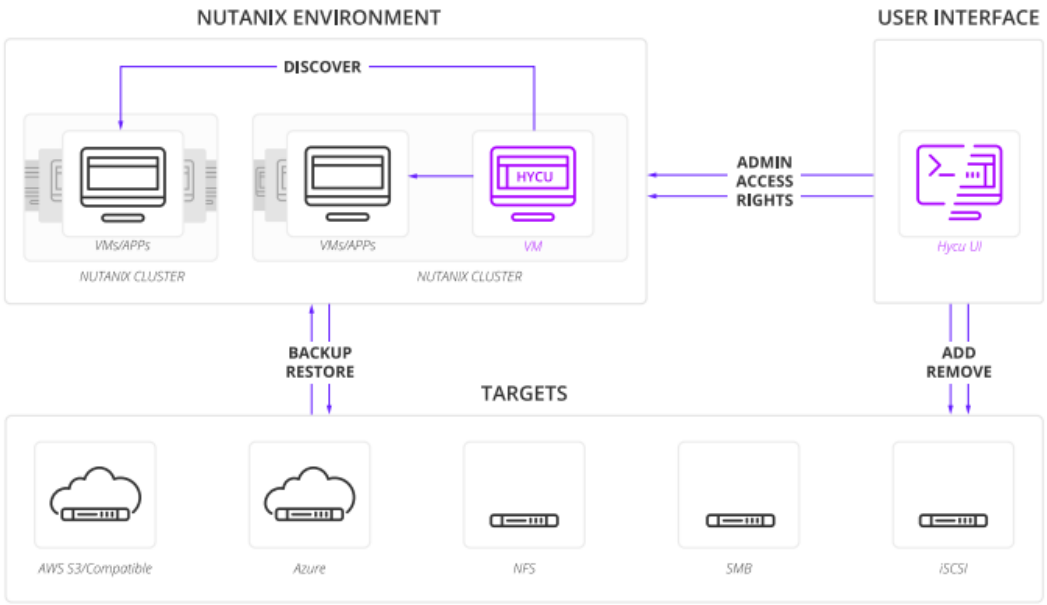
HYCU Data Protection for Nutanix (HYCU) is a high performing backup and recovery solution for Nutanix. It is the first data protection solution that is fully integrated with Nutanix and makes data protection easy to deploy and simple to use. When we started developing HYCU, ease of deployment and time to value were not just key requirements but a basic design principle from the get go.

HYCU is deployed as software and can be spun up as a virtual machine instance within minutes. Administrators simply need to provide credentials to the HCI platform, from where HYCU can identify and immediately start backing up virtual machines. HYCU is app-centric by design. By knowing what resides within the VMs, we know exactly what the configuration of the application is we need to recover and to provide all of the possible recovery options for it.

4.1. Key features and benefits

The following features make HYCU an HCI-focused backup solution:

- Delivers native and reliable data protection for mission-critical applications and data in hyperconverged environments, while ensuring data consistency and easy recoverability.
- Deployment of the HYCU virtual appliance is performed through the Nutanix Prism web console (for Nutanix AHV clusters) or the VMware vSphere Web Client (for Nutanix ESXi clusters).
- Discovery solution provides new-found visibility into virtual machines, pinpointing where each application is running.
- Data protection of virtual machines and applications can be enabled in a few minutes after the deployment.
- Predefined backup policies (Gold, Silver, and Bronze) that come with HYCU simplify the data protection implementation. However, if the needs of the backup environment require, a wide range of opportunities to customize backup policies is provided.
- Automatic backup scheduling provides data protection based on your recovery point objectives (RPOs).
- In-built application awareness provides application discovery and application-specific backup and restore flow and ensures that the entire application data is protected and recovered to a consistent state.
- Using data storage targets and hypervisors is the administrator's choice.
- The HYCU dashboard helps you identify potential problems and bottlenecks to improve the performance of your data protection environment.
- REST API



Picture 2: HYCU Architecture

4.2. Non-disruptive Data Protection

A key objective for any data protection solution is to be non-intrusive. Business applications, and in particular the production environment, should not be affected by your data protection workflows. Never. Ever. One of the biggest headaches for virtual backup solutions is the "VM stun" situation where the VM is non-responsive (stunned) because of the underlying data protection process.

Providing a purpose-built solution for Nutanix, that leverages its storage level snapshotting API, HYCU enables a non-disruptive backup process that completely eliminates the unresponsiveness of your VMs. VM stuns that are a result of the data protection solutions occur when the snapshot manipulation happens within the data protection workflow. You can find the best inside information on VM stuns from VMware's Cormac Hogan in his blog post "When and Why Do We Stun a VM." He also covered the topic in a recent post describing the improvements done in vSphere 6.0 named "Snapshot Consolidation changes done in vSphere 6.0".

If you take a step back, you can see that the higher up on the data storage stack you are, the bigger and longer the stuns will be. This is why, we believe, you need to provide the user the application content for backup, but the software should be smart to perform snapshot management operations on the storage level, i.e. get the best of both worlds.

By tightly integrating with Nutanix and fully exploring its V3 snapshotting APIs, HYCU minimizes the major disruption in the backup workflow. Coupled with our app-centric approach, the customer is able to experience efficiency and simplicity with a natural app-centric interaction.

5. REFERENCES

[1] Magic Quadrant for Hyperconverged Infrastructure, Gartner, February 2018.

[2] storageswiss.com, Why Backup is Breaking Hyper-Converged Infrastructures and How to Fix it

[3] www.hycu.com/blog, Data Protection for Nutanix

[4] www.nutanix.com, Hyperconverged infrastructure

Univerzitetna založba Univerze v Mariboru

ZAPIRANJE IN ODPIRANJE PODATKOV V DRŽAVNIH INFORMACIJSKIH SISTEMIH

SAMO MAČEK, FRANCI MULEC IN FRANC MOČILAR

Povzetek: V informacijske sisteme, ki so namenjeni podpori izvajanju ključnih funkcij države (vlada, zunanje zadeve), se uvrščajo gradiva, ki se razlikujejo po izvoru, vsebinskem področju, vrsti in stopnji zaupnosti. Tudi če se nanašajo na isto zadevo, se lahko za njihovo obravnavo zahtevajo povsem drugačne tehnične rešitve. Za zaščito interesov države določena gradiva zapiramo v izolirane sisteme, kjer jih varujemo pred naraščajočimi grožnjami terorizma, organiziranega in kibernetskega kriminala. Na drugi strani je z vidika odprtosti družbe zaželeno, da se v postopke odločanja, če je le mogoče, vključuje zainteresirano in strokovno javnost ter predstavnike civilne družbe. Zato je treba proaktivno omogočati javno dostopnost določenih vsebin, pod t. i. odprto licenco. Med obema skrajnostma ločujemo sisteme glede na stopnjo tajnosti (interno do strogo tajno), varnostni razred in nivo odprtosti. Čeprav lahko posamezna zadeva presega tehnične okvire določenega sistema, je treba vseeno zagotoviti njeno celovitost z vsebinskega in uporabniškega vidika. V prispevku bomo prikazali nekaj lastnih rešitev, s katerimi obvladujemo navedene izzive in so uporabni tudi za gospodarske ter druge subjekte.

Ključne besede: • varnostni razredi • kibernetski napadi • odprti podatki

NASLOV AVTORJEV: mag. Samo Maček, vodja Sektorja za informatiko, Generalni sekretariat Vlade RS, Gregorčičeva 20, 1000 Ljuvljana, Slovenija. mag. Franci Mulec, CISA, sekretar, Ministrstvo za zunanje zadeve, Prešernova 25, 1000 Ljuvljana, Slovenija. mag. Franc Močilar, CISSP, vodja Oddelka za informacijsko varnost in komunikacije, Ministrstvo za zunanje zadeve, Prešernova 25, 1000 Ljubljana, Slovenija.

DOI https://doi.org/10.18690/978-961-286-162-9.12 ISBN 978-961-286-163-6

Dostopno na: http://press.um.si.

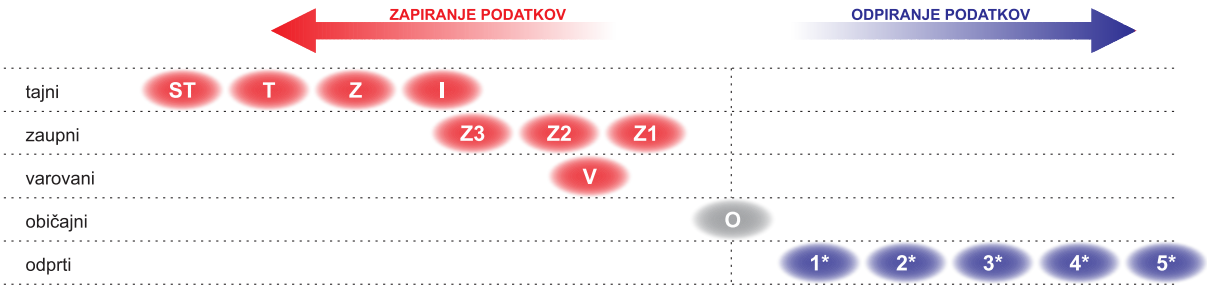
1. UVOD

Hiter razvoj informatike z množico novih konceptov in storitev s seboj prinaša tudi velike izzive na področju varnosti in odpornosti sistemov. V kibernetskem prostoru smo ogroženi posamezniki, poslovni subjekti in tudi države. Zaščitni ukrepi so odvisni od možnih škodljivih posledic varnostnega incidenta. Na ravni posameznika so zahteve povsem drugačne kot pri zaščiti podatkov, ki so pomembni za varnost in delovanje kritične infrastrukture.

Nedavno sprejeta Uredba o informacijski varnosti v državni upravi je uvedla pojem varnostnega razreda. Gre za ravni razvrščanja informacijskega premoženja in sistemov glede na njihov pomen za organizacijo, na njihovo vrednost oziroma občutljivost. Pri tem se upoštevajo vse tri dimenzije informacijske varnosti: zaupnost, celovitost (nespremenljivost) in razpoložljivost. S tem zaščitne ukrepe prilagodimo vsebini, dejavnosti in specifičnim zahtevam delovnih procesov.

Na drugi strani si morajo državni organi prizadevati, da vsem zainteresiranim subjektom omogočijo čim lažji dostop do informacij javnega značaja. S tem se zagotavlja večji nadzor javnosti nad delovanjem državnih organov in njihovim spoštovanjem pravnega reda, zmanjšuje korupcijska tveganja, zlorabe in nepravilnosti ter zagotavlja gospodarnejšo rabo javnih sredstev. Tudi na področju odpiranja vsebin obstaja več razredov oziroma t. i. razvojnih stopenj odprtosti. [2]

Dokumente posamezne zadeve uvrščamo v razrede glede na stopnjo zaupnosti ali odprtosti. To pomeni, da se lahko zaradi različnih zahtev varovanja, nahajajo v ločenih segmentih infrastrukture ali v sistemih, ki med seboj niso povezani. Ne glede na to, je treba z vidika uporabnika zagotoviti čim boljšo preglednost in celostnost zadeve.



Slika 1. Primerjava razredov glede na stopnjo zaupnosti in odprtosti (tajni - Zakon o tajnih podatkih, »zaupni« (zaupnost) - Uredba o informacijski varnosti, varovani - Uredba o upravnem poslovanju (nadomestili so jih »zaupni«); ocena - zaradi različnih pravnih podlag in namena uporabe razredi niso neposredno primerljivi)

2. ZAPIRANJE PODATKOV

Na področju varovanja občutljivih dokumentov, podatkov in informacij ter za varovanje informacijsko podprtih delovnih procesov lahko informacijsko okolje glede na nivo varnosti delimo na tri glavne skupine:

- običajni informacijski sistemi: zajema centralno upravljane in vzdrževane informacijske sisteme, vključno s prenosnimi računalniki in mobilno telefonijo, informatike, vključno z zunanjim izvajanjem;
- varnostno odobreni (akreditirani) informacijski sistemi: tu so zahteve po varnosti višje. Gre za sisteme za obdelavo občutljivih podatkov ali podporo občutljivim delovnim procesom. Lahko se uporabijo za občutljivejše osebne podatke, tajne podatke nižjih stopenj, poslovne skrivnosti, intelektualno lastnino, zdravstvene podatke in podobno. Tovrstni sistemi naj bi bili odporni na »napade« posameznika, ki ima na voljo znanja in splošno dosegljiva orodja. To je lahko tudi oseba

znotraj organizacije;
- sistemi za (naj)višje stopnje varnosti: gre za sisteme za obdelavo najobčutljivejših državnih podatkov, procesov, tajnih podatkov višjih stopenj, za ključne elemente kritične infrastrukture v državi ali na primer za procese v mednarodnih pogajanjih, katerih razkritje nepoklicani osebi bi lahko (hudo) škodovalo varnosti ali interesom Republike Slovenije. Tovrstni sistemi morajo biti odporni tudi na močne interesne skupine, ki imajo dovolj virov, ki ne spoštujejo zakonov, kjer so lahko pasivno ali aktivno v ozadju tuje državne institucije. Upoštevati moramo tudi, da so lahko znotraj naše organizacije osebe, ki lahko aktivno pomagajo nasprotniku. [1]

Glede na vsebino podatkom določamo različne vrste zaupnosti (kot na primer: poslovna skrivnost, davčna tajnost, osebni podatek, tajni podatek ipd.). Okvirno primerjavo med različnimi razredi (slika 1) lahko naredimo na osnovi zahtevanih ukrepov varovanja:

- prepoved uporaba javnega oblaka – Z2 in višje;
- prepoved obravnave podatkov na javnem mestu: Z1 in višje;
- prepoved uporabe javnih brezžičnih dostopnih točk: Z1 in višje;
- prepoved puščanja nosilcev s podatki v pisarnah, mizah ipd.: Z2 in višje;
- varen izbris ali uničenje podatkov na odrabljenih nosilcih: Z1 in višje;
- šifriranje prenosa in hramba izven varovanega območja: Z2, Z3;
- šifriranje prenosa: občutljivi osebni podatek;
- šifriranje prenosa z varnostno odobreno (akreditirano) rešitvijo: TP-I in višje;
- uporaba programskih šifrirnih rešitev – do vključno TP-I;
- uporaba programskih šifrirnih rešitev, izdelanih v državnih organih – do vključno TP-Z;
- uporaba strojnih šifrirnih rešitev – TP-T in višje;
- varnostno ovrednotenje šifrirnih rešitev, funkcionalni preizkus šifrirne rešitve v celoti in posameznih šifrirnih mehanizmov – TP-Z in višje;
- analiza možnosti kompromitiranja varnostno pomembnih parametrov – TP-T in višje;
- prepoved uporabe tujih šifrirnih rešitev – TP-ST;
- zaščita proti neželenemu elektromagnetnemu sevanju – TP-Z in višje;
- varovanje ključnih sestavin sistema v varnostnih območjih – TP-Z in višje.

(legenda: TP - tajni podatki, I - interno, Z - zaupno, T - tajno, ST - strogo tajno, Z1, Z2, Z3 - ravni zaupnosti)

Slika 2. Fizično varovanje podatkov nekoč - aktualno še danes: srednjeveški grad Carcassonne, predsedniška palača Ljubljana (1 – nenadzorovano območje, 2 – notranjost obzidja / nadzor policije, 3 – upravno območje, 4 – dodatno varovana območja – oznake so simbolične)

Pri obravnavanju najobčutljivejših podatkov je treba izvajati tudi določene ukrepe, ki jim pri običajni osebni ali službeni uporabi informacijske opreme ne posvečamo posebne pozornosti. V državnih informacijskih sistemih gre predvsem za podatke, katerih razkritje nepoklicani osebi bi lahko škodovalo varnosti ali interesom Republike Slovenije. Nekaj zaščitnih ukrepov:

- izolacija sistemov (*black / red separation*) – povezava s sistemi z nižjo stopnjo tajnosti je mogoča samo preko podatkovnih diod;
- sistemi ne smejo biti povezani z internetom;
- ustrezna postavitev opreme – upoštevati moramo tveganja optičnih in akustičnih napadov (npr. napadi s teleskopom, laserskim mikrofonom);
- protiprislušni pregledi prostorov;
- namestitev opreme v varnostna območja (posebej varovani prostori z ojačenimi stenami, protivlomnimi vrati, rešetkami na oknih, tehničnim in fizičnim varovanjem);
- fizično varovanje objektov izvaja policija;
- uporaba opreme, ki je zaščitena pred prestrezanjem podatkov prek neželenih elektromagnetnih in konduktivnih emisij;
- temeljito preverjanje oseb, ki delajo s sistemi in podatki;
- osveščanje, usposabljanje in nadzor uporabnikov;
- razvoj lastnih varnostnih programskih rešitev;
- uporaba ne-računalniških metod (papir, svinčnik, kurirji).

3. ODPIRANJE PODATKOV

Z vidika odprtosti družbe in udejanjanja participativne demokracije je zaželeno, da se v postopke odločanja, če je le mogoče, vključuje zainteresirano in strokovno javnost ter predstavnike civilne družbe. Predpogoj je javna dostopnost vsebin, pod t. i. odprto licenco.

Od leta 2003 mora biti, na podlagi direktive EU [3], vsem zainteresiranim subjektom omogočen dostop in ponovna uporaba informacij javnega značaja. Nadalje spremembe direktive iz leta 2013 poudarjajo potrebo po proaktivnem odpiranju podatkov s strani institucij in uvajajo pojem: »odprti podatki« (ang. *Open Data*).

Prenos direktive v slovenski pravni red je bil izveden z novelo Zakona o dostopu do informacij javnega značaja (ZDIJZ) in novo Uredbo o posredovanju in ponovni uporabi informacij javnega značaja.

ZDIJZ državnim organom nalaga, da omogočajo dostop do informacij javnega značaja, s katerimi razpolagajo. Bistvo »dostopa« je v omogočanju seznanitve z vsebino dokumenta. Za razliko od dostopa je bistvo »ponovne uporabe« v tem, da lahko uporabnik določen dokument, zbirko ali surove podatke na enostaven način prenese s spleta ter jih na svojem računalniku nadalje obdeluje, analizira, vključuje v nove aplikacije, storitve ali produkte ipd. Poleg zasledovanja transparentnega in učinkovitega delovanja javnega sektorja je slednje pomembno tudi z gospodarskega vidika. Odprti podatki pomenijo spodbudo za razmah digitalnega gospodarstva, saj so vir za številne spletne aplikacije.

Proaktivno odpiranje podatkov pomeni, da morajo biti vsi javni podatki vnaprej objavljeni na spletu (in ne šele na poziv zainteresiranega subjekta).

Definicija odprtega podatka:

- po vsebini je prosto javno dostopen,
- objavljen je v strojno-berljivem in odprtem tehničnem formatu,
- na voljo je pod t. i. odprto licenco - brezplačno, za katerikoli namen (edini pogoj je navedba vira).

Državni organi morajo javne podatke nuditi praviloma brez zaračunavanja stroškov oz. z zgolj minimalnimi materialnimi stroški. To velja za gradivo, ki je prosto pravic intelektualne lastnine. Izjeme so npr. knjižnice, muzeji in arhivi, ki lahko zaračunavajo stroške dostave, reprodukcije, obdelave.[5]

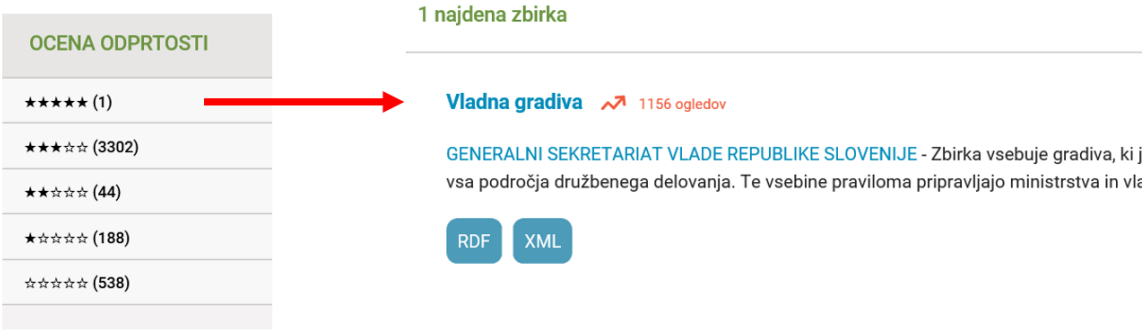
Tabela 1. Značilnosti stopenj odprtosti podatkov

Stopnja	Značilnost	Primer
1 ★	podatek je možno pridobiti na spletu pod odprto licenco	PDF
2 ★	+ podatek je dosegljiv v strukturirani obliki	XLS
3 ★	+ podatek je dosegljiv v nelastniškem odprtem formatu	CSV
4 ★	+ podatki so označeni z URI	RDF
5 ★	+ povezava z drugimi podatki - vsebinski kontekst	RDF

4. PRIMER – »VLADNA GRADIVA«

Zbirka vsebuje gradiva, ki jih je ali jih bo obravnavala vlada. Nanašajo se na vsa področja družbenega delovanja. Praviloma jih pripravljajo ministrstva in vladne službe ter se objavljajo v informacijskem sistemu vlade. Dokumenti se uvrščajo v varnostne razrede različnih stopenj. Generalni sekretariat vlade ima za ta namen akreditiranih več nivojev infrastrukture. Najobčutljivejše vsebine se ščitijo v izoliranih sistemih, kjer so med drugim implementirane tudi zgoraj navedene smernice zapiranja podatkov. Del gradiv pa je odprte narave. Ta segment je namenjen zainteresirani javnosti, zlasti nevladnim in drugim organizacijam civilne družbe. Vsi, ki so v procesu nastajanja gradiv sodelovali, lahko tukaj preverijo, kako so pristojni organi pri pripravi odločitev vlade upoštevali njihove pripombe, pobude in predloge. Način sodelovanja javnosti v postopku sprejema odločitev vlade opredeljuje Poslovnik Vlade RS. Izvzeta so tista gradiva, ki predstavljajo izjemo po Zakonu o dostopu do informacij javnega značaja. [4]

Vladna gradiva so edina med 4000 zbirkami na portalu OPSI z najvišjo stopnjo odprtosti, to je 5★. Prenesti jih je mogoče v odprti, strukturirani in povezani obliki (v zapisu XML ali RDF), in sicer preko izbire mandata vlade.



Slika 3. Razvojne stopnje odprtosti podatkov – portal OPSI (https://podatki.gov.si/data/search)

5. LITERATURA IN VIRI

[1] MULEC Franci, MOČILAR Franc, MAČEK Samo "Načrtovanje in obvladovanje kibernetske odpornosti sistemov na osnovi varnostnih razredov", Dnevi slovenske informatike DSI 2018: Zbornik petindvajsete konference, Portorož, Društvo informatika, 2018.

[2] Ministrstvo za javno upravo »Priročnik za odpiranje podatkov javnega sektorja«. Ljubljana, 2016.

[3] Evropski parlament, Svet EU »Direktiva o ponovni uporabi informacij javnega sektorja«. Uradni list L 345, 31. 12. 2003, str. 90-96.

[4] https://podatki.gov.si/dataset/vladna-gradiva-2, Informacije o zbirki Vladna gradiva, obiskano 18. 5. 2018.

[5] https://www.mju.gov.si, Spletni dostop do javnih evidenc, ponovna uporaba in odprti podatki, obiskano 18. 5. 2018.

Univerzitetna založba Univerze v Mariboru

CENTRALIZACIJA LOGIRANIH PODATKOV Z UPORABO ODPRTOKODNIH REŠITEV ZA UPRAVLJANJE VELIKIH KOLIČIN PODATKOV

MARKO POLAK IN GREGOR SLOKAN

Povzetek: Logiranje v aplikacijah je ključno za ugotavljanje ustreznega delovanja aplikacje, odpravo napak v aplikaciji in ugotavljanja ozkih grl pri delovanju aplikacije. Logiranje lahko razdelimo na beleženje aplikacijskih dogodkov (aplikacijski log) ter shranjevanje in iskanje po zabeleženih aplikacijskih dogodkih. Poslovna pravila beleženja aplikacijskih dogodkov ponavadi predstavljajo osnovo za razvijalce programske opreme kaj in kako beležiti, medtem ko na strani shranjevanja in iskanja po zabeleženih podatkih naletimo na kar nekaj izzivov, ki jih je potrebno rešiti. Pri shranjevanju se moramo odločiti o obliki shranjenih podatkov, načinu shranjevanja podatkov ter retencijskem času hranjenih podatkov. Pri obdelovanju tako shranjenih podatkov se večinoma osredotočimo na posamezne datoteke, ali kratkega časovnega obdobja, ker je časovna in sistemska zahtevnost obdelave celotnega sklopa podatkov prevelika. Dodatno dimenzijo problema uvede komunikacija med različnimi aplikacijami in danes vse bolj razširjen koncept mikro storitev, ter težave z dostopom razvijalcev aplikacije do produkcijskih logov zaradi različnih omejitev dostopa. Centralni logirni sistem ponuja različne možne rešitve opisane problematike, ponuja velike časovne prihranke človeških virov in omogoča enostavno dodeljevanje pravic vpogleda in beleženja dostopov.

Ključne besede: • centralni logirni sistem • velika količina podatkov • NoSQL • skoraj v realnem času(NRT) • skalabilnost • GDPR

NASLOV AVTORJEV: Marko Polak, senior developer in dba, Medius d.o.o, Tehnološki park 21, 100 Ljubljana, e-pošta: marko.polak@medius.si. Gregor Slokan, senior developer, Medius d.o.o, Tehnološki park 21, 100 Ljubljana, e-pošta: gregor.slokan@medius.si.

DOI https://doi.org/10.18690/978-961-286-162-9.13 ISBN 978-961-286-163-6

Dostopno na: http://press.um.si.

1. UVOD

Vsak, ki se dnevno ukvarja z razvojem ali nadzorom nad delovanjem aplikacij se zaveda pomembnosti logiranja različnih nivojev izvajanj aplikacije. Programerji preko logov spremljajo ustreznost delovanja aplikacij in rešujejo nastale težave v delovanju. Sistemski administratorji nadzoruje različne sisteme in aplikacije ter skrbijo za njihovo tekoče delovanje, preverjajo ustreznost delovanja in odkrivajo morebitne zaplete in jih skušajo preprečiti. Varnostni nadzorniki spremljajo možne varnostne težave, nadzorniki aplikacij od uporabnikov prejemajo različna poročila o delovanju ali nedelovanju aplikacij, ki jih filtrirajo in predajajo naprej sistemskim administratorjem, varnostnim nadzornikom ali programerjem, itd. [1]. Vsem naštetim je za vpogled v delovanje sistema in aplikacij skupna točka uporaba logiranih podatkov. Ustrezno logiranje, hranjenje, dostopi in učinkovita uporaba logiranih podatkov so ključni za vzdrževanje sistemov in aplikacij.

Logiranje lahko delimo na sistemsko in aplikacijsko, na upravljanje z logiranimi podatki ter na uporabo logiranih podatkov. V članku se bomo osredotočili:

- na aplikacijsko logiranje razvijalcev aplikacij,
- pri upravljanju s podatki na zbiranje podatkov v centralnem sistemu, hrambo podatkov, obdelavo in retencijo podatkov,
- pri uporabi podatkov pa bomo poudarek naredili na učinkovitem iskanju po logiranih podatkih.

Pri aplikacijskem logiranju obstajajo standardi in dobre prakse na podlagi katerih oblikujemo poslovna pravila logiranja, ki se jih morajo razvijalci programske opreme upoštevati. Del dobre prakse je uporaba standardnih ogrodji (ang. framework) za logiranje, kot je log4j.

Pri upravljanju s podatki se v praksi pojavi kar nekaj izzivov. Pri shranjevanju se moramo odločiti o obliki shranjenih podatkov (tekstovna datoteka, stisnjena datoteka), retencijskem času hranjenja podatkov (koliko časa v kaki obliki) in mestu hranjenja podatkov (običajno imamo več različnih sistemov za shranjevanje podatkov, odvisno od aktualnosti). Pri obdelovanju tako shranjenih podatkov se večinoma osredotočimo na posamezne datoteke kratkega časovnega obdobja, ker je časovna in sistemska zahtevnost obdelave celotnega sklopa podatkov prevelika. Pri omejevanju dostopa do podatkov pa je smiselno postaviti centralni sistem upravljanja z dostopi na katerega se integrira shramba. Dodatni težavnostni nivo predstavlja rastoča količina podatkov, ki jo moramo predvideti pri dimenzioniranju sistema. Rastoče so tudi potrebe po čim več podatkih, na katerih so možne obdelave za podporo poslovnih procesov.

Pri pregledu podatkov se osredotočamo na iskanje po logiranih podatkih s strani prej naštetih uporabnikov. Iskanje je pogosto zaradi (pre)velikih datotek omejeno na konzolne pregledovalnike in iskalnike ali pa izsek log datoteke preko konzolnih ukazov v manjše datoteke, ki jih lahko odpremo z grafičnimi pregledovalniki. Uporabnost takšnega načina iskanja pa je slaba. Poleg opisanega dodatno dimenzijo problema shranjevanja, obdelovanja in pregleda logiranih podatkov vpeljemo s komunikacijo med različnimi aplikacijami ter vpeljavo mikrostoritev. Vzdrževanje shranjenih podatkov, vzporedno obdelovanje in združevanje, ter iskanje preko različnih logov je svojevrsten problem, ki je težko rešljiv preko konzolnih pregledovalnikov oziroma je rešljiv, a časovno in resursko (človeško) zelo potraten.

V članku bomo opisali vpeljavo centralnega logirnega sistema, za podporo naštetih treh delov logiranja. Sistem temelji na odprtokodnih tehnologijah z dodanimi lastnimi komponentami, kjer odprtokodnih nismo mogli uporabiti, s poudarkom na iskanju po logiranih podatkih.

2. APLIKACIJSKI LOG

Aplikacijski log predstavlja seznam dogodkov, ki jih izpiše aplikacija. Vključuje različne tipe dogodkov (napake, opozorila, informacije,...). Oblika zapisa in vsebina posameznega loga pa je definirana na strani aplikacije, bodisi s strani razvijalcev, bodisi kot globalna interna pravila podjetja. Največkrat

aplikacijski log vidimo kot tekstovno datoteko, ki vsebuje logirane podatke [2]. Aplikacijski log tako predstavlja velik vir informacij, vendar so podatki v taki obliki dokaj neuporabni, sploh kadar imamo veliko aplikacij, ki tečejo na veliko aplikacijskih strežnikih, zato se ponavadi uporabljajo v kombinaciji z različnimi orodji za obdelavo aplikacijskih logov.

3. SISTEM CENTRALNEGA LOGIRANJA

Sistem centralnega logiranja predstavlja centralno hrambo vseh logiranih podatkov. To rešuje težave z iskanjem logiranih podatkov preko več strežnikov in aplikacij, hkrati pa nam omogoča oblikovanje enotne strategije shranjevanja, obdelovanja, nadzora nad logiranimi podatki in dostopa do podatkov podjetja ali korporacije.

Centralni logirni sistem nam mora omogočati: zanesljivost delovanja, hiter dostop do podatkov (zaželeno je v skoraj realnem času, za hiter odziv na zabeležene dogodke), varnost podatkov, določanje pravic dostopa do podatkov in fleksibilno razširljivost sistema na različnih nivojih (to nam omogoča hiter odziv trendom logiranih podatkov in nižje stroške upravljanja s strojno opremo).

Za uspešno izvedbo sistema centralnega logiranja in uporabnost podatkov moramo razmisliti najmanj o naslednjih aspektih:

- zbiranje podatkov,
- prenos podatkov v centralni sistem,
- hranjenje podatkov,
- retencijski čas,
- analiziranje podatkov,
- varnost,
- dostop do podatkov.

3.1. Arhitektura sistema

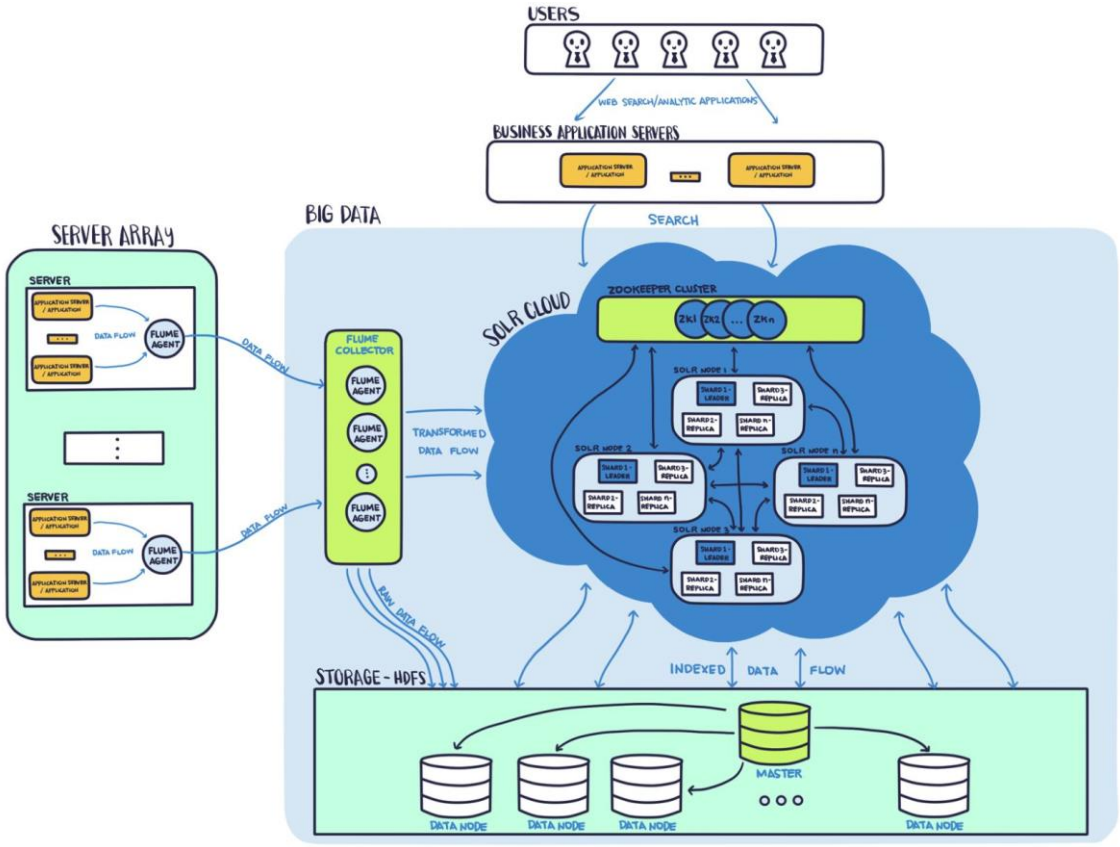
Sistem centralnega logiranja smo postavili z uporabo odprtokodnih tehnologij (Apache Flume, Hadoop, HDFS, Solr Cloud). Omogočajo nam prej naštete funkcionalnosti in hkrati ponujajo skalabilnost sistema na vseh nivojih. To je nujno za dolgoročno vzdržnost in maksimalno prepustnost sistema. Skalabilnost nam na eni strani omogoča, da lahko na produkcijskih sistemih logiramo bistveno več dogodkov ali isto število z več informacijami. Posledica je na eni strani hitrejša odprava napak, na drugi strani pa omogoča podatkovno rudarjenje za podporo poslovnih procesov v podjetju.

Z uporabo namenskih odprtokodnih tehnologij smo zagotovili stabilnost delovanja sistema (komponente so že širše preizkušene in delujejo). Uporabljene komponente je potrebno samo ustrezno konfigurirati, da znajo med seboj komunicirati. Pri tem je potrebno poudariti, da je nujno za vsako komponento posebej upoštevati želje in potrebe stranke za katero gradimo centralni logirni sistem, da komponenta pokrije celotno potrebno funkcionalnost. Dograjevanje funkcionalnosti je bolj kompleksno, kot če gre za naše komponente in se ji želimo izogniti.

Predstavljeni sistem je sestavljen iz:

- zbiranja in prenosa podatkov v centralni sistem,
- shranjevanja podatkov,
- indeksacije podatkov,
- iskanja po podatkih.

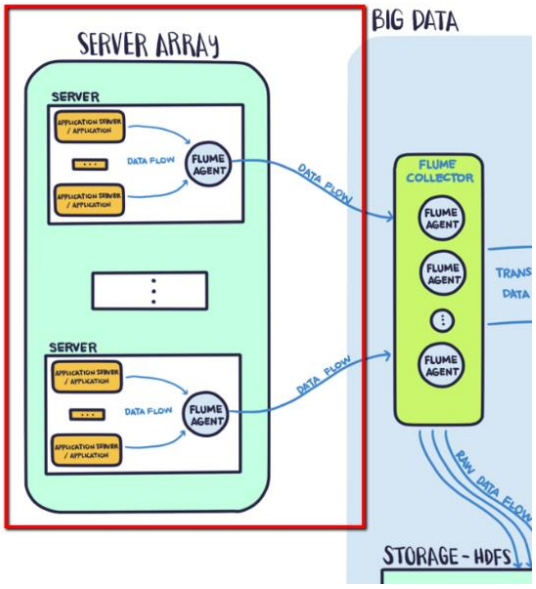
Slika 1 prikazuje arhitekturo sistema centralnega logiranja.



Slika 1: Arhitektura centralnega logirnega sistema [3]

3.2. Zbiranje podatkov in prenos v centralni sistem

Zbiranje podatkov se dogaja na aplikacijskih strežnikih. Slika 2 (izsek arhitekture centralnega logirnega sistema) prikazuje zajem podatkov na aplikacijskih strežnikih in prenos v centralni logirni sistem.



Slika 2: Arhitektura centralnega logirnega sistema - zbiranje podatkov in prenos v centralni logirni sistem [3]

Določiti je potrebno način zbiranja podatkov, strukturo podatkov in učinkovit način za pošiljanje podatkov v centralni sistem.

Določili smo, da se zbira dva tipa podatkov oziroma da zbiranje podatkov poteka na dva načina:

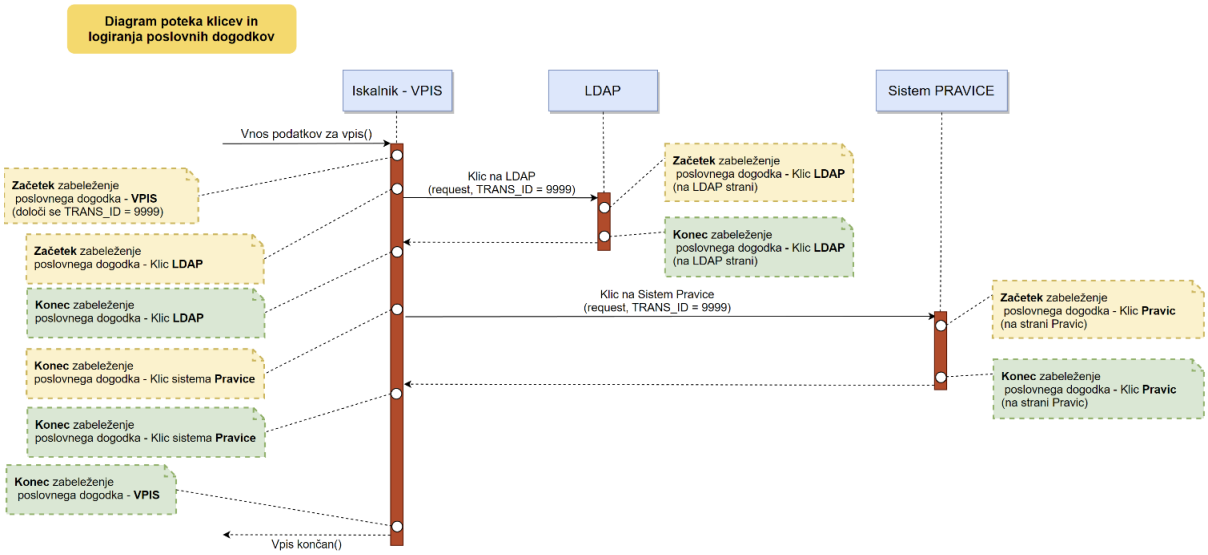
- aplikacijski poslovni dogodki,
- aplikacijski log.

Aplikacijski poslovni dogodki imajo enotno podatkovno strukturo. Definirana je pri postavitvi centralnega sistema in jo morajo upoštevati vse aplikacije, ki se v sistem želijo vključiti. Poslovni dogodki, ki jih želimo beležiti, se deloma definirajo v sklopu pravil logiranja podjetja (npr. zabeležiti je potrebno klic vsake zunanje spletne storitve), deloma pa je prepuščena vsebinskemu delu posamezne aplikacije. Najbolje je, da se konkretna pravila definira ob začetku razvoja aplikacije, ko se definirajo tudi vsebinske zahteve aplikacije. Razvijalec mora na podlagi definiranih pravil na ustrezna mesta v aplikacijo vgraditi beleženje poslovnega dogodka (npr. klic zunanje spletne storitve ali odpiranje okna v aplikaciji) in odgovor oziroma rezultat ob koncu izvajanja poslovnega dogodka. Zabeležen dogodek mora vsebovati osnovne podatke, ki jih potrebujemo za unikatno identifikacijo poslovnega dogodka in za iskanje ter analizo dogodka. Taki parametri so:

- ime okolja,
- ime aplikacije,
- transakcijski identifikator (zaželeno je, da je unikaten čez vsa okolja ali vsaj, da se dolgo časa ne more ponoviti),
- čas začetka izvajanja,
- prijavljenega uporabnika (lahko je sistemski),
- status izvajanja,
- metodo, ki se je klicala, itd.

Dodatno pa je smisleno predvideti, da lahko posamezna aplikacija definira svoje dodatne parametre, brez potrebe po spreminjanju sistema.

Ime okolja, ime aplikacije, transakcijski identifikator in čas začetka izvajanja poslovnega dogodka nam definira unikatni identifikator dogodka. Izvajanje poslovne logike veji na več delov (kliče se več metod) ali pa je kot del klica, klican zunanji sistem. Začetek izvajanja poslovne logike nastavi transakcijski identifikator, ki se nato posreduje celotni hierarhiji klicev. To nam omogoča enostavno sledenje izvajanja poslovne logike čez aplikacijo in čez vse povezane oziroma klicane sisteme.



Slika 3: Diagram poteka klicev in logiranja poslovnih dogodkov

Za lažjo predstavo je na sliki 3 predstavljen diagram poteka beleženja enega takega poslovnega dogodka. Predstavljen je vpis uporabnika v spletno aplikacijo (login).
Opis poteka:

- Uporabnik se s prijavnimi podatki vpiše v spletno aplikacijo.
- Zabeleži se začetek želenega vpisa in definira transakcijski identifikator, ki bo isti po celotnem toku zabeleženih dogodkov.
    - Podatke navedene na vpisni maski je potrebno preveriti v zunanjem sistemu (LDAP). Zabeleži se podatke o začetku klica zunanjega sistema in naredi klic.
        - Sistem LDAP zabeleži klic in preveri podatke ter vrne pozitiven odgovor. Odgovor se doda že zabeleženim podatkom in vse podatke shrani kot celota enega poslovnega dogodka.
    - Vpis s tem še ni končan, saj je potrebno preveriti kakšne pravice dostopa ima uporabnik v spletni aplikaciji. Zabeleži se osnovne podatke o klicu na sistem Pravice ter z istim transakcijskim identifikatorjem naredi klic na sistem.
        - Sistem Pravice zabeleži klic, ter transakcijski identifikator vodi po celotni logiki izvajanja, dokler ne vrne podatkov pravic dostopa. Odgovor se doda že zabeleženim podatkom in vse podatke shrani kot celota enega poslovnega dogodka.
- Vpis je tako končan, doda se še odgovor o uspešnosti klica in shrani kot celota enega poslovnega dogodka.
- Uporabnik je uspešno prijavljen.

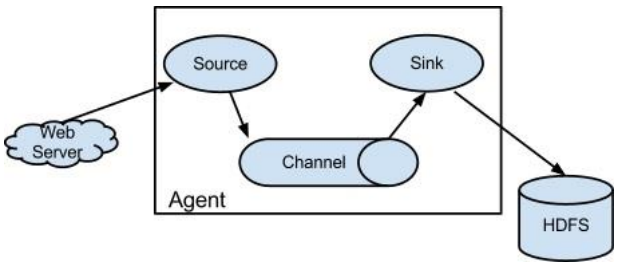
Tako dobimo sled izvajanja poslovnih pravil s celotno hierarhijo klicev znotraj aplikacije in povezanih sistemih, opremljeno z vsemi potrebnimi podatki za kasnejše analize. Smiselno je, da se definira enotno kodo, ki skrbi za logiranje poslovnega dogodka, razvijalec tako samo definira podatke, ki jih mora eksplicitno navesti. Prav tako je potrebno zagotoviti, da v primeru neuspešnega beleženja poslovnega dogodka aplikacija nemoteno deluje.

Aplikacijski log predstavlja celoten log, ki nastane na aplikacijskem strežniku. Predpišemo mu želeno struktura, ki pa jo lahko zagotovimo samo za novo nastale logirane podatke. Za nazaj logiranih podatkov ne moremo spreminjati. Prav tako se v praksi zgodi, da obstoječim sistemom ne moremo spremeniti strukture logiranih podatkov, zaradi že obstoječih odvisnosti ostalih sistemov, ki neposredno uporabljajo logirane podatke. Tako moramo v tem delu to upoštevati in se temu prilagoditi. Najbolje je, da se definirajo standardi za vse nove aplikacije in sisteme, ki bodo vključeni v sistem centralnega logiranja, kar nam poenoti zbiranje podatkov od sedaj naprej. Za vse sisteme pa je zaželeno, da vpeljejo najmanj transakcijski identifikator, preko katerega aplikacijski log enostavno povežemo z logom poslovnih dogodkov. Tako log poslovnih dogodkov daje enostaven vpogled v poslovno logiko aplikacije, aplikacijski log pa celotno zabeleženo izvajanje poslovne logike.

Pošiljanje podatkov aplikacijskih poslovnih dogodkov v centralni sistem ni problematično. Ker gre za dograditev kode aplikacije, lahko pošiljamo podatke v različnih oblikah (avro, jms, Kafka, jdbc,...). Pri pošiljanju aplikacijskih logov pa postanejo stvari bolj zapletene. Novejše programske knjižnice nam znatno poenostavijo zbiranje podatkov in prenos v centralno hrambo (npr. Log4j 2, Logback,...) že preko dodajanja ustrezne konfiguracije na aplikacijski strežnik. Vendar se v velikih podjetjih pogosto srečujemo s starejšimi do zelo starimi tehnologijami, ki nimajo teh možnosti. Za takšne sisteme se uvede drugačen mehanizem prenosa in transformacije v enotno strukturo. Mi smo za prenos uporabili Apache Flume, za transformacijo pa Morphlines.

3.2.1. Apache Flume

Apache Flume je distribuiran, zanesljiv sistem za pridobivanje, agregiranje, filtriranje in prenašanje večjih količin podatkov [4]. Agente je mogoče postaviti zaporedno (podatkovna veriga) za različne transformacije in agregacije in vzporedno za horizontalno skalabilnost.



Slika 4: FLume agent[4]

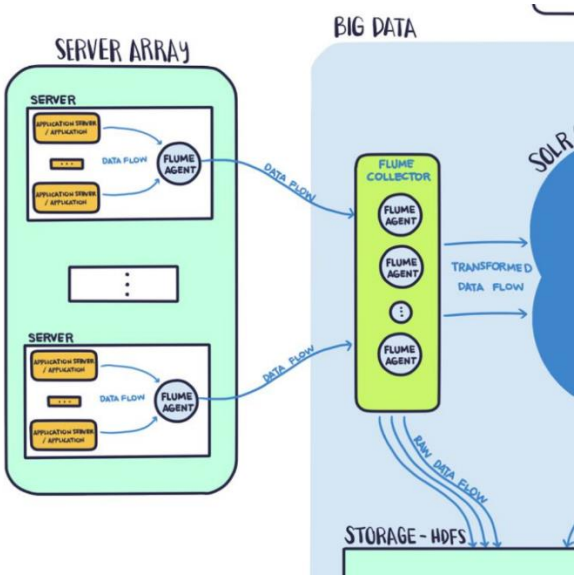
Slika 4 prikazuje interno strukturo podatkovnega toka Flume agenta. Podatkovni tok ima izvor, kanal in ponor.

Izvor podatkov je lahko Avro podatkovni tok, jms vrsta, Kafka, datotečni imenik,ipd, hkrati pa podpira tudi log4j Appender, ki se poveže na Avro podatkovni tok [4]. Takšna fleksibilnost nam omogoča, da isto komponento uporabimo tako za star način prenosa aplikacijskih logov kot novejši način prenosa logiranih podatkov.

Na starejših aplikacijskih strežnikih je potrebno samo nastaviti odlaganje datotek (npr. minutnih logov) v ustrezen imenik ter nastaviti konfiguracijo Flume agenta in že imamo vse pripravljeno za zajem podatkov.

Na novih sistemih pa ustrezno nastavimo konfiguracijo logiranja, da sistem za logiranje pošilja podatke v izvor flume agenta ali v kolektor.

Podatki se iz izvora prenesejo v kanal, kjer se začasno shranijo, dokler se jih uspešno ne pošlje v ponor. Konfiguracija kanala prav tako omogoča različne načine začasne hrambe podatkov (spomin, JDBC, Kafka, datotečni sistem,...), zaradi zanesljivosti prenosa in enostavnosti smo se mi odločili za datotečni sistem. Kot ponor pa se podatki prenesejo v Flume kolektor, ki skrbi za nadaljnjo transformacijo in shranjevanje podatkov. Kolektor Flume agentov predstavlja skupek agentov z isto konfiguracijo, ki zbirajo podatke ostalih Flume agonetov. Število Flume agentov v kolektorju je odvisno od performančnih potreb in od želje po zanesljivosti delovanja (v primeru izpada enega agenta se podatkovni tok preusmeri na ostale agente).



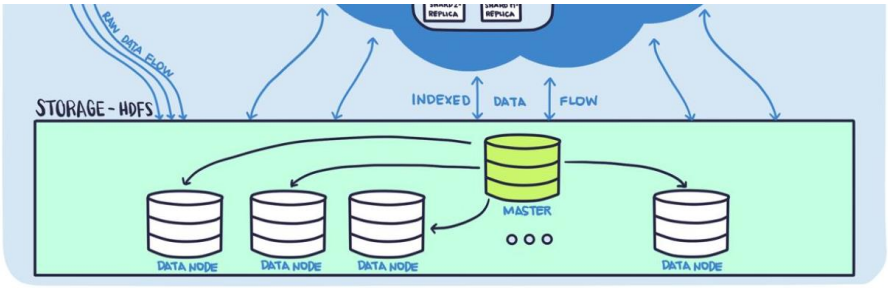
Slika 5: Arhitektura centralnega sistema - Flume agenti in kolektor [3]

Slika 5 prikazuje zajem podatkov na različnih aplikacijskih strežnikih, z uporabo Flume agentov in prenos do kolektorja Flume agentov, ki skrbi za prenos podatkov naprej proti sistemu za indeksacijo (Solr Cloud) in sistemu za shranjevanje raw podatkov (HDFS).

Tako Flume agenti na strežnikih kot Flume agenti v kolektorju so vertikalno skalabilni.

3.3. Hranjenje podatkov

Podatki se preko Flume kolektorja pretakajo v centralno shrambo. Za centralno shrambo smo uporabili HDFS v katerega se stekata dva podatkovna toka. En podatkovni tok z nespremenjenimi podatki, drug podatkovni tok s transformiranimi in indeksiranimi podatki (Slika 6). Za transformacijo, kot del Flume kolektorja, skrbi Morphines, za indeksacijo podatkov pa Solr Cloud, ki prav tako shranjuje podatke v HDFS.

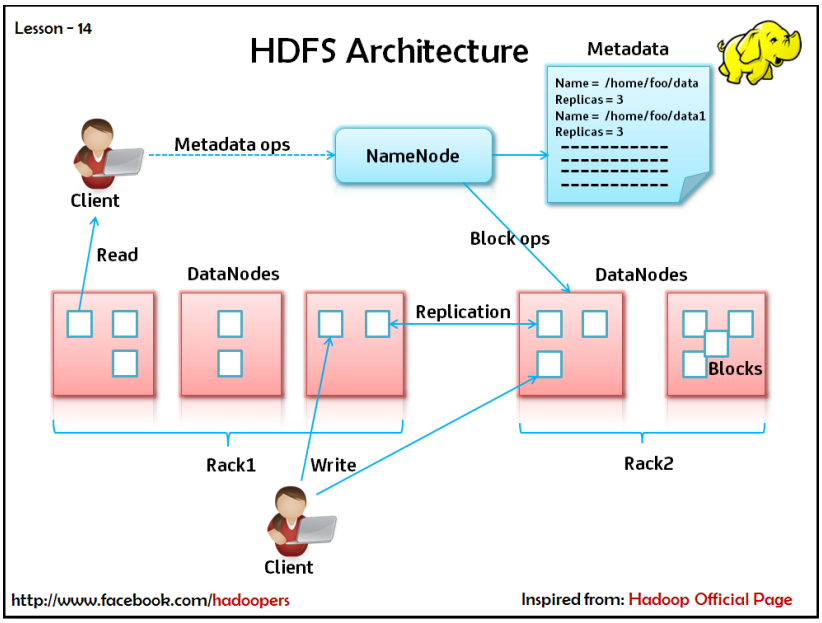


Slika 6: Arhitektura centralnega sistema – centralna hramba [3]

3.3.1. HDFS

Za shranjevanje podatkov smo uporabili HDFS (Hadoop Distributed File System). Je distribuiran datotečni sistem, ki temelji na tehnologiji Java in omogoča skalabilno in zanesljivo hrambo podatkov, ki se lahko širi čez veliko gručo nizko cenovnih strežnikov [5].

Slika 7 prikazuje arhitekturo HDFS sistema.



Slika 7: Arhitektura HDFS sitema [6]

Arhitektura je sestavljena iz imenskega vozlišča (NameNode) in podatkovnih vozlišč (DataNode), Imensko vozlišče je samo eno in upravlja z datotečnim imenskim prostorom ter upravlja dostope klientov, ki želijo pisati ali brati shranjene podatke. Vzdržuje tudi podatke o trenutno delujočih podatkovnih vozliščih in ustrezno preusmerja promet [6].

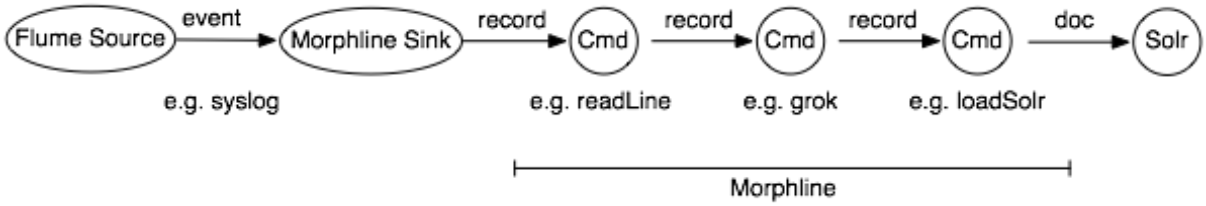
Podatkovna vozlišča upravljajo s podatkovnim prostorom na vozlišču. Skrbijo za shranjevanje in posredovanje podatkov klientom in za replikacijo podatkov. Podatki so shranjeni v blokih, kar omogoča razporeditev ene datoteke preko več podatkovnih vozlišč. Podatkovna vozlišča skrbijo za kreiranje, brisanje in replikacijo blokov na podlagi ukaza imenskega vozlišča. Klienti od imenskega vozlišča dobijo informacije kje so podatki, dejanski prenos podatkov pa gre neposredno do podatkovnih vozlišč [6]. Podatkovnih vozlišč je lahko tudi več tisoč.

Takšen način hranjenja podatkov nam omogoča enostavno razširljivost kapacitet hranjenja, v kombinaciji z Apache Hadoop(zbirka odprtokodnih programskih paketov, ki omogočajo distribuirano procesiranje velikih količin podatkov) pa enostavno in učinkovito kasnejše analiziranje podatkov.

3.3.2. Morphlines

Morphlines je odprtokodno ogrodje za transformacijo podatkov. Konfiguracijo transformacij definiramo v konfiguracijski datoteki morphline. V konfiguraciji lahko gradimo različne verige transformacij, ki jih uporabimo za transformacijo podatkov iz kateregakoli vira, procesiranje in prenos transformiranih podatkov v Solr Cloud [7]. Morphlines je javanska knjižnica vgrajena v Flume agenta. Uporabimo jo preko ustrezne konfiguracije ponora agenta.

Slika 8 prikazuje model procesiranja Morphlines.



Slika 8: Model procesiranja Morphlines [7]

Flume agent dobi na vhodu syslog dogodek in ga pošlje naprej v Flume Morphline Sink. Ta spremeni vsak dogodek v zapis primeren za procesiranje z Morphline. Morphline zapis prebere (readLine), naredi transgormacije (grok) in ga pošlje naprej v Solr (loadSolr) [7].

3.3.3. Solr Cloud

Solr Cloud je fleksibilna distribuirana platforma za iskanje in indeksiranje podatkov brez glavnega vozlišča (kot je to npr. pri HDFS). Glavne lastnosti so centralna konfiguracija za množico vozlišč, mehanizem za avtomatično razporejanje opravil in integracija z ZooKeeper sistemom za distribuirano koordinacijo in konfiguracijo. [9]

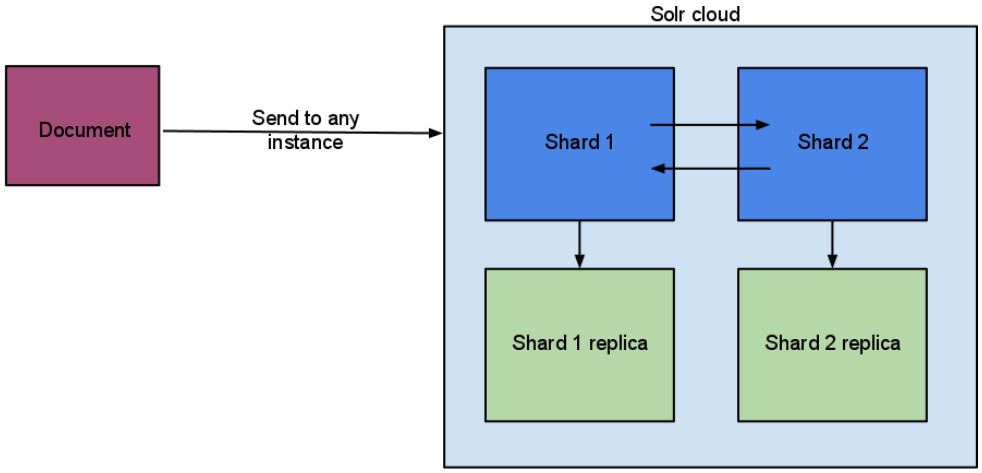
Glavni elementi Solr oblaka so [9]:

- Zbirka (collection) – vsaka zbirka ima ime, število shard-ov na katere je razdeljena in replikacijski faktor za distribuiranje indeksiranih podatkov preko več vozlišč.
- Replikacijsi faktor – kolikokrat se skopira en indeksiran dokument v zbirki.
- Shard – je zbirka logičnih delov indeksiranih podatkov. Vsak shard ima ime, svoje glavno vozlišče, zgoščeno vrednost (and. Hash) in replikacijski faktor. Vsak shard ima najmanj eno

glavno vozlišče in nič ali več replikacijskih. Indeksiran dokument je na podlagi zgoščene vrednosti dodeljen točno enemu shardu v zbirki.
- Replika – hrani kopijo indeksa glavnega vozlišča.

Indeksirani podatki so shranjeni v HDFS, tako da imamo visoko odpornost na odpovedi in skalabilnost v primeru potreb po razširitvi sistema.

Arhitektura Solr Cloud je prikazana na Slika 9.



Slika 9: Arhitektura Solr Cloud[9]

3.4. Indeksacija in iskanje

Iskanje po logiranih podatkih je ena ključnih funkcionalnosti vsakega logirnega sistema. Ob prenosu podatkov v centralni logirni sistem se hkrati izvede indeksacija podatkov za iskanje. To uporabnikom Iskalnika omogoča iskanje po logiranih podatkih v skoraj realnem času.

Indeksacija podatkov je tesno povezana z načinom iskanja po podatkih. Naš centralni logirni sistem zajema dva podatkovna tokova, ki sta različno indeksirana.

Aplikacijski poslovnih dogodki imajo točno določeno podatkovno strukturo in so opremljeni z zahtevkom in odgovorom izvajane operacije. Strukturo morajo upoštevati vse povezane aplikacije, ki poslovne dogodke pošiljajo v centralni sistem. Najbolj pomembna komponenta podatkov je transakcijski identifikator, ki omogoča hierarhično sledenje izvajanja znotraj ene ali več aplikacij.

Aplikacijski log ima nabor nujnih podatkov (kot so ime aplikacije in čas nastanka loga) in nabor možnih podatkov ter celoten log, kot je bil generiran na aplikacijskem strežniku. Dodatno je zaželeno, da je aplikacijski log opremljen s transakcijskim identifikatorjem, ni pa nujno, le iskanje bo malce oteženo, ker toka poslovnih dogodkov ne moremo neposredno povezati z aplikacijskim logom.

To je bila osnova za izgradnjo več nivojskega spletnega iskalnika, ki omogoča:
- iskanje po poslovnih dogodkih,
- iskanje po poslovnih dogodkih in prehod iz najdenih rezultatov v iskanje po transakcijskem identifikatorju za celotno sled v aplikaciji oziroma preko vseh aplikacij vključenih v sistem,
- iskanje po poslovnih dogodkih in prehod iz najdenih rezultatov v iskanje po transakcijskem identifikatorju v aplikacijskih logih v aplikaciji oziroma preko vseh aplikacij vključenih v sistem,
- iskanje po aplikacijskem logu,
- iskanje po poslovnih dogodkih ali po aplikacijskem logu in prehod iz najdenih rezultatov v časovno okno okrog najdenega rezultata (kadar nimamo transakcijskega identifikatorja ali

pa nas zanima ali je naša težava posledica neke druge težava na aplikacijskem strežniku/sistemu).

Vse različne kombinacije iskanja nam omogočajo, da hitro in enostavno najdemo iskane podatke, potem pa iščemo v globino ali v časovnem oknu iskanega dogodka obeh podatkovnih tokov.

3.4.1. Zaščita dostopa do podatkov

Poenostavljen način dostopa do logiranih podatkov z uporabo spletnega iskalnika omogoča dostop do podatkov vsakomur, ki ima dostop do iskalnika. Logirani podatki nemalokrat vključujejo tudi osebne podatke, zato je zaščita podatkov nujna zaradi zakonskih omejitev (Splošna uredba o varstvu podatkov (angl. General Data Protection Regulation – GDPR) [10], skladnosti z določenimi standardi, ki smo jih obvezani spoštovati in naše obveze do strank o ravnanju z njihovimi podatki.

V iskalnik so vgrajeni trije sistemi za upravljanje z dostopi do podatkov:
- omejevanje dostopa,
- zameglitev podatkov,
- revizijska sled.

V iskalnik je vgrajen mehanizem pridobivanja pravic dostopa do podatkov. Za upravljanje s pravicami dostopa skrbi zunanji sistem, iskalnik pa jih pridobi s klicem izpostavljene spletne storitve. Uporabniku se na iskalni maski prikažejo le njemu dostopna okolja in znotraj posameznega okolja njemu dostopne aplikacije. Tako se uporabnik, ki nima pravic do vseh okolji in aplikacij, niti ne zaveda katera obstajajo in po njih ne more iskati. Za pravice dostopa mora kontaktirati administratorja sistema.

Okolje in aplikacija sta še vedno zelo širok pojem dostopa, zato je v iskalnik dodatno vgrajen mehanizem zamegljevanja podatkov (ang. data obfuscation). V tem primeru ima uporabnik dostop do podatkov, so mu pa delčki prikriti (npr. osebni podatki uporabnika). Tako lahko nemoteno išče po podatkih, brez da bi kršili prej naštete razloge za omejevanje dostopa do podatkov.

Krog zaščite dostopa do podatkov pa je sklenjen z revizijsko sledjo. Vsako iskanje se zabeleži v revizijsko sled, ki je dostopna v administracijskem delu aplikacije. Dodatno lahko administrator sistema enostavno ponovno zažene isto iskanje kot ga je zagnal uporabnik.

3.5. Retencija podatkov

Retencija podatkov predstavlja čas hranjenja podatkov pred izbrisom. Vsi podatki niso enaki, ampak se lahko potrebe po izbrisu razlikujejo. Prednost indeksiranih podatkov je v tem, da jih v fazi obdelave lahko opremimo s parametri na podlagi katerih lahko določimo čas hranjenja podatka na log natančno. Tako lahko zagotovimo zakonsko skladnost kot tudi skladnost z različnimi standardi, ki se jih moramo držati.

4. ZAKLJUČEK

V večje podjetje smo uspešno vpeljali centralni logirni sistem s poudarkom na iskanju po logiranih podatkih. Odprtokodne komponente so se izkazale za zanesljive, fleksibilne in skalabilne tudi pod velikimi obremenitvami sistema. Uporabnikom sistem omogoča iskanje po logiranih podatkih vseh vključenih aplikacij v skoraj realnem času.

Ugotovili smo, da je uporaba odprtokodnih komponent dokaj enostavna. S stranko smo natančno opredelili načine uporabe (njene potrebe), želen rezultat in možne tehnologije na vseh nivojih. Na podlagi pridobljenih informacij smo izbrali ustrezne komponente in konfiguracija teh komponent, kar zgradi sistem v fleksibilno in skalabilno celoto na vseh nivojih in nam daje dolgoročno vzdržnost sistema.

Kjer nismo našli ustrezne odprtokodne komponente smo razvili svoje rešitve v duhu celotnega sistema, ki dopolnijo odprtokodne in jih ne omejujejo. Razvite komponente so vpeljale fleksibilnost, ki jo potrebujemo (dograjevanje po potrebi s poslovno logiko stranke). V odprtokodnih komponentah smo se raje držali sprememb konfiguracije in ne sprememb samih komponent.

V fazi načrtovanja vpeljave centralnega logirnega sistema smo naleteli na različne odzive. S strani sistemskih administratorjev, ki bodo zadolženi za vzdrževanje sistema je bil izražen dvom o vzdržnosti in uporabnosti sistema d(niso videli potrebe po centralnem logirnem sistemu in so bili zadovoljni z obstoječim stanjem). Potencialni uporabniki iskalnika so bili navdušeni nad zmožnostmi več nivojskega spletnega iskalnika, ki jim na enem mestu omogoča iskanje preko več aplikacij v skoraj realnem času. Bili so skeptični o uspešnosti postavitve sistema zaradi preteklih neuspehov.

Po skoraj dveh letih uporabe je navdušenje nad sistemom vsestransko. Uporabniki v nekaj minutah pridejo do informacij, za katere so prej porabili nekaj dni in z več iteracij preko različnih oseb.

Postavljen sistem trenutno deluje pod veliko obremenitvijo, je vzdržen in po potrebi razširljiv na vseh nivojih.

Centralni logirni sistem uporabniki dojemajo skozi aplikacijo Iskalnik. Aplikacija Iskalnik je prej kot v enem letu postala ena ključnih in nepogrešljivih aplikacij v podjetju.

5. LITERATURA

[1] https://syslog-ng.com/blog/why-logging-is-important/, Why logging is important?, obiskano 18.05.2018.

[2] https://www.techopedia.com/definition/1819/application-log, Application Log , obiskano 18.05.2018.

[3] https://www.medius.si/services/big-data-management/#logging, Slika arhitekture centralnega logirnega sistema, obiskano 18.05.2018.

[4] https://flume.apache.org/, Apache Flume, obiskano 18.05.2018.

[5] https://searchdatamanagement.techtarget.com/definition/Hadoop-Distributed-File-System-HDFS, Hadoop Distributed File System (HDFS), obiskano 18.05.2018.

[6] http://pramodgampa.blogspot.si/2013/08/hdfs-in-detail.html, HDFS Architecture, obiskano 18.05.2018.

[7] https://blog.cloudera.com/blog/2013/07/morphlines-the-easy-way-to-build-and-integrate-etl-apps-for-apache-hadoop/, Introducing Morphlines: The Easy Way to Build and Integrate ETL Apps for Hadoop, obiskano 18.05.2018.

[8] https://intellipaat.com/tutorial/apache-solr-tutorial/apache-solr-cloud-architecture/, Apache Solr Cloud Architecture, obiskano 18.05.2018.

[9] https://sematext.com/blog/solrcloud-distributed-realtime-search/, The New SolrCloud: Overview, obiskano 18.05.2018.

[10] https://www.ip-rs.si/zakonodaja/reforma-evropskega-zakonodajnega-okvira-za-varstvo-osebnih-podatkov/, Reforma evropskega zakonodajnega okvira za varstvo osebnih podatkov, obiskano 18.05.2018.

Univerzitetna založba Univerze v Mariboru

CIVILIZACIJA DOBRIH SLABIH PROGRAMOV

MATEJ ŠPROGAR

Povzetek: Računalniki so odvisni od programerjev, zato je obvladovanje programerjev primarni cilj velikih korporacij. Programerji pričakujejo vedno nove in udobnejše jezike tudi zaradi potrošniške iluzije, da je novo nujno boljše od starega. Posledično se polenijo in s tem dejansko pripomorejo k povprečnemu upadu znanja in širitvi slabih rešitev, ker nekritična uporaba novega dejansko promovira slabe rešitve. Kritičnost je mogoča samo, če imamo znanje in dostop do izvorne kode in ker tega ni, se raven znanja samo še znižuje. Velike inovacije pri programiranju so že zelo stare, moderna doba pa doživlja glavni problem predvsem v poplavi kompleksnih tehnologij. Potrošniški ideali slabo vplivajo na povprečno znanje programerjev, kar vodi do paradoksa slabšega programerja. Prihodnost je lahko samo v rešitvah, ki bodo preproste in razumljive.

Ključne besede: • programski jeziki • programska orodja • kvaliteta kode • paradoks slabšega programerja • znanje

NASLOV AVTORJA: dr. Matej Šprogar, docent, Univerza v Mariboru, Fakulteta za elektrotehniko, računalništvo in informatiko, Koroška cesta 46, 2000 Maribor, Slovenija, e-pošta: matej.sprogar@um.si.

DOI https://doi.org/10.18690/978-961-286-162-9.14 ISBN 978-961-286-163-6

Dostopno na: http://press.um.si.

1. UVOD

Ljudje smo vedno bolj le figurice, nujni statisti v igri neorgansko rastočega kapitala. Vsakodnevno obstreljevanje z reklamami nam vsiljuje nove in nove izdelke in kot civilizacija smo posledično postali dobesedno odvisni od (domnevnih) izboljšav. Dejansko stanje ocenjujejo za nas drugi, lastno mnenje je postalo nepotreben in včasih celo nevaren luksuz, zgodovinski spomin se je skrčil na minimum. Vse to vpliva tudi na razvoj v programskem svetu.

Na spletu je opazen trend pavšalnih komentarjev, kot "ta program je zastarel", "že pol leta ni bilo commitov", "tega nihče več ne uporablja" ali pa "X je nov!", "Y bo rešil vse težave!" in "novi Z je revolucionaren!"... Vse to je slaba, a nujna posledica spremenjene miselnosti celotne generacije, ki vpliva tako na stanje programske opreme kot tudi programskih jezikov.

S programiranjem se danes ubada ogromno ljudi, a komaj eno človeško življenje nazaj je bil na planetu le en pravi "moderni" programer – oče računalnika Alan Turing. Ker je število programerjev drastično narastlo, je povprečno znanje seveda upadlo. Tudi zato s(m)o izumljali nove in nove prijeme, a jih žal večinoma ne uporabljamo za boljšo kodo, ampak več kode. Dejansko znajo v vsaki naslednji generaciji programerji v povprečju manj, čeprav se zdi, da naredijo več. Je pa številčnost programerjev prinesla nesluteno širino idej in svežine in z GNU/Linuxom celo dostojno alternativo profitno usmerjenim korporacijam.

Vojna za denar pa se dejansko vrti okoli programerjev – kdor "ima" programerje, ima programe in kdor ima programe, ima uporabnike. Današnji programerji zahtevajo in celo pričakujejo vedno bolj udobne jezike in orodja tudi zaradi odseva splošne potrošniške iluzije, da je nov produkt nujno boljši od starega – da je torej novejši prevajalnik že sam po sebi garant za kvalitetnejši program... Posledično se programerji polenijo in s tem dejansko pripomorejo k povprečnemu upadu znanja in tudi širitvi slabih praks: nek problem je vedno mogoče rešiti na več načinov, a razširila se žal ne bo najboljša rešitev, ampak tista z največ reklame. Ko programer nekritično sprejme neko rešitev in jo (slepo) uporabi, jo dejansko promovira! Ampak kritičnost je mogoča samo, če imamo znanje in dostop do izvorne kode; ker pa je znanje omejeno in ker je mnogo kode licenčne in skrite, upada tudi nivo kritičnosti, kar spet zniža raven znanja...

Kapital nujno potrebuje stabilno proizvodnjo, zato od nekdaj vzpodbuja načine, ki podpirajo spremembo intelektualnega procesa programiranja v bolj obrtniško dejavnost. (Če bi to bilo mogoče, bi seveda lahko program nadomestil programerja!) Programerji v posledično vedno bolj birokratsko vodenem okolju ne delajo optimalno. Temeljni razkorak med programerji in uporabniki je skušal preseči agilni manifest iz leta 2001, ki je lepo ubesedil težave tedanjih razvojnih praks. Ampak agilne ideje so dosegle resnični preboj le na procesnem področju (Scrum), na tehnološkem (XP) pač ne. Posledica je bila razvodenitev gibanja, nastanek novih iniciativ ("Software Craftmanship" 2009) in borba za XP nasledstvo.

2. PROBLEM CIVILIZACIJE

Elektronski računalnik je verjetno najpomembnejši izum moderne dobe, ki je tudi že postal temelj obstoja civilizacije. Od skromnih začetkov do danes je doživel ogromno sprememb in izboljšav. Izjemen napredek je bil dosežen tako na strojnem kot tudi na programskem področju, kar je dodatno povečalo uporabnost in dostopnost računalniške tehnologije. Nujna posledica vsega tega je velika potreba po programerjih, ki dejansko upravljajo računalnik. Profesionalno se s programiranjem danes ukvarja več kot 22 milijonov programerjev [1], kar predstavlja enormno količino tako znanja kot neznanja. In seveda tržni potencial, kar pomeni, da so programerji tudi ciljna skupina korporacij.

Denar v borbi za tržni delež ne izbira sredstev in v današnjem svetu so polresnice ustaljena praksa, sprejemljive so postale celo laži in zavajanja. Učinkovito sredstvo prodaje je (umetna) iluzija, da je nekaj boljše, lepše... Rezultat vsega tega je neizmerna množica oglaševano "novega", ki pa je pogosto zgolj preoblečeno "staro", ali obljub, ki enostavno niso izpolnjene. (Lep primer tega je uspešno reklamiranje

Jave kot superiornega jezika [2].) Uradnim (reklamnim) virom tako ne moremo zaupati, večina ostalih informacij pa je težko preverljivih ali subjektivnih. Paradoksalno imamo v času interneta preveč informacij, ki zamegljujejo preprost dostop do kvalitetnega znanja; programiranje pa je dejavnost, ki temelji na znanju.

2.1. Moderna doba

Moorov zakon ne opisuje več pravilno razvoja vedno manjših tranzistorjev z nižjo porabo energije in višjo hitrostjo delovanja, kot je to okvirno veljalo celih 50 let. Zaostanek pri razvoju novih procesorjev zahteva nove pristope k snovanju tako strojne kot tudi programske opreme [3]. Do takrat pa smo obsojeni na praktično nepregledno mnogo "modernih" in "pošminkanih" ogrodij in jezikov...

Ampak velike revolucije v svetu programske opreme (prevajalnik, vse osnovne paradigme programskih jezikov...) so vse že zelo stare, večina iz petdesetih, šestdesetih letih prejšnjega stoletja; zadnji je bil svetovni splet skoraj 30 let nazaj. Od takrat pa vse do danes praktično ni bilo večjega kvalitetnega preskoka, samo neskončna serija nadgradenj, izboljšav, preoblek in podvajanja. Danes nimamo programskega orodja ali jezika, ki ne bi izhajal ali uporabljal principov in rešitev iz prvega obdobja računalništva!

Tudi pomembna kasnejša odkritja na področju programiranja, na primer programski vzorci [4], niso izumi, ampak empirična opažanja o lastnostih že udejanjenih programskih sistemov in arhitektur. Dejansko so najboljši programerji preteklosti uspešno uporabljali vsa ta načela, le da jih niso poimenovali. Dober programer bo nezavedno udejanil večino znanih programerskih priporočil, ker se mu drugačna koda enostavno zdi napačna. In to je mogoče le, če programer razume problem, ki ga rešuje, in če seveda obvlada tudi orodje (računalnik), ki ga pri tem uporablja.

2.2. Velika slika

In tukaj pridemo do osrednjega problema modernega časa: mladi programerji so podvrženi tolikšni količini informacij, jezikov, ogrodij, orodij, tehnik, pravil, priporočil... da ne zmorejo več videti velike slike. Posledično ne zmorejo doseči kvalitete v vseh njenih dimenzijah. Tipično trpi učinkovitost, ki je v prvi vrsti odvisna od sposobnosti programskega jezika, da je hkrati blizu strojni opremi in hkrati omogoča visoko stopnjo abstrakcije te iste strojne opreme [5]. Večina modernih jezikov tega enostavno ne omogoča, nasprotno, to smatra kot nevarnost in posledično raje reklamira iluzijo nekega "boljšega" virtualnega računalnika. Druga žrtev je ponavadi razumljivost, saj se vsi jeziki in ogrodja in knjižnice prepogosto spreminjajo, kar praktično pomeni, da se mora začetnik naučiti tudi slabe "stare" sintakse, ki je uradno že zastarela. Dodatno je večina ogrodij in celo nekaj jezikov v svoji želji, da bi skrili podrobnosti delovanja pred uporabnikom, postalo netransparentnih. In kako razumeti kodo, ki ni vidna? Nerazumljiva koda pa je začetek konca.

3. TEHNIČNI IN DRUGI RAZLOGI

Moderno zavijanje programerjev v vato, ponujanje prekomerne pomoči (razvojno okolje resda lahko občutno pomaga, ampak nekje je meja, ko ta pomoč postane škodljiva za programerjevo znanje, ga uspava) in preveč sposobnih pripomočkov (ko se vse "nekako" zgodi samo od sebe) preprečujejo, da bi se programer dejansko naučil obvladati računalnik. Vzporednico bi lahko potegnili ali z vzgojo razvajenega in posledično nesamostojnega otroka ali s pilotom, ki ne zna več pristati brez elektronske pomoči.

Stvari, ki vplivajo na sposobnosti programerjev, so vse tako prepletene, da je nemogoče izpostaviti le eno. Tukaj se bom osredotočil na poplavo prekompleksnih tehnologij, ki nujno zahtevajo specializirana znanja, ki pa jih je pravzaprav težko pridobiti zaradi poplave dezinfomacij modernega časa.

3.1. Preobilica in potrošniška mrzlica

Včasih je bil dostop do strojne opreme omejen v tej meri, da se je bil programer prisiljen močno poglobiti v program in se "pripraviti" na izvedbo. Hkrati je imel na razpolago le omejeno število orodij, ki pa jih je v osnovi zato obvladal in iz njih iztisnil maksimum.

Danes je obratno – svetovni splet je praktično neobvladljivo velika veleblagovnica, kjer so po policah razmetana programerska orodja na razpolago za takojšnji nakup, vse ovito v bleščeč reklamni papir, ki pa samo prikriva vsebino. Programski jeziki se pravzaprav prodajajo kot zelenjava, vsak dan sveža! Pa ni tako preprosto. Odločitev je namreč izrazito daljnosežna in bo izredno pomembno krojila programerjevo prihodnost in njegovo znanje. Dejansko programer ne "kupi" izdelka, ampak izdelek "ulovi" programerja!

3.2. Poplava dezinformacij

Pravzaprav je dostop do znanja omejen bolj, kot si domišljamo. Javne šole in univerze so resda vir znanja za vse, ampak vsebina, ki jo ponujajo, tudi postaja vedno bolj žrtev dobičkonosne miselnosti moderne dobe – pomembno je, kaj se "prodaja", da je všečno, novo, lepo, bleščeče... Posledično večina študentov namesto učenja skozi knjigo (dolgočasno, zastarelo, počasno) izbere raje lažjo pot spletnih mnenj in tutorialov. Spletno znanje je postalo moderni programerski učbenik, ki pa ima neskončno strani. Težava sedaj ni v dostopu, ampak v filtriranju. Za iskanje po spletu dandanes seveda poskrbi Google. Kot preprost primer si poglejmo Googlov odgovor na relativno popularno vprašanje o prihodnosti funkcijskih (FP) in objektno-usmerjenih (OO) programskih jezikov:

Poizvedba "functional vs oop" na Googlu vrne 689.000 rezultatov s prvim zadetkom na blog codenewbie.org [6], kjer je avtor B. Gathen leta 2015 predstavil in s primeri v Ruby-ju podkrepil svoje stališče, da je izbira FP ali OOP odvisna od problema, ki ga rešujemo. Kar je vsaj na prvi pogled dokaj neinformativno (in zatorej nekoristno) stališče.

Drugi zadetek prihaja iz portala Medium.com in je praktično povzetek prvega zadetka, z vsebinsko istim(!) primerom, le da brez kode. Tretji je prav tako iz portala Medium, le da sedaj avtor daje bolj specifične nasvete za izbiro med OO in FP, spotoma subtilno preferira Scalo pred Javo in šele v zaključku omeni, da se paradigmi ne izključujeta in da ju torej lahko kombiniramo, pač odvisno od potreb.

Četrti zadetek [7] je iz portala Stack Overflow (kjer je bila nit zaprta po 7 letih zaradi subjektivnosti odgovorov), ki tudi vzpodbuja razlikovanje med primernostjo FP in OO. Peti zadetek na Raganwald.com iz leta 2013 pravzaprav nasprotuje sam sebi ko trdi, da pri poslovnih aplikacijah dominirajo funkcije in ne objekti zaradi funkcijske narave samega SQL-a!?

Šele šesti zadetek [8] na Oreilly.com nas usmeri na poročilo R. Walburtona, ki je posvečeno ravno temu vprašanju. Tam že v uvodu jasno piše, da gre pri izbiri med OO in FP pravzaprav le za mnenjsko vojno in da izbira ni izključujoča.

Kaj je torej tako problematičnega v mnenju, da je izbira med OO ali FP problemsko odvisna? Odgovor sam po sebi ni napačen, problematično je pa to, kar NI povedano, ampak je bralcu posredno sugerirano. Čeprav izbira med OO in FP sploh ni potrebna (ti dve paradigmi se med seboj lahko lepo dopolnjujeta oziroma jih spreten programer odlično kombinira), bo neinformiran bralec (začetnik) vseeno zaključil, da se koncepta izključujeta. Ker zakaj bi se sicer sploh toliko člankov ubadalo z vprašanjem OO vs FP?!!!

Pomembna pa ni toliko debata OO vs FP, ampak razmislek, zakaj pride do tega, da slabi odgovori splavajo na vrh iskalnih zadetkov. Dejansko sta razloga dva: (1) Google ne razume objavljenih vsebin in jih pač indeksira po nekih sintaktičnih merilih; in (2) Google sam vpliva na svoje rezultate. Dejstvo, da je bil "pravilen" šele 6. zadetek, je pripeljalo do tega, da je Google postal ojačevalec javnega mnenja.

Prvi zadetek na Googlu namreč pomeni večjo prepoznavnost, kar spet pripomore k boljšemu SEO rezultatu... Konec koncev se stran vzpenja v indeksu ravno s številom ljudi, ki jo citirajo od zunaj, ki ji torej "verjamejo". In višje kot je stran na Googlu, bolj ljudje verjamejo, da je vsebina resnična in kvalitetna in da se ne "splača" gledati nižje po seznamu. Na vrhu Googla je pa "itak" sveta resnica, sicer bi jo že zdavnaj razkrinkali! Posledično je zatajil tudi StackOverflow, ker je večina ljudi že kupila prepričanje, ki ga je pravzaprav vsilil Google in je tako v dobri veri, da izbira prav, podprla slabši odgovor. Dejansko je prišlo do pojava samoojačevalne zanke, ko visokoležeč odgovor kotira vedno više zgolj zato, ker je že sicer visoko.

3.3. Kompleksnost

In tudi če se mlad programer znajde v tej poplavi podatkov in uspe izluščiti koristne informacije, je pred njim še vedno izziv kompleksnosti. Kompleksni problemi namreč zahtevajo izredno specifična znanja in tesno sodelovanje mnogih ekspertov, dodatno pa postaja tudi kompleksnost današnjih programerskih orodij tolikšna, da so neobvladljiva brez (spet) kompleksnih razvojnih okolij in procesov. Kar predstavlja veliko težavo za začetnika.

Ironično je, da je bila kompleksnost problematizirana že davno nazaj, pa je od takrat samo še rastla. Rešitve, ki so se nalašč reklamirale kot manj kompleksne, pa so s svojim nastankom postale del problema množičnosti; in danes so vse, ki so preživele, močno prerastle svoje začetne okvirje ali kako drugače poteptale lastne korenine (ker vsi dobri programski jeziki prej ali slej prerastejo svoje prvotne okvirje [9]). Kompleksnost je namreč nujna slaba lastnost modernega programja, saj produkt, ki nečesa ne omogoča, ne "preživi" spletne kritike, ki usmerja javno mnenje. In spletna kritika je izredno neinformirana in pristranska... Preobilica funkcionalnosti programerskih orodij povzroči, da povprečen programer uporablja in resnično pozna zgolj del(ček) sposobnosti svojih orodij.

3.4. Produktivnost in specializacija

Prevelika količina (ne)znanja, ki ga mora obvladati mlad programer, preprečuje, da bi lahko v svojih najbolj produktivnih letih pisal res kvalitetno kodo. Ker pa mora (žal) producirati čim več v čim manj časa, se zateče k jezikom in orodjem, ki omogočajo večjo produktivnost. Oziroma ga v to prisilijo, priučijo...

Večja produktivnost (z manj kode) je mogoča samo, če uporabimo ali specializiran ali kako drugače prilagojen programski jezik, ogrodje... Skladno z No Free Lunch teoremom [10] pa moramo nekje plačati ceno za to "ugodnost" in to je ponavadi ali pri učinkovitosti ali funkcionalnosti kode. V večini primerov to ni težava ali pa programerji rešujejo novo pomanjkljivost kako drugače (več procesorjev, pomnilnika...). Šele če to ni mogoče ali je vsaj zelo težko, opazimo, da smo zašli v ozko in slepo ulico in da ravno zaradi specializacije nimamo širokega znanja, ki je sicer nujno potrebno za rešitve izven tradicionalnih, specializiranih okvirjev.

3.5. Ko orodje postane cilj

Žal se tudi na vseh ravneh pojavlja fenomen, ko sintaksa pričenja nadvladovati semantiko, ko postaja orodje pomembnejše od produkta, proces pomembnejši od izdelka: miselnost, da če "uporabljamo" Scrum, Javo, Maven, jMock, git... ne samo, da bomo končali pravočasno, naredili bomo tudi boljši izdelek kot sicer! Problem so pravzaprav vse dobronamerne ideje, ki se osredotočajo izključno na proces in zapostavljajo programiranje. Nobena od prej naštetih ali drugih stvari ni slaba sama po sebi, dejansko so izredno koristne. Problematično pa je neskočno kombiniranje in vpeljava novih in novih magičnih strategij in s tem delanje megle okoli dejanskega problema. Dogaja se, da naše (ne)znanje orodij usmerja naš način razmišljanja, načrtovanja rešitev ter s tem praktično definira rešitev [11]. Začetniki pogosto uporabljajo neko tehniko zgolj zato, ker je na voljo, in ne zato, ker bi bila optimalna za njihov problem.

4. USODA GENERACIJE

Glede na vse našteto ni čudno, da je veliko mladih programerjev še vedno prepričanih, da obstajajo boljši in slabši jeziki, da so primerjave smiselne in njihovi rezultati pravilni in relevantni. Ampak šele

izkušnje nas naučijo, da je za vsak problem potrebno svoje orodje in da idealen programski jezik ne obstaja; nasprotno, dober programer pač pozna več jezikov, predvsem pa računalnik.

Ampak rastoča množica "novincev" dejansko ustvarja in usmerja javno mnenje in s tem prihodnost branže (v smislu medtem ko pametni razglabljajo, nori zavzamejo trdnjavo). Industrija namreč vidi tržni potencial v veliki količini neizoblikovanih programerjev, ki zaradi bolezni časa pričakujejo, da se bo delo nekako naredilo samo in so v toliko "lačni" novih magičnih rešitev. Posledično industrija in programerji sami lansirajo nove in nove stvari in s tem samo še poslabšajo že tako kaotične razmere.

Jutrišnja generacija pa bo morala živeti in vzdrževati današnjo kodo, kodo, ki prihaja z vedno večjim tehničnim dolgom. Dejansko prihajamo do *paradoksa slabšega programerja*: ko bi naj težave preteklosti v povprečju reševal programer, ki je slabši od programerja, ki je težave sploh povzročil!

4.1. Inflacija kode

Podobno, kot se tehnični dolg pogosto predstavlja v obliki finančnega dolga in obresti, ki jih je potrebno prej ali slej plačati, lahko preobsežno in prekompleksno kodo magičnih jezikov in ogrodij smatramo kot preveč natiskanega denarja, kar povzroči inflacijo in potrebo po še več kode...

Rešitev iz te neskončne zanke je v širitvi znanja, ki edino omogoča nastanek kvalitetne kode. Znanje mora biti podano v nezamegljeni obliki, to je v obliki kvalitetne delujoče kode. Kar pomeni, da je dostop do dobre delujoče kode edini pravi način učenja. Utopija, da bo vsa koda kdaj na voljo v resnično prosti obliki (Free Source), je danes bliže kot kadarkoli, dejansko smo že na točki, ko vse velike korporacije objavljajo svojo kodo pod bolj omejenimi pogoji odprte kode (Open Source).

4.2. Prihodnost je v enostavnosti

Upanje, da bo prihodnost bolj svetla, kot se mogoče zdi iz zgoraj povedanega, izhaja iz tega, da na dolgi rok slabe rešitve vedno propadejo, dobre pa še pridobijo na kvaliteti. Kriterij za to selekcijo, ki se je znova in znova potrdil v praksi, je *enostavnost*. Dokler bo človek glavni vir programske kode, bo le z obvladljivimi orodji in kodo mogoče producirati kvalitetne rešitve z minimalnim tehničnim dolgom. Obvladljivost pa je sorazmezna z razumljivostjo in kompaktnostjo kode, skratka z njeno enostavnostjo.

Prihodnost ni v uporabi novih in novij orodij in jezikov, ki naj bi rešili stare težave, ampak v poenostavitvi obstoječih programerskih orodij, ker je trajnostno gledano to edina rešitev, ki reši prej omenjeni paradoks slabšega programerja. Tudi žal ne moremo zaupati vsemu, kar je reklamirano kot enostavno, ker pravi test je edinole test časa.

5. LITERATURA

[1] https://evansdata.com/reports/viewRelease.php?reportID=9,
Global Developer Population and Demographic Study 2017 Vol. 2, obiskano 14.5.2018.

[2] GOSLING James, MCGILTON Henry "The Java Language Environment: A White Paper", Sun Microsystems Computer Company, 1995.

[3] rebootingcomputing.ieee.org, IEEE Rebooting Computing Initiative, obiskano 10.5.2018.

[4] GAMMA Erich, HELM Richard, JOHNSON Ralph, and VLISSIDES John "Design Patterns: Elements of Reusable Object-Oriented Software", Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1995.

[5] STEPANOV Alexander "Notes on Programming", 2006.

[6] www.codenewbie.org/blogs/object-oriented-programming-vs-functional-programming,
Object Oriented Programming vs. Functional Programming, obiskano 7.5.2018.

[7] stackoverflow.com/questions/2078978/functional-programming-vs-object-oriented-programming,
Functional programming vs Object Oriented programming [closed], obiskano 7.5.2018.

[8] www.oreilly.com/programming/free/object-oriented-vs-functional-programming.csp, WARBURTON Richard "Object-Oriented vs. Functional Programming - Bridging the Divide Between Opposing Paradigms", O'Reilly ebook, dostopano 7.5.2018.

[9] http://www.stroustrup.com/bs_faq.html#understand, STRUSTRUP Bjarne "Did you really not understand what you were doing?", FAQ, obiskano 10.5.2018.

[10] WOLPERT David H., MACREADY William G. "No Free Lunch Theorems for Optimization", IEEE Transactions on Evolutionary Computation 1, 67, 1997

[11] BROOKS Rodney A. "Intelligence without reason" In *Proceedings of the 12th international joint conference on Artificial intelligence - Volume 1*(IJCAI'91), John Mylopoulos and Ray Reiter (Eds.), Vol. 1. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 569-595, 1991.

Univerzitetna založba Univerze v Mariboru

DRUŽINSKO CENTRIČNA ZASNOVA SPLETNE APLIKACIJE MYFAMILY

VID ČERMELJ, JURE TRILAR, VERONIKA ZAVRATNIK IN EMILIJA STOJMENOVA DUH

Povzetek: V prispevku je predstavljen razvoj in testiranje interaktivnega prototipa aplikacije MyFamily, ki je bila razvita v projektu RRP3 Aktivno življenje, dobro počutje, ki se izvaja v okviru programa EkoSmart. Namen aplikacije je spodbujanje aktivnega življenja med družinskimi člani. Namenjena je poenostavljeni komunikaciji med družinskimi člani, obveščanju o aktivnostih in doseganju skupnih ciljev. Predstavili smo tehnologijo in arhitekturo aplikacije ter postopek uporabniškega testiranja grafičnega vmesnika in funkcionalnosti aplikacije. V okviru razvoja smo izvedli več anket in intervjujev, s katerimi smo poskušali ugotoviti, kako družinske člane čim bolj motivirati za uporabo aplikacije. Za testiranje uporabnosti in uporabniške izkušnje aplikacije smo izvedli uporabniško testiranje. Rezultate testiranja smo podrobno analizirali in jih predstavili v prispevku.

Ključne besede: • aktivno življenje • uporabniško testiranje • razvoj spletne aplikacije • interaktivni prototip • uporabniško usmerjen razvoj

NASLOV AVTORJEV: Vid Čermelj, Univerza v Ljubljani, Fakulteta za elektrotehniko, Tržaška cesta 25, 1000 Ljubljana, Slovenija, e-pošta: vid.cermelj@ltfe.org. univ. dipl. soc. Jure Trilar, Univerza v Ljubljani, Fakulteta za elektrotehniko, Tržaška cesta 25, 1000 Ljubljana, Slovenija, e-pošta: jure.trilar@ltfe.org. mag. etn. in kult. antrop. Veronika Zavratnik, Univerza v Ljubljani, Fakulteta za elektrotehniko, Tržaška cesta 25, 1000 Ljubljana, Slovenija, e-pošta: veronika.zavratnik@ltfe.org. doc. dr. Emilija Stojmenova Duh, Univerza v Ljubljani, Fakulteta za elektrotehniko, Tržaška cesta 25, 1000 Ljubljana, Slovenija, e-pošta: emilija.stojmenova@ltfe.org.

DOI https://doi.org/10.18690/978-961-286-162-9.15 ISBN 978-961-286-163-6

Dostopno na: http://press.um.si.

1. UVOD

Tehnologija je v današnjem času vse bolj prisotna in predstavlja vedno večji del našega vsakdana. Kamor koli gremo, se ne moremo izogniti uporabi različnih tehnologij. Smiselno je združevanje funkcionalnosti, ki jih posamezne tehnologije nudijo, z namenom, da nam postanejo še bolj v pomoč. Na evropskem programu EkoSmart razvijamo rešitve, ki bodo to omogočile. Univerza v Ljubljani s številnimi slovenskimi podjetji in ustanovami razvija sistem pametnega mesta z vsemi podpornimi mehanizmi, ki so potrebni za učinkovito optimizirano in postopno integracijo posameznih področij v enovit in povezan sistem vrednostnih verig.

Program se osredotoča na tri ključne domene pametnega mesta. To so: zdravje, aktivno življenje in mobilnost. Program EkoSmart sestavlja šest projektov, ki vsak na svoj način prispevajo k uresničevanju vizije programa. Aplikacija MyFamily spada v področje aktivnega življenja in je bila razvita v okviru projekta RRP3[24] Aktivno življenje, dobro počutje.

V prispevku so predstavljeni cilji in potek projekta RRP3 po posameznih sklopih ter pridobljeni rezultati. Podrobneje je predstavljen razvoj interaktivnega prototipa aplikacije MyFamily, ki je bila razvita v sklopu projekta. Aplikacijo smo testirali z uporabniki, ki so po koncu uporabniškega testiranja rešili tri vprašalnike.

2. IDEJA IN CILJI APLIKACIJE MYFAMILY

Obstoječa rešitev 24aLife, ki usmerja uporabnike k zdravemu življenjskemu slogu, je namenjena štirim ciljnim skupinam: odraslim rekreativcem, zaposlenim v podjetjih, fitnes centrom in starostnikom. V projektu RRP3 smo prepoznali potrebo po nadgradnji te aplikacije z rešitvami za aktivno življenje in dobro počutje družine kot modela trajne in intimno povezane skupnosti, v obliki prototipne rešitve MyFamily.

Aplikacija MyFamily bo v ekosistemu EkoSmart omogočala večjo povezanost družine, boljšo komunikacijo med družinskimi člani ter spodbujala bolj dejavno in kakovostno preživljanje skupnega družinskega časa.

Ključni cilji projekta so:

- Identificirati potrebe uporabnikov (družinskih članov).
- Opredeliti koncept delovanja aktivne in zdrave družine v pametnem mestu.
- Identificirati mehanizme za spodbujanje aktivnega in zdravega življenja družin v pametnem mestu.
- Izdelati načrt funkcionalnosti za delovanje aktivne in zdrave družine v pametnem mestu.
- Razviti prototipe storitev in rešitev za preizkušanje v laboratorijskem okolju.

3. RAZVOJNE FAZE APLIKACIJE MYFAMILY

Projekt Aktivno življenje, dobro počutje se je začel avgusta 2016. Za lažje doseganje zastavljenih ciljev je bilo delo na projektu razdeljeno na pet delovnih paketov razloženih v tem poglavju. Paketi so bili izpolnjeni po vrsti. Trenutno v projektu prehajamo na zadnji delovni paket, kjer bomo interaktivni prototip aplikacije MyFamily povezali v integracijsko platformo ekosistema EkoSmart.

3.1. Analiza uporabniških navad v družinah

Pri načrtovanju in razvoju novih izdelkov se podjetja prepogosto osredotočajo na poslovne cilje, napredne lastnosti in funkcionalnosti ter tehnološke zmogljivosti programske in strojne opreme. Pri načrtovanju pristopov pa pozabljajo na ključen del načrtovanja, saj spregledajo končnega uporabnika.

[2] RRP3 – Raziskovalno-razvojni projekt številka 3 z imenom Aktivno življenje, dobro počutje

V projektu smo rešitve in storitve reševali s pristopom usmerjenim k ljudem[25]. V vsaki fazi njegovega razvoja in izdelave smo veliko pozornosti namenili potrebam, željam, zahtevam in omejitvam uporabnikov izdelka. Namen prvega delovnega paketa je bilo razumevanje načina življenja, potreb in želja končnih uporabnikov. Za lažjo izvedbo raziskave je bila razvita spletna anketa z 32 glavnimi vprašanji ter s podvprašanji, ki obsegajo tematske sklope: demografija, oblika gospodinjstva, delitev dela, skupno preživljanje časa, uporaba sodobnih tehnologij, zdravje in skrb za družinske člane, organizacija časa ter skupni cilji. Anketa je bila izvedena spomladi 2017, sodelovalo je 101 anketirancev, večino so predstavljali starši (71%), otroci le manjši delež (27%), stari starši pa v anketi skoraj niso bili zajeti (2%).

S pomočjo rezultatov ankete smo bolje razumeli mentalne modele uporabnikov. Pomagali so uskladiti konceptualni model za načrtovanje sistema z mentalnim modelom končnih uporabnikov.

3.2. Opredelitev konceptualnega modela za načrtovanje sistema

V drugem delovnem paketu smo preučevali delovanje članov znotraj družine, njihove medsebojne odnose in povezljivost za aktivno življenje. Raziskovali smo razmerje med družino in pametnim mestom in s tem pridobili boljše razumevanje vloge družine in potreb njenih članov.

Ugotovili smo, da se družine spopadajo s težavami, ki bi jih inovativne IKT rešitve lahko močno olajšale in posledično razbremenile družine. S primernimi aplikacijami lahko družinam zagotovimo boljšo organiziranost, povezanost in informiranost, kar vodi do manjšega stresa in več prostega časa za skupno druženje. Poleg tega lahko z aplikacijami spodbujamo zdrav življenjski slog na vseh nivojih.

3.3. Mehanizmi za vzpodbujanje aktivnega zdravega življenja v družini

Pomanjkljivost velikega dela IKT rešitev, ki vzpodbujajo zdrav življeni slog, je pomanjkanje mehanizmov za dolgoročno in vzdržno motiviranje uporabnikov. Družina je s socio-psihološkega stališča kompleksen subjekt za motiviranje. Posebej zahtevno je oblikovanje pristopov, ki družinske člane spodbujajo k zdravemu življenjskemu slogu. Že uveljavljeni motivacijski mehanizmi so: določanje in spremljanje doseganja skupnih ciljev, skupnih izzivov, nagrajevanja ter možnost spodbujanja s t.i. resnimi igrami.

V tem delovnem paketu smo se ukvarjali z iskanjem primernih motivacijskih mehanizmov. V analizi najnovejših raziskav in konkurenčnih motivacijskih modelov na trgu smo ugotovili, katere so bistvene lastnosti, ki jih imajo uspešni mehanizmi. Ugotovili smo, da je notranja motivacija pomembnejša od zunanje. Zunanja je lahko večkrat kratkoročno uspešna, vendar pa za razvoj dolgotrajnejših navad potrebujemo notranjo. Eden izmed osnovnih mehanizmov zunanje motivacije je nagrajevanje. V primeru da ljudje spremenijo svoje vedenje na podlagi nagrajevanja, po zaključku le tega to navado opustijo.

Pomemben motivacijski mehanizem je pripadnost. To je osnovna človekova potreba, ki lahko vzdržuje motivacijo. Vključuje povezovanje z drugimi glede na skupni cilj ali skupino, ki ji pripada. Če posamezniki čutijo, da s svojim sodelovanjem pomagajo, je to lahko ključnega pomena za motivacijo. Posamezni člani skupine lahko druge spodbujajo, jim pomagajo ali postanejo njihovi mentorji.

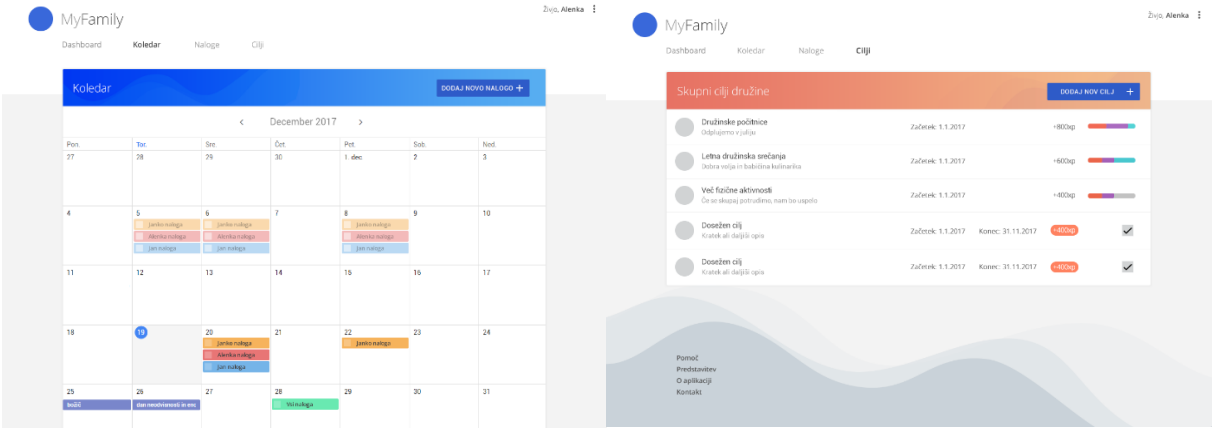
V aplikaciji je glavni motivacijski mehanizem pripadnost družini. Člani družine lahko dodajajo naloge in jih skupaj izpolnjujejo. Za izpolnjeno nalogo dobijo določeno število točk, ki so del sistema nagrajevanja.

[3] Pristop usmerjen k ljudem – angl. "human-centered design"

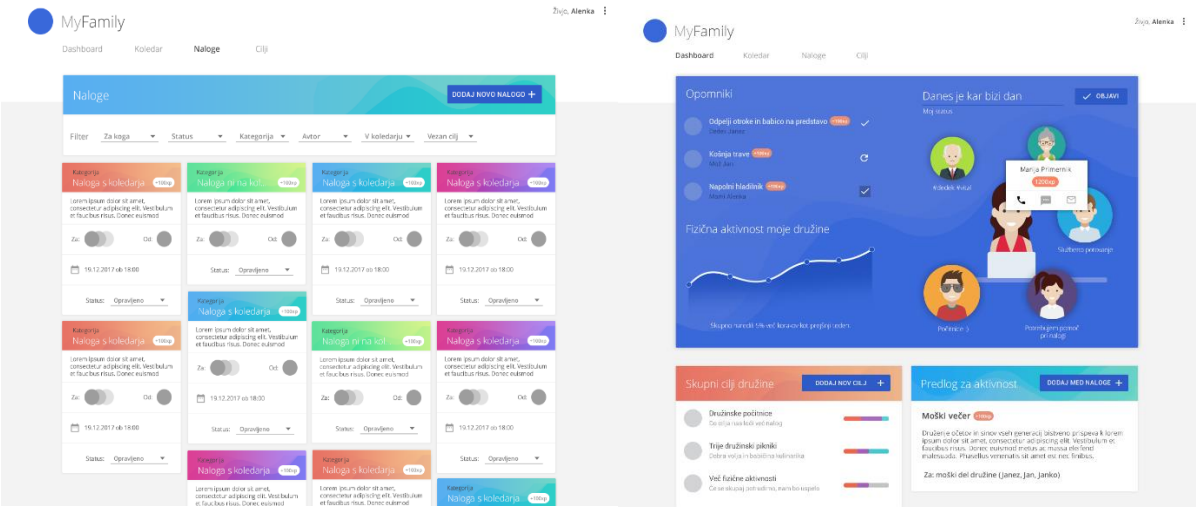
3.4. Načrtovanje funkcionalnosti

V prejšnjih delovnih paketih smo raziskovali različna področja, s pomočjo katerih smo dobili informacije, ki smo jih uporabili pri načrtovanju funkcionalnosti aplikacije. V četrtem delovnem paketu smo najprej pripravili specifikacije posameznih funkcionalnih modulov ter prioritetno listo za implementacijo, nato pa smo določili tudi specifikacije posameznih modulov, ki še niso bili definirani v okviru drugih področnih rešitev.

Specifikacije smo začeli testirati z zasloni. Na sliki 1 in sliki 2 so osnovni štirje zasloni aplikacije. Opazovali smo prijaznost do uporabnika in uporabniško izkušnjo. Ugotovitve smo upoštevali v implementaciji interaktivnega prototipa.



Slika 1. Zaslon s koledarjem (levo). Zaslon s cilji (desno).



Slika 2. Zaslon z nalogami (levo). Osnovni zaslon (desno).

3.5. Razvoj storitev in rešitev

Zadnji delovni paket bo namenjen integraciji s platformo EkoSmart z že obstoječo aplikacijo 24aLife. Izbrane funkcionalne module aplikacije MyFamily bomo najprej povezali z rešitvijo 24aLife. Nato bomo obe aplikaciji povezali s platformo EkoSmart. V prejšnjem delovnem paketu smo ta povezovanja nakazali v prototipih, v tem pa jih bomo tudi izvedli. Po implementaciji povezav bo prototip pripravljen na testiranje, ki ga bomo najprej izvedli v laboratorijskem okolju, nato pa še v realnem operativnem okolju pri posameznih družinah.

4. RAZVOJ INTERAKTIVNEGA PROTOTIPA APLIKACIJE MYFAMILY

V tem poglavju je predstavljen razvoj interaktivnega prototipa. Predstavljena je arhitektura aplikacije, uporabljene tehnologije in izkušnje z njimi ter grafični vmesnik aplikacije.

4.1. Arhitektura in uporabljene tehnologije

Aplikacijo smo razvijali s tehnologijami MongoDB, Express in Node.js, ki jih najdemo v MEAN svežnju[26]. Namesto Angular ogrodja[27] smo uporabili jezik za avtomatsko generiranje predlog[28] EJS [3]. Za uporabo EJS smo se odločili, ker je bolj preprost za impementacijo kot Angular ter ponuja vse funkcionalnosti, ki jih potrebujemo v naši aplikaciji.

4.1.1. Zaledni del

Na strežniškem delu smo uporabili tehnologijo Node.js z ogrodjem[4] Express. Aplikacijo smo zasnovali tako, da jo je preprosto vzpostaviti tudi v oblaku. Med uporabniškim testiranjem smo aplikacijo prenesli na Heroku [4], podatkovno bazo aplikacije pa smo vzpostavili pri spletnem ponudniku mLab [5].

4.1.2. Podatkovna baza

Za uporabo mongoDB smo se odločili, ker je sestavljena iz dokumentov, ki jih je preprosto prejemati in pošiljati. V ekosistemu EkoSmart se bodo aplikacije pogovarjale med seboj, zato je pomembna hitra in preprosta komunikacija. To najhitreje dosežemo z dokumenti.

Glavna težava, ki smo jo imeli pri načrtovanju baze, je bila shranjevanje podatkov o družini. Družino lahko v bazo zapisujemo na dva načina, statično ali dinamično. V dinamičnem načinu se družina sestavlja tako, da je uporabnik vedno v središču. To pomeni, da kot uporabnik lahko dodamo/odstranimo člane družine. Družina se gradi dinamično glede na uporabnika. Ta način ne omogoča, da bi imeli več različnih družin.

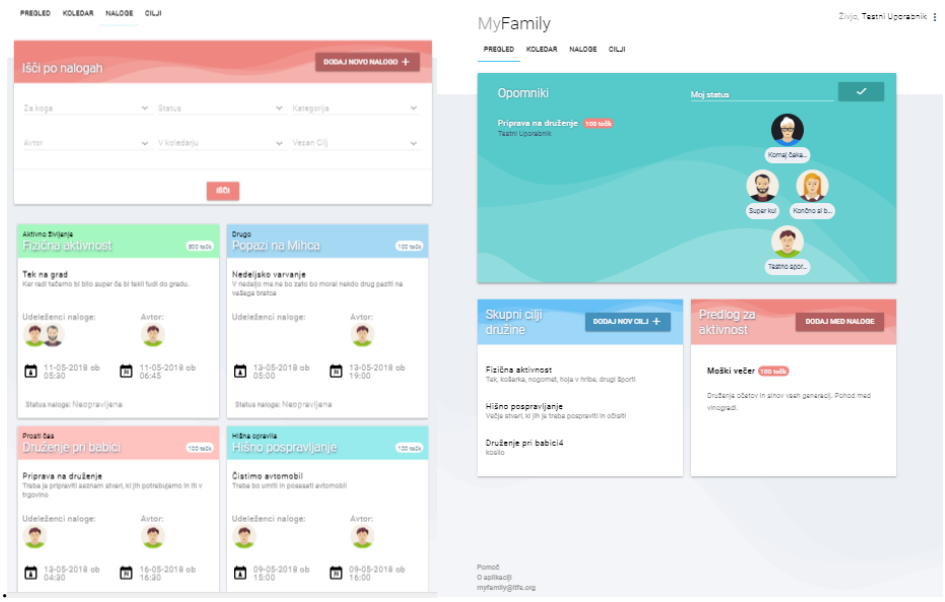
Statičen način ima v središču družino. Vsi uporabniki lahko dodajajo nove člane v družino in imajo enake pravice. Ta način omogoča uporabniku članstvo v več družinah hkrati.

Dinamičen način nam omogoča gradnjo socialnega omrežja, vendar ni primeren za našo aplikacijo. Naloge in cilji, ki jih lahko člani družine dodajajo so vidni vsem članom, kar je v primeru dinamičnega shranjevanja slabo, saj lahko uporabniki, ki se med seboj ne poznajo, vidijo medsebojne naloge. Zato smo se odločili za statičen način shranjevanja. Ob registraciji se uporabniku ustvari nova družina, s povabilom se lahko pridruži drugim družinam oziroma lahko druge uporabnike povabi v svojo družino.

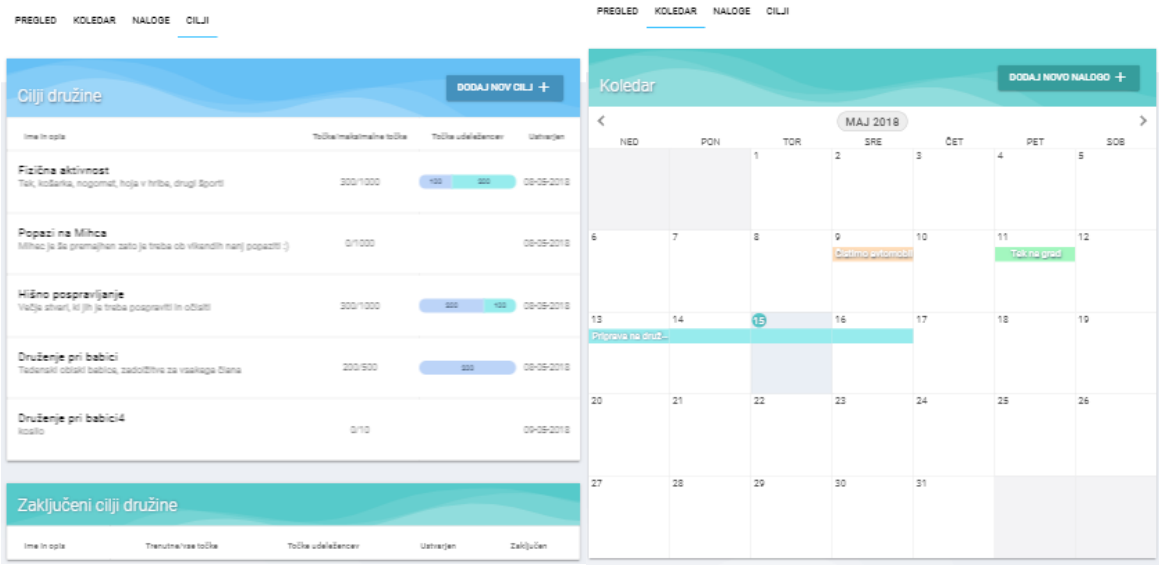
4.1.3. Uporabniški del

Uporabniški del aplikacije smo zasnovali na podlagi ugotovitev iz raziskav, ki smo jih izvedli pred začetkom razvijanja interaktivnega prototipa. Aplikacijo smo oblikovali v skladu z Google Material Design standardi [6]. Za lažji in hitrejši razvoj smo uporabili njihovo implementacijo tega standarda imenovano Material Design Lite [7]. Uporabili smo jezik za označevanje nadbesedila (HTML[29]) in kaskadne stilske predloge (CSS[30]). Za dinamičnost strani smo poskrbeli z uporabo EJS in jQuery [8]. Elemente uporabljene v slikah zaslonov smo v prototipu prilagodili standardu Material Designa. Za potrebe novih funkcionalnosti smo dodali nove elemente, ki jih ni bilo v zaslonih.

[26] Sveženj – angl. "stack"
[27] Ogrodje – angl. "framework"
[28] Jezik za avtomatsko generiranje predlog – angl. "templating engine"
[29] HTML – Hyper Text Markup Language
[30] CSS – Cascading Style Sheets



Slika 3. Stran z nalogami (levo). Uvodna stran (desno).



Slika 4. Stran z cilji (levo). Stran z koledarjem (desno).

Na levi strani slike 3 je stran z nalogami. Izris in iskanje po nalogah je zelo podobno tistemu, ki je bilo predstavljeno na zaslonu nalog. Na desni strani slike je prikazana stran s pregledom. Za takšno poimenovanje smo se odločili, ker se je ime »Dashboard« zdelo testirancem neustrezno, saj je ta beseda angleška v sicer slovenski aplikaciji.

Na levi strani slike 4 je prikazana stran z cilji. Stran je podobna tisti, ki je bila predstavljena v zaslonih. Spremenjen je prikaz ciljev. Ti so razdeljeni na cilje, ki jih je še potrebno opraviti ter na že opravljene cilje. Na desni strani je prikazana stran z koledarjem. Stran je vizualno malo drugačna, funkcionalnosti pa so enake kot so bile zastavljene v zaslonih.

5. UPORABNIŠKE MERITVE

Pri testiranju uporabniške izkušnje in uporabnosti aplikacije smo uporabili različne meritve, ki so se nanašale na uspešnost uporabnika, merjene glede na specifične cilje uspešnosti, ki so potrebni za

izpolnjevanje zahtev uporabnikov. Uporabili smo stopnjo uspešnosti zaključka scenarija, zapiske pogovorov, stopnje napak in subjektivno oceno. Zbirali smo tudi podatke o času, potrebnem za opravljanje scenarijev, ki so vključevali subjektivno vrednotenje.

5.1. Zaključek scenarija

Vsak scenarij je od udeleženca zahteval, da izpolni tipično nalogo. Scenarij je bil zaključen, ko je udeleženec pokazal/navedel, da je bil cilj scenarija dosežen (uspešno ali neuspešno) ali če je udeleženec zahteval in prejel zadostne napotke, ki opravičujejo točkovanje scenarija kot kritično napako.

5.2. Kritične in nekritične napake

Kritične napake so odstopanja od cilja scenarija ob njegovem zaključku. Poročanje o napačni vrednosti podatkov zaradi napačnega toka akcij udeleženca je kritična napaka. Udeleženci se lahko zavedajo ali pa ne, da je cilj naloge nepravilen ali nepopoln.

Nekritične napake so napake, ki jih udeleženec popravi/obnovi, ali napake, ki, če jih udeleženec ne zazna, ne povzročajo težav pri obdelavi ali nepričakovanih rezultatov. Čeprav lahko nekritične napake ostanejo udeležencu skrite, pa so v primeru, da jih udeleženec odkrije, zanj ponavadi neprijetne.

5.3. Subjektivne ocene

Subjektivne ocene glede enostavnosti uporabe in zadovoljstva smo zbirali s pomočjo vprašalnikov ter prek polstrukturiranih intervjujev. Vprašalniki so vsebovali nestrukturirane odgovore in ocenjevalne lestvice.

5.3.1. NASA TLX

Orodje NASA-TLX[31][9] smo uporabili za ocenjevanje delovne obremenjenosti ter s tem ocenili učinkovitost naloge. Vprašalnik se osredotoča na mentalno zahtevnost, fizično zahtevnost, časovno zahtevnost, trud, izvedbo in nivo frustracije.

5.3.2. UEQ

Vprašalnik UEQ[32][10] smo uporabili neposredno po uporabi aplikacije, preden udeleženec govori z drugimi udeleženci, saj je njegov namen ujeti takojšnji vtis. Gre za orodje, ki ocenjuje uporabniško izkušnjo pri čemer se osredotoča na privlačnost, jasnost, zanesljivost, spodbudo in novosti aplikacije.

5.3.3. SUS

S pomočjo vprašalnika SUS[33][11] smo preverjali uporabnost aplikacije. Uporabnost je pokrivala področja učinkovitosti, zmogljivosti in zadovoljstva uporabnikov.

5.4. Izvedba uporabniškega testiranja

Uporabniško testiranje smo izvedli v prostorih Fakultete za elektrotehniko Univerze v Ljubljani, od koder smo pridobili tudi vse udeležence. Na naše vabilo se je prostovoljno odzvalo dvanajst zaposlenih na fakulteti.

Pred začetkom testiranja je vsak udeleženec rešil vprašalnik v katerem so bila vprašanja o rabi računalnika in pametnega telefona ter vprašanja o rabi aplikacij in pametnih pripomočkov, ki so povezani s skrbjo za zdravje. Pri testiranju so udeleženci uporabljali prenosni računalnik in miško, trajalo pa je med 30 in 60 minut. Testiranje smo izvedli v laboratorijskih razmerah. Celoten intervju je

[31] NASA-TLX – NASA Task Load Index
[32] UEQ – User Experience Questionnaire
[33] SUS – System Usability Scale

bil pripravljen vnaprej in pri vseh udeležencih enak. Ena oseba je vodila testiranje, medtem ko je druga oseba zapisovala komentarje udeležencev. Udeležence smo vodili prek scenarijev, s katerimi smo preverjali različne funkcionalnosti in izgled aplikacije. Med reševanjem nalog so udeleženci glasno komentirali njihova dejanja in utemeljili morebitne težave. Po uspešnem oziroma neuspešnem zaključku vseh devetih scenarijev smo uporabnike prosili, če lahko rešijo še tri vprašalnike (SUS, UEQ in NASA-TLX), s katerimi smo merili uporabnost aplikacije, uporabniško izkušnjo in delovno obremenjenost pri uporabi aplikacije.

5.5. Analiza rezultatov

5.5.1. Demografija

Pri uporabniškem testiranju je sodelovalo dvanajst udeležencev, sedem moških in pet žensk. Enajst udeležencev je imelo vsaj univerzitetno izobrazbo, en udeleženec pa je imel srednješolsko izobrazbo. Vsi so bili zaposleni na Fakulteti za elektrotehniko. Najmlajši udeleženec je imel 28, najstarejši pa 53 let. Povprečna starost udeležencev je bila 37,42 let.

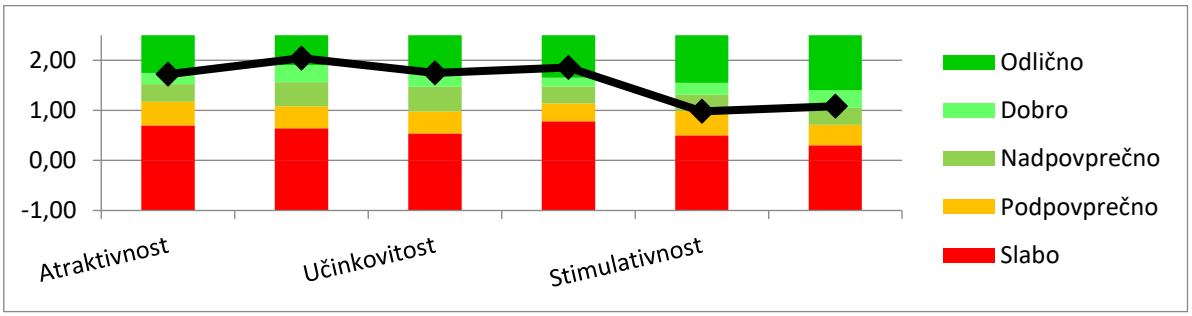
Vsi sodelujoči so imeli lasten računalnik, ki so ga večinoma uporabljali več kot osem ur na dan. Vsi so imeli v lasti tudi pametni telefon, ki ga je večina uporabljala med eno in dvema urama na dan. Računalnik so najpogosteje uporabljali za delo, organizacijo, branje novic ter branje in pisanje elektronskih sporočil. Pametni telefon so najpogosteje uporabljali za klice in sporočila sms, branje in pisanje elektronskih sporočil, organizacijo in za uporabo socialnih omrežij. Nekaj več kot polovica udeležencev je uporabljala aplikacije, ki so povezane s skrbjo za zdravje. Trije redko, dva pogosto in dva izredno pogosto. Polovica udeležencev je uporabljala pametne pripomočke, dva redko in štirje izredno pogosto.

5.5.2. Rezultati UEQ

Vprašalnik UEQ je najdaljši izmed vseh treh, saj vsebuje 26 vprašanj. Zaradi velikega števila vprašanj nam poda podrobnejšo oceno naše aplikacije. Z njim ugotavljamo atraktivnost, preglednost, učinkovitost, vodljivost, stimulativnost in originalnost aplikacije. Na sliki 3 so rezultati vprašalnika sešteti po kategorijah. Različne barve označujejo povprečne vrednosti pri drugih aplikacijah. Aplikacija MyFamily v kategorijah Preglednost in Vodljivost dosega najvišjo možno oceno, kategorije Atraktivnost, Učinkovitost in Originalnost so ocenjene kot dobre, Stimulativnost pa je ocenjena slabše od povprečne aplikacije. Na desni strani slike 4 so rezultati in standardna deviacija za posamezne kategorije.

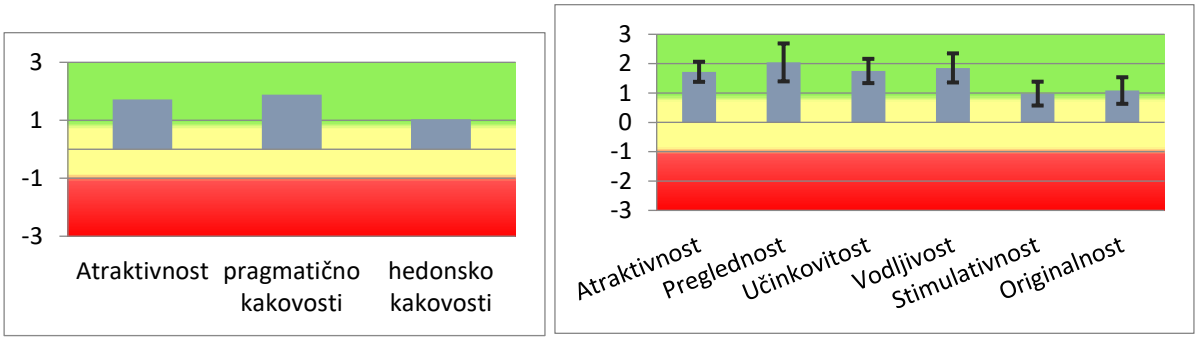
Rezultati so bili zelo nekonsistentni v kategorijah Učinkovitost (Alfa 0,28) in Originalnost (Alfa 0,58), zato teh rezulatov nismo preveč upoštevali.

Najbolj zaskrbljujoč rezultat je ocena v kategoriji Stimulativnost, saj je slabši od stimulativnosti povprečne aplikacije. V procesu razvoja aplikacije smo to področje podrobno raziskali, vendar bomo morali rezultate še izboljšati. Potrebno bo še enkrat razmisliti, kako bi lahko uporabnika bolj učinkovito motivirali k uporabi aplikacije.



Slika 5. Rezultati vprašalnika UEQ v primerjavi z povprečnimi rezultati aplikacij

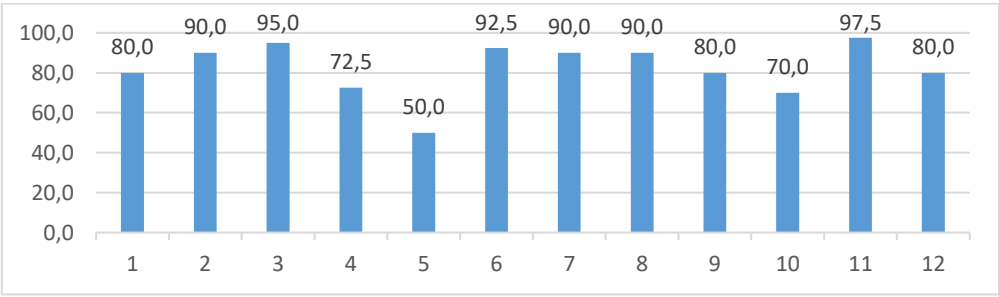
Na sliki 6 so rezultati, ki jih je aplikacija dosegla v atraktivnosti, pragmatični kakovosti in hedonski kakovosti. V pragmatično kakovost spadajo kategorije Preglednost, Učinkovitost in Vodljivost. Iz rezultatov je razvidno, da je aplikacija pregledna. Uporabniki so razumeli ter pravilno uporabljali njene funkcionalnosti. V hedonsko kakovost uvrščamo Stimulativnost, Originalnost in Atraktivnost. Uporabniki so slabše ocenili stimulativnost aplikacije. Ta kategorija je ena izmed pomembnejših v naši aplikaciji, zato bo potrebno preizkušati še druge načine stimulativnosti in poiskati najboljšega. Aplikacija se je uporabnikom zdela atraktivna, kar pomeni, da so med uporabo doživljali pozitivna čustva.



Slika 6. Pragmatična in hedonska kakovost (levo). Skale po posameznih kategorijah (desno).

5.5.3. *SUS*

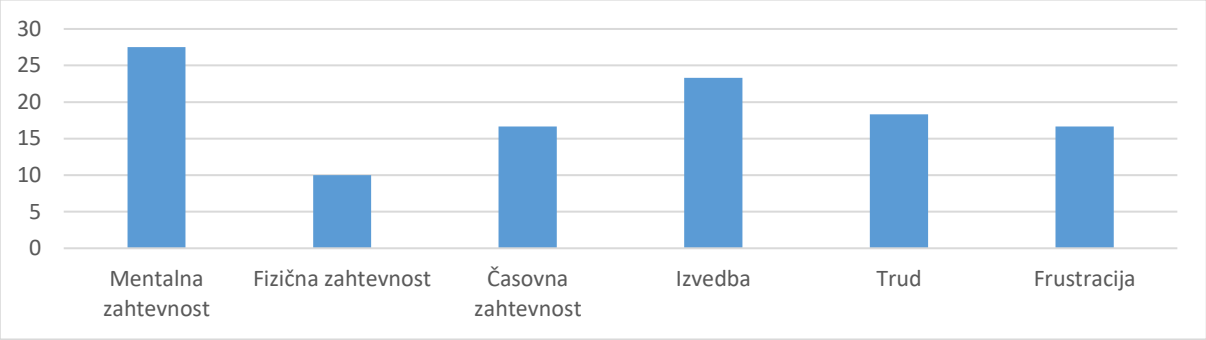
Na sliki 7 so rezultati vprašalnika SUS predstavljeni po točkah, ki so jih dosegli udeleženci. Ti so presenetljivi zaradi zelo dobrih rezultatov že pri prvem testiranju z uporabniki. Samo en udeleženec od 12 je bil z aplikacijo nezadovoljen (50 točk), ostali pa so jo ocenili kot zelo dobro, o čemer priča tudi povprečni rezultat 82.3 točk. Rezultat nad 85 točk pomeni, da se je testirancem zdela aplikacija odlična in da je ni potrebno popravljati. Rezultat, ki smo ga dosegli, nam torej pove, da je naša aplikacija zelo dobra, vendar jo lahko še izboljšamo.



Slika 7. Rezultati SUS vprašalnika

5.5.4. *NASA-TLX*

Na sliki 8 so prikazani rezultati NASA-TLX vprašalnika. Aplikacijo smo hoteli narediti preprosto in razumljivo za uporabo, kar nam je tudi uspelo. Pri uporabnikih, ki so imeli težave z izpolnitvijo scenarijev, je bila najvišja ocena mentalne zahtevnosti in frustracije. Obstaja možnost, da bi bili rezultati drugačni, če bi uporabniki rešili vprašalnik po vsaki nalogi in ne po koncu vseh scenarijev. Tako so uporabniki podali splošno oceno uporabe aplikacije in ne posameznih nalog, ki so jih morali izpolniti.



Slika 8. Rezultati NASA-TLX vprašalnika.

6. ZAKLJUČEK

Prvi del testiranja interaktivnega prototipa MyFamily smo opravili na omejenem vzorcu uporabnikov z namenom izločiti očitne napake in delno dopolniti aplikacijo, preden bi jo testirali na večjem vzorcu uporabnikov. Testiranje je služilo predvsem za preizkušanje funkcionalnosti in uporabniške izkušnje ob uporabi aplikacije. Rezultati prvega testiranja služijo kot orientacija in jih je potrebno interpretirati z zadržki. Zaobjeta je bila manjša skupina uporabnikov, ki so vešči uporabe elektronskih naprav ter vsakodnevno uporabljajo podobne aplikacije. Za boljšo oceno bi morali vključiti tudi uporabnike, ki so manj vešči upravljanja z računalnikom. Udeleženci so bili večinoma stari med 30 in 45 let. Za boljšo predstavo, kako bodo aplikacijo sprejeli starejši in otroci, bomo morali v naslednje testiranje vključiti tudi uporabnike iz teh starostnih skupin. Obstaja možnost, da so udeleženci na vprašanja odgovarjali pristransko, saj prihajajo iz iste fakultete kot ekipa, ki deluje na projektu.

V prihodnosti bomo ugotovitve s testiranja implementirali v interaktivni prototip in ponovno raziskali, kako lahko izboljšamo stimulativnost uporabnikov. Po uspešni povezavi v platformo EkoSmart bomo izvedli novo testiranje s pravimi družinami. Izbrali bomo približno 10 družin, ki bodo dva tedna uporabljale aplikacijo. S tem testiranjem bodo pridobljeni najbolj verodostojni rezultati, saj bodo družine podale oceno po daljši rabi aplikacije MyFamily.

7. LITERATURA

[1] http://ekosmart.net/sl/o-projektu/, Ekosmart, obiskano 10. 5. 2018
[2] Projektna dokumentacija, Eko Sistem Pametnega Mesta
[3] http://ejs.co/, Embedded JavaScript templating, obiskano 20. 2. 2018
[4] https://www.heroku.com/, Heroku: Cloud Application Platform, obiskano 25. 2. 2018
[5] https://mlab.com/, mLab Database-as-a-Service, obiskano 15. 3. 2018
[6] https://material.io/design/, Material Design, obiskano 10. 1. 2018
[7] https://getmdl.io/, Material Design Lite, obiskano 10. 1. 2018
[8] https://jquery.com/, jQuery, dostopno 10. 1. 2018
[9] S. G. Hart, »Nasa-Task Load Index (NASA-TLX); 20 Years Later«, Proceedings of the Human Factors and Ergonomics Society Annual Meeting, letnik. 50, št. 9, oktober 2006, stran 904-908.
[10] http://www.ueq-online.org/, User Experience Questionnaire (UEQ), obiskano 10.5.2018
[11] https://www.usability.gov/how-to-and-tools/methods/system-usability-scale.html, System Usability Scale (SUS), obiskano 10. 5. 2018

Univerzitetna založba Univerze v Mariboru

PLATFORMA ZA ENOSTAVNO PROTOTIPIRANJE MOBILNIH APLIKACIJ

BLAGOJ SOKLEVSKI, BOJAN BRUMEN, UROŠ PERNAT IN GREGOR PLAVČAK

Povzetek: Izzivi današnjega časa nas silijo v učinkovito izkoriščanje dragocenega časa. Prodati idejo v svetu informacijskih tehnologij ni lahko brez prototipov, s katerimi lažje predstavimo koncept, ki ga želimo prodati. Izdelava prototipov terja strokovna znanja, denar in čas, ki bi jih lahko uporabili v druge namene. V članku bomo prikazali, kako je lahko izdelava prototipov mobilnih aplikacij enostavna s pomočjo naše rešitve. Vzpostavili smo platformo za enostavno prototipiranje mobilnih aplikacij, ki je na voljo v oblaku 24/7, je enostavna za uporabo in dostopna vsem, tudi tistim, ki jim je programiranje tuje.

Ključne besede: • Android OS • generiranje mobilne aplikacije • uporabniški vmesnik • oblačne storitve

NASLOV AVTORJEV: Blagoj Soklevski, msg life odateam d.o.o., Titova cesta 8, 2000 Maribor, Slovenija e-pošta: blagoj.soklevski@msg-life.com. Bojan Brumen, msg life odateam d.o.o., Titova cesta 8, 2000 Maribor, Slovenija e-pošta: bojan.brumen@msg-life.com. Uroš Pernat, msg life odateam d.o.o., Titova cesta 8, 2000 Maribor, Slovenija, e-pošta: uros.pernat@msg-life.com. Gregor Plavčak, msg life odateam d.o.o., Titova cesta 8, 2000 Maribor, Slovenija e-pošta: gregor.plavcak@msg-life.com.

DOI https://doi.org/10.18690/978-961-286-162-9.16 ISBN 978-961-286-163-6

Dostopno na: http://press.um.si.

1. UVOD

Življenje v moderni družbi nas na podlagi principa »čas je denar« sili v pametno in produktivno uporabo časa ter s tem seveda tudi denarja. Kadar prodajamo kakšno svojo idejo, je najhitrejši način, da stranki opišemo to idejo, le-ta pa si poskuša slednje vizualizirati v svoji glavi. To nas je pripeljalo k temu, da smo začeli razmišljati kako stranki na intuitiven način predstaviti nek produkt, ki bi pripomogel k samemu razumevanju in vizualizaciji idejne zasnove. V večini primerov vsakokrat poskušamo razložiti, kako naj bi neka stvar izgledala v praksi, recimo na neki mobilni aplikaciji, z uporabo skiciranja, naših besed, oziroma s pomočjo tvorjenja stavkov. Ker se prvotne idejne zasnove lahko razlikujejo že v osnovi, smo kljub vsem omejitvam staknili glave in iskali preprost način, kako optimizirati proces prototipiranja tako, da bi to lahko počeli tudi ljudje iz poslovnega sveta oz. tudi takšni z manj tehničnega znanja. Ideja je bila rojena. Potrebujemo rešitev, ki bi na podlagi našega tehničnega znanja ter tehnološke ekspertize omogočala uporabo vsakomur ob vsakem času brez specifičnih znanj z namenom ustvarjanja prototipa mobilne aplikacije.

Platforma je sestavljena iz dveh delov, in sicer: spletni vmesnik, ki služi za uporabo dizajniranja izgleda aplikacije ter zaledni sistem, ki generira demo aplikacijo na podlagi vnesenih vhodnih podatkov. Za ta namen smo kot naše osnovne sestavne dele uporabili moderne tehnologije, kot so Angular, Kotlin, mikrostoritve ter oblačne storitve. Uporabnik začne proces na spletnem vmesniku, kjer določi barvo aplikacije ter doda posamezne poglede in vnosna polja. Iz tega dizajna nastane JSON datoteka, ki služi kot definicija mobilne aplikacije in jo je mogoče z gumbom »Generiraj« prenesti na sam oblak. Proces se nadaljuje na oblaku, kjer mikrostoritev ob prejemu JSON datoteke naredi prazen projekt Android narejen v programskem jeziku Kotlin in le-tega nadgradi z definicijo iz JSON datoteke. Po končanem opravilu se s pomočjo avtomatske skripte generira .apk datoteka ter QR koda, ki je takoj za tem vrnjena v spletni vmesnik in služi kot povezava do prenosa same mobilne aplikacije. Ob skeniranju te kode s pametnim telefonom se prične prenos in kasneje tudi inštalacija mobilne aplikacije. Ker pa vse skupaj biva v samem oblaku, je platforma seveda na voljo 24/7.

2. PLATFORMA ZA PROTOTIPIRANJE

Osnovno ogrodje platforme se sestoji iz dveh osnovnih komponent: uporabniški spletni vmesnik s katerim uporabnik izdela osnovo aplikacije, in generator kode, ki ustvari mobilno aplikacijo z namestitveno .apk datoteko za naprave z Android OS. Ta dva dela izmenjujeta podatke preko REST storitev z JSON meta-modelom. Za spletni uporabniški vmesnik smo uporabili tehnologijo Angular 6 s podporo Bootstrap 4. Vse skupaj je nameščeno v oblaku. Generator je javanska aplikacija napisana s pomočjo Spring Boot ogrodja v Javi 8, zapakirana v Docker zabojnik skupaj z Android SDK, Android build tools in ogrodjem Gradle. Zabojnik je orkestriran v Amazon Elastic Container Service (Amazon ECS).

2.1. Spletni uporabniški vmesnik

Zasnova mobilne aplikacije poteka preko spletnega uporabniškega vmesnika. Ta omogoča izdelavo novega prototipa aplikacije, pregled nad že izdelanimi aplikacijami in tudi neposredno namestitev na Android napravo s pomočjo QR kode. JSON meta model mobilne aplikacije se gradi s pomočjo urejevalnika aplikacije, ki omogoča enostavno dodajanje zaslonov in gradnikov in nastavljanje posameznih parametrov. Urejevalnik zajema celotno širino zaslona in je sestavljen iz treh vertikalnih sekcij.

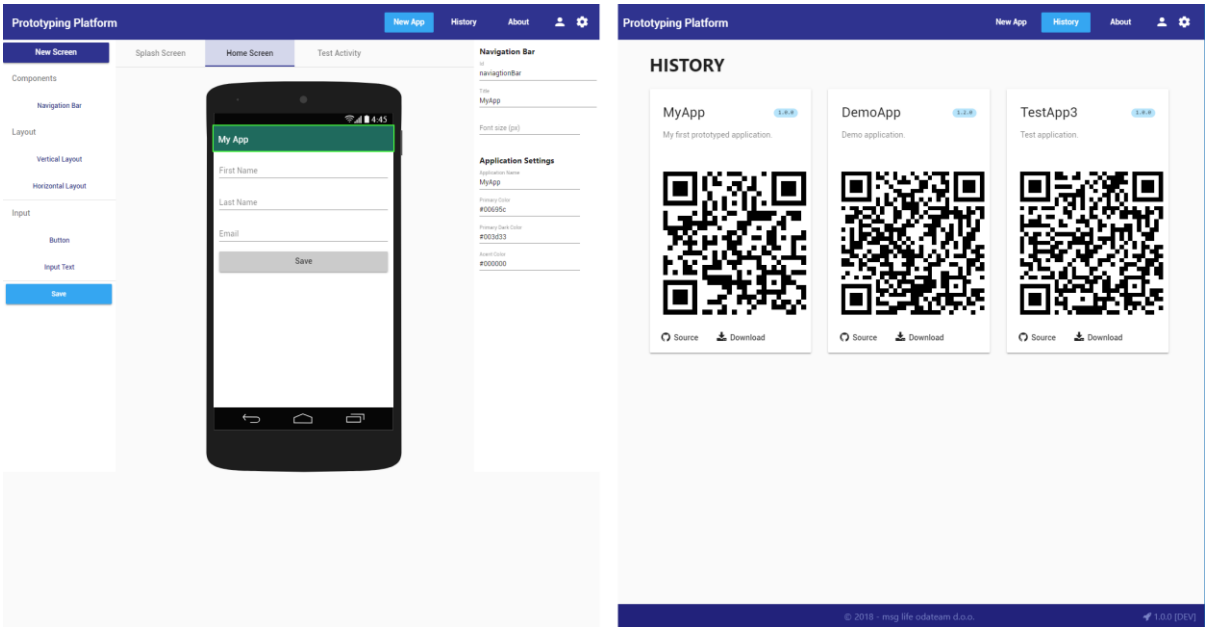
Na vrhu prve sekcije je gumb za dodajanje novih zaslonov v aplikacijo. Sledijo komponente, ki jih je možno dodati na posamezni zaslon. Zaradi načina delovanja generatorja aplikacij, komponente v veliki meri sovpadajo s komponentami, ki jih je mogoče najti tudi v Android Studio IDE-ju. V isti sekciji se nahaja tudi gumb za shranjevanje in generiranje aplikacije.

Osrednja sekcija predstavlja nabor zaslonov v aplikaciji, ki so pregledno organizirani v obliki zavihkov. Posamezni zavihek v osnovi predstavlja podlago v obliki mobilnega telefona, na katerega je možno dodajati posamezne komponente iz prve sekcije. Vsako komponento je možno izbrati in ji še dodatno nastaviti parametre (ali odstraniti) v tretji sekciji. Izbrana komponenta na zaslonu aplikacije je označena s svetlo zeleno obrobo.

Tretja sekcija urejevalnika prototipirane aplikacije je namenjena dodatnim nastavitvam. Tukaj je možno nastaviti ime aplikacije in barvno shemo. V primeru, da je izbrana tudi ena izmed komponent, je za le-to prikazan nabor parametrov (identifikator komponente, besedilo, velikost pisave, itd.).

Urejanje prototipne aplikacije se zaključi s potrditvijo na gumbu »Shrani«. V tem koraku se sestavljen meta model aplikacije posreduje generatorju, ki ga interpretira in generira izvorno kodo, in zgradi prenosljivo aplikacijo v obliki arhiva (.apk).

Nabor vseh aplikacij se nahaja na strani »Zgodovina«. Prototipne aplikacije so predstavljene v obliki kartic in vsebujejo ime aplikacije, verzijo, dodaten opis, povezavo na prenos izvorne kode in povezavo za prenos namestitven datoteke (.apk). Aplikacijo je možno enostavno namestiti na željeno napravo s skeniranjem QR kode posamezne aplikacije.



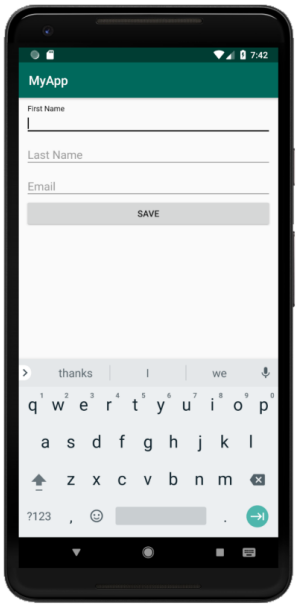
Slika 1. Spletni uporabniški vmesnik

2.2. Omejitve platforme za prototipiranje in JSON meta-model

Stili in teme na Android aplikacijah omogočajo ločevanje podrobnosti dizajna aplikacije od strukture uporabniškega vmesnika in njegovega obnašanja. Stili so zbirke atributov, ki določajo izgled posamezne forme ali pogleda. Stil lahko določa atribute, kot so barva in velikost pisave, barva ozadja in še mnogo več. Tema je tip stila, ki jo uveljavimo nad celotno aplikacijo, Activity gradnikom ali nad celotno hierarhijo pogledov. Kadar uporabimo stil kot temo, vsak gradnik aplikacije ustrezno prevzame stil.[1] Na platformi smo si zamislili omejitve glede izgleda. Trenutno lahko uporabnik spreminja samo barve colorPrimary, colorPrimaryDark in colorAccent. Zaenkrat generiranje aplikacije omogočasamo pokončnen pogled, ležeč pogled še ni podprt. Glede funkcionalnosti omogočamo samo aplikacije, ki vsebujejo forme, kar pomeni, da lahko uporabnik izbira med vnosnimi polji in gumbi, lahko pa spreminja tudi pozicijo posameznega gradnika. Aplikaciji lahko določimo vrstni red prikaza zaslonskih mask in navigacijo med njimi. Trenutno generirana aplikacija deluje v načinu brez povezave do interneta (offline mode), zato komunikacija z REST storitvami ni mogoča. Dostop do strojnih komponent in

senzorjev (kamera, Blutooth, NFC, …) trenutno ni mogoč, vendar bo to omogočeno v prihajajočih verzijah. JSON model predstavlja DNK generirane aplikacije. Vsebuje vse potrebne podatke za generiranje končne aplikacije. Ker je naš cilj dinamična generacija androidnih mobilnih aplikacij s strani uporabnikov, ki niso programerji, smo se morali omejiti na samo določene funkcionalnosti in omejenim naborom vizualnih gradnikov. Meta podatki v JSON obliki so razporejeni v več nivojev. Prvi nivo vsebuje ime aplikacije, URL do slike, ki se bo prikazala ob zagonu aplikacije, seznam barv in seznam aktivnosti. Vsaki aktivnosti lahko določimo ime in gradnike. Vsak gradnik je določen s tipom, oznako in identifikatorjem. Vsak gumb ima tudi dodatni atribut »akcija«, v katerem je ime aktivnosti, ki se bo ob kliku odprla. Na Sliki 2 je levo prikazan model in desno aplikacija, zgrajena na podlagi tega modela.


```
{
    "appName": { "name":"app_name", "value":"MyApp"},
    "splashScreenImageURL" :"https://image.freepik.com/free-icon/apple-logo_318-40184.jpg",
    "colors": [
        { "name":"colorPrimary", "value":"#00695c"},
        { "name":"colorPrimaryDark", "value":"#003d33"},
        { "name":"colorAccent", "value":"#000000"}

    ],
    "activities": [
      {"name":"MainActivity",
        "components":[
            {"label": { "name":"first_name", "value":"First Name"},
            "componentId":"firstNameField",
            "type":"textInput"},
            {"label": { "name":"last_name", "value":"Last Name"},
            "componentId":"lastNameField",
            "type":"textInput"},
            {"label": { "name":"email", "value":"Email"},
            "componentId":"emailField",
            "type":"textInput"},
            {"label": { "name":"button_save", "value":"Save"},
            "componentId":"saveButton",
            "type":"button",
            "openActivity":"TestActivity"
            }]
      },
      {"name":"TestActivity"}
    ]
}
```

Slika 2. Levo je met podatkovni model, desno je končni izgled aplikacije

2.3. Generator

Generator je osrčje platforme. To je zaledna javanska aplikacija napisana v ogrodju Spring Boot, ki sprejme JSON model, posredovan s strani uporabnika, na podlagi katerega se v ozadju zgradi projekt s programsko kodo. Programska koda se nato prevede v končni izdelek, ki je .apk datoteka. Preden preidemo na podroben opis delovanja generatorja, bomo opisali, kaj vse nastane pri generaciji. Rezultat generiranja je androidna aplikacija z izvorno kodo v Kotlinu, ki temelji na Android OREO 8.1 (API level 27). Minimalna verzija, ki je še podprta, je Android Nougat 7.0 (API level 24). Končno aplikacijo je mogoče prenesti na vse naprave, ki ustrezajo zahtevam, ne glede na velikost zaslona. A najboljšo uporabniško izkušnjo dosežemo, kadar jo namestimo na mobilni telefon, saj sta pri večjih zaslonih uporabniška izkušnja in ergonomija slabša.

Generiranje aplikacije poteka v več korakih. Prvi korak po prejetju JSON podatkovnega meta modela je kopiranje predloge projekta imenovane »base Application«, ki je prazen Android projekt, ki vsebuje samo osnovno »splash« aktivnost (Aktivnost ob zagonu aplikacije). Ta kopija predloge se preimenuje v konkreten projekt skladno z željami uporabnika.

V projektu je pripravljenih nekaj razredov, ki služijo kot različni generatorji, ločenih z vlogami:

- Color Resource Generator prepiše vrednosti iz meta podatkov v colors.xml datoteko. Ta datoteka je osnova za barvni izgled gradnikov v celotni aplikaciji. Komponente teme, ki se barvno lahko prilagajajo so: colorPrimary, colorPrimaryDark in colorAccent.
- Number Resource Generator zapiše spremenljivko »splash_screen_duration« v numbers.xml datoteko z vrednostjo 0. V kolikor se v meta podatkih nahaja povezava na sliko, se ta vrednost poveča na 3000. Ta vrednost upravlja zakasnitve v aplikaciji, kar bi simuliralo zakasnitev nalaganja podatkov v primeru izrisa slike.
- String Resource Generator prepiše vse vrednosti potrebne za izpise (labele, ki jih vidi uporabnik) iz meta podatkov v strings.xml datoteko. Trenutno ni večjezične podpore.
- Activity Generator izdela dejansko programsko kodo v obliki Kotlin razredov, kar so datoteke z končnico ».kt«. V kodi vključi import stavke na začetku in izdela telo razreda v katerem prepiše onCreate metodo in doda klice funkcij, ki so posredovane v meta podatkih.
- Layout Generator ustvari datoteke za vsak izgled vsake aktivnosti. Kot izhodišče se uporabi osnovni gradnik imenovan »LinearLayout«. Ta vsebuje »ScrollView«, ki služi kot vsebnik za vse ostale gradnike. Nastanejo s tem povezane datoteke imenovane z imenom aktivnosti, ter končnico ».xml«.
- Manifest Generator ustvari AndroidManifest.xml, ki spremeni ime aplikacije in registrira vse ostale aktivnosti, ki jih bo aplikacija poznala.
- Application Generator služi kot orkestrator za ostale generatorje.

Po ustvarjenem projektu v oblaku je možno prožiti generiranje .apk datoteke na dva načina. Klic »generate« pokliče ukaz Gradle ogrodja »gradlew assembleDebug«. Drug klic, ki je primeren za razvijalce in ni možen v oblaku, pa omogoča klic ogrodja Gradle z ukazom »gradlew installDebug«. Ta ustvari .apk datoteko in jo namesti na virtualno mobilno napravo (emulator). Zadnji klic je prenos .apk datoteke na ciljno mobilno napravo.

2.4. Namestitev

Ogrodje platforme sestoji iz dveh osnovnih komponent, ki ju ločeno zapakiramo v dva Docker zabojnika. Prvi vsebuje spletno aplikacijo v Angular ogrodju in spletni strežnik Nginx, ki servira statično vsebino. Drugi del platforme je z zalednim generatorjem in se namesti v drug zabojnik. Ta ima nameščena zraven Spring Boot aplikacije tudi razvojna ogrodja za mobilne aplikacije, kot so Android SDK, Android Build Tools in Gradle.

Oba zabojnika sta nameščena v Amazon ECS, ki je visoko zmogljiva, skalabilna storitev za orkestriranje Docker zabojnikov, kar omogoča uporabniku enostavno zaganjanje ter skaliranje aplikacij v zabojnikih na AWS. [2]

2.5. Težave in načrti za izboljšave v prihodnosti

Priporočen način dela z »Gradle« orodjem za pakiranje aplikacij je z uporabo »Gradle Wrapper«-ja (na kratko Wrapperja). Wrapper je skripta, ki kliče deklarirano verzijo Gradle orodja. Kot rezultat lahko razvijalec dobi delujoč projekt hitro, brez sledenja navodil kako le-tega namestiti [3]. Večina težav je izhajala iz tega, da se ob prvem zagonu Graddle Wrapper-ja vse morebitno manjkajoče knjižnice prenesejo na sam računalnik. Tega nismo opazili, dokler nismo naše Spring Boot aplikacije zapakirali v Docker zabojnik. Zgodilo se je, da smo dobili napako prvič, ko smo poklicali servis »generiraj« preko našega REST API-ja. Naša predpostavka, da je vsebina Docker zabojnika, ki vključuje Gradle ter Android SDK dovolj, se je izkazala kot napačna. Ker imamo na strežniku vedno čisto stanje, ko namestimo nek zabojnik, je naš inicialni REST klic servisa »generiraj« zmeraj znova nalagal Gradle .zip datoteko in zatem tudi to razpakiral ter instaliral. Seveda je to rezultiralo k daljšemu času generiranja aplikacije, kar se pa žal ni videlo v prikazu napake. Po ugotovitvi vzroka za to napako smo našli dve možnosti za sam popravek. Prva možnost je bila spremeniti atribut »distributionUrl« v datoteki gradle-wrapper.properties, ki kaže na lokacijo v zabojniku, kjer se nahaja .zip datoteka. S tem smo pridobili,

da ne zapravljamo časa s samim nalaganjem .zip datoteke iz spleta. Druga opcija je bila proženje REST klica asinhrono, kar je pomenilo, da ne čakamo na odgovor iz samega zaledja.

V prihodnosti bi radi razširili zaledno aplikacijo na način, da bi bila sposobna poleg dizajna sprejeti tudi definicijo kalkulacij in jih smiselno povezati z vnosnimi polji. Tak pristop bi omogočal dizajn recimo zavarovalniškega produkta, ki temelji na aktuarskih kalkulacijah.

Druge manjše razširitve pa bi bile narediti aplikacijo še bolj prilagodljivo, omogočiti prenos še več podatkov med samimi aktivnostmi (»activities«), omogočiti uporabo kamere in podobno.

3. ZAKLJUČEK

Za kreiranje preproste Android aplikacije , ki vsebuje nekaj vnosnih polj in gumb, razvijalec potrebuje nekje od 10 do 30 minut. To vključuje zagon Android studia, pa vse do končne izdelave .apk datoteke. Vse skupaj je seveda odvisno od razvojnega računalnika, razvijalčevih znanj, spretnosti, predhodnih izkušenj s programiranjem takšnih aplikacij in drugih različnih dejavnikov. Z našo platformo smo zagotovili eliminiranje mnogih od zgoraj naštetih dejavnikov. Za start takšnega projekta s pomočjo platforme sedaj več ne potrebujemo izkušenih Android razvijalcev. Uporabniški vmesnik je tako preprost, da ga lahko uporablja kdorkoli. Prav tako je kreiranje in zagon takega projekta na voljo takoj, brez nameščanja kakšne posebne programske opreme. Ker pa je platforma seveda nameščena v oblaku, končni uporabnik ne potrebuje veliko procesorske moči in drugih virov za kreacijo. Preprosto odpremo brskalnik in začnemo z dizajniranjem in kreacijo. Kot dodatek smo z oblačnim pristopom rešili tudi problem domovanja kreirane aplikacije, saj je le ta po končani kreaciji odložena v hrambo na oblaku in vedno dostopna s pomočjo skeniranja QR kode. Če bi razvijali aplikacijo na klasičen način, bi morali imeti omogočen način razvijalca na fizični Android napravi in le to povezano z računalnikom pri testiranju in namestitvi same aplikacije. Ostaja slabost, da moramo v vsakem primeru omogočiti namestitev aplikacij izven »Google play« trgovine na način da omogočimo / dovolimo nameščanje neznanih virov v nastavitvenem meniju.[4]

Naj še navedemo, da nismo samo eliminirali različnih dejavnikov v samem kreiranju preproste Android aplikacije, kot so zahtevana znanja, vendar smo tudi prepolovili čas kreacije le-te. Tako bi lahko kritično ocenili, da sedaj potrebujemo le 10 do 30 minut od vstopa v sam uporabniški vmesnik, do nameščanja same aplikacije na napravo. Največja značilnost same platforme pa je seveda, da je sama platforma lahko uporabljena kjerkoli, kadarkoli in od kogarkoli, ne glede na tehnična znanja in izkušnje. V tej tehnološki avanturi ob združevanju različnih znanj in ekspertiz smo razvili platformo, ki nam bo prihranila veliko dragocenega časa v prihodnosti. Pri tej celotni izkušnji smo prav tako testirali veliko konceptov, ki jih nameravamo uporabiti pri sami produkcijski aplikaciji. Kot primer smo za generiranje Android aplikacije uporabili Kotlin namesto Java programskega jezika, da bi si dokazali kaj le-ta prinese kot dodatek pri samem razvoju mobilnih aplikacij.

Na koncu smo kot ekipa zadovoljni z razvitimi funkcionalnostmi, seveda pa že gledamo naprej kako bi-le te v prihodnosti nadgradili in s tem omogočili še širši spekter značilnosti, ki bi v končni verziji rezultirale k dizajniranju ter kreiranju aplikacij brez večjih omejitev.

4. LITERATURA

[1] https://developer.android.com/guide/topics/ui/look-and-feel/themes, Android Developer Guide, obiskano 23. 5. 2018.

[2] https://aws.amazon.com/ecs/ , Amazon Elastic Container Service, obiskano 23. 5. 2018.

[3] https://docs.gradle.org/current/userguide/gradle_wrapper.html , Gradle Wrapper, obiskano 23.5.2018.

[4] https://www.androidauthority.com/how-to-install-non-market-third-party-apps-on-your-phone-31494/, How to Install Non-Market, Third-party Apps on Your Phone, obiskano 23. 5. 2018.

Univerzitetna založba
Univerze v Mariboru

NEPREKINJENA DOSTAVA V VZPOREDNEM KRITIČNEM POSLOVNEM OKOLJU

TADEJ JUSTIN, ŽIGA CIGLAR IN ANDREJ OREŠNIK

Povzetek: Razvoj in realizacija obsežnejšega programja je navadno razdeljena na več zaključenih enot, ki jih lahko realizira poljubna razvojna skupina oz. razvojno podjetje. Pred izvedbo deljenih nalog je v takih primerih vnaprej izbrano in specificirano tudi produkcijo okolje. Vsem skupinam mora biti na voljo čim bolj podobna infrastruktura, kot je na voljo v produkcijskem okolju, za preverjanje realizacije izvedenega dela. Le tako so posamezne skupine lahko dovolj samozavestne, da bo njihov izdelek deloval tudi v produkciji. Poseben primer deljenega razvoja predstavlja zaprto produkcijsko okolje naročnika. Velikokrat se izkaže, da je produkcijsko okolje razvijalcem nedostopno, hkrati pa je nedostopno tudi celotno omrežje naročnika, takrat je dostava programske kode v zaprto poslovno omrežje s strani razvijalcev nemogoča. V takšnih primerih je potrebno celoten proces neprekinjene dostave prilagoditi in upoštevati dodatne možne zaplete pri izdaji skupnega programja v produkcijo. V tem prispevku predstavljamo izkušnje in realizacijo neprekinjene dostave izvorne programske kode z usklajenim delovanjem dveh podjetji, pri čemer je eno v vlogi lastnika programske kode, kontrolorja kvalitete in realizatorja izdaj programa v produkcijsko okolje, drugo pa v vlogi razvijalca. Posebnost te izkušnje je, da slednje žal nima dostopa do nobenega izmed programskih okolij prvega podjetja, nima možnosti niti "odriva" (ang. push) programske kode v omrežje prvega.

Ključne besede: • neprekinjena dostava • neprekinjena integracija • GitLab • Kubernetes

NASLOV AVTORJEV: dr. Tadej Justin, raziskovalcec, Medius d.o.o,. Tehnološki park 21, 1000 Ljubljana, Slovenija, e-pošta: tadej.justin@medius.si. Žiga Ciglar, višji razvijalec programske opreme, Medius d.o.o, Tehnološki park 21, 1000 Ljubljana, e-pošta: ziga.ciglar@medius. Andrej Orešnik, arhitekt informacijskih rešitev, Medius d.o.o., Tehnološki park 21, 1000 Ljubljana, e-pošta: andrej.oresnik@medius.

DOI https://doi.org/10.18690/978-961-286-162-9.17 ISBN 978-961-286-163-6

Dostopno na: http://press.um.si.

1. UVOD

V Sloveniji je dan danes vedno več manjših razvojnih podjetij, ki ponujajo razvoj programske opreme le za določen segment programja ali svetovanje iz specifičnih ozkih podjetij področja IKT. Združeni specialisti ali skupine lahko uspešno konkurirajo tudi večjim podjetjem, pri razvoju visoko tehnoloških programskih rešitev, čeprav lahko dostavijo le ozko usmerjen segment želenega proizvoda. Opažamo, da je vedno več sodelovanja med ozko usmerjenimi razvojnimi podjetji programske opreme pri izdelavi končnih aplikacij. Razvoj in realizacija obsežnejšega programja je navadno razdeljena na več zaključenih enot ali segmentov, ki jih lahko realizira poljubna razvojna skupina ali razvojno podjetje. V kolikor, se naročnik odloči za porazdeljen in segmentiran razvoj programja, ponavadi sam predpiše obliko sodelovanja in usklajevanja razvoja končnega produkta.

Pred izvedbo deljenih nalog je pogosto vnaprej izbrano in podrobno določeno tudi produkcijo okolje. Za lažjo dosego končnega cilja - razvoja visoko tehnološkega programja, mora biti na voljo razvojnim skupinam čim bolj podobna infrastruktura in programska oprema (okolje), kot bo/je ta na voljo v produkcijskem delovanju. Razvojno okolje, ki je v večji meri skladno s produkcijskim okoljem, skupine uporabijo za preverjanje, verifikacijo in validacijo razvitega programja. Le s tovrstnimi preizkusi so posamezne skupine lahko dovolj samozavestne, da bo njihov izdelek deloval tudi v produkciji. Dostop do enake ali čimbolj podobne infrastrukture, pa ni edini pogoj za oddajo realizirane rešitve naročniku. Skupine morajo imeti na razpolago tudi vse ostale enote ali module celotnega programa. Na takšen način lahko skupina dobro preizkusi tudi integracijske funkcionalnosti svojega doprinosa oz. realizirane naloge.

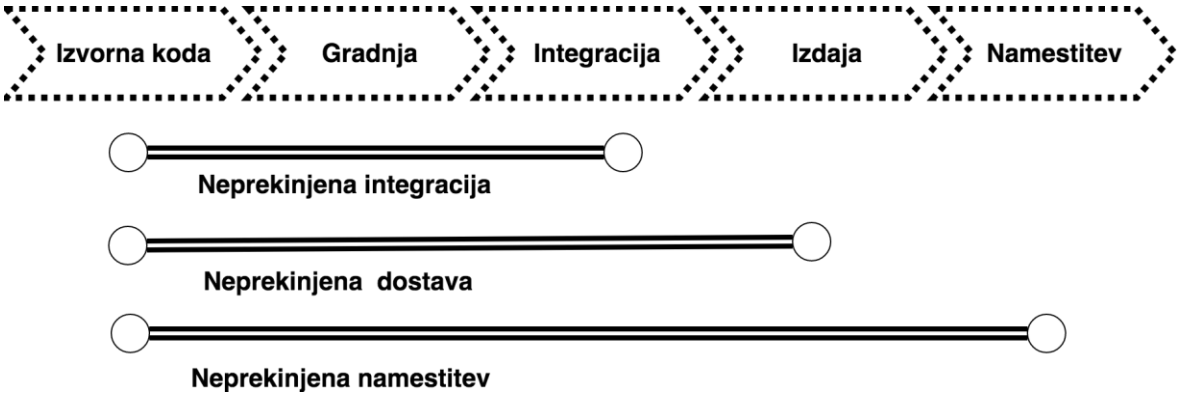
V idealnih primerih naročnik poskrbi za dokumentacijo okolja, njegovih specifik in/ali celo njegove skriptirane postavitve. V redkih primerih razvojnim podjetjem ponudi tudi namensko programje, ki omogoča enostavno in hitro dostavo razvite funkcijonalnosti po predpisanih kriterijih. S tem omogoči razvijalcu osredotočenost le na razvoj specifičnih funkcionalnosti, za ostalo pa poskrbi naročnik ali njegov podizvajalec, specialist za neprekinjeno dostavo. Pri porazdeljenem razvoju programja se pokaže, da je to ena ključnih komponent za učinkovit in hiter razvoj končnega izdelka. Razvojnim skupinam omogoča dostavo programja v skladu z razvojnimi kriteriji naročnika, vnaprej definiranimi povezavami in relacijami tako med infrastrukturnimi specifikami in hkrati tudi samim razvojnim ciklom. Za razvijalce se pokaže prednost takega programja tudi v tem, da se končnim razvijalcem ni potrebno neposredno zavedati vseh specifik in kriterijev naročnika, saj so le-ti lahko že določeni znotraj programja za neprekinjeno dostavo [1].

V prispevku predstavljamo poseben primer porazdeljenega razvoja s specializiranim naročnikovim programjem za neprekinjeno dostavo, ki omogoča razvijalcem nemoten in osredotočen razvoj naročnikovih želja. Programje zajema specifiko produkcijskega okolja, njegovo skriptirano postavitev in neprekinjeno dostavo razvitih funkcionalnosti. V predstavljenem primeru naročnika opredeljuje popolnoma zaprto okolje in hkrati popolna kontrola nad izvorno kodo in oddajo programja v produkcijsko okolje. V takih primerih razvojne skupine nimajo neposrednega dostopa do naročnikove infrastrukture, kar je morda posebnost, vendar tipičen realni primer kritičnega poslovnega okolja. Odprto kodna orodja za neprekinjeno dostavo pa tipično ne premorejo podpore za tak primer, kar pa narekuje zasnovo novega namenskega orodja, ki omogoča razvojnim skupinam dostavo v okolje naročnika.

2. ZASNOVA SISTEMA NEPREKINJENE DOSTAVE V ZAPRTO OKOLJE NAROČNIKA

Definicija neprekinjene dostave je pristop k avtomatizaciji procesa pri izdaji verzije programja od razvoja do izdaje. Oblikovalo ga je gibanje “DevOps” [2], ki izhaja iz povezave med angleškima besedama “Development” (razvoj) in “Operations” (operacije). Proces tako zajema avtomatične postopke izločanja slabih verzij programa in s tem povečuje zaupanje v izvorno kodo programja.

Diagram na sliki 1 prikazuje razlike med pojmi, ki so v zadnjih letih na področju DevOps močno zastopani, vendar pogosto zamešani.



Slika 1. Diagram razlik med pojmi: neprekinjena integracija, neprekinjena dostava in neprekinjena namestitev

Z dobrim poznavanje procesa neprekinjene dostave lahko hitro posplošimo proces tudi za vzporedne sisteme in ga priredimo za uporabo, tudi za primer zaprtega kritičnega poslovnega okolja. Z gradnjo in razvojem lastnega programskega orodja za neprekinjeno dostavo lahko enostavno omogočimo tudi izpolnitev naročnikovih specifičnih želja pri postopku izdaje različice razvitega programja v produkcijo. Razvijalcem pa omogočimo čim lažji razvoj in izdajo produkcijske verzije programja po ustaljenih praksah, kot jih predlaga in predvideva gibanje "DevOps". Pri razvoju takega programja se je zato nedvomno potrebno ozirati tudi na praktično uporabo uveljavljenih praks in razvojnih ciklov. Pri razvoju programja za neprekinjeno dostavo so bile upoštevane naslednje izboljšave v primerjavi z tradicionalnim načinom predaje izvorne kode:

- avtomatizacija razvojnega cikla,
- pregleden način za ugotavljanje sprememb v novih verzijah,
- konsistentno upravljanje konfiguracij,
- avtomatizacija ročnih posegov,
- in prenosljivost gradnikov programja.

Z razvojem programja za neprekinjeno dostavo pri upoštevanju kriterijev sodobne neprekinjene dostave se pokažejo ne le prednosti za zunanje razvojne skupine, pač pa predvsem za samega naročnika, ki je v obravnavanem primeru lastnik izvorne kode, kontrolor kakovosti in izdajatelj novih verzij programja. Programje neprekinjene dostave naročniku olajša izdajo in testiranje realiziranih funkcionalnosti. Prav tako je celoten postopek do oddaje programja v produkcijo avtomatiziran. Specifike realiziranega programja in visoka stopnja varnosti razvite programske rešitve vseeno predvideva ročno proženje namestitve novih verzij programja v produkcijo, kljub temu pa je v celoti avtomatizirana.

Zaprto okolje naročnika narekuje razvojnim skupinam, da same realizirajo postavitev razvojno/testnega okolja. Ker v splošnem želimo, da bi bila konfiguracija in postavitev čim bolj podobna produkcijskemu okolju, se lahko s pridom uporabi skriptirano postavitev okolja, obenem, pa olajša njegovo vzpostavitev, tako v javnih oblakih kot na lastni infrastrukturi. Obenem naročniku omogoči tudi postavitev identičnega testno/produkcijskega okolja (ang. staging envronment) ali replikacijskega okolja. S tako specifikacijo lahko naročnik prepreči morebitne nesporazume, ki jih opredeljuje produkcijsko okolje in omogoči razvojnim skupinam njihovo obravnavo že pri razvoju naročene funkcionalnosti. Z uporabo odprtokodnih programskih orodij in ogrodij je bila z naročnikom zasnovana skriptirana namestitev, ki tudi razvojnim skupinam omogoča vzpostavitev testno-razvojnega okolja. Naročnik je med širokim naborom orkestracijskih sistemov izbral sistem za orkestracijo izvajanja programov v vsebnikih - Kubernetes [3,4]. Izbira takega produkcijskega okolja prinaša v poslovno kritično okolje določena

tveganja pri izolaciji aplikacij, vendar obenem omogoča hitrejšo in bolj pregledno opravljanje pri visoki stopnji skalabilnosti.

2.1. Zakaj Kubernetes?

Z vidika podjetja, ki je v vlogi naročnika in razvija programje na porazdeljen način, je ključnega pomena zmožnost Kubernetesa, da lahko zelo hitro uvajamo nove postavitve aplikacij, seveda v okviru kapacitet. Z vnosom deklaracijskih datotek ("Kubernetes YAML") lahko postavimo tudi več primerkov iste aplikacije (za različne faze testiranja) ali enostavno ustvarimo delitev obstoječih aplikacij na več manjših, obvladljivejših enot.

Programje, ki ga želimo poganjati, zapakiramo v obliko Docker vsebnika (ang. container). Tako je podprto poganjanje skoraj kakršnekoli programske opreme, ki lahko teče pod operacijskim sistemom Linux, posredno pa je lahko aplikacija razvita v praktično kateremkoli programskem jeziku. V konkretnem primeru je sicer aplikacija bila razvita v programskem jeziku Java.

Pri razvoju aplikacij namreč želimo že pred produkcijo poganjati aplikacijo v pogojih testiranja v različnih okoljih:

- v okviru sistema neprekinjene integracije poganjati avtomatizirane teste ob vsaki gradnji ali periodično poganjati temeljitejše avtomatske teste,
- v okviru ročnega testiranja vzdrževati primerek aplikacije v verziji, ki je kandidat za oddajo v produkcijo,
- za testiranje funkcionalnosti API s strani drugih podjetij vzdrževati poseben primerek testne namestitve s primerno pripravljeno podatkovno bazo,
- za demo predstavnikom naročnika morebiti pripraviti poseben primerek različice v znanem stanju, da s tem ne oviramo vzporednega razvoja in testiranja novih funkcij.

Zato seveda potrebujemo način za aplikacijo v razvoju pognati. Če pripravimo vsebnik v obliki Docker, imamo s tem tudi binarno referenčno okolje izvajanja aplikacije, ki je lahko enako vse od izvajanja neprekinjene integracije oz. neprekinjenega testiranja do produkcije. V praksi sicer Docker ne zagotavlja popolne izolacije in Docker slika tudi ne vsebuje jedra operacijskega sistema (Linux), zato je v praksi potrebna še uskladitev operacijskih sistemov na gostiteljih izvajanja Docker slik.

Kubernetes omogoča, da lahko prijavljeni uporabniki (v okviru pravic in kvot) sami ustvarijo nove primerke aplikacije, ki jih ločijo npr., z različnimi imeni gostitelja, s čimer je lažje dodajanje okolij in pospešen je začetek dela na novem namestljivem gradniku. Cilj, da bi lahko že pri razvojnem podjetju postavili čim bolj podobno infrastrukturo oziroma "maketo infrastrukture", kot je pri naročniku, je s Kubernetes lažje dosegljiv.

Omenjena zmožnost - poenostavljeno ustvarjanje novih namestitev aplikacije - je seveda uporabna tudi za namene produkcijskih okolij in zaradi tega se Kubernetes hitro uveljavlja tudi v poslovnih okoljih. Predvsem je to pomembno v primerih, da želimo uvesti arhitekturo mikrostoritev, ker moramo v tem primeru imeti zmožnost učinkovitega obvladovanja življenjskih ciklov velikega števila manjših programov.

Vse, kar omogoča Kubernetes, je izvedljivo tudi s strežniško infrastrukturo na osnovi virtualizacije ali celo s fizičnimi strežniki, vendar terja bistveno več dela, tako razvijalcev, kot sistemskih administratorjev. Kubernetes (in druge rešitve orkestracije vsebnikov) pa postavljanje kompleksnejših strežniških namestitev aplikacij poenostavijo, tako da te postanejo stroškovno učinkovite in lahko pripomorejo k testiranju ter s tem izboljšanju kakovosti aplikacij tudi pri poslovnih aplikacijah, katerih razvoj je bolj stroškovno občutljiv, kot razvoj aplikacij za množični trg.

Vendar ni nujno, da bo raba Kubernetesa vedno pomenila manj porabe strojnih virov - lahko se zgodi ravno obratno, ker lahko uvedba boljšega izvajalnega okolja privede do tega, da organizacija, uvede nove aplikacije, saj je za njih to lažje. Možni scenarij je tudi, da postane obvladljivo poganjanje rešitev, ki so sicer strojno in upravljalsko zahtevnejše, ampak omogočajo boljšo podporo poslovanja organizacije.

Poleg Kubernetesa obstajajo tudi drugi sistemi za orkestracijo vsebnikov, npr. Docker Swarm, Rancher Cattle, Apache Mesosphere Marathon, CoreOS Fleet, ... ter lastniške rešitve posameznih ponudnikov javnega oblaka, kot so Amazon ECS, Google Container Engine, ... Naročnik se je odločil za rešitev na lastnih sistemih ("on-premises"). Med temi se je odločil za Kubernetes, ker ima najširšo skupnost razvijalcev, uporabnikov ter ponudnikov plačljive podpore. Zato je možno računati tudi na prihodnji razvoj te platforme. Kubernetes ni najenostavejša, je pa ena od bolj fleksibilnih rešitev, kar pomeni manjša tveganja v luči dejstva, da v poslovnih sistemih ne moremo vedno zanesljivo napovedati vseh prihodnji poslovnih potreb, ki jim bo izvajalno okolje moralo zadostiti.

2.2. Namestitev Kubernetesa v izoliranem produkcijskem okolju

Značilnost poslovno kritičnih aplikacij je, da se uporabljajo za takšne namene, kjer neavtoriziran dostop predstavlja veliko grožnjo, kar lahko posledično prinaša tudi resne finančne ali strateške težave pri poslovanju naročnika. Zato podjetja vlagajo v informacijsko varnost s ciljem preprečiti vse možne načine napadov.

Docker in posledično rešitve na osnovi Dockerja se zanašajo na to, da je slika z izvajalno kodo dostopna iz registra (registry), torej javnega ali zasebnega strežnika za hrambo in distribucijo slik z izvajalno kodo. Gradniki Kubernetes so tudi sami distribuirani kot Docker slike, ki se namestijo iz javno dostopnih registrov, kot so gcr.io, quay.io ter hub.docker.io. To zahteva, da so omenjeni registri dostopni v času namestitve, pa tudi kasneje se dogaja, da Docker dostopa do teh registrov iz različnih razlogov (preverjanje za posodobitve, prvi zagon posameznih gradnikov na novo dodanem Kubernetes vozlišču, …).

Z vidika informacijske varnosti to predstavlja tveganje. Možni napadi so npr.:

- Napadalec uspe vdreti v javni Docker register ali v sisteme za gradnjo samega Kubernetesa ter vnese spremenjene Docker slike z dodano zlonamerno kodo (ang. malware).
- Napadalec, z možnostjo vpliva na omrežje na katerikoli točki od namestitve Kubernetesa do teh registrov, lahko prestreže ta promet in npr. ta dostop organizaciji blokira ter s tem povzroči motnje v delovanju (ang. "denial of service").
- V primeru odkritih ranljivosti v protokolih TLS, lahko napadalec ta promet prestreže in spremeni ter na omrežju vstavlja zlonamerno kodo.

Ker Kubernetes predstavlja podlago za aplikacije, je posebnega pomena zagotavljati integriteto samega Kubernetes okolja in upravičena so vlaganja v zavarovanje.

Minimizacija tveganj je, da Kubernetes postavimo na od interneta izoliran način (ang. air gapped), kar si je naročnik tudi določil kot pogoj, za rabo Kubernetes in je to tudi implementiral v svojih namestitvah. Pri tem zajamemo ves nabor Docker slik - gradnikov Kubernetes za določeno verzijo v datoteke ter uvozimo v krajevni Docker register, ki ga konfiguriramo kot zrcalo (ang. mirror) javnega registra. Pri tem je bolj zanesljivo izvajati različne oblike preverjanj (npr. iskanje znanih ranljivosti in znanih primerkov zlonamerne kode) tudi na eni kopiji podatkov, kot pa ta preverjanja samo izvajati na vstopnih točkah omrežja.

Kot za vsako programsko opremo, ki poganja grozd strežnikov, je tudi namestitveni proces Kubernetes relativno zapleten. Seveda je mogoče namestitev izvesti v precejšnji meri ročno, vendar je priporočeno namestitev skriptirati. Pri tem je težava, da sicer obstajajo številni načini namestitve Kubernetesa

(kubeadm, kops, kubespray, CDH, ...), ampak nobeden od teh sistemov nima implementirane posebne podpore za "air gapped" namestitev - vsi se zanašajo na stalno dostopnost interneta. Podprt je sicer posredniški strežnik oz. "proxy", ki ne prispeva toliko k zmanjšanju zgoraj omenjenih tveganj.

Naročnik je razvil posebne skripte za postavitev zrcalnih strežnikov v njihovem okolju, za namestitev Kubernetesa pa je uporabil projekt kubespray [5], ki je nabor "receptov" za sistem Ansible. Tudi konfiguracija namestitve je lahko verzionirana v sistemu za upravljanje različic (npr. Git), kar olajša uskladitev različic Kubernetesa v testni in produkcijski postavitvi.

Skupnost razvijalcev Kubernetesa se zaveda te ovire za rabo le tega v kritičnih poslovnih okoljih. Formirana je delovna skupina za izboljšanje namestitvenega procesa (SIG-cluster-lifecycle), ki ima enega od ciljev tudi podporo za nameščanje v okoljih z varnostnimi zahtevami za izolacijo od interneta, vendar je trenutno v teh okoljih še vedno potrebno v veliki meri ročno postaviti zrcalni register (ang. registry mirror) [6], kar je naročnik tudi naredil.

Poleg varnosti so prednosti rabe zrcalnih registrov tudi hitrejše postavljanje novih vozlišč in Kubernetes clustrov (ker je krajevno omrežje hitrejše od internetne povezave) in večje zaupanje v konsistentnost različnih okolij (testno in produkcijsko Kubernetes okolje), ker imajo administratorji kopijo vseh potrebnih datotek..

2.3. Ostali vidiki varnosti

2.3.1. Omrežna segregacija

Drugi varnostni ukrep v poslovnih omrežjih je omrežna segregacija. V sodobnih časih ne zadostuje več le postaviti točke informacijske obrambe na vstopu v omrežje (ang. perimeter defense), ampak je potrebno varnost zasnovati v globino (ang. defense in depth), torej tudi samo omrežje razdeliti v območja ter postaviti "požarne pregrade" med njimi.

To je še zlasti posebnega pomena, če lahko za nek informacijski sistem rečemo, da v okviru istega obsega (ang. same perimeter) spada več organizacij.

Pri načrtovanju varnosti je potrebno privzemati, da bodo napadalci uspešno vdrli do vsakega posameznega elementa omrežja, proti čemur se lahko borimo tako, da onemogočimo in otežimo uporabo tega elementa kot odskočno desko za napade na druge elemente omrežja.

Ločitev okolij gradnje ter nadzorovan prenos sprememb med njima je tudi element omrežne segregacije, tako, da to prispeva tudi k varnosti.

2.3.2. Izolacija izvajanja

Koncept vsebnikov, tudi Docker, prinaša prednosti z vidika upravljanja postavitve aplikacij ter porabe virov. Po drugi strani pa predstavlja tveganja z vidika varnosti, ker je stopnja izolacije manjša kot pri izvajanju na klasičnih navideznih strojih ali fizičnih strežnikih [7].

Vsi vsebniki na istem gostitelju tečejo na istem jedru operacijskega sistema. Procesi v različnih vsebnikih so torej procesi istega jedra, ki so med seboj ločeni samo prek funkcij ločitve virov (Linux CGROUP) in ločitve vidnosti (Linux kernel namespaces). Implikacije so, da varnostne napake v jedru, ki omogočijo napadalcu pridobitev korenskih (root) privilegijev na posameznem vsebniku, omogočajo napad tudi na vse ostale vsebnike na istem vozlišču in posredno lahko omogočijo napad na celotno postavitev Kubernetesa.

Zato je na Kubernetes vozliščih še pomembneje redno nameščati varnostne nadgradnje, zlasti za jedro Linux. Nekatere distribucije Linux omogočajo celo uveljavitev popravkov Linux jedra med izvajanjem,

brez ponovnega zagona, kar je priporočljivo. Razvita je sicer rešitev gVisor [8], ki omogoča približati raven izolacije virtualizacijskim rešitvam, ampak še ni dovolj preverjena za poslovno kritično produkcijo.
Omejena izolacija je eden od ključnih povodov, da se je naročnik odločil postavitve na CI in testnih okoljih poganjati na ločeni Kubernetes gruči kot produkcijo.

2.3.3. Onemogočenje znane nevarne kode

Tako kot pri tradicionalnih strežniških infrastrukturah je pomembno tudi pri infrastrukturi na osnovi Dockerja poganjati iskalnike znane zlonamerne kode (protivirusne programe, mrežne filtre, …) in iskalnike gradnikov z znanimi ranljivostmi.

Pri tem je potrebno uporabiti iskalnike, ki podpirajo iskanje v zasebnem Docker registru organizacije, ker je ta register točka, skozi katero potuje zgrajena izvajalna koda do točk izvajanja.

Posebnega pomena je izbira osnovnih slik. Pri gradnji Docker slike moramo bodisi ustvariti celoten datotečni sistem od začetka, bodisi izbrati neko osnovno sliko z distribucijo Linux. Praviloma izberemo že neko obstoječo namestitev izmed slik, ki so na voljo na osrednjem javnem registru Docker Hub. Z vidika varnosti je pomembno, da izberemo zaupanja vredno osnovno sliko, ker se izbrana slika dejanko prenese, namesti in poganja tudi kot osnova vsebinka v produkciji. Zlonamerna koda, ki jo napadalcem uspe namestiti v osnovne slike v register osnovnih slik Docker Hub, bo tako pognana v samem središču produkcijskega sistema. Najbolje je izbrati uradne slike, ki jih pripravijo skrbniki posameznih Linux distribucij in kjer posodabljajo vdelane programske pakete, seveda pa je potem potrebno redno ponovno graditi Docker slike aplikacij in jih nameščati v produkcijo.

2.3.4. Varnostne prednosti Dockerja

Pristop Docker vsebnikov z vidika varnosti pomeni nekatere prednosti:

- Izvajalna slika se pri vsakem zagonu ponovno prenese iz zasebnega registra (“ephemeral OS installation”). Morebitna zlonamerna koda ima zaradi tega zaprto eno izmed možnih poti za trajno namestitev v sistem, ker tudi, če se uspešno zažene z okužbo obstoječih procesov v vsebniku ali če v vsebnik namesti svoje procese, bodo ti pri naslednjem zagonu “pozabljeni”. Pri tem je sicer pomanjkljivost, da se lahko zgodi, da izgubimo sledi morebitnih varnostnih vdorov.
- Enostavno izvedljivo je urediti, da je datotečni sistem slike samo berljiv (ang. read-only). Nabor morebitnih potrebnih konfiguracijskih datotek se pripravi ročno, potem pa se v času zagona vsebnika s strani Kubernetes (ali drugega orkestratorja, pod katerim teče) preslika v datotečni sistem vsebnika. Če je datotečni sistem samo berljiv, je tudi ob nekaterih vrstah uspešnih vdorov oteženo namestiti in zagnati trajno kopijo zlonamerne kode.

2.4. Ogrodja in orodja za neprekinjeno dostavo

Realizacija neprekinjene dostave je močno odvisna od izbire programskega okvirja oz. programa za opravljanje različic in programa namenjenega splošnemu avtomatskem testiranju komponent programa. Ob novem razvoju posebne različice programja za neprekinjeno dostavo je smiselno preveriti ali uporabiti obstoječe sodobne programe za opravljanje različic, testiranje in izdajo različic. Na tem mestu najdemo več možnosti, kot so npr. Jenkins, GitLab, TeamCity, Travis CI, Bamboo, Go CD , Spinaker in drugi. Nekatere od naštetih lahko uporabimo kot specializirano rešitev za neprekinjeno dostavo ali pa je le-ta zajeta kot posamezen gradnik produkta. Če si želimo, da razvijalcem ponudimo dobro prakso razvoja, ki predvideva, da je realizirana posamezna komponenta programja v največji meri tudi testirana in da je tudi pregledana s strani drugih razvijalcev, kot le samega avtorja, lahko našo izbiro programa za izbiro različic zožimo na take, ki omogočajo tudi enostavno proženje testnih scenarijev, in sicer kar z odrivom izvorne programske kode v upravljalnik različic. V svetu se je za tako prakso že uveljavilo skupno ime “GitOps”, saj lahko prožimo izdajo različice programa na podlagi sprejete revizije izvorne kode v strežnik upravljalnika različic Git.

Poleg skript, ki omogočajo avtomatično izdajo verzije, proženje integracijskih testov in testov programskih enot, je ključna komponenta tudi sinhronizacija izvorne kode med razvojnimi skupinami in naročnikom. Ta je realizirana na tak način, da naročnik izvede akcijo sinhronizacije po vnaprej določenem časovniku.

Kot ogrodje za realizacijo programa za neprekinjeno dostavo v vzporednih kritičnih poslovnih okoljih smo izbrali GitLab, ki omogoča enostavno postavitev in tudi enostavno realizacijo vseh zahtevanih komponent naročnika. Gradnja cevovoda je tako omejena na pravila, določena v eni sami datoteki. Ta datoteka služi kot skupek pravil, ki prožijo posamezne dele neprekinjene dostave.

2.5. Upravljanje različic programske kode in sinhronizacija izvorne kode programja

Dostava programske kode naročniku je izvedena s pomočjo sistema sinhronizacije verzioniranja kode v sistemu Git [9], ki je integriran v več neodvisnih strežnikov GitLab-CC [10]. V kolikor sodeluje pri razvoju le ena razvojna skupina sta za prevzem potrebna dva strežnika GitLab. Eden je nameščen v razvojem podjetju, drugi pa v zaprtem omrežju naročnika. Slednji ima možnost funkcije prevzema (ang. pull) in dostop do GitLab strežnika pri razvijalskem podjetju.

Razvojno podjetje razpolaga le s testno-razvojnim okoljem. Sinhronizacija je zasnovana tako, da na največji možen način lajša razvoj, testiranje in pred-izdajo naročniku. Vodja razvojne skupine izvede ročno izdajo programja pred-izdane različice. To stori z izdelavo kreacije nove značke znotraj strežnika za opravljanje različic v okolju razvojne skupine. Izdaja pred-izdane različice je omogočena le na delih programja, kjer je bil uspešen cikel zvezne integracije. Ob izdaji značke se avtomatsko kreira stisnjena potrditev (ang. squash commit) programske kode v vejo, ki je namenjena sinhronizaciji z naročnikovim strežnikom za upravljanje različic. Sinhronizacija se izvaja z izvedbo sinhronizacijske naloge, ki jo proži časovni razporejevalnik v naročnikovem upravljalniku različic. Izvede se prenos izvorne kode s sinhronizacijo Git repozitorijev.

3. REZULTATI

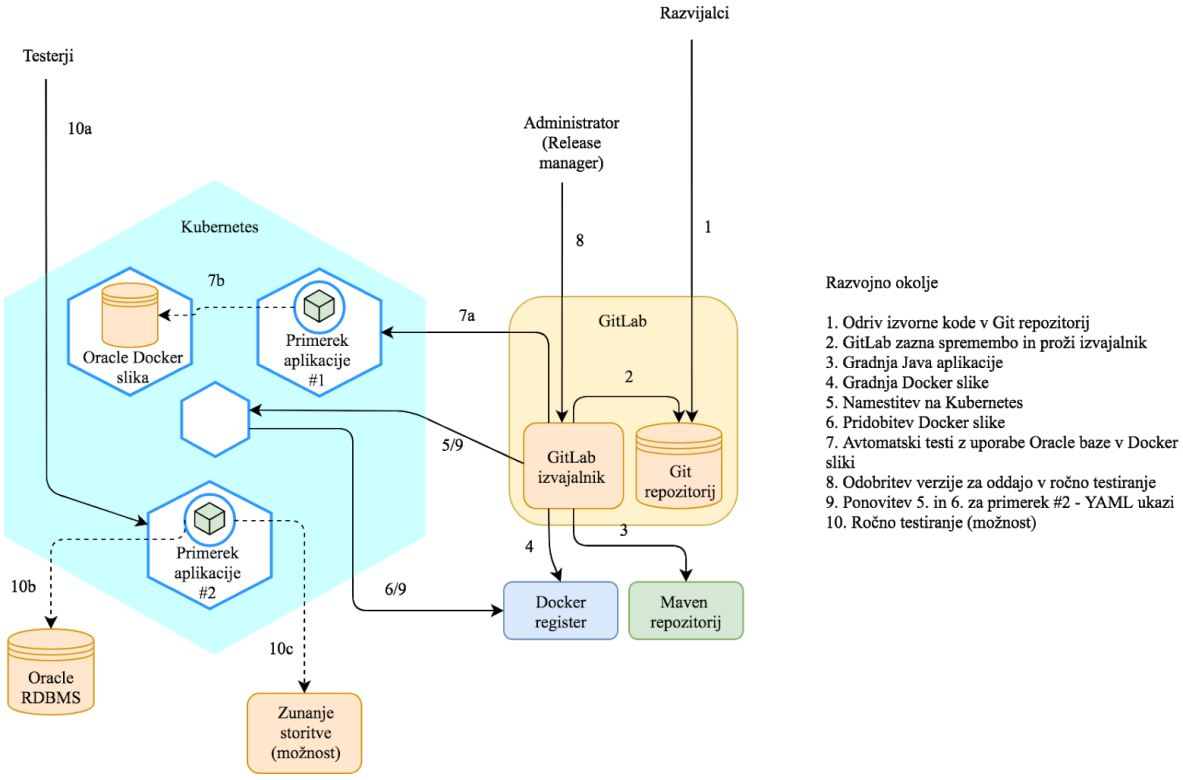
Rezultat tega prispevka opredeljuje izvedba programja osnovanega za GitLab cevovode in Git upravljalnika različic za neprekinjeno dostavo izdaj. Upošteva spremenjen protokol, ki ga zaradi posebnosti v zaprtem kritičnem okolju naročnika implementiramo in omogoča enostavno dostavo/sinhronizacijo izvorne kode programja v upravljalca različic naročnika. Zaradi parametriziranih postopkov programja je mogoče ta program uporabiti na popolnoma ločenih okoljih, le z nastavitvijo konfiguracije na osnovi okoljskih spremenljivk. Razvojne skupine lahko enostavno uporabljajo in tudi same same prispevajo k dopolnitvam ali spremembam cevovoda, saj je le ta vključen v razvojni projekt kot Git pod modul (ang. Git submodule). To predstavlja neodvisen Git projekt, ki vključuje v svojem programju funkcionalnosti, ki so potrebne in osredotočene le na zastavljeno metodologijo neprekinjene dostave in dodatne želje naročnika, ki opredeljujejo naročnikov način dostave nove različice programja v produkcijo.

Razvojne skupine ali podjetja si ponavadi želijo dostaviti svoj izdelek čim bolj dovršen, saj le tako lahko skrajšajo čas od samega razvoja do uveljavitve funkcionalnosti v produkciji. V splošnem si želijo zagotoviti delovanje v skladu z naročnikovimi željami tako iz funkcionalnega, kot tudi tehničnega vidika. Za preverjanje delovanja se pogosto uporabljajo avtomatski in ročni testni postopki, ki jih v delimo na testiranje programskih enot (ang. unit testing), integracijsko testiranje, sistemsko testiranje, sistemsko integracijsko testiranje, regresijsko testiranje, testiranje sprejemljivosti, alfa testiranje, beta testiranje. Način samega testiranja ponavadi opredeli naročnik. V našem primeru uporabljamo testiranje programskih enot, integracijsko testiranje in testiranje sprejemljivosti.

Naročnikova zahteva je, da razvojne skupine same realizirajo in testirajo tako testiranje programskih enot, kakor tudi integracijsko testiranje v testno-razvojem okolju. Šele po uspešni izvedbi navedenih testov lahko realizirajo oddajo izvorne programske kode s pripadajočimi testi. Naročnik nato še na lastnem razvojno-testnem okolju ponovno izvede testiranje osnovnih programskih enot in avtomatsko integracijsko testiranje. V kolikor je testiranje uspešno lahko dostavljeno funkcionalnost tudi namesti v testnem okolju za testiranje sprejemljivosti, ki se izvede ročno po protokolu naročnika. Na tem mestu se morebitne pomanjkljivosti sporoči razvijalcem in se hkrati zahteva njihova odprava.

Testiranje programskih enot in integracijsko testiranje pa od razvojne skupine/podjetja zahteva postavitev razvojno-testnega okolja, kakor tudi GitLab strežnika in GitLab izvajalnika, ki omogoča izvajanje akcij, proženih z odrivom izvorne kode v posamezno vejo Git projekta. Vse to lahko razvojna skupina naredi hitro in učinkovito, saj ima na voljo tudi dokumentirano in skriptirano postavitev oz. namestitev za vse odvisne strežnike in ostale programe, ki sestavljajo razvojno-testno okolje.

Slika 2 prikazuje trenutno zadnjo različico implementacije razvojno-testnega okolja. Upravljalnik različic GitLab mora za nemoteno delovanje cevovoda neprekinjene dostave imeti dostop do GitLab izvajalnika, Docker registra in Maven repozitorija (slednji je potreben za podporo gradnje projektov v jeziku Java in predstavlja strežnik za potrebne knjižnice, ang. dependency). Poleg tega GitLab izvajalnik proži testne protokole dodane funkcionalnosti z namestitvijo razvojne različice programja v čim bolj podobno okolje, kot je produkcijsko - Kubernetes grozd. Skriptirana postavitev Kubernetes grozda omogoča postavitev tovrstnega okolja na lastni ali najeti infrastrukturi razvojne skupine oz. podjetja. S tem razvijalcem omogočimo hitro in učinkovito testiranje novo razvitih funkcionalnosti.

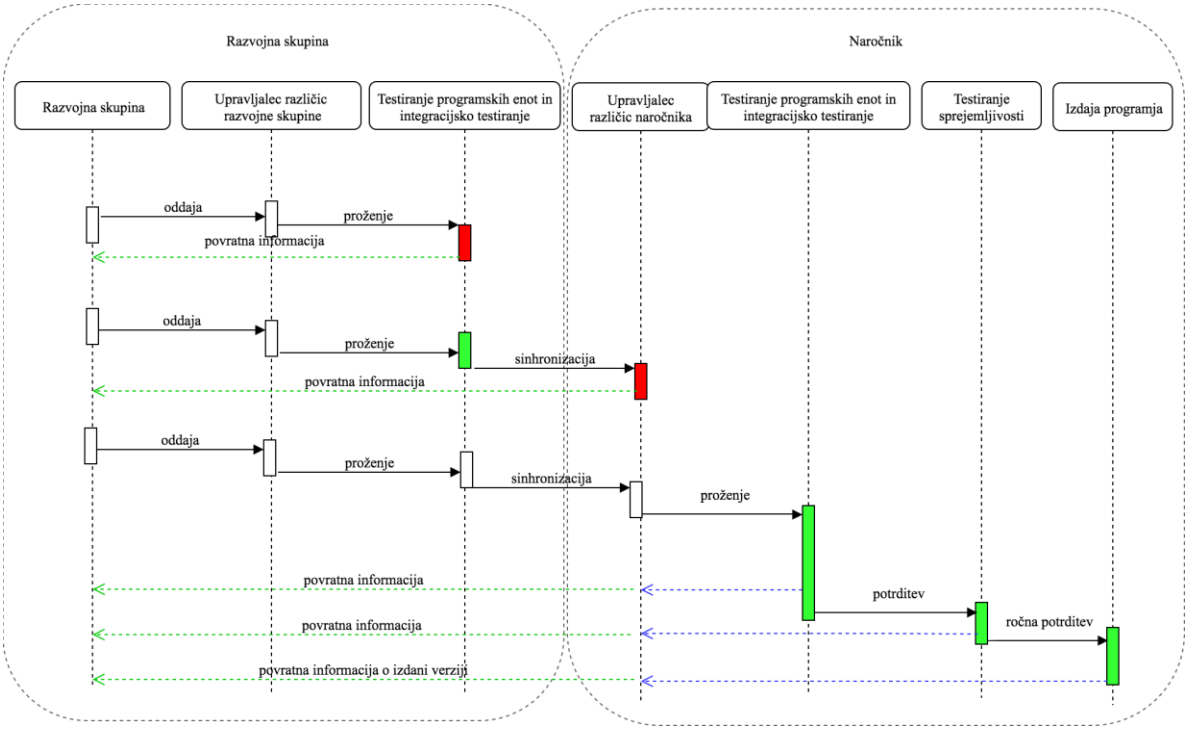


Slika 2. Diagram postavitve razvojnega okolja pri razvojni skupini

Slika 2 poleg podrobnega pregleda vseh odvisnih entitet v razvojno/testnem okolju opisuje tudi razčlenjen potek cevovoda pri testiranju nove funkcionalnosti. Razvijalci odrinejo spremembo izvorne kode v upravljalnike različic GitLab, ki zazna spremembo in proži cikelj neprekinjene integracije. GitLab izvajalnik zgradi Java aplikacije in Docker slike, katere različice odloži v Docker register in

Maven repozitorj. Nato namesti različico aplikacije v okolje Kubernetes. Ta v namestitvenem postopku prevzame zgrajeno različico Docker slike in zažene aplikacijo. Obenem tudi namesti podatkovno zbirko, ki je potrebna za izvedbo testov aplikacije. V primeru uspešno izvedenih testov je možno ročno tudi narediti dodatno značko, ki omogoča ročno testiranje z uporabo zunanje stalne podatkovne zbirke. Vzporedno lahko teče več vzporednih neodvisnih različic aplikacije, ki se jih preizkuša neodvisno.

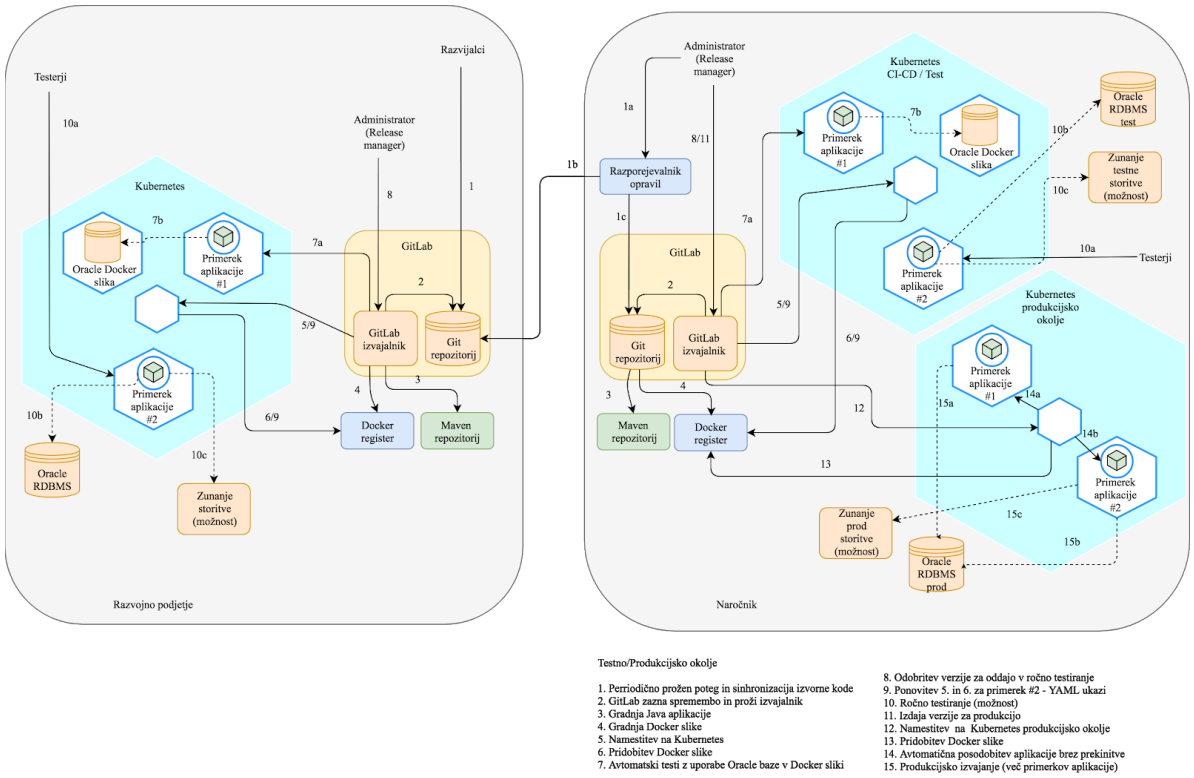
Sekvenčni diagram na sliki 3 opisuje cevovod razvoja nove funkcionalnosti. Posebnost pri razvoju, ki ga izpostavljamo v tem prispevku, je, da razvita funkcionalnost poteka med dvema ločenima okoljema, med okoljem razvojne skupine in testnim okoljem naročnika. Oddaja izvorne kode v naročnikovo okolje definira akcija sinhronizacije na sekvenčnem diagramu na sliki 3. Vsa izvorna koda se izteka v naročnikov upravljalnik različic, kjer se dodatno preizkusi novo funkcionalnost, tudi predhodno funkcionalnost programja in se na ta avtomatičen način izloči slabe kandidate za izdajo. Šele, ko funkcionalnost uspešno prestane tudi teste skladnosti, je nova funkcionalnost avtomatično izdana v novi različici programja. Komunikacija v naročnikovem okolju je zabeležena v upravljalniku različic in tudi hkrati posredovana v upravljalnik različic razvojne skupine.



Slika 3. Sekvenčni diagram neprekinjene dostave v vzporednem kritičnem poslovnem okolju

Avtomatično se zažene naloga ponovne zvezne integracije. Če je ta uspešna, se vzpostavi novo okolje s pripadajočo novo funkcionalnostjo programja v okolju namenjenemu kontroli kakovosti oz. v okolju za izvedbo testov sprejemljivosti. Hkrati je lahko v testnem okolju nameščenih več različnih verzij programja. Vsaka verzija je testirana po protokolu, ki omogoča testiranje nove in starih funkcionalnosti. Če so preizkusi testnih postopkov uspešni, lahko kontrolor kakovosti naredi zahtevek za izdajo nove verzije programa. Kreiranje značke proži avtomatično namestitev programja v produkcijo.

Celoten vpogled v realizirano metodologijo neprekinjene dostave lahko predstavimo na sliki 4. Opazimo vez med okoljem razvojnega podjetja in naročnika, ki je realizirana s pomočjo sinhronizacije izvorne kode med razvojnih podjetjem in naročnikom. Sinhronizacija je izvedena glede na nastavljeno periodično preverjanje sprememb v razvojnem upravljalniku različic v točno določeni veji, ki je namenjena za sinhronizacijo. V kolikor se na tem mestu zazna sprememba, se avtomatično izvede ponovno testiranje programskih enot in integracijsko testiranje v testnem okolju naročnika.



Slika 4. Diagram neprekinjene dostave v vzporednem kritičnem poslovnem okolju

4. ZAKLJUČEK IN DISKUSIJA

Sodoben pristop k razvoju programske opreme za strežnike - DevOps - predvideva, da je okolje produkcijskega izvajanja programja v visoki meri integrirano z okoljem razvoja programja. Javni oblak omogoča upravljanje strežniške infrastrukture s spletnimi upravljalskimi portali (ang. self-service) in programskimi vmesniki (API). Zato je izvedljivo, da iste delovne skupine izvajajo tako razvoj programja ("development") kot upravljanje produkcije programja ("operations") - meja med tema specializacijama je zabrisana.

V svetu programja za podjetja (ang. enterprise software), kjer sta ti dve specializaciji ločeni tudi prek mej organizacij, prinaša uporaba sodobnih pristopov nekaj dodatnih izzivov. Naročnik je podjetje, ki ni specializirano za razvoj programja, zato za ta namen običajno najame zunanje podjetje. Istočasno pa je programje dovolj pomembno za poslovanje, da naročnik zahteva največji možni nadzor nad izvajanjem. Zato razvojna skupina praviloma ne more hkrati še upravljati produkcijskega okolja.

Po drugi strani pa so se uveljavila orodja za porazdeljeno upravljanje različic (ang. distributed version control system), kot je npr. Git [9]. Ta orodja omogočajo, da za isti projekt obstaja več repozitorijev izvorne kode, pri čemer lahko na sistematičen način prenašamo spremembe med njimi.

To omogoča zasnovati neprekinjeno dostavo na tak način, da naročnik v svojem omrežju postavi okolje za gradnjo aplikacij, skupaj z repozitorijem porazdeljega upravljalnika različic, tudi če sam ne izvaja razvoja. Izvajalci razvoja pa potem iz svojih repozitorijev na dogovorjen način in ob dogovorjenih časih dostavljajo kodo v repozitorij, postavljen pri naročniku. Naročnik za tem sam izvaja gradnjo in nameščanje aplikacij s parametriziranimi skripti, ki jih pripravijo izvajalci po njegovih standardih.

Takšna rešitev se ujema z interesom naročnika, da izvaja lastništvo kode (ang. ownership); vedno ima izvorno kodo verzije, ki dejansko teče v produkciji in vedno lahko odloča glede nadaljnjih sprememb te kode. Istočasno pa to ne ovira razvojnega procesa pri izvajalcih, ker lahko izvajalci interno zasnujejo svoje strategije ravnanja s kodo ter neprekinjeno integracijo, ki lahko v odvisnosti od aplikacije dajejo prednost hitrosti razvoja (za npr. uporabniški vmesnik, ki lahko terja hitrejše iteriranje za dosego dobre uporabniške izkušnje) ali morebiti natančnejšemu preverjanju (za bolj kritične dele). Pomembna prednost je, da ob tem ohranimo pomemben varnostni element, da je naročnikovo okolje maksimalno mrežno zaprto.

Kot prednosti predstavljenega pristopa k neprekinjeni dostavi v poslovno kritičnih okoljih lahko izpostavimo hitrejši čas od zaključka razvoja posameznega projekta do namestitve v produkcijskem okolju. Obenem je s parametrizirano skriptirano postavitvijo izboljšana testabilnost, kar omogoča višjo samozavest razvojnikov in višjo stopnjo zaupanja naročnika v kakovost dostavljenega izdelka.

Določitev glavnih komponent izdelave parametriziranega skriptiranega nameščanja ter postavitev potrebnih izvajalnih okolij je zahteven proces, ki terja večja začetna vlaganja in lahko vzame precej časa. Kasneje pa se obrestuje z večjo agilnostjo ne samo razvoja programja, ampak tudi zmožnostjo hitrejšega uvajanja transformativnih sprememb informacijskega sistema v produkcijo. Pri tem istočasno podpira učinkovito delitev dela med različne izvajalce. Čedalje več podjetij za konkurenčnost na tržišču potrebuje tudi odzivnost svojih informacijskih sistemov za podporo poslovanja na hitre spremembe zahtev.

5. LITERATURA

[1] CHEN Lianping, “Continuous delivery: overcoming adoption challenges”, Journal of Systems and Software, letnik 128, junij 2018, str. 72-86

[2] Bass, Len, Ingo Weber inLiming Zhu, DevOps: A Software Architect's Perspective, Addison-Wesley Professional, 2015.

[3] https://kubernetes.io/, Informacije o sistemu Kubernetes, obiskano 15.05.2018

[4] Saito, Hideto, Hui-Chuan Chloe Lee, in Cheng-Yang Wu, DevOps with Kubernetes: Accelerating software delivery with container orchestrators, Packt Publishing Ltd, 2017.

[5] https://github.com/kubernetes-incubator/kubespray, Informacije o namestitvenem postopku Kubespray, obiskano 15.5.2016

[6] https://schd.ws/hosted_files/kccnceu18/87/kubeadm%20deep%20dive%20-%2020180504.pdf, Prestavitev namestitvenega postopka Kubadm na konfenreci KubeCon 2018, obiskano 15.5.2016

[7] Suresh, Salini in Manjunatha Rao, "A Methodical Review Of Virtualization Techniques In Cloud Computing.", International Journal of Computing Science and Information Technology, april 2018, str-. 50-53.

[8] https://github.com/google/gvisor, obiskano 15.5.2016

[9] LOELIGER Jon in MCCULLOUG Matthew, Version Control with Git: Powerful tools and techniques for collaborative software development, O'Reilly Media, Inc., 2012

[10] https://about.gitlab.com, Informacije o aplikaciji Gitlab, obiskano 15.05.2018

OTS 2018 SODOBNE INFORMACIJSKE TEHNOLOGIJE IN STORITVE
ZBORNIK TRIINDVAJSETE KONFERENCE, MARIBOR, 19. IN 20. JUNIJ 2018
M. HERIČKO IN K. KOUS

Univerzitetna založba Univerze v Mariboru

PRIMERJAVA ODPRTOKODNIH PODATKOVNIH MREŽ

BOJAN ŠTOK, CIRIL PETR IN ANDREJ KRAJNC

Povzetek: V prispevku bomo primerjali nekaj popularnih odprtokodnih podatkovnih mrež (ang. data grid). Na običajnih računalnikih nam je v zadnjih letih na voljo vedno večja količina notranjega pomnilnika (RAM), kar nam sicer prinaša vertikalno skaliranje, z večanjem števila strežnikov v mreži, pa imamo tudi možnost horizontalnega skaliranja. Diskovno podprte podatkovne baze vse težje dohajajo vse večje zahteve po vzporedni obdelavi in shranjevanju. Zato večina kombinira podatkovne baze in podatkovne mreže, ki hranijo podatke v pomnilniku. Osnovne funkcionalnosti primerjanih podatkovnih mrež so podobne, razlikujejo pa se v arhitekturi in zmogljivosti. Funkcionalno najbolj bogat je sorazmerno mlad projekt Apache Ignite, ki omogoča enak programski vmesnik kot relacijske baze (SQL, JDBC, ODBC). Medtem ko Redis in Infinispan nista strogo konsistentna (ang. strongly consistent), je Apache Ignite strogo konsistenten in podpira dvofazno potrjevanje (ang. two-phase commit). Najbolj razširjen je Redis, ki pogosto zamenjuje rešitve, ki so temeljile na uporabi Memcached.

Ključne besede: • podatkovne mreže • porazdeljeni predpomnilniki • podatkovne baze • vzporednost • redundanca • skalabilnost

NASLOV AVTORJEV: mag. Bojan Štok, IskraTEL d.o.o., Tržaška c. 37a, 2000 Maribor, e-pošta: stok@iskratel.si. dr. Ciril Petr, IskraTEL d.o.o., Tržaška c. 37a, 2000 Maribor, Slovenija, e-pošta: petr@iskratel.si. mag. Andrej Krajnc, IskraTEL d.o.o., Tržaška c. 37a, 2000 Maribor, Slovenija, e-pošta: krajnc@iskratel.si.

DOI https://doi.org/10.18690/978-961-286-162-9.18 ISBN 978-961-286-163-6

Dostopno na: http://press.um.si.

1. UVOD

Eden od mehanizmov za pohitritev aplikacij je uporaba predpomnilnika (cache). Tako nam v aplikaciji ni potrebno pri vsaki zahtevi izvesti dostopa do podatkovne baze bodisi SQL ali NoSQL.

Predpomnilnik je strojna ali programska komponenta, ki shrani podatke za bodočo uporabo na način, da bodo ti podatki dostavljeni hitreje. Podatki v predpomnilniku so lahko rezultat predhodnih izračunavanj ali pa so duplikat podatkov, ki so shranjeni nekje drugje. Primeri strojnih predpomnilnikov so procesorski (CPU) predpomnilnik, grafični (GPU) predpomnilnik, mikroprocesorski DSP predpomnilnik itd. Primeri programskih predpomnilnikov so diskovni predpomnilnik (disk cache), spletni predpomnilnik (web cache), DNS, iskalni predpomnilniki, bazni predpomnilniki itd.

V tradicionalnih aplikacijah ni bilo takšne potrebe po uporabi predpomnilnika, saj so ponavadi klasični pristopi zadoščali za izdelavo performančno dovolj zmogljivih aplikacijah. Sodobne aplikacije dandanes so pogosto zelo kompleksne, delajo z velikimi količinami podatkov, uporabljajo porazdeljene arhitekture, povezujejo najrazličnejše storitve, vse skupaj pa mora biti tudi performančno učinkovito. Da dosežemo želene cilje, se moramo pogosto odločiti za uporabo predpomnilnikov, še posebej je vedno večja potreba po uporabi porazdeljenih predpomnilnikov.

Porazdeljeni predpomnilniki so razširitev tradicionalnega koncepta predpomnilnika, ki se uporablja le na enem računalniku. Porazdeljen predpomnilnik se lahko razširja čez več strežnikov in na ta način omogoča shranjevanje večje količine podatkov, hkrati pa omogoča večje transakcijske kapacitete. Največkrat gre pri porazdeljenih pomnilnikih za shranjevanje podatkov, ki so sicer shranjeni v podatkovni bazi ali pa gre za podatke o spletnih sejah. Ideja o porazdeljenih predpomnilnikih je vedno bolj zanimiva tudi zaradi tega, ker je postala strojna oprema (predvsem diski) cenovno ugodna, omrežja pa so hitrejša (1 GBit je standard, 10 GBit je vedno bolj pogost). Porazdeljeni predpomnilniki delajo dobro tudi na cenovno ugodnih računalnikih, medtem ko bazni strežniki pogosto zahtevajo drago strojno opremo. Na področju porazdeljenih predpomnilnikov je na voljo veliko produktov, pri čemer so nekateri na voljo že dalj časa, vedno znova pa se pojavljajo novi produkti, ki se želijo uveljaviti na tem področju. Trenutno najbolj razširjeni produkti so Infinispan, Redis, Apache Ignite, Memcached, Oracle Coherence, Riak, Couchbase, Aerospike itd.

V zadnjem času se vse bolj uveljavljajo podatkovne mreže (data grid), predvsem v situacijah, ko nam ni dovolj zgolj osnovna funkcionalnost porazdeljenih pomnilnikov (manipulacija podatkov preko put in get). Podatkovne mreže so nadgradnja porazdeljenih pomnilnikov, saj ponujajo vse osnovne funkcionalnosti porazdeljenih pomnilnikov, hkrati pa omogočajo računalniško obdelavo v predpomnilniku shranjenih podatkov. Tako lahko nad podatki izvajamo razna kompleksna indeksiranja, Map Reduce procesiranja, omogočen je porazdeljen SQL ipd. Tako fokus tovrstnih orodij ni več zgolj čisto upravljanje s podatki, temveč se fokus premika v smer hibridnega upravljanja s podatki in s procesiranjem podatkov. Uporaba tovrstnih orodij predstavlja nov konceptualni preskok v razvoju programskih rešitev.

2. PRIMERJAVA REŠITEV

V prispevku bomo predstavili in primerjali tri implementacije: Infinispan, Redis in Apache Ignite. Infinispan in Apache Ignite sta napisana v programskem jeziku Java, zato seveda podpirata tudi JSR 107 - Java Caching API. Redis je napisan v ANSI C-ju. Kljub temu pa je pred približno enim letom dobil podporo tudi za JSR 107. Redis je verjetno najbolj popularna implementacija podatkovne mreže. Pogosto zamenjuje rešitve, ki so temeljile na uporabi Memcached. Omogoča tudi, da se obstoječi Memcached odjemalec (v kateremkoli jeziku) priključi na Redis.

Zmogljivost oz. hitrost podatkovne mreže je precej odvisna od narave aplikacije (veliko ali majhno število vpisov, pomembnejša zanesljivost (redundanca) ali hitrost branja, ….). Zato moramo podatkovno mrežo konfigurirati tako, da dobimo optimalne zmogljivosti za našo aplikacijo.
Podatkovno mrežo tipično uporabljamo za naslednje primere uporabe:

- Porazdeljeni predpomnilnik (ang. distributed cache), pogosto pred podatkovno bazo
- Shramba za začasne podatke kot so npr. seje
- Obdelava podatkov v pomnilniku (ang. In-memory data processing) in analitika
- Komunikacija med JVM stroji in deljena shramba
- Map Reduce implementacija v podatkovni mreži pomnilnika (ang. in-memory data grid)

3. INFINISPAN

Infinispan je naslednik JBoss Cache. Projekt se je startal leta 2009 [1] pod okriljem podjetja Red Hat. Infinispan teče znotraj aplikacijskega strežnika Wildfly. Če naša aplikacija teče znotraj aplikacijskega strežnika Wildfly, pomeni, da lahko uporabljamo Infinispan kot lokalni ali porazdeljeni predpomnilnik. Če pa želimo, da aplikacija teče ločeno od podatkovne mreže oz. aplikacija ni napisana v programskem jeziku java, namestimo Infinispan server in uporabimo oddaljen dostop do podatkovne mreže preko protokola Hot Rod. Poleg Jave so podprti še naslednji programski jeziki: Ruby, Python, C++, C#, Javascript.

Infinispan lahko uporabimo kot lokalni (standalone). Tipično ga uporabljamo v gruči (cluster). Gručo lahko konfiguiramo na naslednje načine [2]:

- Način razveljavitve (invalidation mode). V tem načinu se med vozlišči ne delijo podatki. Infinispan le zagotavlja, da ko je podatek vpisan v neko vozlišče, razveljavi podatek v ostalih vozliščih. Ta način je uporaben, ko imamo permanentne podatke shranjene npr. v podatkovni bazi, Infinispan pa se uporablja le za optimizacijo dostopa. Vedno, ko se podatek spremeni, se pošlje sporočilo o razveljavitvi na ostala vozlišča. To je učinkovito, saj pošljemo le sporočilo o razveljavitvi, ne pošljemo pa celotne vrednosti (npr. vrstice). Ta način se lahko uporablja tudi s ClusterLoader nalagalnikom. V tem primeru, če ključ ni v lokalnem vozlišču, se bo pridobil podatek iz drugega vozlišča.
- Repliciran način (replicated mode) - podatek vpisan na kateremkoli vozlišču se pošlje vsem ostalim vozliščem. Pridobivanje tega podatka pa je potem hitro, saj ga vedno pridobimo lokalno. To je uporabno le za gruče z največ 10 vozlišči in za primere, ko imamo sorazmerno malo vpisov.
- Porazdeljen način (distributed mode) - poskuša hraniti ključ v "numOwners" vozliščih. To nam omogoča linearno skalabilnost in hranjenje več podatkov, če dodamo več vozlišč. Število kopij (numOwners) je kompromis med zmogljivostjo in zanesljivostjo (durability). Kopije se particionirajo glede na konfiguracijo (KeyPartitioner). Prvo vozlišče, ki hrani podatek je primarni lastnik (primary owner), ostala vozlišča, ki hranijo kopijo so rezervni lastniki (backup owners).
- Razpršeni način (scattered mode) - je podoben porazdeljenemu načinu, vzdržuje pa le dve kopiji (numOwners=2). Za razliko od porazdeljenega načina, lokacija podatka ni fiksna. Primarni lastnik se določi po enakem algoritmu, rezervna kopija pa se hrani v vozlišču, ki je zadnji shranil podatek.

Neodvisno od tega pa lahko uporabljamo asinhroni in sinhroni način. Pri asinhronem načinu uporabniška nit ni blokirana. Vendar se moramo v tem primeru zavedati, če kličemo cache.put(k1, v1); cache.put(k1,v2); da ne vemo ali se bo vpisala vrednost v1 ali v2.

Ko razpakiramo infinispan-server-9.2.2.Final-bin.zip ga zaženemo s parametrom, kjer povemo pot do konfiguracijske datoteke. Uporabimo clustered.xml, kjer je že konfigurirana gruča.

```
bin/standalone.sh -c clustered.xml
```

Tako na različnih strežnikih zaženemo več instanc. Gruča se avtomatsko vzpostavi preko multicast omrežja. V konfiguracijski datotetki je privzeta nastavljena naslednja multicast grupa 234.99.54.14 in port 45700. Za Hod Rod protokol je privzeto nastavljen port 11222. Konfiguriran je tudi že port 11211

za memcached protokol. Če želimo oddaljeno dostopati do podatkovne mreže, moramo konfigurirati interaface "public", zamenjamo locahost z IP-jem.

Primer:

```
   <interface name="public">
           <inet-address
value="${jboss.bind.address:172.18.120.50}"/>
   </interface>
```

Če uporabimo Hot Rod odjemalca (maven knjižnica name: infinispan-cachestore-remote, version: 9.2.2.Final) v datoteki hotrod-client.properties naštejemo seznam IP-jev. Načeloma je dovolj IP le enega živega strežnika.

Primer datoteke hotrod-client.properties:

```
infinispan.client.hotrod.server_list=172.18.120.50:11222;172.18.120.51:11222;172.18
.120.52:11222
```

4. REDIS

Ime Redis pomeni **RE**mote **DI**ctionary **S**erver. Začel se je razvijati leta 2010 pod okriljem podjetja VMware. Od leta 2013 ga sponzorira Pivotal Software.

Redis se primarno uporablja za izboljšanje performans v aplikacijah za predpomnilnik. Podatke lahko shranimo tudi persistentno. Poleg enostavnega tipa kot je string, omogoča tudi hranjenje bolj kompleksnih tipov kot so: seznami (lists), mape (maps), množice (sets) in bitne mape. Za vsakega od teh tipov omogoča tudi posebni API, ki omogoča "delni" vpis in "delno" branje iz te strukture.

Za začetek je zelo uporaben redis-cli, s pomočjo katerega testiramo funkcionalnosti Redis [3]. Primeri uporabe

- key/value
  - **set** k v
  - **get** k
- liste
  - **rpush** mylist a // dodamo na konec seznama element a v listo s ključem mylist
  - **lpush** mylist b // dodamo na začetek seznama element b v listo s ključem mylist
  - **lrange** mylist 0 -1 // vrne vse elemente od začetka do konca liste mylist
  - **lrange** mylist 2 5 // vrne elemente mylist[2], mylist[3], mylist[4], mylist[5]
  - **rpop** mylist // vrne in odstrani zadnji element iz liste
  - **lpop** mylist // vrne in odstrani prvi element iz liste
- mape
  - **hmset** k a1 v1 a2 v2 a3 v3 // v mapo dodamo element s ključem k in vrednost v1 za atribut a1, vrednost v2 za attribut a2 ter vrednost v3 za atribut a3
  - **hget** k a1 // vrne vrednost atrbuta a1 za ključ k
  - **hgetall** k // vrne vse atribute za ključ k
- množice
  - **sadd** myset a b c // dodamo tri elemente: a, b, c v množico s ključem myset
  - **srem** myset b // odstrani element b iz množice myset
  - **smembers** myset // vrne vse elemente množice s ključem myset
  - **sismember** myset a // vrne 1, če je "a" element množice myset, sicer vrne vrednost nič
  - **scard** myset // vrne število elementov množice myset
  - **sinter** myset1 myset2 myset3 // vrne presek - elemente, ki so v vseh treh množicah
  - **sunion** myset1 myset2 // vrne unijo - elemente iz obeh množic
  - **sdiff** myset1 myset2 // vrne elemente iz množice myset1, ki niso v myset2
- administrativni ukazi

- **cluster** nodes // izpiše seznam vseh vozlišč (master in slave)
- **cluster** info // informacije o gruči
- **cluster** replicate <node-id> // vozišče postane slave za podan master node-id
- **cluster** slaves <node-id> // izpiše seznam slave vozlišč za podan master node-id

Obstajajo odjemalske knjižnice za večino programskih jezikov: ActionScript, Bash, C, C#, C++, Clojure, Lisp, Dart, Delphi, Erlang, Go, Haskell, Java, Julia, Objective-C. Pascal, Perl, PHP, Python, R, Ruby, Scala, Smalltalk, Swift, VB in drugi.

Za Java programski jezik obstaja več knjižnic. Mi smo uporabili knjižnico Jedis (group: redis-client, name: jedis, version 2.9.0.).

Imena metod so enaka kot cli ukazi.

Primeri:

```
jedis.set("car", "Toyota");
jedis.get("car");
jedis.rpush("mylist", x);
List<String> list = jedis.lrange("mylist", 0, -1);
Map<String,String> m = new HashMap<>();
m.put("name", "Bill");
m.put("age", "40");
jedis.hmset("h1", m);
jedis.hget("h1", "name")
jedis.sadd("myset", "a", "b");
jedis.sadd("myset", "c", "d", "e");
System.out.println("is member: " + jedis.sismember("myset", "b"));
jedis.smembers("myset"));
```

Redis particionira podatke glede na število master vozlišč. Za vsakega masterja lahko določimo enega ali več repliciranih strežnikov (slave). Če ne določimo vsaj enega strežnika, ob izpadu i-tega master strežnika, izgubimo vse ključe i-te particije. Med masterjem in slaveom se privzeto uporablja asinhrona komunikacija.

Instalacija je malo bolj zahtevna. Na enem strežniku lahko zaganjamo več instanc Redis strežnika. Zato tipično uporabimo številko porta za ime konfiguracije. Privzet port je 6379. Pred prvim zagonom je potrebno popraviti konfiguracijsko datoteko: /etc/redis/6379.conf.

V konfiguraciji ponavadi popravimo naslednje parametre:

- **bind** - IP naslov, privzeto localhost
- **deamonize** na yes, privzeto no
- spremenimo **loglevel** in **logfile** postavimo na /var/log/redis_6379.log
- syslog-enabled postavimo na yes
- pomembno je, da parameter **dir** postavimo na direktorij, kjer se naj shranjujejo podatki, tipično na: /var/redis/6379
- **cluster-enables** postavimo na yes
- **cluster-config-file** nodes-6370.conf - s tem parametrom povemo le v katero datoteko si bo Redis vpisal konfiguracijo. Datoteke nodes-6370.conf ne spreminjamo sami.
- s parametrom **save** določimo po koliko spremembah se shranijo podatki na disk: save <seconds> <changes>

Privzeto so nastavljene naslednje vrednosti za shranjevanje:.

```
save 900 1
save 300 10
```

```
save 60 10000
```

Redis streže odjemalcem na TCP portu s privzeto vrednostjo 6379. Za komunikacijo med strežniki v clustru pa se uporablja port 16379 (portu prištejemo 10000).

Ko smo pripravili konfiguracijo, zaženemo Redis deamone na vseh vozliščih:

- `/etc/init.d/redis_6379 start`

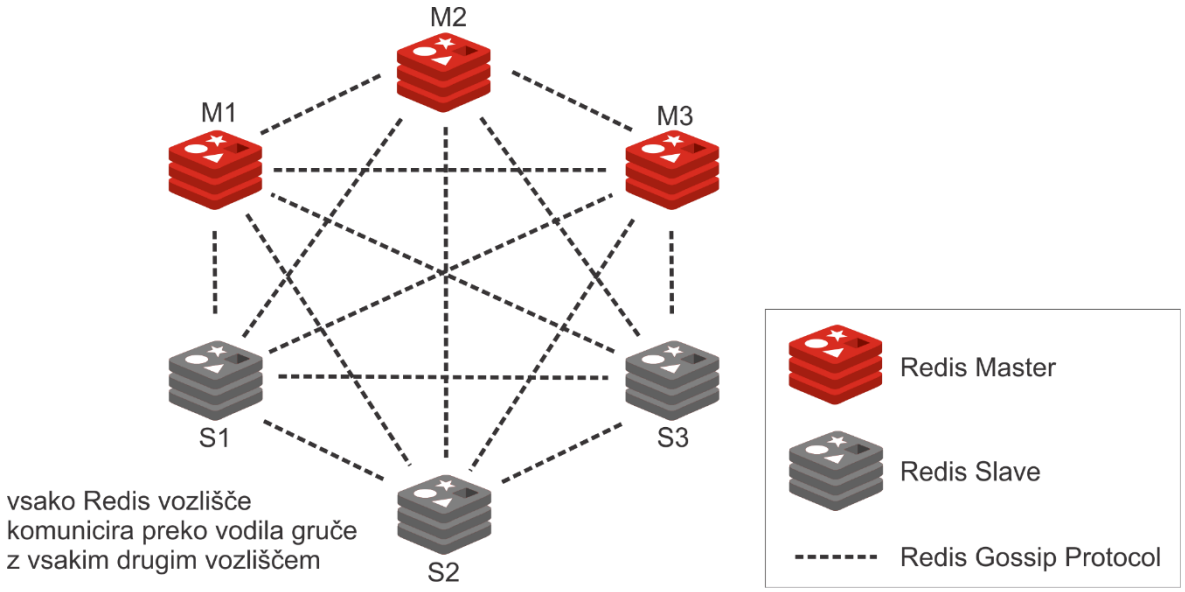
Vozlišča se med seboj še ne poznajo. Potrebno je konfigurirati katera vozlišča bodo masterji, katera pa slave. Pri konfiguriranju gruče si lahko pomagamo z Ruby skripto redis-trib [4].

Če želimo postaviti 3 mastereje in 3 slave, moramo imeti 6 vozlišč. Izvedemo naslednji ukaz:

```
./redis-trib.rb create --replicas 1  172.18.120.50:6379 172.18.120.51:6379
172.18.120.52:6379 72.18.120.53:6379 172.18.120.54:6379 172.18.120.55:6379
```

Če želimo postaviti 3 masterje brez slave vozlišč, izvedemo naslednji ukaz:

```
./redis-trib.rb create --replicas 0  172.18.120.50:6379 172.18.120.51:6379
172.18.120.52:6379
```



Slika 1. Okolje minimalne Redis gruče

Minimalno Redis gručo sestavljajo tri master vozlišča vsak s svojim slave vozliščem. Vsakemu master vozlišču je dodeljena poddomena med 0 in 16.384. Na obeh master in slave vozliščih tečeta dva TCP servisa. Prvi sprejema REST sporočila, drugi pa predstavlja komunikacijsko vodilo za gručo, po katerem teče komunikacija po Redis Gossip protokolu.

V Redis gruči imamo 16384 slotov. Da bi izračunali kakšen je »hash slot« za nek ključ, enostavno izračunamo CRC16 ključa z modulom 16384 [4].

Tako je vsako Redis vozlišče odgovorno za poddomeno hash funkcije. Če imamo tri master vozlišča, potem:

- vozlišče A vsebuje poddomeno od 0 do 5500
- vozlišče B vsebuje poddomeno od 5501 do 11000
- vozlišče C vsebuje poddomeno od 11001 do 16383.

5. APACHE IGNITE

Apache Ignite je začelo razvijati podjetje GridGain System. V Apache incubator program se je vključil konec leta 2014. Prva verzija je bila izdana leta 2015.

Apache Ignite se lahko uporablja kot podatkovna mreža, lahko pa tudi kot in-memory porazdeljena SQL podatkovna baza [5].

Ignite obravnava pomnilnik (RAM) ne samo kot predpomnilnik ampak polno funkiconalno shrambo. Persistenca je lahko vključena ali izključena. Če je persistenca izključena, potem lahko obravnavamo Apache Ignite kot bazo podatkov v pomnilniku (in-memory database - SQL) ali kot podatkovno mrežo v pomnilniku (in-memory data grid - key-value API). Če pa persistenco vključimo, potem Ignite postane porazdeljena horizontalno skalabilna podatkovna baza, ki omogoča konsistenco podatkov in je odporna na napake (resilient to full cluster failures).

Persistenco vključimo s konfiguracijo parametra persistenceEnable:

```
   <bean class="org.apache.ignite.configuration.DataStorageConfiguration">
      <property name="defaultDataRegionConfiguration">
        <bean
class="org.apache.ignite.configuration.DataRegionConfiguration">
          <property name="persistenceEnabled" value="true"/>
        </bean>
      </property>
    </bean>
```

Če imamo 1000 vpisov in kapaciteta pomnilnika omogoča, da shranimo le 200, potem bo vseh 1000 shranjenih na disku, v pomnilniku pa bo shranjenih v predpomnilniku le 200 vpisov.

Ker je arhitektura pomnilniško naravnana (memory-centric), se RAM uporablja kot prvi pomnilnik, kjer se izvaja procesiranje. Vsi podatki in indeksi so shranjeni izven Java kopice, kar omogoča procesiranje petabyte (PB) podatkov, ki so shranjeni v gruči.

Ignite lahko opcijsko razlikuje med odjemalcem in strežniškimi vozlišči. Strežniki izvajajo caching, izvajanje (compute exceuting), obdelavo tokov (stream processing) itd., medtem ko se odjemalci povežejo oddaljeno na strežnike.

Apache Ignite implementira tudi Hadoop-ov MapReduce API, ki omogoča bistveno hitrejše izvajanje kot običajne (native) Hadoop MapReduce implementacije. Poleg tega IGFS (Ignite File System) ne potrebuje imenskega vozlišča (name node), ko se uporablja IGFS, kar pomeni, da gredo Ignite MapReduce opravila direktno v IGDS podatkovno vozlišče (data node).

Ko namestimo Apache-Ignite, najdemo na direktoriju /usr/share/apache-ignite/libs/optional precej opcijskih komponent. Za začetek je koristno, če si namestimo opcijski REST vmesnik. Iz poddirektorija optional prenesemo celotni direktorij na nivo više.

Primeri uporabe REST vmesnika [6]

- podatki o topologiji, seznam vseh vozlišč v gruči
  `curl "http://172.18.120.50:8080/ignite?cmd=top"`
- podatki o vozlišču
  `curl "http://172.18.120.50:8080/ignite?cmd=node&id=8b48b419-bfea-46fd-bb9a-6730e"`
- kreiranje cache
  `curl "http://172.18.120.50:8080/ignite?cmd=getorcreate&cacheName=mycache"`

- brisanje cache
  ```
  curl
  "http://172.18.120.50:8080/ignite?cmd=destcache&cacheName=mycache"
  ```
- vpis vrednosti "newValue" v ključ newKey
  ```
  curl
  "http://172.18.120.50:8080/ignite?cmd=put&key=newKey&val=newValue&cac
  heName=mycache"
  ```
- branje vrednosti ključa newKey
  ```
  curl
  "http://172.18.120.50:8080/ignite?cmd=get&key=newKey&cacheName=mycach
  e"
  ```

Na direktoriju /etc/apache-ignite imamo lahko več konfiguracijskih datotek. Privzeta datoteka je default-config.xml. Ignite servis zaženemo z ukazom:

```
systemctl start apache-ignit@<config-name>
```

Privzeto konfiguracija datoteka je prazna in je primerna za zagon standalone strežnika. Za postavitev clustra lahko uporabimo multicast odkrivanje vozlišč, statično odkrivanje (discovery) s seznamom IP-jev, ali kombinacijo multicast in seznama IP-jev [7].

Če so vsi strežniki v isti multicast skupini, je najbolj enostavno uporabiti multicast odkrivanje (TcpDiscoveryMulticastIpFinder). S parametrom muliticast group določimo multicast grupo.

```
<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="
       http://www.springframework.org/schema/beans
       http://www.springframework.org/schema/beans/spring-beans.xsd">
    <bean id="grid.cfg" class="org.apache.ignite.configuration.IgniteConfiguration">
     <property name="discoverySpi">
       <bean class="org.apache.ignite.spi.discovery.tcp.TcpDiscoverySpi">
        <property name="ipFinder">
          <beanclass="org.apache.ignite.spi.discovery.tcp.
                ipfinder.multicast.TcpDiscoveryMulticastIpFinder">
            <property name="multicastGroup" value="228.10.10.157"/>
          </bean>
        </property>
       </bean>
      </property>
    </bean>
</beans>
```

Aplikacija lahko teče v istem JVM kot Apache-Ignite. V našem primeru pa smo želeli, da je aplikacija le odjemalec in se oddaljeno poveže v podatkovno mrežo. Ker je bil odjemalec v drugem multicast omrežju smo v odjemalcu uporabili IP-je strežnikov (TcpDiscoveryVmIpFinder). Dovolj je navesti IP vsaj enega živega Apache-Ignite strežnika.
Uporabili smo knjižnico ignite-core (group: org.apache.ignite, name: ignite-core, version: 2.4.0)

Primer inicializaciijske kode v odjemalcu:

```
IgniteConfiguration cfg = new IgniteConfiguration();
cfg.setClientMode(true);  // client mode
TcpDiscoverySpi discoSpi = new TcpDiscoverySpi();
TcpDiscoveryVmIpFinder ipFinder = new TcpDiscoveryVmIpFinder();
```

```
List<String> address= Arrays.asList("172.18.120.52", "172.18.120.51",
"172.18.120.50");
ipFinder.setAddresses(address);
discoSpi.setIpFinder(ipFinder);
cfg.setDiscoverySpi(discoSpi);
Ignite ignite = Ignition.start(cfg);
```

Ignite pozna tri načine delovanja predpomnilnika:

- CacheMode.LOCAL - lokalni predpomnilnik
- CacheMode.REPLICATED - vsi ključi se replicirajo na vsa vozlišča. To je primerno za "manjše" podatkovne sete, kjer imamo precej več branj kot vpisovanj
- CacheMode.PARTITIONED - množica ključev je razdeljena v particije med vsemi vozlišči. Podatek se hrani na primarnem vozlišču. Opcijsko pa na enem ali več rezervnih. To je primerno za "zelo" velike sete podatkov. Spremembe podatkov so lahko tudi pogoste.

Za vsak cache lahko določimo drug način. Če načina pri kreiranju ne navedemo, je privzeti način PARTITIONED.

Primer kreiranja predpomnilnika:

```
CacheConfiguration cc = new CacheConfiguration(CACHE_NAME);
cc.setCacheMode(CacheMode.PARTITIONED);
cc.setBackups(1); // privzeto število rezervnih vozlišč je nič
cc.setAtomicityMode(TRANSACTIONAL); // privzeti način je ATOMIC
cache = ignite.getOrCreateCache(cc);
```

Primer uporabe predpomnilnika:

```
cache.put("car", "Toyota");
String car = cache.get("car");
```

Apache Ignite podpira, da se lahko obstoječi Memcached ali Redis odjemalci povežejo na Ignite. Pri Redisu je nekaj omejitev, ker niso podprti čisto vsi ukazi.

6. PRIMERJAVA

V naslednji tabeli smo poskušali primerjati različne parametre. Številke v tabeli pomenijo vrstni red: 1 – prvi (najboljši), 2 - drugi, 3 – tretji.

Tabela 1. Primerjava lastnosti

	Infinispan	Redis	Apache Ignite
Programski jezik	Java	C	Java
Popularnost/razširjenost	2	1	3
Enostavna instalacija/konfiguracija	1	3	2
Zmogljivost	3	1	2
Podprti programski jeziki	2	1	3
Podprte funkcionalnosti	3	2	1
Java knjižnica za odjemalca	infinispan-cachestore-remote	Jedis	Ignite-core
Možnost izvajanja aplikacije znotraj podatkovne mreže	da	ne	da

	Infinispan	Redis	Apache Ignite
Podpora za Memcached odjemalce	da	da	da
Uporabljena verzija	9.2.2 (prva verzija 4.0)	4.0.8	2.4.0
Začetek razvoja	2009	2010	2014
Sponzor	Red Hat	Pivotal Software	GridGain System

Izvedli smo tudi performančno primerjavo funkcionalnosti, ki jih implementirajo vse tri implementacije: vpis put(key,value) in branje get(key). Primerjavo smo izvedli tako, da smo namestili podatkovno mrežo na tri starejše strežnike (8GB RAM, 2 core). Med njimi je bila vzpostavljena 100 MB povezava. V praksi se tipično uporablja 1GB ali pa celo 10GB povezava. V primeru hitrejše povezave bi bili rezultati seveda še precej boljši. V primeru Infinispan in Ignite bi lahko naša aplikacija tekla kar znotraj istega JVM navideznega stroja, v tem primeru bi bili rezultati še boljši. Mi pa smo aplikacijo namestili v drugo omrežje, ki je bilo povezano s podatkovno mrežo preko 100 MB povezave. Tako je multicast omrežje delovalo le med strežniki, ki so sestavljali podatkovno mrežo.

Za nastavitev podatkovnih mrež smo uporabili privzete vrednosti. Za dostop do Infinispan smo uporabili oddaljen dostop oz. Hot Rod protokol. Za dostop do Redis smo uporabili knjižnico Jedis. Pri Apache Ignite smo uporabili odjemalski način in knjižnico ignite-core.

Test smo izvedli tako, da smo v zanki izvedli tisoč ali milijon ponovitev na različnih ključih v eni nit ter test ponovili s sto hkratnimi nitmi.

Tabela 2. Primerjava branje/pisanje

	Branje (get) [sec]	Pisanje (put) [sec]	100 hkratnih branj [sec]	100 hkratnih pisanj [sec]
Infinispan n=1000	1,82	2,59	0,10	0,28
Infinispan n=1000000	439,36	1.809,37	161,34	192,79
Redis n=1000	**0,90**	**0,92**	**0,08**	**0,08**
Redis n=1000000	**945,80**	**915,81**	101,82	102,92
Apache Ignite Replicated n=1000	1,73	1,91	0,14	0,19
Apache Ignite Replicated n=1000000	1.652,57	1.728,36	63,08	55,85
Apache Ignite Partitioned n=1000	1,81	1,77	0,11	0,14
Apache Ignite Partitioned n=1000000	1.619,76	1.689,41	**61,70**	**52,73**

Pokazalo se je, da je Redis skoraj dvakrat hitrejši od ostalih dveh. Pri milijon ponovitvah in 100 nitih pa je bil Apache Ignite hitrejši od Redis. Infinispan je bil cca. 10% počasnejši od Apache Ignite. Na treh strežnikih in sorazmerno počasni povezavi (100MB) se pri pisanju ni opazila razlika, če smo uporabili

predpomnilnika tipa Cached ali Replicated. Rezultati se seveda lahko bistveno spremenijo, če spremenimo konfiguracijo podatkovne mreže.

7. ZAKLJUČEK

Vse tri implementacije dobro opravljajo svojo nalogo, odločitev za izbiro podatkovne mreže pa je odvisna od naših potreb. Če naše aplikacije oz. mikroservisi tečejo znotraj aplikacijskega strežnika Wildfly, priporočamo uporabo Infinispan. Če pa imamo mikroservise implementirane v različnih programskih jezikih ali pa ne uporabljamo aplikacijskega strežnika Wildfly, priporočamo uporabo Redis. Zavedati se moramo, da je za postavitev strežnika Redis v produkcijskem okolju potrebnih vsaj šest strežnikov. Če poleg podatkovne mreže želimo uporabljati SQL podatkovne baze v pomnilniku, ali želimo hitro SQL podatkovno bazo za analitiko, je smiselno izbirati Apache Ignite.

8. LITERATURA

[1] Infinispan: the Start of a New Era in Open Source Data Grids, https://blog.infinispan.org/2009/04/, obiskano 12. 5. 2018.

[2] Infinispan 9.2 User Guide, http://infinispan.org/docs/stable/user_guide/user_guide.html, obiskano 12. 5. 2018.

[3] Redis commands, https://redis.io/commands, obiskano 12. 5. 2018.

[4] Redis Cluster Tutorial https://redis.io/topics/cluster-tutorial, obiskano 12. 5. 2018.

[5] Apache Ignite, Documentation, https://apacheignite.readme.io/docs, obiskano 12. 5. 2018.

[6] Apache Ignite, Cluster Configuration, https://apacheignite.readme.io/docs/cluster-config, obiskano 12. 5. 2018.

[7] Apache Ignite REST API, https://apacheignite.readme.io/v2.4/docs/rest-api, obiskano 12. 5. 2018.

Univerzitetna založba Univerze v Mariboru

UMETNA INTELIGENCA ZA TELEBANE – PLATFORME STROJNEGA UČENJA

SAŠO KARAKATIČ, GREGA VRBANČIČ, JERNEJ FLISAR IN VILI PODGORELEC

Povzetek: Uporaba procesov strojnega učenja za večino podjetij predstavlja velik izziv, saj zahteva spremembo obstoječih poslovnih procesov, drag in talentiran kader ter njihovo dodatno izobrazbo. Najnovejše tehnike strojnega učenja predstavljajo velik napredek pri obdelavi podatkov, vendar je področje preobširno za spoznavanje »čez vikend« ali »v popoldanskem času« za ljudi, ki niso strokovnjaki s področja podatkovne znanosti (angl. Data Science). Če gradimo nov Uber, Netflix ali Facebook je zahtevnost res ogromna, ampak uporaba obstoječih storitev strojnega učenja znatno olajša ta postopek. Z uporabo oblačnih storitev strojnega učenja lahko začnemo graditi svoje prve modele umetne inteligence in tako vključiti dragocene napovedi in inteligenco v obstoječe sisteme. V članku bomo razpravljali o obstoječih platformah strojnega učenja – tako o spletnih kakor tudi o lokalnih.

Ključne besede: • strojno učenje • podatkovna znanost • platforme • storitve

NASLOV AVTORJEV: dr. Sašo Karakatič, docent, Univerza v Mariboru, Fakulteta za elektrotehniko, računalništvo in informatiko, Koroška cesta 46, 2000 Maribor, Slovenija, e-pošta: saso.karakatic@um.si. Grega Vrbančič, mladi raziskovalec, Univerza v Mariboru, Fakulteta za elektrotehniko, računalništvo in informatiko, Koroška cesta 46, 2000 Maribor, Slovenija, e-pošta: grega.vrbancic@um.si. Jernej Flisar, asistent, Univerza v Mariboru, Fakulteta za elektrotehniko, računalništvo in informatiko, Koroška cesta 46, 2000 Maribor, Slovenija, e-pošta: jernej.flisar@um.si. dr. Vili Podgorelec, redni profesor, Univerza v Mariboru, Fakulteta za elektrotehniko, računalništvo in informatiko, Koroška cesta 46, 2000 Maribor, Slovenija, e-pošta: vili.podgorelec@um.si.

DOI https://doi.org/10.18690/978-961-286-162-9.19 ISBN 978-961-286-163-6

Dostopno na: http://press.um.si.

1. UVOD

Strojno učenje predstavlja velik mejnik pri analizi velepodatkov, vendar je njegova uporaba zastrašujoča za ljudi, ki niso strokovnjaki za področje podatkovne znanosti in niso tehnično podkovani iz algoritmov strojnega učenja . Platforme podatkovne znanosti in strojnega učenja uporabljajo znanstveniki in razvijalci pri opravljanju nalog v celotnem podatkovnem in analitičnem procesu. Vključujejo naloge, ki se nanašajo na kreacijo in zajemanje podatkov, pripravo podatkov v primerno obliko, interaktivno raziskovanje in vizualizacijo, napredno modeliranje, testiranje, učenje, uporabo ter inženiring modelov strojnega učenja.

Izzivi uporabe podatkovne znanosti in platform za strojno učenje niso omejeni na izbiro prave platforme za zadostitev analitičnih potreb organizacij. Hkrati je potrebno obravnavati podatke, ljudi in procese, kar predstavlja še dodatni nivo težavnosti pri upravljanju organizacije. Upravljanje informacij ima zato pomembno vlogo pri zagotavljanju, da modeli temeljijo na najnovejših znanstvenih dosežkih tega področja in dobrih praksah, ki pridejo le z izkušnjami. Mnogokrat pa organizacije ne zaposlujejo strokovnjakov, ki bi posedovali ta znanja in izkušnje. Namesto najema zunanjih človeških virov, zaposlovanja in formiranja ekip podatkovnih znanstvenikov, je vedno bolj popularna uporaba namenskih storitev, ki so enostavne za uporabo in rešujejo specializirane naloge podatkovne znanosti.

Včasih so algoritme in ogrodja strojnega učenja večinoma uporabljali znanstveniki, tehnološki strokovnjaki ali domenski strokovnjaki. Vendar pa vse več organizacij zdaj uporablja storitve strojnega učenja, ki so vse bolj dostopne širšemu spektru razvijalcev in raziskovalcev. Podobno, kakor tradicionalne spletne storitve pomagajo razvijalcem ustvariti aplikacije, tako tudi storitve strojnega učenja (angl. *machine learning as a service* ali *MLaaS*) omogočajo enostavno uporabo strojnega učenja povprečnemu razvijalcu. Storitve strojnega učenja prikrijejo kompleksnost pri ustvarjanju in uporabi modelov strojnega učenja, tako da se lahko razvijalci osredotočijo na pripravo podatkov, uporabniško izkušnjo, oblikovanje, eksperimentiranje in uporabo znanja ter odkrivanje vzorcev iz podatkov.

Storitve strojnega učenja ponujajo abstraktni sloj za razvijalce, ki jim omogoča integracijo strojnega učenja v aplikacije v realnem svetu, ne da bi morali skrbeti za skaliranje algoritmov na svoji infrastrukturi in se ukvarjati s podrobnostmi metod strojnega učenja. Razvijalci aplikacij vedno iščejo različne načine za olajšanje življenja svojih uporabnikov z uvedbo novih in inovativnih funkcij, ki uporabnikom omogočajo prihranek časa. To je razlog za priljubljenost storitev strojnega učenja pri razvijalcih aplikacij. Nekateri standardni primeri teh storitev vključujejo inteligentno označevanje, priporočila, napovedovanje, segmentiranje in kategoriziranje. [1]

Najbolj pomemben del uporabe storitev strojnega učenja je prepoznava poslovnega problema, ki ga je treba rešiti in oblikovanje toka podatkov skozi celotno aplikacijo. Vsake težave ni mogoče rešiti s storitvami strojnega učenja, zato je pomembno identificirati problem, cilj in scenarij uporabe metod strojnega učenja. Mnogokrat namreč uporaba takih storitev ni mogoča, saj zastavljeni primeri uporabe zahtevajo aplikacijo podatkovne znanosti, ki jih ponujene storitve ne omogočajo.

Tekom naše raziskave smo identificirali tri tipe uporabnikov storitev in platform strojnega učenja:

(1) **Tipični razvijalci programske opreme, hobi znanstveniki, podatkovni raziskovalci in novinarji**, ki potrebujejo podatkovno znanost in strojno učenje za reševanje specifičnega problema ali primera uporabe.
(2) **Operativni delavci, razvijalce strojne opreme in odločevalci**, ki vsakodnevno sprejemajo odločitve na podlagi modelov strojnega učenja.
(3) **Visokokvalificirani inženirji strojnega učenja in podatkovni znanstveniki**, ki oblikujejo poskuse in učijo modele strojnega učenja za predstavljanje in optimizacijo poslovnih odločitev.

Vsak tip uporabnikov lahko uporabi platformo strojnega učenja, ki je prilagojena in optimizirana za njegov primer uporabe. V sledečem poglavju so opisane specializirane storitve, ki rešujejo ozko določen primer uporabe in so namenjene specifičnim zahtevam. Temu sledi poglavje, ki predstavi storitve in

platforme z vizualnim uporabniškim vmesnikom in so zasnovane bolj široko – potrebujejo prilagoditev in posledično poznavanje strojnega učenja. Zadnje poglavje pa predstavi platforme, ki so osnova vsem storitvam strojnega učenja in jih lahko uporabimo tudi v naših aplikacija, ampak za to zahtevajo tako odlično poznavanje algoritmov strojnega učenja kakor tudi programiranja v splošnem.

2. SPECIALIZIRANE STORITVE STROJNEGA UČENJA

V zadnjih nekaj letih je v podjetjih prišlo do premika paradigme gradnje tehnoloških skladov v smeri platform in mikrostoritev. Ta premik je bil omogočen zaradi razcveta računalništva v oblaku, še posebej pa zaradi povečane rasti števila javnih oblačnih storitev, ki jih ponujajo glavni igralci na področju računalništva v oblaku (Amazon, Google, Microsoft). Omenjena podjetja so močno poudarjala in zagovarjala poslovni model "kot storitev" (angl. *as a service*), kateri zunanjim podjetjem omogoča izbiro zgolj tistih mikrostoritev, ki so za njih potrebne oz. nujne.

Infrastruktura kot storitev (angl. *infrastructure as a service* ali *IaaS*) ter platforma kot storitev (angl. *platform as a service* ali *PaaS*) sta dve najpogosteje uporabljeni storitveni ponudbi ponudnikov računalništva v oblaku. S hitrim napredkom in razvojem strojnega učenja in umetne inteligence v splošnem pa se v zadnjem letu ali dveh na trgu vedno bolj razširjajo namenske – specializirane storitve. Za takšne specializirane storitve strojnega učenja sta se uveljavila dva izraza in sicer strojno učenje kot storitev (angl. *machine learning as a service* ali *MLaaS*) ter umetna inteligenca kot storitev (angl. *artificial intelligence as a service – AIaaS*). Z integracijo orodij in storitev umetne inteligence oz. strojnega učenja lahko podjetja izboljšajo zmožnosti svojih produktov, bolje komunicirajo s strankami, racionalizirajo poslovanje in ustvarjajo natančne napovedne poslovne strategije. MLaaS storitve omogočajo podjetjem brez namenskih oddelkov za strojno učenje oz. umetno inteligenco, da s svojim obstoječim kadrom – razvijalci, hitro in učinkovito oplemenitijo in nadgradijo svoje produkte. Podatki so gonilna sila strojnega učenja in ker velika podjetja – glavni igralci na področju IaaS in PaaS – ustvarjajo ogromne količine podatkov do katerih imajo tudi dostop, obenem pa imajo tudi veliko računskih virov, so sami zmožni zgraditi in naučiti modele, kar jim omogoča, da te ponujajo v obliki MLaaS zunanjim podjetjem. Takšne storitve torej vsebujejo že v naprej pripravljene algoritme in modele, za katere bi sicer potrebovali ogromno virov, da bi jih zgradili popolnoma od začetka. Hitrost, enostavnost in stroškovna učinkovitost integracije z obstoječimi produkti so tako ključne prednosti MLaaS v primerjavi s klasičnimi pristopi k strojnemu učenju oz. umetni inteligenci. [2]

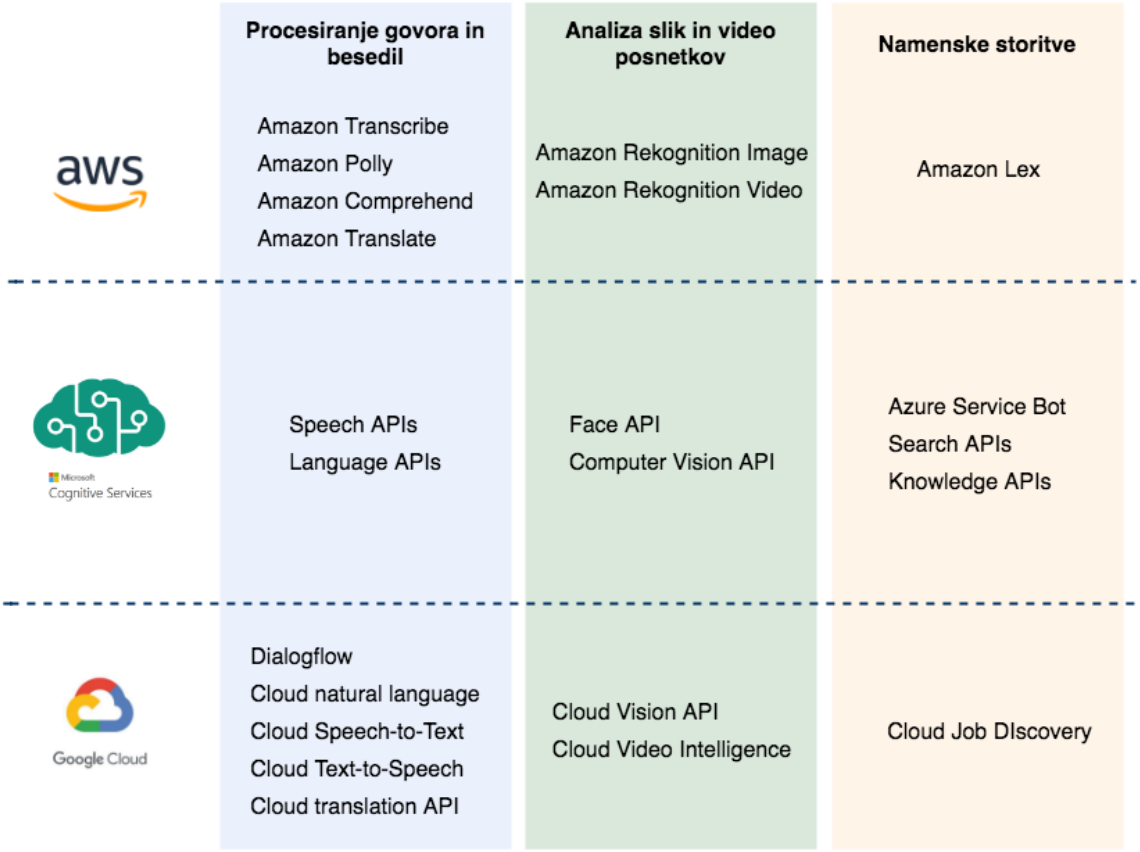
MLaaS oz. strojno učenje kot storitev je krovna definicija avtomatiziranih in delno avtomatiziranih oblačnih platform, ki naslavljajo problematiko večine infrastrukturnih problemov kot so pred-procesiranje podatkov, učenje modelov, evalvacija naučenih modelov in napovedovanje s pomočjo takšnih modelov. Rezultati napovednih modelov so z interno infrastrukturo podjetja povezani preko REST API povezav. [4]

Amazon Machine Learning Services, **Microsoft Azure Machine Learning** in **Google Cloud AI** so trije vodilni ponudniki oblačnih MLaaS storitev, ki omogočajo hitro učenje modelov in njihovo namestitev z malo ali celo brez domenskega znanja s področja podatkovne znanosti in strojnega učenja. Ponudniki MLaaS storitev ponujajo širok nabor funkcionalnosti teh storitev vse od prepoznave obrazov, zaznave predmetov iz slik, pretvarjanja govora v besedilo in obratno, pa vse do namenskih funkcionalnosti kot so pogovorni roboti (angl. *chatbot*) in podobno. V splošnem lahko glavne funkcionalnosti vodilnih podjetij na področju MLaaS razdelimo v tri osnovne kategorije (glej Slika 1):

- procesiranje govora in besedil,
- analiza slik in video posnetkov, ter
- namenske storitve.

Kategorija procesiranja govora in besedil je najbolj zastopana pri treh vodilnih ponudnikih. V to kategorijo spadajo storitve, z integracijo katerih lahko podjetje svoje produkte nadgradi na nivoju

interakcije z uporabniki. Z apliciranjem takšnih storitev lahko na primer omogočimo strojno prevajanje besedila v različne jezike ali pa analizo čustev iz besedila, s pomočjo katere lahko dodatno obogatimo uporabniško izkušnjo. Tehnike procesiranja naravnega besedila so med drugim tudi primarna tehnologija, ki stoji za raznimi pogovornimi roboti ali asistenti. V to skupino storitev pa spadajo tudi storitve, ki omogočajo ”razumevanje” govora in so ga zmožne tudi pretvarjati v besedilo. Najbolj poznani primeri aplikacij storitev te kategorije so pametni asistenti **Apple Siri**, **Google Now**, **Microsoft Cortana** ter **Amazon Alexa**.



Slika 1. MLaaS storitve ponudnikov Amazon, Microsoft in Google

Storitve iz kategorije analiza slik in video posnetkov v glavnem bazirajo na metodah in tehnikah globokega učenja. S pomočjo teh lahko podjetja svoje produkte nadgradijo z zmožnostjo prepoznave in ”razumevanja” slik ter video posnetkov. Storitve omogočajo analizo vizualnih vsebin vključno z identifikacijo objektov kot tudi klasifikacijo zaznanih objektov. Dodatno storitve omogočajo napredno analizo obraznih mimik subjektov, sledenje subjektom ipd.

V kategorijo namenske storitve so uvrščene storitve, ki so namenjene reševanju točno določenega problema. Na primer **Amazon Lex** in **Azure Service Bot** sta namenski storitvi za gradnjo pogovornega robota s katerim lahko komuniciramo preko govora ali besedila. Microsoftovi skupini aplikacijskih programskih vmesnikov **Search API** in **Knowledge API** med drugim omogočata dostop do funkcionalnosti iskalnika Bing in pa do naprednih funkcionalnosti za upravljanje z znanjem. **Google Cloud Job Discovery** je storitev, ki omogoča kadrovskim službam podjetij ”*plug and play*” dostop do iskalnika Google in njihovih zmogljivosti strojnega učenja, s čimer združujejo celoten ekosistem: strani podjetij za iskanje kadrov, zaposlitvene oglase, sisteme sledenja prošenj za službo in kadrovske agencije.

Ena od največjih prednosti uporabe MLaaS v primerjavi s klasičnimi metodami in tehnikami strojnega učenja je, da MLaaS podjetjem oz. organizacijam omogoča dostop do zmogljive infrastrukture, ki si je sama najverjetneje ne bi mogla ali ne morejo privoščiti. Strojno učenje zahteva veliko računske moči in sistemi, ki zagotavljajo to raven moči, so tradicionalno zelo dragi. Druga prednost, povezana s stroški, je dostop do cenovno ugodne podatkovne hrambe. Količina podatkov kontinuirano raste in mnoga podjetja se odločajo, da je stroškovno učinkoviteje, če podatke hranijo v oblaku. Ko ima podjetje podatke že v oblaku, pa je tudi uporaba MLaaS obstoječega ponudnika oblačnih storitev še bolj smiselna. Ključna prednost MLaaS je hitra in enostavna uporaba ter integracija v lastne produkte, brez potrebnega predhodnega znanja o metodah in tehnikah strojnega učenja.

Kljub številnim prednostim MLaaS se podjetja oz. organizacije, ki uporabljajo te storitve, lahko soočijo tudi z nekaj slabostmi. Največji problem pri uporabi MLaaS je skupen vsem storitvam, ki tečejo v javnem oblaku in to je odvisnost od ponudnika. Podjetja skrbi, da v primeru, da bi uporabljali preveč storitev posameznega ponudnika, prehod na drugega več ne bi bil mogoč. Obenem pa lahko to predstavlja tveganje v primeru, če ponudnik storitev zviša ceno svojih storitev. Vključevanje podatkov iz različnih virov prav tako lahko prestavlja oviro pri uporabi MLaaS. Mnogi produkti, ki uporabljajo strojno učenje, se opirajo na podatke, ti pa navadno prihajajo iz veliko različnih virov. Takšno zbiranje podatkov ter njihovo pred-procesiranje in obdelovanje je lahko težavna naloga ne glede na to ali uporabljamo MLaaS ali ne. [2, 3, 5]

3. STROJNO UČENJE Z UPORABNIŠKIM VMESNIKOM

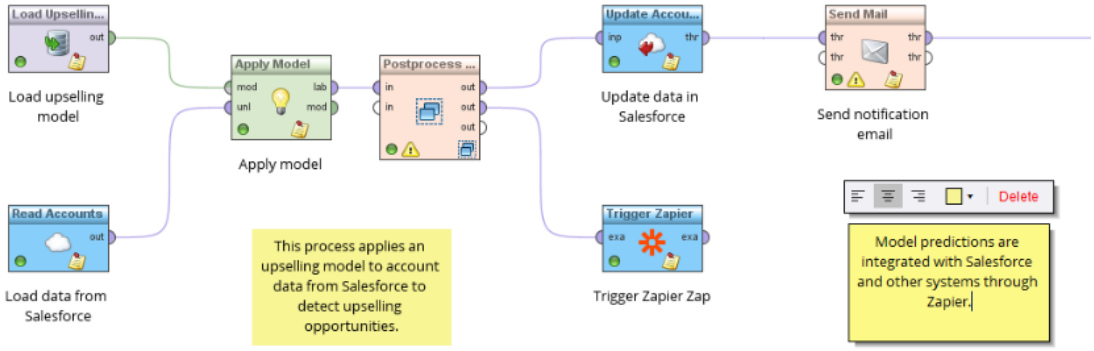
Zaradi razvijajočih se tehnologij strojnega učenja ter možnosti njihove uporabe v realnih projektih in aplikacijah se je pojavila zahteva po enostavni integraciji takih tehnologij na različnih nivojih in domenah aplikacij [5]. Metode strojnega učenja so namreč v domeni znanstvenikov oz. raziskovalcev v raziskovalnih institucijah, ki imajo veliko domenskega znanja, katerega pa razvijalci v podjetjih, ki razvijajo večinoma programsko opremo, nimajo. To je podaljšalo razvoj in apliciranje metod strojnega učenja v podjetjih. S časom so se razvila orodja, ki omogočajo enostaven razvoj inteligentnih aplikacij oz. integracijo naprednih metod strojnega učenja, brez naprednega domenskega znanja o samih metodah strojnega učenja.

Orodja za uporabo strojnega učenja pri razvoju oz. apliciranju le-tega lahko uporabljamo tudi preko naprednih grafičnih vmesnikov. Zaradi enostavne uporabe ni večje potrebe po poznavanju različnih algoritmov in metod strojnega učenja. Reševanja domensko specifičnih problemov se lahko lotimo s tako-imenovanim "*povleci in spusti*" načinom, kjer posamezne komponente odlagamo na delovno površino ter jih povežemo v smiseln proces (ang. *workflow*), kateri nam na koncu zgradi želen model. Model v tem primeru lahko predstavlja naučen klasifikator, ki nam npr. klasificira dokumente v določene kategorije. Vsaka posamezna komponenta v procesu predstavlja posamezno nalogo iz domene strojnega učenja za gradnjo učnega modela. Na voljo imamo številne metode podatkovnega rudarjenja, od priprave in čiščenja podatkov, preko algoritmov gručenja in klasifikacije vse do validacije zgrajenih modelov.

Ker moramo sami definirati cevovod procesa strojnega učenja, potrebujemo ustrezno osnovno znanje s področja podatkovnega rudarjenja in strojnega učenja, saj se morajo posamezne komponente ustrezno uporabiti. Večina orodij sicer preprečuje "*napačno*" uporabo komponent, ki nas vodi pri razvoju modela. Poleg velikega števila posameznih algoritmov in metod imamo pri teh na voljo še mnogo različnih nastavljivih parametrov, ki jih lahko dober strokovnjak s področja podatkovnega rudarjenja optimizira glede na specifiko problema, ki ga rešuje, kar lahko občutno izboljša uspešnost razvitega modela. Kljub temu je mogoče dobro razviti oz. aplicirati modele strojnega učenja tudi s pomanjkljivim domenskim znanjem, saj določena orodja sama predlagajo metode ter njihove nekatere privzete vrednosti.

Vzdrževanje modela, zgrajenega z grafičnim vmesnikom, je zelo enostavno. Zaradi vizualizacije je zgrajen model zelo pregleden. Tudi nadgrajevanje, spreminjanje in izboljševanje modela je dokaj

enostavno, saj lahko na enostaven način spremenimo podatkovni vir, iz katerega model črpa učne podatke, ali pa zamenjamo klasifikacijski algoritem, katerega želimo uporabiti. Tipični predstavniki orodij z grafičnim vmesnikom za namen obdelave podatkov so: RapidMiner (primer na Sliki 2), Knime, Azure ML in Waikato Weka.

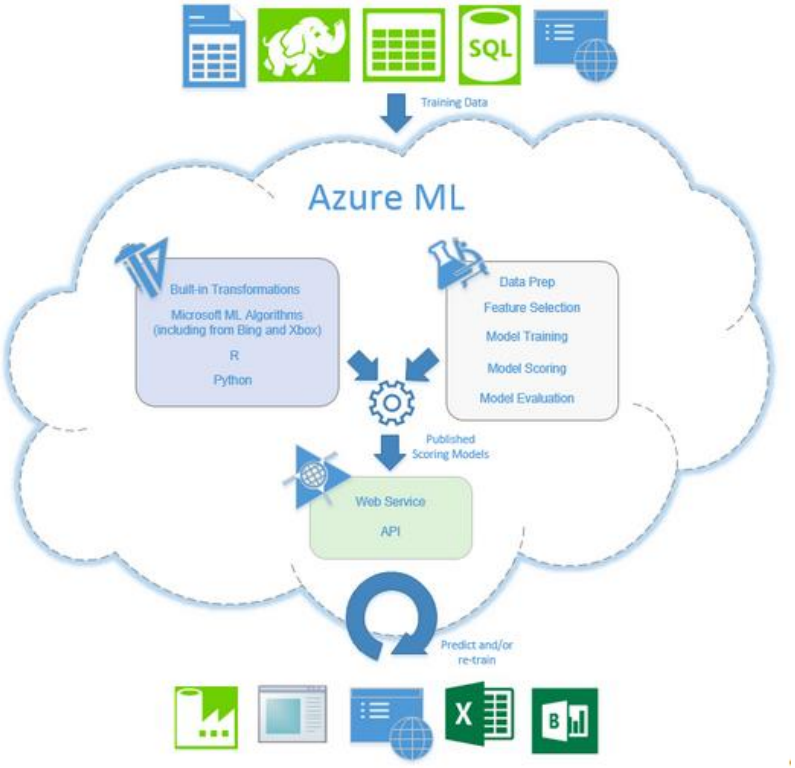


Slika 2. Primer gradnje modela v RapidMiner-ju.

Orodja strojnega učenja z uporabniškim vmesnikom lahko razdelimo v dve skupini, glede na "lokacijo" izvajanja – **lokalno** ali na **strežnikih** ("oblakih").

Orodja lahko namestimo in zaganjamo lokalno. V tem primeru potrebujemo dovolj dobro oz. problemu primerno strojno opremo, saj so lahko nekateri napredni algoritmi strojno zelo zahtevni. Prednost takšnih orodij je enostavnejša integracija z lastnim, že obstoječim sistemom, ali celo lastnimi aplikacijami. Pomemben je tudi nadzor nad podatki, saj se ti vedno nahajajo na lastni infrastrukturi in tako niso izpostavljeni na zunanjih strežnikih, ki niso v celoti pod našim nadzorom.

Na voljo so tudi orodja, ki se izvajajo na zunanjih strežnikih, ki smo jih že prej imenovali MLaaS. Takšna orodja so primerna predvsem, kadar imamo potrebe po obdelavi velikih količin podatkov, saj so infrastrukturno že pripravljena za izvajanje preko različnih sistemov. Slaba lastnost uporabe orodij v oblaku je izguba popolnega nadzora nad podatki ter izguba nadzora nad konfiguracijo in parametrizacijo nekaterih učnih algoritmov. Primer takšnih orodih so npr: MS Azure ML, AWS Machine Learning, RapidMinerCloud. Slika 3 prikazuje arhitekturni pregled MLaaS.



Slika 3. Primer Arhitekturni pregled MLaaS za storitev Azure ML. [10]

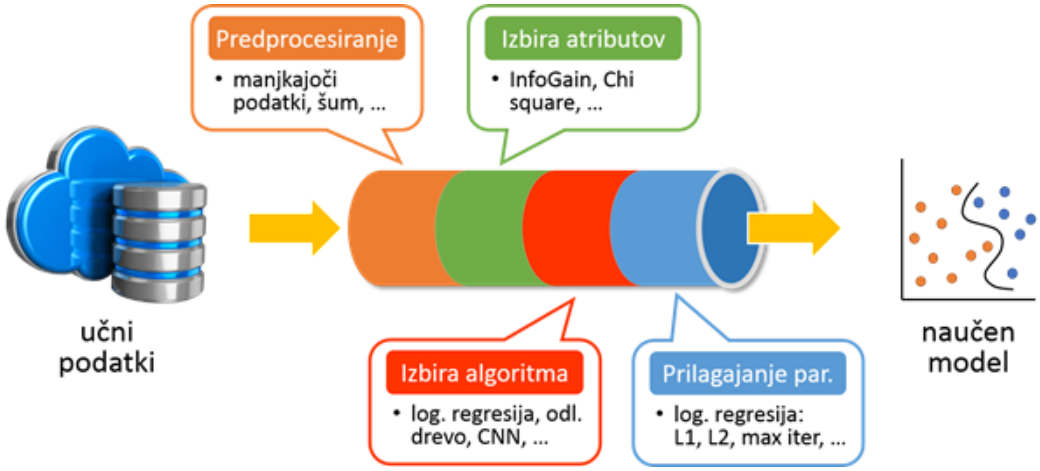
Tabela 1 prikazuje prednosti in slabosti razvoja modelov strojnega učenja z uporabo grafičnih vmesnikov v orodjih.

Tabela 1. Prednosti in slabosti orodij strojnega učenja z uporabniškim vmesnikom.

Prednosti	Slabosti
Hitrejši in enostavnejši razvoj	Slabša kakovost/točnost modelov
Manjša razvojna ekipa	Počasnejše izvajanje
Enostavnejše vzdrževanje	Težja implementacija za specifične probleme
Manjša potreba po domenskih znanjih	
Vizualizacija procesa	

4. STROJNO UČENJE IZ NIČ

Tretji možen pristop k razvoju inteligentnih informacijskih rešitev z uporabo metod in tehnik strojnega učenja je popolnoma samostojen razvoj, tako rekoč iz nič. Kot velja že za ostala področja razvoja, lahko ugotovimo tudi tukaj – samostojnega razvoja »*iz nič*« se je smiselno lotiti predvsem takrat, ko želimo imeti popoln nadzor nad uporabljenimi metodami in algoritmi v celotnem življenjskem ciklu razvoja in uporabe tehnik strojnega učenja – t.i. cevovodu strojnega učenja (Slika 4). Ker obstoječe rešitve MLaaS neenakomerno pokrivajo posamezne korake ML cevovoda (Slika 5), je tudi od tega, kateremu izmed korakov želimo posvetiti večjo pozornost, odvisno, ali in v kolikšni meri se bomo lotili samostojnega razvoja.

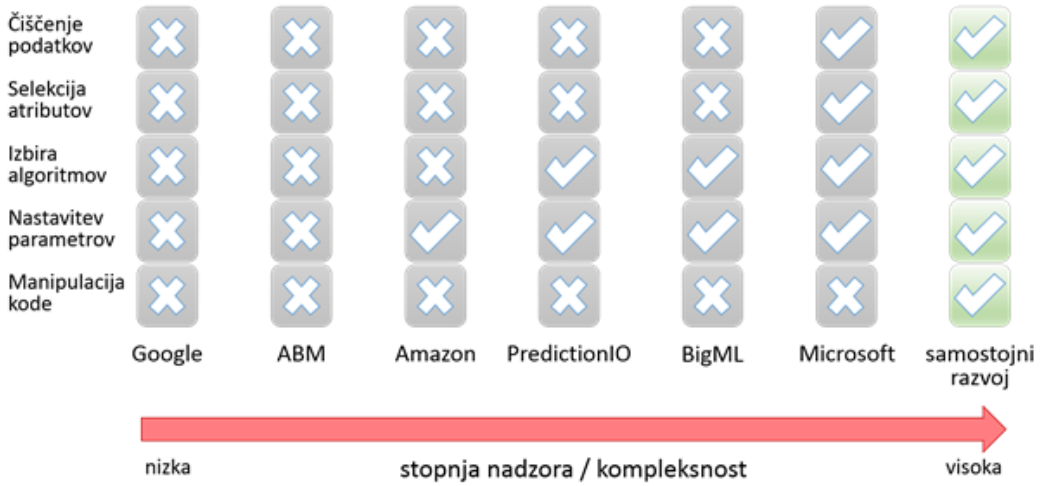


Slika 4. Standardni cevovod strojnega učenja. Od posamezne platforme je odvisno, katere korake je mogoče nadzorovati.

Povsem intuitivno je, da večji nadzor nad vsakim korakom v cevovodu omogoča dobro obveščenim uporabnikom, da gradijo bolj kakovostne modele [6]. Funkcija, model in izbira parametrov lahko pomembno vplivajo na uspešnost nalog strojnega učenja (npr. točnost napovedi). Vendar pa je za uspešno optimizacijo vsakega koraka potrebno preseči precejšnjo kompleksnost, kar je težko brez poglobljenega znanja in izkušenj. Po drugi strani pa seveda ni samo po sebi umevno, da lahko specializirane storitve, katerih uporaba seveda precej zmanjša kompleksnost, samodejno opravijo učinkovito upravljanje cevovoda ter nastavitev parametrov algoritmov in metod do te mere, kot je včasih potrebno.

Pri razumevanju odnosov med kompleksnostjo, uspešnostjo in preglednostjo na platformah MLaaS se lahko osredotočimo na tri ključna vprašanja:

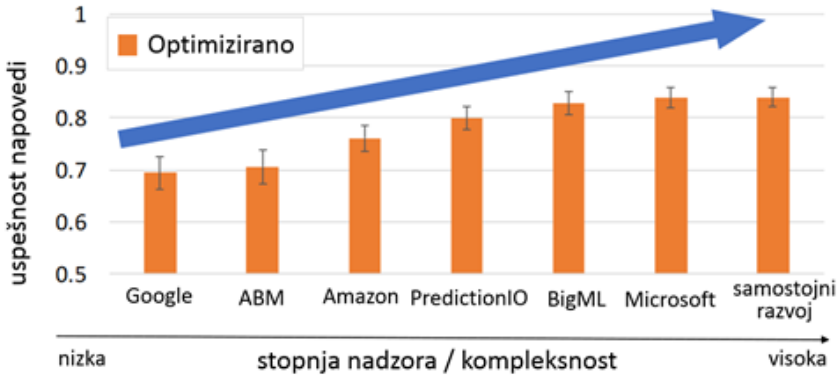
- Kako kompleksnost (stopnja nadzora) sistemov strojnega učenja vpliva na točnost modelov?
- Ali lahko povečan nadzor vodi do večjih tveganj pri oblikovanju slabih modelov?
- Do kakšne mere lahko sistemi MLaaS optimizirajo avtomatizirane dele svojega cevovoda?



Slika 5. Različne platforme strojnega učenja omogočajo različno stopnjo nadzora nad posameznimi koraki v procesu strojnega učenja; podatki povzeti po [6].

V [6] so avtorji opravili eksperimentalno primerjavo napovedne uspešnosti posameznih platform strojnega učenja nad 119 podatkovnimi množicami. Rezultati so pokazali, da z večanjem kompleksnosti

oz. stopnje nadzora posamezne platforme raste tudi uspešnost najboljših zgrajenih modelov (Slika 6) – tako lahko pri samostojnem razvoju dejansko zgradimo najboljše napovedne modele. A po drugi strani z večanjem kompleksnosti platforme zelo raste tudi tveganje, da bomo dejansko uspeli zgraditi najboljši možni model, ki ga orodje zmore (Slika 7). Preprost sklep bi lahko bil: uporabljajmo tisto platformo oz. orodje, ki smo ga sposobni obvladati.

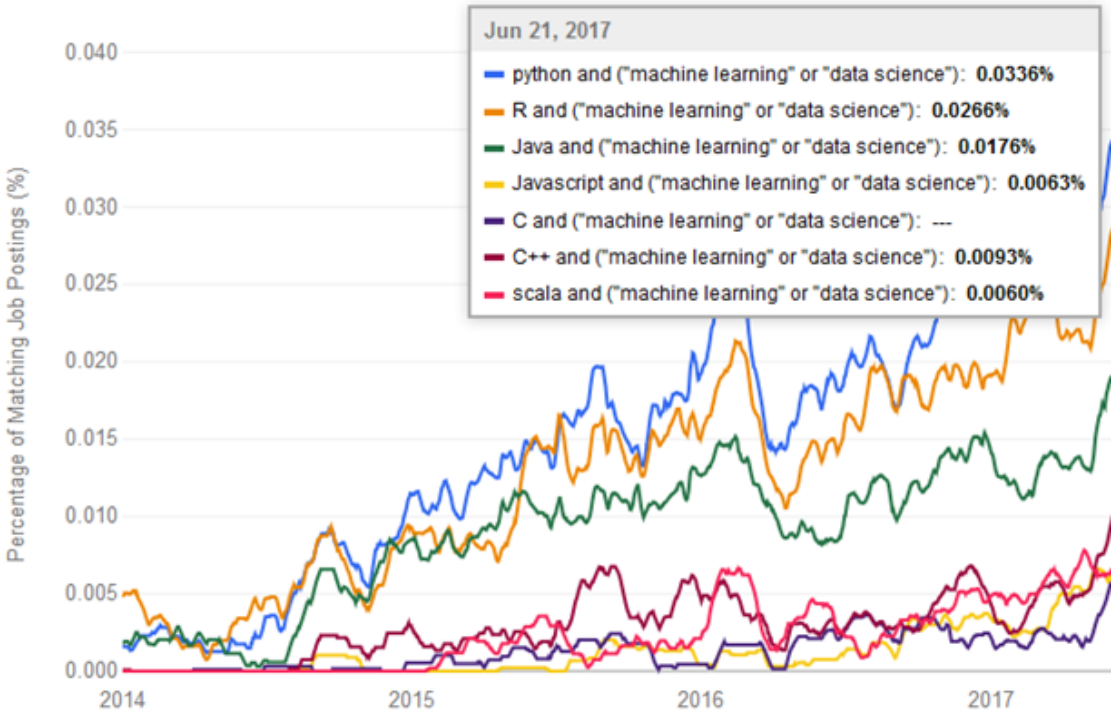


Slika 6. Z večanjem stopnje kompleksnosti se veča uspešnost zgrajenih modelov; povzeto po [6].

Slika 7. Z večanjem stopnje kompleksnosti se zelo veča potencialno tveganje; povzeto po [6].

No, resnici na ljubo »nič«, iz katerega gradimo rešitve strojnega učenja, zajema kopico odličnih programskih knjižnic, večinoma odprtokodnih in podprtih z zelo močno skupnostjo razvijalcev. Tako se potencialnemu razvijalcu ni potrebno ukvarjati z implementacijskimi podrobnostmi samih metod strojnega učenja, hkrati pa njihova programska uporaba omogoča poln dostop do vseh parametrov, ki vplivajo na učinkovitost izvajanja ter uspešnost izvedbe.

Med programskimi jeziki, v katerih se razvijalci najpogosteje lotevajo razvoja, izstopajo predvsem Python, R, Java, C, C++, Scala in morda nekoliko presenetljivo tudi Javascript. Slika 8 prikazuje trend števila oglasov za službe, ki zajemajo strojno učenje oz. podatkovno znanost, in zahtevajo poznavanje določenega programskega jezika. Vidimo lahko, da predvsem Python vse bolj prednjači, medtem ko Java med prvimi tremi jeziki, ki sicer od ostalih že precej odstopajo, nekoliko izgublja zagon.



Slika 8. Zahtevano znanje programskih jezikov v oglasih za službe na področju strojnega učenja in podatkovne znanosti [vir: https://www.indeed.com/jobtrends/].

Pri podatkih na Sliki 8 je treba upoštevati dejstvo, da v prikaz niso zajeti le oglasi za razvijalce, pač pa tudi podatkovne analitike ipd. Prav jezik R pa je, spet poleg Pythona, najbolj pogost prav med analitiki, statistiki in drugimi, ki pri svojem delu obdelujejo in analizirajo podatke, ne razvijajo pa programske opreme. Posledično velja, da so za razvoj inteligentnih rešitev najpogosteje uporabljeni jeziki Python, Java, C/C++, R in Javascript. Tako kar 57% razvijalcev že sedaj uporablja Python, še 33% uporabnikov pa meni, da je prav Python ustrezen za razvoj tovrstnih rešitev [7]. R po drugi strani sicer uporablja 31% uporabnikov, ki se ukvarjajo s strojnim učenjem, vendar bi ga za namen razvoja uporabilo zgolj 5%. Java in C/C++ sta precej izenačena, saj za oba jezika okrog 20% razvijalcev meni, da sta primerna izbira. V Tabeli 2 je zbranih nekaj podatkov o uporabi različnih programskih jezikov za namene razvoja inteligentnih rešitev.

Kot je razvidno iz Tabele 2, so jeziki C/C++, R in Javascript namenjeni precej specifičnim namenom – tistim, pri katerih se tudi sicer posamezni programski jezik najpogosteje uporablja: C/C++ za razvoj iger in vgrajenih sistemov, R za namene statističnih obdelav in analitike, Javascript pa za razvoj spletnih rešitev, predvsem na strani odjemalca.

Tako ostaneta Python in Java tista programska jezika, ki sta najbolj primerna za razvoj raznovrstnih inteligentnih rešitev. Pri tem se, spet podobno kot sicer, Java osredotoča na poslovno uporabo in predvsem večje poslovne sisteme, medtem ko je Python primeren za praktično karkoli in je daleč najbolj priljubljen predvsem pri vseh vrstah zagonskih podjetij in manjših, drznejših projektih.

Tabela 2. Pregled uporabe programskih jezikov v projektih strojnega učenja.

	Python	Java	C/C++	R	Javascript
Najpogostejše aplikacije	procesiranje naravnega jezika, analiza sentimenta, tekstovno in rudarjenje po spletu	upravljanje podpore strankam, omrežna varnost in detekcija vdorov, detekcija goljufij	AI v igrah, upravljanje robotov, omrežna varnost in detekcija vdorov	bioinženiring, bioinformatika, analiza sentimenta, detekcija anomalij	iskalniki, upravljanje podpore strankam, različne specifične naloge
Najmanj pogoste aplikacije	AI v igrah, omrežna varnost in detekcija vdorov	analiza sentimenta, bioinženiring, bioinformatika, specifične naloge	detekcija goljufij, priporočilni sistemi, analiza sentimenta	prepoznavanje govora, AI v igrah, upravljanje robotov	diagnostika v industriji, bioinženiring, bioinformatika
Poklicno ozadje	podatkovni znanstvenik	razvijalec aplikacij za namizne računalnike	inženir za vgrajene sisteme	podatkovni analitik, statistik	spletni razvijalec
Razlogi za uporabo AI	priključiti se valu vpeljave metod strojnega učenja	navodilo vodstva podjetja	dodajanje umetne inteligence v obstoječe aplikacije	podatkovna znanost je (bila) del študija	zagotovitev oz. pridobitev profitabilnih projektov

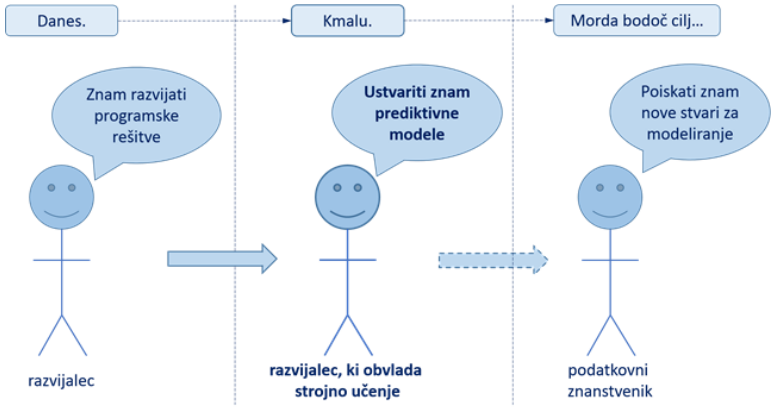
Za konec omenimo še nekatera trenutno najpogosteje uporabljana ogrodja in knjižnice za strojno učenje. Za Javo je smiselno pregledati naslednje: ADAMS, Deeplearning4j, ELKI, JavaML, JSAT, Mahout, MALLET, Massive Online Analysis, RapidMiner in Weka. Za Python pa je nabor še bistveno obširnejši. Med temeljne knjižnice sodijo: NumPy, SciPy in Pandas. Za namene vizualizacije so na voljo: Matplotlib, Seaborn, Bokeh in Plotly. Najpogostejša ogrodja za strojno učenje so: SciKit-Learn, Theano, TensorFlow in Keras. Za obdelavo naravnega jezika imamo na voljo NLTK in Gensim, medtem ko je Scrapy odličen za zajem podatkov iz (nestrukturiranih) vsebin, za statistiko pa velja uporabiti Statsmodels.

5. BISTVENO VPRAŠANJE RAZVOJA INTELIGENTNIH REŠITEV NI VPRAŠANJE IZBIRE TE ALI ONE TEHNOLOGIJE

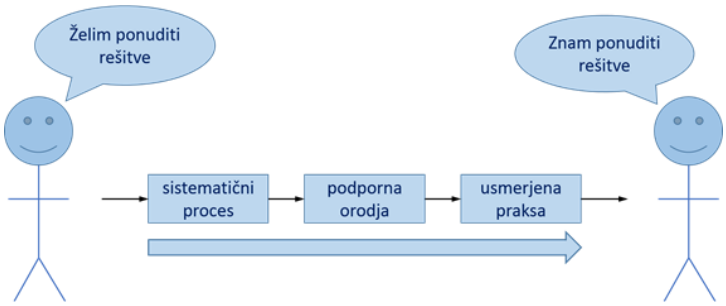
Tipična opravila strojnega učenja, ki se danes vse pogosteje vgrajujejo v sodobne aplikacije, z vidika razumevanja strojnega učenja običajno niso preveč zahtevna. Samodejno identifikacijo neželene e-pošte, razvrščanje izdelkov v priporočilnih sistemih, profiliranje uporabnikov spletne storitve in podobne primere uporabe metod strojnega učenja bi praviloma povprečno usposobljen razvijalec moral obvladati že po osnovni seznanitvi s področjem strojnega učenja. Tudi na videz zahtevne naloge, kot so npr. prepoznava obrazov, identifikacija objektov na slikah ali ugotavljanje sentimenta v podanem besedilu, ob uporabi sodobnih knjižnic ne bi smele predstavljati pretrdega oreha. Praviloma je v teh primerih potrebno zbrati znane podatke, z njimi naučiti model znanja, ki ga lahko nato uporabimo za predikcijo novih, neznanih situacij. Nekaj klicev ustreznih funkcij in rešitev je na dlani. S tega vidika izbira med specializiranimi rešitvami za strojno učenje (MLaaS), uporabo metod strojnega učenja preko uporabniškega vmesnika ali samostojen razvoj v izbranem programskem jeziku in z izbrano programsko knjižnico ni vprašanje obvladanja strojnega učenja, pač pa drugih, poslovnih in osebnih faktorjev.

Vse nekaj drugega pa je, ko v teoriji delujoč in priporočen recept v praksi ne deluje. Ko rezultati niso niti približno takšni, kot smo pričakovali ali si jih obetali. Takrat ni pravo vprašanje, katera tehnologija je boljša, temveč kako rešiti problem, pred katerim smo se ustavili. Največji problem, ki ga mora razvijalec na svoji poti do uspešnega razvijalca inteligentnih rešitev premagati, tako ni v obvladovanju tehnologij, kompleksnosti knjižnic ali obilju algoritmov in njihovih nastavitev. Največji problem je ustrezno povezati teorijo, algoritme in matematiko strojnega učenja s problemom, ki ga moramo rešiti. Ko priporočen recept, ki običajno dobro deluje, in smo mu dosledno sledili, ne pričara pravega rezultata, se zavemo prave, ogromne vrzeli.

Proces transformacije iz (običajnega) razvijalca v razvijalca, ki obvlada strojno učenje in morda še naprej v podatkovnega znanstvenika, je prikazan na Sliki 9. Za čim lažjo dosego tovrstne transformacije se velja držati top-down pristopa, prikazanega na Sliki 10. Za učinkovito uporabo strojnega učenja se ne smemo ustaviti zgolj pri učnih algoritmih in ogrodjih, ki jih implementirajo in nam ponujajo možnost njihove uporabe. Potrebujemo sistematičen pristop od zgoraj navzdol, kjer se osredotočimo na dejanski rezultat, ki ga želimo: razvijati resnične rešitve »na ključ« z uporabo najboljših, sodobnih orodij in platform. V ta namen pa moramo razumeti, za kaj pri strojnem učenju dejansko gre, ne zgolj znati uporabiti vnaprej pripravljene rešitve.



Slika 9. Transformacija iz razvijalca v razvijalca, ki obvlada strojno učenje.



Slika 10. Pristop, ki se začne z dejanskimi problemi strojnega učenja in konča z rešitvijo »na ključ«.

6. ZAKLJUČEK

V prispevku smo pregledali tri nivoje storitev, kjer vsak nivo služi svojemu primeru uporabe. Če se navežemo na tri tipe uporabnikov takih storitev, vidimo da so razvijalske ekipe s strokovno usposobljenim kadrom na področju strojnega učenja in izdelkom, katerega osrednji del je inteligentna obdelava podatkov, primarni uporabniki platform na najnižjem nivoju. Te namreč ponujajo tem uporabnikom največjo svobodo pri implementaciji, a poleg strokovno usposobljenega kadra zahtevajo mnogo več časa za implementacijo ter višje stroške vzdrževanja in skaliranja teh storitev. Tisti, ki s pomočjo metod podatkovne znanosti in rezultatov teh metod sprejemajo poslovne odločitve, so primarni uporabniki storitev na drugem nivoju, kjer platforme z vizualnim uporabniškim vmesnikom omogočajo rešitve strojnega učenja. Te namreč omogočajo dovolj velik spekter prilagodljivosti, uporabi se jih lahko v oblaku ali lokalno, in ne zahtevajo programerskega znanja. Razvijalci programske opreme, ki ne temeljijo na strojnem učenju, raziskovalni novinarji in raziskovalci pa so primarni uporabniki specializiranih in že optimiziranih storitev, ki jih kot storitve ponujajo različni ponudniki.

Uporaba tehnologije podatkov in strojnega učenja je glavna prioriteta ali celo izdelek mnogih organizacij. Na voljo so vedno bolj izpopolnjeni analitični postopki in storitve strojnega učenja, ki jim omogočajo, da se ogromne količine podatkov, ki so na voljo, v celoti izkoristijo za namen optimizacije poslovnega procesa. Zelo enostavno se je izgubiti v različnih razpoložljivih rešitvah. Te se razlikujejo po ponujenih algoritmih, primerih uporabe in zahtevanem poznavanju strojnega učenja. Ustvarjanje mostu med podatkovno znanostjo in poslovno vrednostjo je težaven proces, predvsem če primanjkuje znanja o podatkovni znanosti ali domenskih izkušenj. Spodnji diagram prikazuje napoved Gartner-ja o vodilnih platformah in vizionarjih na področju platform strojnega učenja. Gartner v naslednjih dveh do treh letih pričakuje nadaljevanje turbulence na trgu podatkovne znanosti in platform strojnega učenja. V njem bodo še naprej v celoti novi ponudniki, pa tudi obstoječi ponudniki sosednjih trgov (kot so analitika in BI). [8]

Prihodki iz računalniških znanosti in platform za strojno učenje so se v letu 2016 povečali za 9,3%, na 2,4 milijarde USD. Ta rast je več kot dvakrat večja od celotnega analitičnega trga (4,5%), 2,4 milijarde pa predstavlja 14,1% celotnega svetovnega analitičnega in BI prihodka (v letu 2015 je bil ta delež 13,5%). Rast trga podatkovne znanosti in strojnega učenja spodbuja željo končnih uporabnikov, da uporabijo bolj napredne analitike za izboljšanje odločanja v celotnem podjetju. [7]

Po vseh prognozah rasti trga podatkovne znanosti je tako pričakovano, da bo število različnih ponujenih platform, na vseh treh nivojih, le še raslo. To bo vsekakor imelo pozitiven vpliv na kvaliteto storitev, saj bodo ponudniki takih platform in storitev tekmovali za enak ali podoben profil uporabnikov in razvijalcev, kar pa je za končnega uporabnika pozitivno v smislu višje kvalitete storitev in nižjih stroškov uporabe teh storitev.

Slika 11. Gartner-jev kvadrant za leto 2017 za podjetja, ki razvijajo platforme strojnega učenja [8].

7. LITERATURA

[1] https://www.gartner.com/document/3868668, Market Guide: Machine Learning Infrastructure as a Service, obiskano 17.5.2018.

[2] https://blog.g2crowd.com/blog/trends/artificial-intelligence/2018-ai/machine-learning-service-mlaas/, AI Trends 2018: Machine Learning as a Service (MLaaS), obiskano 17.5.2018.

[3] https://www.datamation.com/cloud-computing/artificial-intelligence-as-a-service-ai-meets-the-cloud.html, Artificial Intelligence as a Service: AI Meets the Cloud - Datamation, obiskano 17.5.2018.

[4] https://www.altexsoft.com/blog/datascience/comparing-machine-learning-as-a-service-amazon-microsoft-azure-google-cloud-ai/, Comparing MLaaS: Amazon AWS, MS Azure, Google Cloud AI, obiskano 17.5.2018.

[5] https://www.datamation.com/cloud-computing/cloud-machine-learning-is-it-right-for-you.html, Cloud Machine Learning: Is It Right for You? - Datamation, obiskano 17.5.2018.

[6] Yao Y, Xiao Z, Wang B, Viswanath B, Zheng H, Zhao BY. Complexity vs. performance: empirical analysis of machine learning as a service. Proceedings of the 2017 Internet Measurement Conference 2017, ACM, pp. 384-397.

[7] https://towardsdatascience.com/what-is-the-best-programming-language-for-machine-learning-a745c156d6b7, What is the best programming language for Machine Learning?, Developer Economics, 2017, obiskano 17.5.2018.

[8] https://www.gartner.com/document/3860063, Magic Quadrant for Data Science and Machine-Learning Platforms, obiskano 17.5.2018.

[9] https://www.gartner.com/document/3772081, Hype Cycle for Data Science and Machine Learning, 2017, obiskano 17.5.2018.

[10] https://www.blue-granite.com/blog/bid/404378/azure-machine-learning-an-overview, Aazure Machine Learning: An Overview, obiskano 16.5.2018.

Univerzitetna založba Univerze v Mariboru

ANALIZA INTELIGENTNIH OBLAČNIH STORITEV NA PRIMERU PREPOZNAVE OBRAZOV

GREGA VRBANČIČ IN VILI PODGORELEC

Povzetek: V zadnjih letih je strojno učenje postalo eden od temeljnih elementov informacijske tehnologije in s tem osrednji, čeprav navadno prikrit del našega življenja. Uporabniki so ozavestili inteligentno obnašanje naprav ter storitev in to od ponudnikov v vedno večji meri tudi zahtevajo. Podjetja so tako primorana nadgrajevati ter razvijati produkte in rešitve v smeri integracije z metodami in tehnologijami strojnega učenja oziroma s pomočjo umetne inteligence. Kljub enormnemu napredku na področju strojnega učenja je slednje še vedno zahtevna naloga, ki zahteva obilico domenskega znanja s področja podatkovne znanosti. Z namenom zapolnitve omenjene vrzeli se na trgu pojavlja množica rešitev. Med najbolj obetajočimi rešitvami omenjenega problema so metode strojnega učenja v obliki storitev, integrirane z računalništvom v oblaku.

Ključne besede: • inteligentne oblačne storitve • strojno učenje • prepoznava obrazov

NASLOV AVTORJEV: Grega Vrbančič, mladi raziskovalec, Univerza v Mariboru, Fakulteta za elektrotehniko, računalništvo in informatiko, Koroška cesta 46, 2000 Maribor, Slovenija, e-pošta: grega.vrbancic@um.si. dr. Vili Podgorelec, redni profesor, Univerza v Mariboru, Fakulteta za elektrotehniko, računalništvo in informatiko, Koroška cesta 46, 2000 Maribor, Slovenija, e-pošta: vili.podgorelec@um.si.

DOI https://doi.org/10.18690/978-961-286-162-9.20 ISBN 978-961-286-163-6

Dostopno na: http://press.um.si.

1. UVOD

V zadnjih letih smo priča razcvetu na področju strojnega učenja oziroma umetne inteligence. Ključno vlogo igra v širokem spektru aplikacij, kot so podatkovno rudarjenje, obdelava naravnega jezika, prepoznava slik in podobno. Zaradi vedno večje količine podatkov, ki so na voljo, obstaja dober razlog za domnevo, da bo inteligentna analiza podatkov postala še bolj razširjena in prisotna kot ključna sestavina za tehnološki napredek. [1]

Kljub širokemu naboru uporabnosti pa je za uspešno uporabo metod strojnega učenja še vedno potrebna kopica domenskega znanja s področja umetne inteligence, poleg tega pa je za uporabo strojnega učenja nujna tudi velika količina računskih virov. Z namenom približanja metod in tehnik strojnega učenja širši množici ljudi, tako posameznikom kot tudi podjetjem, je večina spletnih velikanov (Google, Amazon, Facebook) ponudila produkte, ki omogočajo enostavno apliciranje metod strojnega učenja z uporabo oblačne infrastrukture, brez oz. z minimalno potrebo po domenskem znanju s področja strojnega učenja. Tako združeni tehnologiji strojnega učenja in računstva v oblaku sta poznani pod imenom »inteligentni oblak«. Trenutna uporaba oblačnih storitev v večini predstavlja računstvo, hrambo podatkov ter mreženje. Z dodano funkcionalnostjo strojnega učenja, vgrajenega neposredno v obstoječo oblačno infrastrukturo, pa se zmožnosti takšnega oblaka močno povečajo. Z združitvijo omenjenih tehnologij oblak postane zmožen učenja iz masovnih podatkov, predhodno hranjenih ali novo ustvarjenih, z namenom izgradnje raznih napovednih modelov, iskanja vzorcev in analizo dogodkov, kar se v končni fazi lahko odraža v obliki sprejemanja boljših, bolj racionalnih poslovnih odločitev ali pripomore k izboljšanju, nadgradnji samega produkta ali storitve.

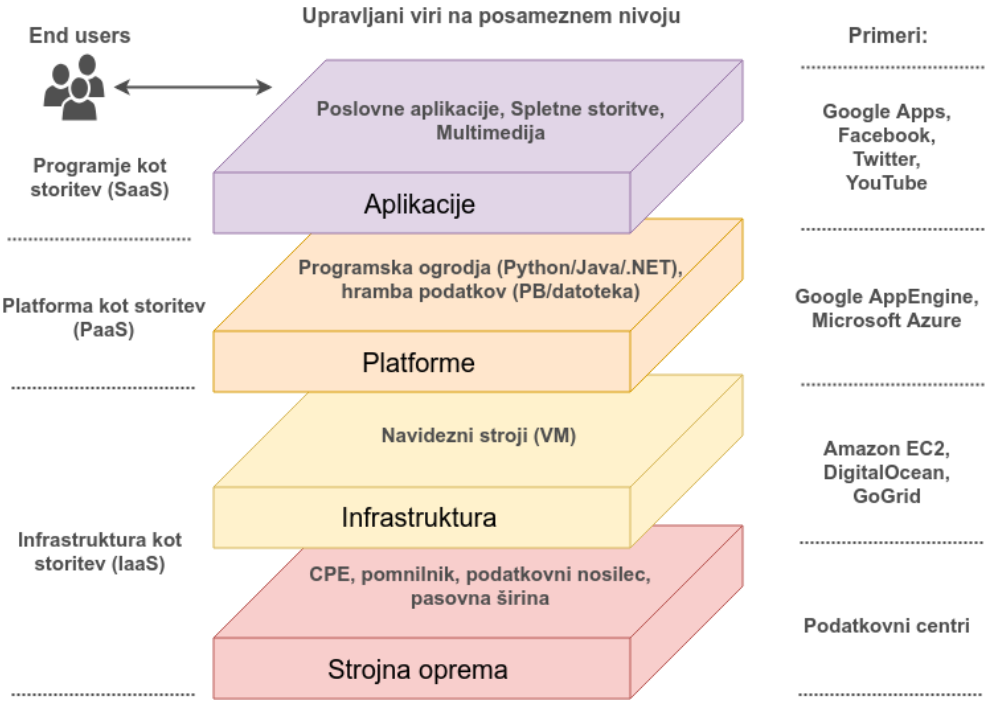
V nadaljevanju bomo predstavili računalništvo v oblaku ter inteligentne oblake oz. storitve večjih ponudnikov ter njihove zmožnosti, zatem pa predstavili primer aplikativne uporabe inteligentnega oblaka za problem prepoznave obrazov.

2. RAČUNALNIŠTVO V OBLAKU

Odkar je bila paradigma računalništva v oblaku zasnovana, je bilo podanih več definicij. Nekatere izmed njih so se osredotočale na dinamično zagotavljanje virov procesiranja in shranjevanja, druge pa so poudarjale storitveno usmerjen vmesnik in izkoriščanje tehnik virtualizacije. Ameriški nacionalni inštitut za standarde in tehnologijo (angl. National Institute of Standards and Technology – NIST) je podal referenčno opredelitev. NIST je računanje v oblaku opredelil kot model, ki deluje po načelu »plačaj glede na porabo« in zagotavlja priročen, omrežen dostop na zahtevo do bazena nastavljivih računalniških virov v skupni rabi (npr.: omrežja, strežniki, hramba podatkov, aplikacije, storitve), katere lahko z minimalnim trudom enostavno upravljamo – instantno zagotavljamo in sproščamo. [2]

Računalništvo v oblaku v splošnem delimo na dva načina – glede na model dostave ter glede na namestitev. Glede na model dostave so oblaki razdeljeni v tri skupine (glej sliko 1): [3]

- infrastruktura kot storitev (angl. Infrastructure as a Service – IaaS),
- platforma kot storitev (angl. Platform as a Service – PaaS) ter
- programje kot storitev (angl. Software as a Service – SaaS).



Slika 1: Opredelitev oblakov glede na model dostave

Glede na namestitev poznamo štiri modele oblakov: [4]

- javni oblak (angl. Public Cloud) – infrastruktura je dostopna splošni javnosti ali veliki gospodarski skupini in je v lasti organizacije, ki prodaja oblačne storitve.
- zasebni oblak (angl. Private Cloud) – infrastruktura operira zgolj za namene oz. potrebe organizacije; upravljana je lahko s strani organizacije ali tretje osebe in je lahko vzpostavljena znotraj ali izven organizacije.
- skupnostni oblak (angl. Community Cloud) – infrastruktura je deljena med več organizacijami in podpira specifično skupnost, ki ima skupne interese; upravljana je lahko s strani organizacije ali tretje osebe.
- hibridni oblak (angl. Hybrid Cloud) – infrastruktura je kompozija dveh ali več oblakov (zasebnih, skupnostnih ali javnih), kateri ostajajo unikatne entitete, ki so med seboj povezane s standardizirano ali lastniško tehnologijo za namene omogočanja podatkovne in aplikativne prenosljivosti.

3. INTELIGENTNI OBLAK

Računalništvo v oblaku zagotavlja dve osnovi zahtevi za izvajanje sistema umetne inteligence: učinkoviti in stroškovno sprejemljivi viri (v glavnem hramba podatkov) ter računska moč, zmožna obdelave velikih količin podatkov. Kot prvo, računalništvo v oblaku zagotavlja prilagodljivo, cenovno dostopno računsko moč in kot drugo, odlično možnost hrambe in procesiranja velikih količin podatkov. Taka združitev računskega oblaka z metodami in pristopi strojnega učenja koristi obema disciplinama. [5]

Strojno učenje in umetna inteligenca postajata ključna elementa novega vala inovacij na trgu PaaS. Začetniki PaaS kot so Amazon z AWS, Microsoft z Azure, IBM z Bluemix in Google s Cloud storitvijo nenehno poskušajo z inoviranjem na področju strojnega učenja in umetne inteligence prehiteti svoje konkurente. Medtem ko se je prvi val inovacij na področju PaaS osredotočal na ključno infrastrukturo in aplikacijske platforme, je novi val osredotočen na podatke, pridobivanje poglobljenega vpogleda v njih ter pridobivanje znanja iz podatkov. V bližnji prihodnosti se lahko nadejamo, da bodo storitve

strojnega učenja (angl. Machine Learning as a Service – MLaaS) in umetne inteligence (angl. Artificial Intelligence as a Service – AIaaS) splošno uporabne kot so trenutno uporabni PaaS produkti.

Boj med največjimi konkurenti na trgu PaaS je osredotočen na zelo specifične vrste in domene strojnega učenja oz. umetne inteligence in zdi se, da so si konkurenti kar se tiče vpeljave novih rešitev za specifične domene kar precej konsistentni.

Trenutno aktualna področja, na katera se osredotočajo vodilni na področju PaaS, lahko opredelimo sledeče: [6]

- platforme za strojno učenje (angl. Machine learning platforms): domorodne oblačne storitve, ki omogočajo ustvarjanje, izvajanje ter upravljanje tradicionalnih modelov strojnega učenja (npr.: klasifikacija, regresija).
- storitve obdelave naravnega jezika (angl. Natural language processing services): storitve, ki omogočajo sintaktično in semantično analizo povedi v naravnem jeziku. Najpogostejše discipline v tej kategoriji vključujejo analizo čustev, analizo namere, modeliranje pogovora ipd.
- storitve za analitiko vida (angl. Vision analytic services): storitve, ki omogočajo razumevanje vsebine iz slik. Najpogostejše tehnike v tej kategoriji vključujejo klasifikacijo slik, analizo čustev, zaznavo obrazov in predmetov, itd.
- storitve za procesiranje govora (angl. Speech processing services): storitve, ki omogočajo zmožnosti analize govora, kot so pretvorba govora v besedilo, prevod govora ipd.
- storitve za upravljanje z znanjem (angl. Knowledge services): storitve, ki dopolnjujejo oz. nadgrajujejo obstoječe podatke z operacijami kot so ekstrakcija konceptov, povezovanje entitet ipd.
- storitve za pridobivanje vpogleda v podatke (angl. Data insights services): storitve, ki omogočajo vpogled v ciljne vire podatkov.

Poleg gigantov na področju PaaS se v boj za pridobitev strank oz. tržnega deleže na področju MLaaS oz. AIaaS vključujejo tudi številna zagonska podjetja. Cilj večine takšnih zagonskih podjetij seveda ni direktno konkurirati omenjenim gigantom, ampak čim bolje predstaviti svoje znanje s področja strojnega učenja, z namenom privabiti čim več pozornosti pri velikih podjetjih, katera bi jih potencialno lahko prevzela. Po drugi strani pa je ogromno zagonskih podjetij svoj trud usmerilo v apliciranje strojnega učenja oz. umetne inteligence na zelo nišne industrijske probleme ter se na tak način poskušajo prebiti na trgu. Na primer zagonsko podjetje Enlitic uporablja globoko učenje za pomoč zdravnikom pri prepoznavanju določenih bolezni oz. zdravstvenih stanj iz zajetih rentgenskih slik ali posnetkov magnetne resonance. Eno izmed trenutno bolj zastopanih področij je področje zaznave in prepoznave ljudi in objektov iz slik ter video posnetkov, s katerim se ukvarja veliko število zagonskih podjetij (Kairos, CloudSight, …). [7]

3.1. Inteligentne storitve

MlaaS oz. AIaaS je krovna definicija avtomatiziranih in polavtomatskih oblačnih platform oz. storitev, ki pokrivajo večino infrastrukturnih vprašanj, kot so predhodna obdelava podatkov, učenje in vrednotenje modelov ter nadaljnje napovedovanje s pomočjo omenjenih modelov. Rezultate napovedi je mogoče povezati z obstoječo notranjo IT infrastrukturo prek REST API. Omenjene storitve strankam pomagajo izkoriščati zmožnosti strojnega učenja brez večjih dodatnih stroškov v obliki časa in tveganja za vzpostavitev internega oddelka strojnega učenja. [8, 9]

Inteligentne storitve združujejo nabor predpripravljenih, generičnih rešitev strojnega učenja, ki naslavljajo različne probleme vse od prepoznave obrazov iz slik, procesiranja naravnega jezika in napovednih analitik pa do vizualizacije podatkov. V zaledju teh storitev so uporabljeni različni algoritmi strojnega učenja za namene iskanja vzorcev iz podatkov. Z uporabo teh vzorcev in algoritmov so nato zgrajeni matematični modeli, ki omogočajo napovedovanje na novo ustvarjenih oz. nepoznanih podatkov. Ključno je, da organizacije ki uporabljajo takšne storitve, ne potrebujejo poglobljenega domenskega znanja s področja strojnega učenja. [9]

3.2. Tipi inteligentnih storitev

Strojno učenje oz. še bolj splošno umetna inteligenca naslavlja širok nabor problemskih področij kot tudi različnih tehnologij. Trenutno lahko obstoječ nabor inteligentnih storitev razdelimo v sledeče kategorije: kognitivno računalništvo, osebni asistenti in roboti za pogovor (angl. Chatbot), ogrodja za strojno učenje ter popolnoma upravljane storitve strojnega učenja.

3.2.1. Kognitivno računalništvo

Velike količine podatkov, hranjenih v oblaku, predstavljajo za proces strojnega učenja ogromen vir informacij, ki skupaj z milijoni uporabnikov računalništva v oblaku in milijoni dnevno izvedenih procesov tvorijo veliko učno množico, nad katero se lahko stroj uči. Trenutno so na trgu nekateri primeri kognitivnega računalništva naredili izjemen napredek, predvsem na področju umetne inteligence. Tukaj lahko izpostavimo predvsem Microsoft z njihovim paketom kognitivnih storitev (Microsoft Cognitive Services), IBM s platformo Watson ter AWS s skupkom aplikacijskih storitev strojnega učenja (AWS ML Application Services). Ob tem je potrebno poudariti, da so kljub izjemnemu napredku v splošnem kognitivni računalniški sistemi trenutno še vedno v začetni fazi. Pričakovati pa je, da se bodo v naslednjih nekaj letih razvili do te mere, da bodo lahko takšni sistemi prevzemali tudi pomembnejše odločitve. [10]

3.2.2. Osebni asistenti in roboti za pogovor

Osebni asistenti posameznikom do določene mere že olajšujejo življenje. Produkti kot so Microsoft Cortana, Google Assistant in Apple Siri so sistemi za prepoznavo govora, ki strojem dajejo občutek človeškosti. Vendar pa imajo takšni asistenti omejene zmožnosti, saj so njihovi odzivi precej splošni in ne toliko personalizirani. Z masovnimi podatki v oblaku, zmožnostmi strojnega učenja in kognitivnimi sposobnostmi pa postajajo osebni asistenti vedno bolj personalizirani in sposobni ljudem podobne glasovne interakcije.

Roboti za pogovor so bili v svojih začetkih v središču pozornosti v storitveni industriji s ponujanjem rešitev in informacij uporabnikom preko spletnega klepeta, kar je delno olajšalo uporabniško podporo storitvenih podjetij. Z implementacijo strojnega učenja lahko zvišamo kognitivne sposobnosti takšnih botov, saj se ti lahko učijo iz predhodnih pogovorov in namesto golih sej »vprašanje – odgovor« med uporabnikom in robotom ponudijo interakcijo, podobno človeški. Glavni cilj pri tem je ustvariti robote čim bolj »človeške« in čim bolj personalizirane, kolikor je to pač mogoče. [10]

3.2.3. Ogrodja za strojno učenje

Ogrodja za strojno učenje v oblaku omogočajo razvijalcem ustvarjanje aplikacij, ki imajo sposobnost izboljšanja skozi čas. V splošnem zahtevajo, da razvijalci oz. podatkovni znanstveniki zgradijo model ter ga učijo z obstoječimi podatki. Takšna ogrodja so še posebej priljubljena pri razvoju aplikacij, povezanih z analizo masovnih podatkov (angl. Big Data Analytics), uporabiti pa jih je možno za ustvarjanje številnih drugih aplikacij. Dostop do teh ogrodij preko oblačnih storitev je navadno lažji in cenejši kot pa vzpostavitev lastne infrastrukture za opravljanje nalog strojnega učenja, kar je tudi ena izmed največjih prednosti uporabe takšnih storitev. [11]

3.2.4. Popolnoma upravljane storitve strojnega učenja

Včasih si organizacije želijo integracije zmožnosti strojnega učenja v aplikacije, vendar se pri tem soočijo s pomanjkljivim ali neobstoječim znanjem, potrebnim za uporabo metod in tehnik strojnega učenja. V takšnih primerih lahko uporabijo popolnoma upravljane storitve strojnega učenja, ki jim ponujajo vnaprej izdelane modele in / ali »povleci in spusti« orodja, ki omogočajo poenostavitev in pospešitev uporabe strojnega učenja. [11]

3.3. Prednosti in slabosti inteligentnih storitev

Mnoge organizacije, navadno večje, investirajo v lasten oddelek za strojno učenje oz. umetno inteligenco, kot tudi v nakup ustrezne lastne infrastrukture. Znaten delež organizacij pa se vendarle odloči za uporabo MLaaS oz. AIaaS predvsem zaradi številnih koristi, ki jih te vrste storitev prinašajo. Med njimi velja izpostaviti predvsem napredno infrastrukturo, manjše stroške, prilagodljivost računskih zmožnosti ter uporabnost. Aplikacije, ki uporabljajo metode in tehnike strojnega učenja, se najbolj učinkovito izvajajo na hitrih grafičnih procesnih enotah, katere naloženo delo izvajajo paralelno. Takšni sistemi so praviloma dragi in marsikateri organizaciji stroškovno nedosegljivi. MLaaS omenjenim organizacijam omogočajo dostop do visoko zmogljivih, prilagodljivih virov po ceni, katero si lahko privoščijo. Čeprav je večina najbolj razširjenih orodij, ogrodij in knjižnic odprtokodnih in tako organizacijam lahko dosegljivih, pa ta niso vedno enostavna za uporabo. Storitve MLaaS naslavljajo tudi to problematiko, saj strankam oz. organizacijam ponujajo vmesnike, navadno REST API, ki so enostavni za uporabo in integracijo v obstoječe okolje ter za njihovo uporabo ne zahtevajo domenskih strokovnjakov s področja strojnega učenja oz. umetne inteligence.

Dve največji pomanjkljivosti MLaaS oz. AIaaS kot storitve sta težavi, ki sta skupni tudi klasičnim računalniškim storitvam v oblaku – varnost in skladnost. Veliko aplikacij, ki uporablja metode in tehnike strojnega učenja, se zanaša na velike količine podatkov. Če se bodo ti podatki prenašali oz. bodo hranjeni v oblaku, morajo organizacije zagotoviti ustrezne varnostne ukrepe in šifriranje tako prenašajočih se, kot tudi hranjenih podatkov. V nekaterih primerih lahko predpisi tudi preprečijo hrambo določenih vrst občutljivih podatkov v oblaku ali pa je hramba podatkov omejena na določeno geografsko področje (kontinent, država), kar lahko predstavlja organizacijam veliko težavo. [11, 12]

4. UPORABA INTELIGENTNIH OBLAČNIH STORITEV

Za primer aplikativne uporabe inteligentnih oblačnih storitev smo se osredotočili na področje prepoznave obrazov. Kot cilj smo si zadali razvoj REST storitve, ki omogoča registracijo in avtorizacijo uporabnika s pomočjo prepoznave obraza iz slike, zajete s pomočjo spletne kamere, ter zajem video posnetka in analizo čustev osebe na posnetku. Ključen del pri implementaciji primera predstavljajo spletni aplikacijski vmesniki MLaaS storitev. V našem primeru smo implementirali MLaaS storitve za prepoznavo obrazov z rešitvami treh ponudnikov in sicer z Amazon Rekognition, Microsoft Face API ter Kairos Human Analytics API. Za demonstracijske namene delovanja ustvarjene REST storitve smo implementirani tudi spletnega odjemalca.

4.1. Primerjava funkcionalnosti storitev

Za namen pridobitve boljšega vpogleda v funkcionalnosti oz. zmožnosti inteligentnih oblačnih storitev izbranih ponudnikov smo opravili primerjavo med njimi, prikazano v tabeli 1. Iz rezultatov je razvidno, da imajo različni ponudniki večinoma podprte enake oz. zelo podobne funkcionalnosti.

Tabela 1: Primerjava funkcionalnosti inteligentnih oblačnih storitev med posameznimi ponudniki

	Amazon Rekognition	Microsoft Face API	Kairos Human Analytics API
Zaznava obraza	✓	✓	✓
Prepoznava obraza (slika)	✓	✓	✓
Prepoznava obraza (video)	✓[39]	✓[40]	✓
Čustvena globina (%)	✗	✓	✓
Prisotna čustva (Da/Ne)	✓	✓	✓
Starost in spol	✓	✓	✓
Sledenje več obrazom	✓	✓	✓
SDK (brez povezave s spletom)	✗	✗	✓[41]
API	✓	✓	✓

Amazon Rekognition ponuja prepoznavo slik, temelječo na globokem učenju. Storitev je integrirana v obstoječ Amazonov AWS ekosistem. Sama tehnologija temelji na tehnologiji leta 2015 prevzetega podjetja Orbeus in je bila prvič predstavljena konec leta 2016. Glavne funkcionalnosti storitve so analiza objektov in scen, zaznava ter prepoznava obraza in analiza čustev. Omejitve storitve se odražajo v:

- maksimalnem številu zaznanih obrazov na sliki (15),
- maksimalni velikost (15MB) hranjene slike v obliki Amazonovega S3 objekta,
- minimalni ločljivosti slike; minimalna višina in širina je 80 slikovnih točk,
- maksimalni velikosti slike (5MB) v obliki surovih bajtov, poslanih kot parameter API-ju,
- formatu slike (zgolj JPG in PNG),
- brez poglobljene analize čustev,
- maksimalnem številu obrazov, ki jih lahko hranimo v eni sami kolekciji (1 milijon), ter
- v maksimalnem številu ujemajočih se obrazov (4096), ki jih vrača iskanje preko API-ja.

Microsoft Face API (poznan tudi kot »Project Oxford«) je storitev, ki omogoča analizo slik in video posnetkov. Storitev je del Microsoftove platforme za kognitivne storitve Microsoft Cognitive Services. Glavne funkcionalnosti storitve so zaznava obrazov, verifikacija obrazov, prepoznava obrazov oz. identifikacija ter zaznava emocij. Omejitve storitve se odražajo v:

- maksimalnem številu slik v galeriji (1000),
- zaznava samo 27 obraznih značilnosti,
- maksimalnem številu zaznanih obrazov na posamezni sliki (64),
- maksimalni velikosti videa (100MB; 10 – 20 sekund videa pri ločljivosti 1080p in manj kot 30 sekund videa pri ločljivosti 720p) za zaznavo emocij,
- maksimalni velikosti slik (4MB), ter
- maksimalnem številu transakcij na sekundo (10).

Kairos Human Analytics API je osrednja storitev podjetja Kairos, ustanovljenega leta 2012, specializiranega za področje prepoznave obrazov. Glavne funkcionalnosti storitve so zaznava obrazov, identifikacija in prepoznava obrazov, zaznava in prepoznava emocij ter prepoznava demografskih podatkov. Omejitve storitve se odražajo v:

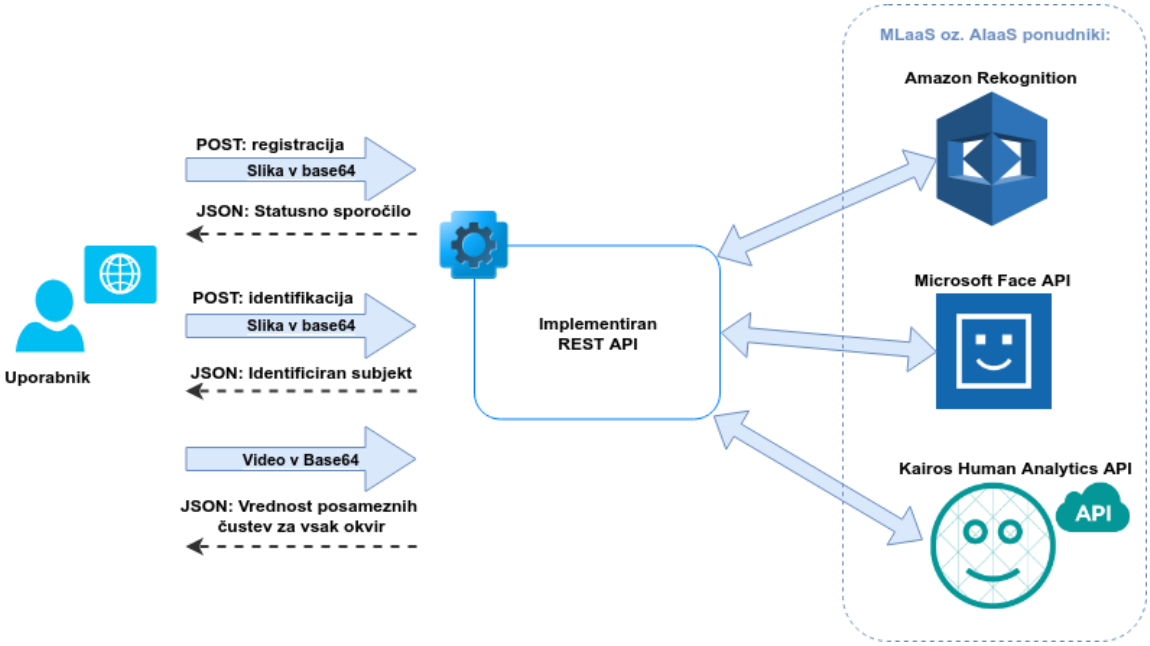
- zahtevanem minimalnem razmiku (75 slikovnih točk) med očmi fotografirane osebe,
- formatu podprtih slik (JPG, PNG) ob uporabi API-ja,

[39] Predhodno je potrebno ustvariti kolekcijo obrazov (funkcionalnost CreateCollection) iz video posnetka ter nato izvesti prepoznavo obraza oz. obrazov.
[40] Predhodno je potrebno iz video posnetka pridobiti posamezne okvirje (angl. Frame) ter nato nad temi okvirji izvesti prepoznavo obraza oz. obrazov.
[41] Samo v primeru uporabe plačljive naročnine na storitev.

- samodejni kompresiji in pretvorbi slik v sivinski spekter,
- brez podpore za hrambo slik ali video posnetkov,
- vsak klic na API se šteje v kvoto klicev,
- omejitev števila klicev na interval minuta/dan/mesec glede na zakupljen paket, ter
- SDK, ki je na voljo samo pri plačljivih paketih.

4.2. Zasnova primera uporabe

Primer uporabe inteligentnih oblačnih storitev smo zasnovali na način (slika 2), da ta deluje kot ovojnica obstoječim inteligentnim oblačnim storitvam. Zamislili smo si tri scenarije in sicer registracijo uporabnika, identifikacijo uporabnika ter analizo čustev iz zajetega video posnetka. Pri registraciji uporabnik določi ponudnika inteligentnih oblačnih storitev, preko katerega želi, da se proces izvede, določi svoje uporabniško ime ter zahtevku doda sliko, zakodirano z algoritmom Base64. Implementiran REST API poskrbi za nadaljnjo izvedbo scenarija pri izbranem ponudniku ter ob zaključku le-tega vrne statusno sporočilo v obliki JSON. Identifikacija uporabnika deluje na podoben način, le da v tem primeru uporabniku ni potrebno zraven slike pošiljati še uporabniškega imena. Po končani izvedbi procesa na implementiranem REST API-ju pa le-ta s strani izbrane inteligentne oblačne storitve vrne identificiran subjekt, prav tako v obliki JSON. Pri analizi video posnetka je enako kot pri identifikaciji, s to razliko, da v slednjem z algoritmom Base64 zakodiramo video posnetek, ga shranimo na datotečni sistem naše REST storitve ter ga nato pošljemo v analizo izbranemu ponudniku MLaaS storitve. Po končani analizi posnetka se odjemalcu vrne odgovor v obliki JSON, ki vsebuje prepoznane vrednosti oz. nivo zaupanja v posamezna čustva subjekta za vsak okvir video posnetka.



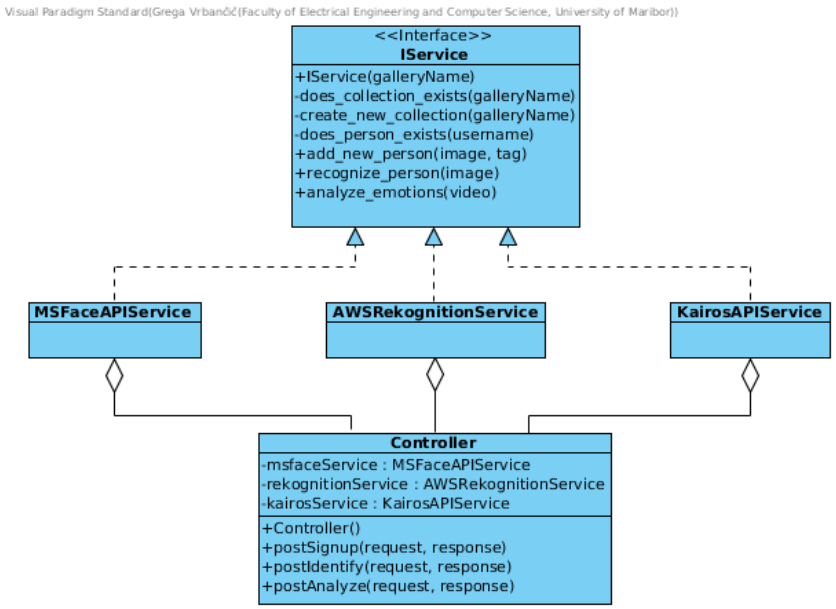
Slika 2: Konceptualni model primera uporabe

Primer uporabe smo implementirali v programskem jeziku Python z uporabo podpornih knjižnic (Boto3 za Amazon Rekognition, Kairos Face SDK Python za Kairos Human Analytics API ter Cognitive Face Python za Microsoft Face API) – ovojnic za posamezno inteligentno oblačno storitev. Za vzpostavitev spletne REST storitve smo uporabili ogrodje Flask.

4.3. Implementacija primera uporabe

Implementacije REST API smo se lotili na način, da smo najprej definirali posplošen vmesnik servisnih razredov, kateri v nadaljevanju, prav tako preko REST API, komunicirajo s posameznimi MLaaS

storitvami ponudnikov. Implementirane servisne razrede smo povezali z REST aplikacijskim vmesnikom s pomočjo metod kontrolnega razreda. Razredni diagram (slika 3) prikazuje zasnovo implementiranega primera uporabe.



Slika 3: Razredni diagram primera uporabe

Kontrolni razred implementira tri metode, katere so izpostavljene preko REST, in sicer metodo za registracijo, metodo za identifikacijo ter metodo za analizo čustev iz video posnetka. Metoda za registracijo v telesu zahtevka HTTP prejme tri parametre: uporabniško ime, preko katerega ponudnika želimo izvesti registracijo, ter sliko, zakodirano z algoritmom Base64. Prejete parametre validiramo, pridobimo format poslane slike ter jo shranimo na datotečni sistem za nadaljnjo uporabo. Glede na s parametrom izbranega ponudnika MLaaS kličemo razredno metodo za dodajanje osebe razreda, ki implementira nadaljnjo komunikacijo z REST API izbranega ponudnika. Po uspešnem vnosu nove osebe vrnemo v odgovoru na zahtevek HTTP stanje izvedene operacije v obliki JSON ter izbrišemo prej shranjeno sliko.

Metoda za identifikacijo v telesu zahtevka HTTP prejme dva parametra: ponudnika storitev in sliko, zakodirano z algoritmom Base64. Kot pri predhodni metodi, se tudi v tej naprej validirajo prejeti podatki. Za uspešno validacijo podatkov pridobimo format prejete slike ter jo shranimo na datotečni sistem. Glede na izbranega ponudnika MLaaS storitev nato kličemo metodo za identifikacijo osebe ustreznega razreda, ki implementira nadaljnjo komunikacijo z REST API. S strani REST API ponudnika MLaaS dobimo v obliki JSON polje oseb, za katere je model našel podobnosti s poslano sliko ter nivo prepričanosti oz. zaupanja za posamezno osebo. Kot odgovor na zahtevek HTTP vrnemo identificirano osebo z najvišjim nivojem prepričanosti oz. zaupanja, prav tako v obliki JSON. V nasprotnem primeru, torej v primeru, da oseba ni bila prepoznana, pa vrnemo statusno sporočilo. Na koncu, tako kot v prejšnji metodi, tudi v tej predhodno shranjeno sliko izbrišemo iz datotečnega sistema.

Metoda kontrolnega razreda za analizo čustev iz video posnetka deluje podobno kot metoda za identifikacijo, s to razliko, da namesto slike v telesu zahtevka HTTP prejme video posnetek, zakodiran z algoritmom Base64. Tega najprej shranimo na datotečni sistem ter zatem glede na izbranega ponudnika MLaaS storitev kličemo metodo za analizo čustev iz video posnetka ustreznega servisnega razreda. S strani REST API ponudnika MLaaS storitev dobimo v obliki JSON vrednosti za posamezna identificirana čustva (jeza, gnus, strah, veselje, žalost, presenečenost) v obliki realnih števil (med 0 in

100) za vsak okvir poslanega video posnetka. Prejete rezultate vrnemo kot odgovor na prejet zahtevek HTTP. Predhodno lokalno shranjen video posnetek izbrišemo iz datotečnega sistema.

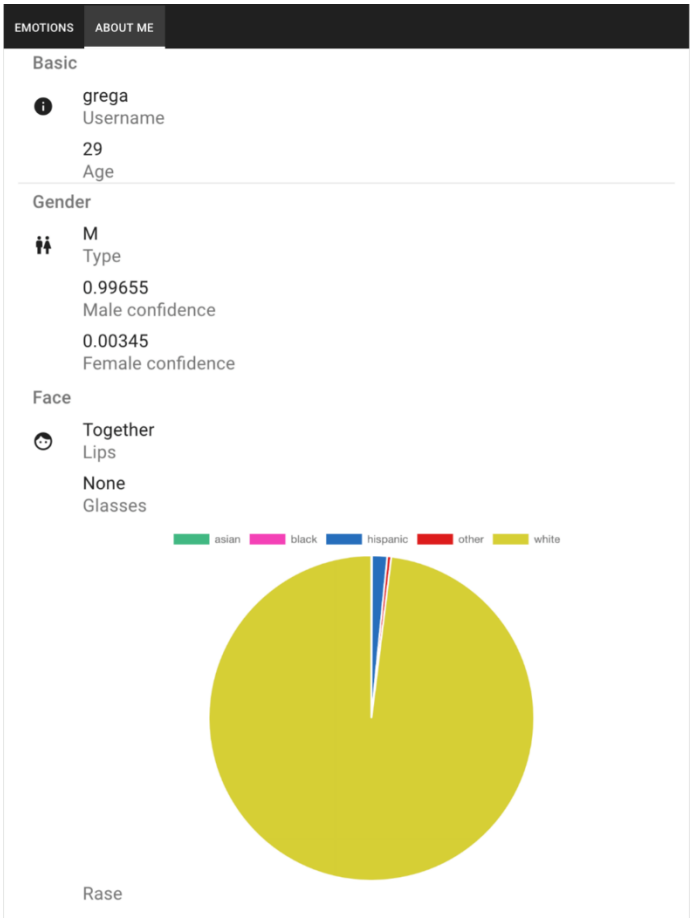
4.4. Rezultati

4.4.1. Prikaz delovanja

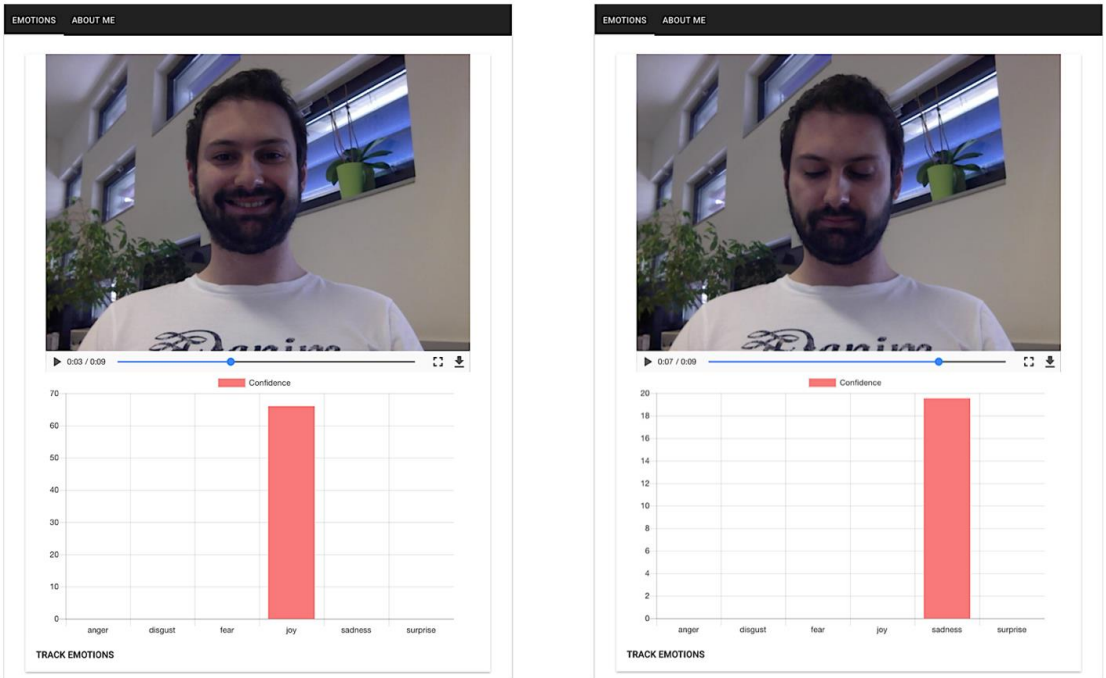
Za potrebe testiranja delovanja razvite storitve v praksi smo razvili enostaven odjemalec v obliki spletne aplikacije. Razvita aplikacija z uporabo spletne kamere preko brskalnika omogoča zajem slik, ki se uporabijo za potrebe registracije ter identifikacije, ter zajem video posnetka, iz katerega s pomočjo MLaaS storitev poskušamo izluščiti čustva posnetega subjekta.

Za identifikacijo uporabnika smo s pomočjo spletne kamere zajeli sliko, s pomočjo katere smo z uporabo inteligentnih oblačnih storitev identificirali predhodno registriranega uporabnika. Uporabljene storitve nam kot rezultat identifikacije, poleg identificiranega uporabnika, vrnejo še razpoznane lastnosti obraza (Slika 4). V prikazanem primeru je storitev s slike razpoznala starost osebe 29 let, pravilna starost pa je 27 let. Z 99,66% nivojem prepričanja je pravilno določila spol osebe. Prepoznala je, da oseba ne nosi očal ter ima zaprta usta. Dodatno je s slike razbrana rasa osebe, v našem primeru je z 98,04% razpoznala belca, z 1,5% latino-američana ter z 0,4% drugo oz. nekategorizirano raso.

V primeru razpoznave čustev z video posnetka smo s spletno kamero posneli 10 sekundni video posnetek ter ga analizirali z uporabo omenjenih storitev. Rezultati analize čustev so prikazani v obliki stolpičnega diagrama pod video posnetkom, kjer vsak izmed stolpcev prikazuje nivo prepričanja glede posameznega čustva. Na sliki 5 sta zaslonska posnetka spletnega odjemalca, kjer je na levi strani prikazana prepoznava čustva veselja z nivojem prepričanja 66%, na desni strani pa ravno nasprotje – žalosti, katerega je storitev prepoznala z nivojem prepričanja 19,5%.



Slika 4: Prikaz prepoznave lastnosti osebe za podlagi fotografije zajete za potrebe identifikacije uporabnika



Slika 5: Prikaz delovanja razpoznave čustev iz video posnetka

4.4.2. Uspešnost prepoznave

Za potrebe testiranja uspešnosti prepoznave obraza posamezne inteligentne oblačne storitve smo uporabili slike iz zbirke slik »Georgia Tech Face Database«[42] . V vsako storitev smo dodali novega uporabnika z eno sliko, pri čemer je subjekt na sliki imel pogled usmerjen direktno v kamero brez obraznih mimik. Identifikacijo uporabnika smo testirali s štirimi različnimi fotografijami:

- Slika 1: pogled naravnost v kamero, rahlo odprta usta.
- Slika 2: pogled naravnost v kamero, z očali.
- Slika 3: obraz obrnjen proti kameri, pogled usmerjen v tla.
- Slika 4: pogled naravnost v kamero, nasmeh.

Rezultati v tabeli 2 predstavljajo nivo zaupanja oz. prepričanja storitve, da je uporabnik pravilno identificiran. Kot je razvidno, se je najbolje, v vseh pogledih, odrezala storitev Amazon Rekognition, medtem ko imata preostali dve storitvi podobno povprečje zaupanja oz. prepričanja. Zanimivo je, da imata obe omenjeni storitvi slabši nivo zaupanja pri testu identificiranja uporabnika s fotografijo z dodanimi očali in z nasmeškom fotografirane osebe, medtem ko imata pri ostalih fotografijah bistveno boljše rezultate, a še vedno opazno slabše kot storitev Amazon Rekognition.

Tabela 2: Rezultati uspešnosti identifikacije subjekta

	Amazon Rekognition	Microsoft Face API	Kairos Human Analytics API
slika 1	0,9997	0,8949	0,9146
slika 2	0,9999	0,7244	0,7609
slika 3	0,9998	0,9174	0,9265
slika 4	0,9998	0,7776	0,686
Povprečje zaupanja	0,9998	0,8286	0,822

5. ZAKLJUČEK

V članku smo predstavili osnovni koncept računalništva v oblaku ter inteligenten oblak oz. inteligentne oblačne storitve, ki predstavljajo združitev že uveljavljenega koncepta računalništva v oblaku z naprednimi metodami in pristopi strojnega oz. globokega učenja. Povzeli smo potencialne vplive omenjene združitve tehnologij ter predstavili osnovne trenutne smernice oz. gibanja na trgu.

V nadaljevanju smo podrobneje analizirali inteligentne oblačne storitve treh različnih ponudnikov ter se pri tem osredotočili na področje prepoznave obrazov. Primerjali smo funkcionalnosti storitev ponudnikov Amazon, Microsoft in Kairos ter izpostavili njihove omejitve. Zatem smo predstavili praktični primer uporabe teh storitev – implementacijo registracije oz. dodajanje novega uporabnika ter identifikacijo uporabnika glede na podano fotografijo. Analizirali smo tudi rezultate uspešnosti prepoznave posameznih storitev, pri čemer smo ugotovili, da se je najbolje izkazala storitev Amazon Rekognition, medtem ko sta preostali storitvi imeli nekoliko slabše rezultate v primerih, ko smo poskušali identificirati uporabnika s korekcijskimi očali ter obrazno mimiko, kar tudi sicer na področju prepoznave obrazov še vedno predstavlja izziv.

Ne glede na vse pozitivne vidike uporabe inteligentnih oblačnih storitev, pri čemer je potrebno v prvi vrsti izpostaviti predvsem dejstvo, da za nadgradnjo obstoječe oz. razvoj nove rešitve z uporabo strojnega učenja ne potrebujemo poglobljenega domenskega znanja s področja strojnega učenja, je po drugi strani potrebno izpostaviti tveganja, ki izhajajo iz tveganj uporabe računalništva v oblaku. Kot dve ključni tveganji bi izpostavili nadzor in varovanje osebnih podatkov, ki v primeru uporabe takšnih storitev ni več v naši domeni, ter odvisnost od ponudnika storitve.

[42] Prosto dostopna zbirka slik Georgia Tech Face Database dostopna na: http://www.anefian.com/research/face_reco.htm

V splošnem smo z uporabo vseh omenjenih storitev imeli pozitivne izkušnje. Izpostaviti bi veljalo, da izmed vseh ponudnikov Kairos ne omogoča fleksibilnosti v tolikšni meri kot jo omogočata preostala dva ponudnika. Ugotovljeno pravzaprav ne preseneča, če vemo, da Kairos predstavlja specializirano storitev, medtem ko sta obe ostali rešitvi precej bolj splošno namenski, s čimer ohranjata višjo stopnjo nadzora. Razlika med preskušenimi rešitvami je razvidna tudi pri kvaliteti in obširnosti dokumentacije, ki prav tako govori v prid ponudnikoma Amazon in Microsoft.

6. LITERATURA

[1] SMOLA Alex, Introduction to Machine Learning, Cambridge University Press, 2010.

[2] MELL Peter, Grance Timothy, "The NIST Definition of Cloud Computing", Recommendations of the National Institute of Standards and Technology, National Institute of Standards and Techonology, 2011.

[3] HOFER C. N., KARAGIANNIS Georgios, "Cloud computing services: taxonomy and comparison.", Journal of Internet Services and Applications, številka 2, 2011, str. 81-94.

[4] MARINESCU Dan C, Cloud Computing: Theory and Practice, Morgan Kaufmann, 2017.

[5] https://www.botmetric.com/blog/machine-learning-impact-on-cloud-computing/, Machine Learning's Impact on Cloud Computing, obiskano 2. 4. 2018.

[6] https://medium.com/@jrodthoughts/which-paas-is-winning-the-machine-learning-and-artificial-intelligence-race-2640e1e96eed, Which PaaS is Winning the Machine Learning and Artificial Intelligence Race?, obiskano 3. 4. 2018.

[7] HSU Jeremy, "For Sale: Deep Learning", IEEE Spectrum, številka 8, Avgust 2016, str. 12-13.

[8] https://www.altexsoft.com/blog/datascience/comparing-machine-learning-as-a-service-amazon-microsoft-azure-google-cloud-ai/, Comparing MLaaS: Amazon AWS, MS Azure, Google Cloud AI, obiskano 12. 5. 2018.

[9] https://analyticsindiamag.com/what-is-machine-learning-as-a-service-mlaas/, What Is Machine Learning As A Service (MLaaS)?, obiskano 8. 5. 2018.

[10] https://www.botmetric.com/blog/machine-learning-impact-on-cloud-computing/, Machine Learning's Impact on Cloud Computing, 9. 5. 2018.

[11] https://www.datamation.com/cloud-computing/artificial-intelligence-as-a-service-ai-meets-the-cloud.html, Artificial Intelligence as a Service: AI Meets the Cloud – Datamation, obiskano 9.5.2018.

[12] https://www.datamation.com/cloud-computing/cloud-machine-learning-is-it-right-for-you.html, Cloud Machine Learning: Is It Right for You? – Datamation, obiskano 12. 5. 2018.

Univerzitetna založba Univerze v Mariboru

RAZVOJ INTELIGENTNE REŠITVE WATSON ZLITINE

SARA HMELAK, MATIC STRAJNŠAK IN GREGOR KOVAČEVIČ

Povzetek: Prispevek predstavlja tehnologijo uporabljeno pri razvoju inteligentne rešitve Watson Zlitine. Gre za rešitev, ki s pomočjo kognitivnih algoritmov in z uporabo umetne inteligence vodi pogovor z virtualnim asistentom o pripravi in sestavi nove aluminijaste zlitine ter njenih ključnih mehanskih lastnosti. S pomočjo virtualnega asistenta lahko simuliramo sestavo nove zlitine ter na ta način minimiziramo porabo primarnih elementov in maksimiziramo porabo odpadkov (scrap-a). Hkrati pa z naprednimi kognitivnimi algoritmi predvidimo lastnosti končnega izdelka. S tem pa se že v fazi načrtovanja zlitine optimizirajo stroški, poraba surovine na skladišču ter predvidi ustreznost končnih lastnosti zlitine.

Ključne besede: • virtualni asistent • kognitivno računalništvo • umetna inteligenca • BigData • speech to text – text to speech

NASLOV AVTORJEV: Sara Hmelak, BI Specialist, Alcad d.o.o., Mroževa 5, 2310 Slovenska Bistrica, e-pošta: sara.hmelak@alcad.si. Matic Stajnšak, Študent FERI Maribor, e-pošta: matic.strajnsak@gmail.com. Gregor Kovačevič, Študent FERI Maribor, e-pošta: gregor.kovacevic@gmail.com.

DOI https://doi.org/10.18690/978-961-286-162-9.21 ISBN 978-961-286-163-6

Dostopno na: http://press.um.si.

1. UVOD

Krepko smo že vstopili v čas kognitivne sfere, velikih količin podatkov in novih tehnologij obvladovanja le-teh. Zaradi digitalizacije in mobilnosti podatki nastajajo neprestano, hkrati pa jih zaradi vedno zmogljivejših računalnikov lahko shranjujemo in obdelujemo. Izkazalo se je, da ni pomembno koliko podatkov generiramo in shranimo, temveč predvsem kako učinkovito znamo le-te izkoriščati. Podatki sami zase ne predstavljajo konkurenčne prednosti, dodano vrednost prinese hitra in učinkovita analiza.

Kako podatke napraviti učinkovite in hitre? Učinkovite, take ki z analitično obdelavo dobijo odločitveno vrednost in hitre, da so odločitvenih zmožnosti zmožni v realnem času. Hkrati pa ne posegamo več le po strukturiranih podatkih. Podatki niso več le številčni, strukturirani objekti in tabele, vedno več je nestrukturiranih podatkov – slike, grafi, tekst… - teh pa ne zmoremo več obvladovati na klasičen način. Zato posegamo po novih konceptih kot je *kognitivno računalništvo,* ki zajema naslednjo stopnjo razumevanja.

Doba množičnih podatkov in korelacij, ki nam jih te zbirke podatkov ponujajo kaže, da se bomo morali navaditi na način odzivanja, ki ni ne intuitivni in ne vzorčni, ampak korelacijski. Gre za analizo množice podatkov, ki ne išče vzrokov ampak zgolj verjetne povezave. In to omogočajo napredni, inteligentni, kognitivni algoritmi. Kognitivno računalništvo se nanaša na sisteme, ki so sposobni učenja iz izkušenj, namesto da bi jim znanje naložili s programom oz. jih sprogramirali.

Kako množico podatkov (BigData) pametno uporabiti ter vključiti dodano vrednost kognitivne tehnologije v rednih procesih proizvodnje?

Proizvodnji procesi ustvarjajo velik nabor najrazličnejših podatkov. Končni izdelek je odvisen od množice nastavljenih parametrov. Če te nastavljene parametre sistematično zbiramo, jih uparimo še z izmerjenimi parametri ter jih oblikujemo v BigData matrike, lahko s pomočjo umetne inteligenco simuliramo obnašanje v proizvodnji.

Osnovna ideja našega projekta je bila povezati procesne parametre različnih proizvodnih procesov in s pomočjo umetne inteligence ter kognitivnega računalništva ugotoviti medsebojne vplive kemijske sestave in ključnih parametrov procesa na mehanske lastnosti izdelkov in obratno. Vse to pa povezati z virtualnim asistentom, ki zahtevke sprejema kot nativni pogovor z uporabnikom ter zna iz teksta ali govora izluščiti bistvo, sprožiti akcijo ter podati rešitev.

2. OSNOVNI POJMI

2.1. Nameni pogovora

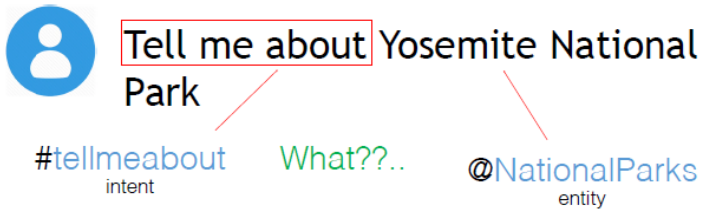
Nameni pogovora (angleško *intents*) se pri virtualnem asistentu uporabljajo, da iz danega teksta ugotovimo kakšni so nameni uporabnika. Preko razpoznanih namenov pogovora tako virtualni asistent ve o kateri temi ga sprašujemo in zna pripraviti odgovor. Definiramo toliko namenov kolikor funkcionalnosti (odgovorov) želimo, da naš virtualni asistent podpira in prepozna v tekstu.

2.2. Entitete

Z *entitetami* virtualnemu asistentu povemo kaj so stvari zanimanja v pogovoru. Z entitetami določamo objekte, ki so relevantni glede na naše namene pogovora. Entiteta definira tip objekta ter njegov kontekst za namen pogovora.

Primer: entiteta »Zlitina« z vrednostmi »EN 6082, EN 5005, EN 3100«, kjer so EN** primeri različnih serij zlitin. Virtualni asistent v stavku, kjer se katera izmed zgoraj naštetih zlitin pojavi, prepozna, da se v danem besedilu z določenim namenom pojavi entiteta »Zlitina«.

Slika 1 prikazuje primer *namena* in *entitete*, ki nastopita v stavku. Virtualni asistent, ki bi bil naučen na namen »tellmeabout« in entitete »NationalParks«, bi nam stavek razpoznal kot je razvidno iz slike.

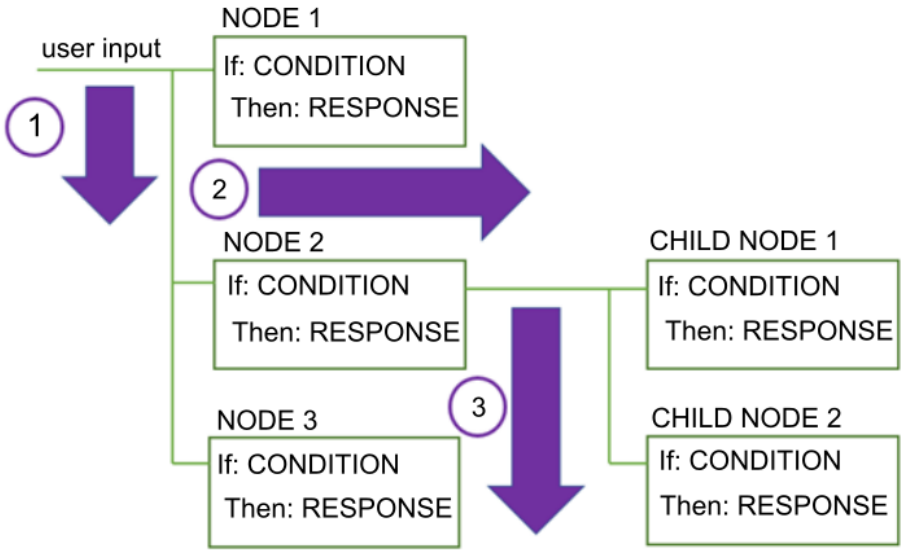


Slika 1. Primer entitete in namena pogovora v stavku.

2.3. Pogovorna drevesa (dialogi)

Pogovorna drevesa so vnaprej definiran diagram poteka pogovora z virtualnim asistentom. Definirajo tok pogovora in dane odgovore v odvisnosti od zaznanih *namenov pogovora* in *entitet*. Pogovorno drevo zgradimo na podlagi potreb poteka pogovora.

Iz slike 2 je razvidno, da se pogovorna drevesa uporabljajo za vodenje pogovora. Komunikacija med virtualnim asistentom in uporabnikom tako ni samo *vprašanje - odgovor*, vendar lahko ob zaznanih specifičnih *namenih* in *entitetah* podamo specifične odgovore ali pa postavimo dodatna vprašanja na katera podan uporabnikov odgovor odloči v katero smer drevesa se bo pogovor nadaljeval. Gre za neke vrste kreiranje *odločitvenega drevesa* za pogovor.



Slika 2. Primer preprostega pogovornega drevesa

3. RAZVOJ POGOVORNEGA ASISTENTA

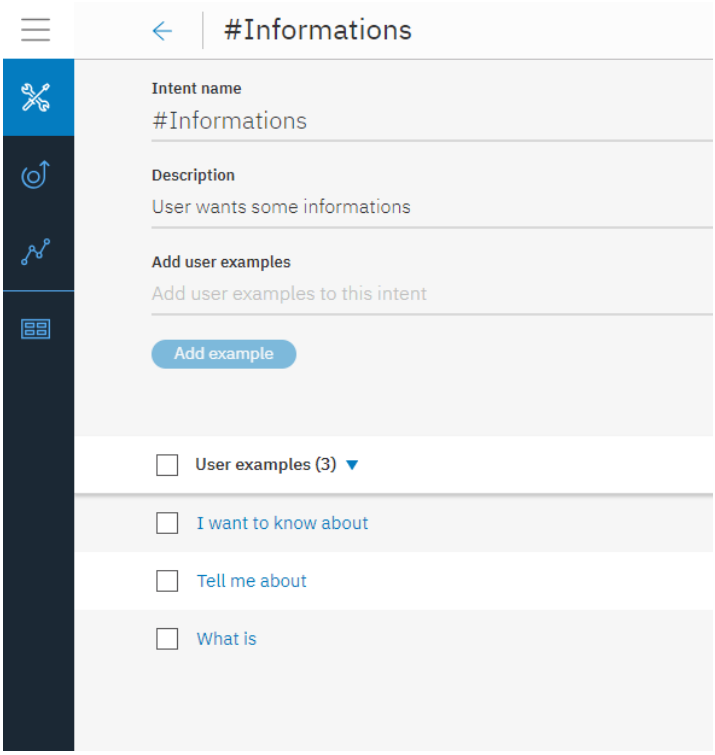
3.1. Ustvarjanje novega okolja

Virtualni asistent rešitve *Watson Zlitine* je napravljen s pomočjo IBM spletne platforme IBM Cloud, prej poznane kot IBM Bluemix. Za razvoj virtualnega asistenta ni potrebno eksplicitno znanje programiranja zato razvoj poteka precej hitro in enostavno.

Na platformi lahko ustvarimo več okolij za razvoj same rešitve. To pomeni, da lahko hkrati načrtujemo več različnih verzij asistenta ter se kasneje odločimo katero bomo uporabili v dejanski aplikaciji. Seveda vsake izmed verzij ni potrebno razvijati od začetka. Asistenta lahko v kateremkoli stadiju razvoja izvozimo ter ga po potrebi uvozimo v novo okolje. Končna razlika za razvijalca je le v tem kateri servis se kliče iz IBM strežnika.

3.2. Kreacija namenov

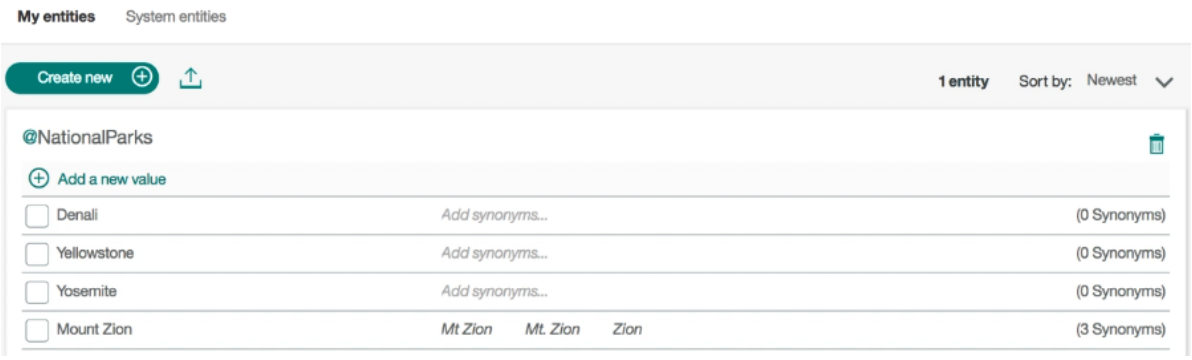
Razvoj virtualnega asistenta se prične z dodajanjem *namenov*. Asistentu lahko dodamo poljubno število namenov, ki jih bo razumel. Število je v celoti odvisno od tega kakšno funkcionalnost želimo podpreti. Za vsak namen, ki ga podpremo, moramo asistenta naučiti kakšne vnose uporabnika lahko pričakuje. Zadostuje že, da za vsak namen podamo 6 primerov uporabe, večje kot pa bo število primerov uporabe, boljši bo asistent pri razpoznavanju končnega namena uporabnika. Iz podanih primerov uporabe se nato IBM-ova programska oprema, preko umetne inteligence, sama nauči razpoznavanja namena, tudi če uporabnik asistenta ni povprašal povsem na enak način kot je podano v primerih.



Slika 3: Dodajanje namena »Informations«, ter primerov uporabe

3.3. Dodajanje entitet

Podobno kot dodajanje namenov je tudi ustvarjanje *entitet* zelo hitro opravilo. Definirati je potrebno ime entitete - torej stvar, ki jo hočemo v besedilu uporabnika zaznati, ter njene sopomenke v kolikor te obstajajo in pričakujemo, da se bodo uporabljale.



Slika 4: Dodajanje entitet »NationalParks«, ter njihovih sinonimov

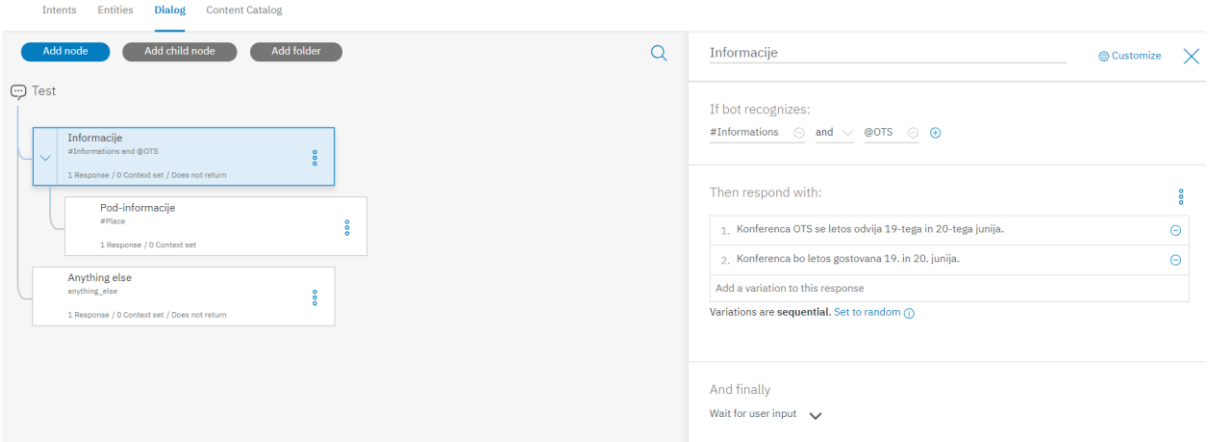
3.4. Ustvarjanje dialoga

V *dialogu* razvijalec nastavi na kakšen način se bo virtualni asistent odzval glede na zaznane *namene* in *entitete*. Odziv se lahko specificira samo na določen namen ter entiteto ali pa na kombinacijo le-teh. Tako lahko natančno definiramo odgovor, ki ga prejme uporabnik oz. strežnik v kolikor podatke na strežniku pred posredovanjem končnemu uporabniku še dodatno obdelamo.

Odziv asistenta na enako zaznane namene in entitete pri različnih časovnih in lokalnih povpraševanjih se lahko razlikuje v kolikor je razvijalec definiral več možnih odgovorov. Ti odgovori se lahko izbirajo zaporedno ali pa naključno pri čemer se naključen izbor odgovora vrši preko IBM-ove programske opreme.

Ko virtualni asistent poda odgovor, ponovno čaka na nov vnos uporabnika ali pa preide v neko drugo stanje, v kolikor je razvijalec to specificiral, ter tam poda nek drug odgovor.

Možno je tudi gnezdenje zaznavanja namenov in entitet. S tem poskrbimo, da vodimo tok pogovora. Tako lahko na primer uporabnik zahteva neke splošne informacije o nekem dogodku npr. kakšen dogodek je, potem pa pričakujemo, da ga bodo zanimale še ostale informacije v zvezi s tem dogodkom kot so čas in kraj dogodka. Zato je logično, da te odgovore ugnezdimo. S tem poskrbimo, da je tok pogovora bolj naraven, hkrati pa izvzamemo neposredne odgovore na prvo vprašanje.

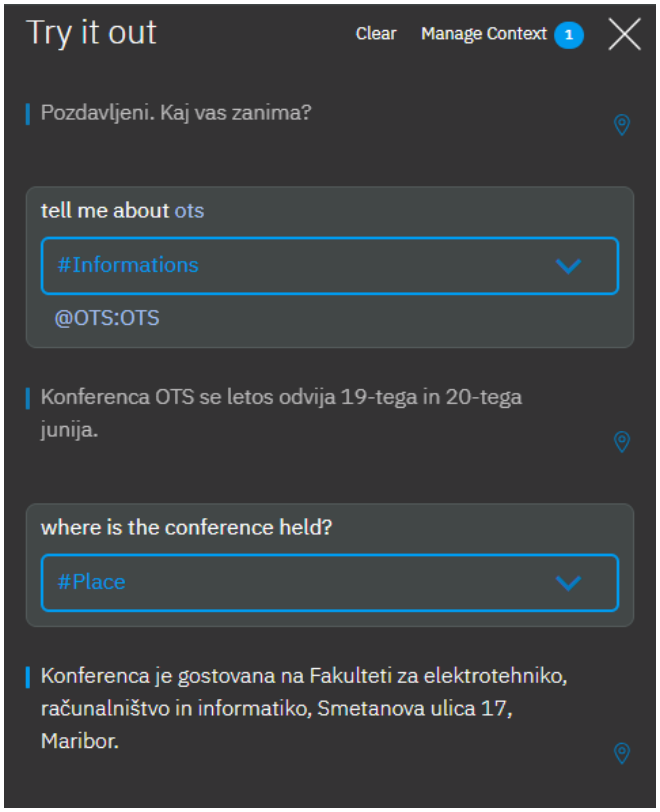


Slika 5: Ustvarjanje dialoga z več možnimi odgovori ter ugnezdenim zaznavanjem

3.5. Hitro testiranje ustvarjenega asistenta

Ko je osnutek virtualnega asistenta končan ga lahko preprosto preskusimo kar v IBM-ovi spletni aplikaciji. Tam vidimo, ali so zaznani ustrezni nameni in entitete ob določenem vnosu, ter v kolikor niso to tudi ročno ustrezno popravimo. V tem primeru se asistent s pomočjo umetne inteligence ponovno nauči novih povezav. V kolikor pa testiramo preko našega strežnika, imamo dostop tudi do dodatnih podatkov, kot je npr. prepričanost asistenta v pravilno zaznan namen, kar nam poda nek dodaten vpogled v delovanje.

Pri testiranju je omogočeno tudi dodajanje določenih spremenljivk v kontekst, kar se večinoma uporablja kasneje pri povezovanju s strežnika, kjer na takšen način prenašamo podatke med asistentom in strežnikom.



Slika 6: Preprost primer testiranja ustvarjenega virtualnega asistenta. V modrem so prikazani nameni in entitete, ki jih je asistent zaznal ter odgovori, ki jih je podal.

4. REŠITEV WATSON ZLITINE

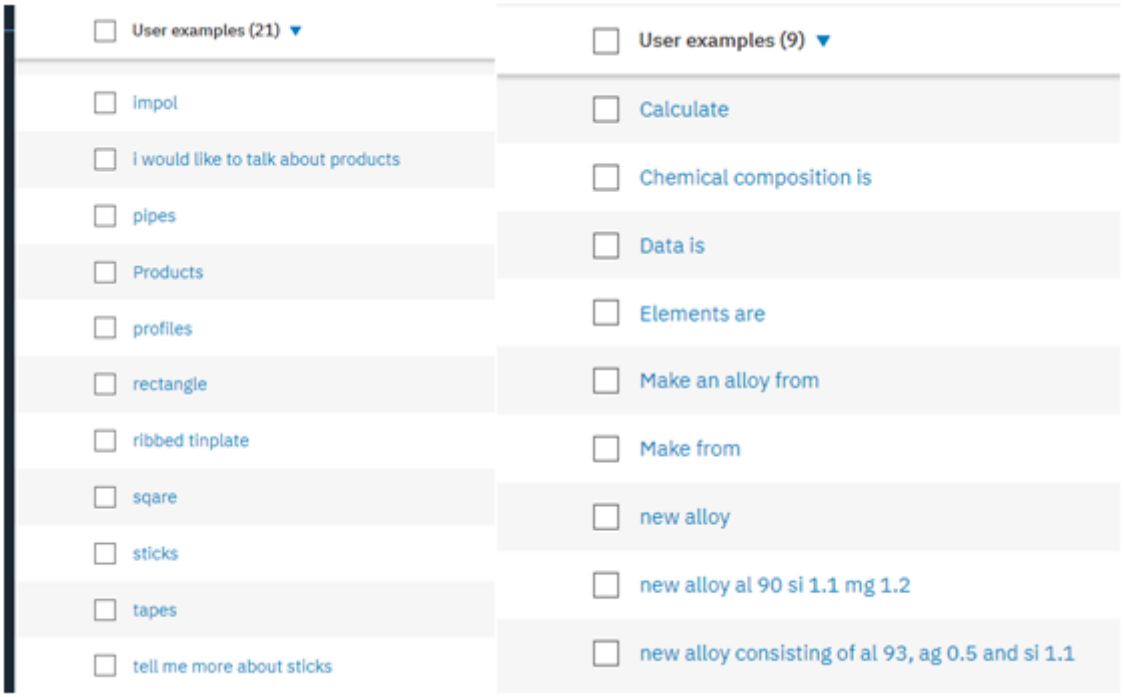
4.1. IBM Cloud konfiguracija

Na platformi IBM Cloud smo ustvarili servis IBM Watson Conversation (sedaj Watson Assistant) do katerega smo nato dobili url naslov, uporabniško ime ter geslo. Te podatke kasneje v NodeJS delu uporabimo za dostop do naše storitve. Nato smo ustvarili novo delovno okolje imenovano »Zlitine«.

Glede na zahteve aplikacije Watson Zlitine smo ustvarili štiri namene pogovora:

- Namen »tellmeabout« je splošen namen, ki nam določa tematiko Impolskih zlitin. Zraven sodijo informacije o internih nomenklaturah[44] podjetja za te zlitine, kot tudi splošne informacije o zlitinah iz spletne enciklopedije Wikipedija.
- Namen »impolProducts« je bil snovan za potrebe splošnih informacij o produktih podjetja Impol.
- Namen »maxValue« uporabljamo za pogovore o maksimalnih mehanskih lastnostih (npr. natezna trdnost, raztezek …). Sem sodi iskanje zlitine pri kateri podana sestava (npr. vsebnost aluminija v zlitini je vsaj 96%) ustreza zlitini in je najboljša glede na želeno mehansko lastnost.
- Namen »newalloy« uporabljamo za iskanje hipotetičnih novih zlitin tako da podamo sestavo zlitine in nam Watson ob pomoči IBM SPSS Modeler, kjer je zgrajen napovedni model nevronskih mrež, vrne teoretične mehanske lastnosti zlitine za podano sestavo.

Slika 7 prikazuje primer namenov za namenov »impolProducts« levo, ter »newalloy« desno.



Slika 7: Namen pogovora

Ko so *nameni* bili definirani je bilo potrebno specificirati in definirati *entitete.* Entitete nam definirajo objekte zanimanja.

V našem primeru to pomeni, da potrebujemo glavne entitete:
- »impolProduct« da lahko razločimo za katere produkte podrobneje uporabnik želi informacije
- »kemijski_elementi« za sestavo zlitin - tako imamo definirane vse elemente, ki nastopajo v zlitinah
- »mehanske_lastnosti« preko katere pridobimo informacijo katera mehanska lastnost je uporabniku v interesu
- »metals« za potrebe določevanja zlitin - entiteta predstavlja primerke imen vseh zlitin, ki se v podjetju pojavljajo in tudi vse njihove sinonime
- »procesni_parametri« za potrebe pridobivanja informacij o procesnih parametrih zlitine, s katero tudi določamo za kateri parameter uporabnik išče informacije

[44] Nomenklatura je sestava zlitine podana v intervalih posameznih kemijskih elementov.

Slika 8 prikazuje uporabljene entitete in nekaj vidnih primerov, ki določajo posamezne entitete.

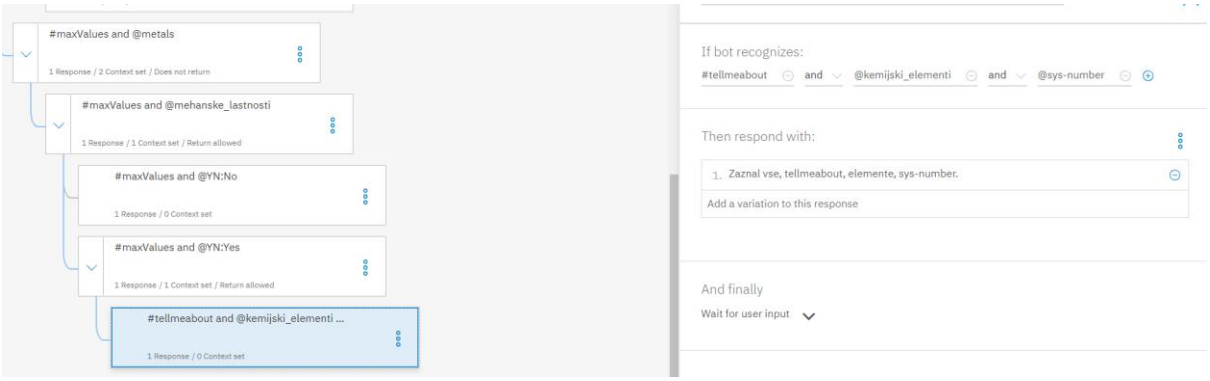
Entity (8)	Values
@impolProduct	tapes, colored products, sticks, profiles, tinplate, pipes, f...
@kemijski_elementi	Ga, Mo, Tl, Cd, Fe, Mn, Mg, Si, Zn, Zr, Be, Cr, Cu, Ni, Pb, Sn,...
@mehanske_lastnosti	RAZTEZ, NATTRD, DOGNAPT
@metals	5049, 5050A, 5052, 5083, 5086, 5087, 5182, 5183, 525...
@nomenclature	internal, general, both
@procesni_parametri	flow rate, speed, pressure, temperature
@typeOfAluminiumSticks	Square, Hexagon, Flat, Round
@YN	Yes, No

Slika 8: Ustvarjene entitete

Po definiranju *namenov* in *entitet* je na vrsti *pogovorno drevo*. Veje pogovornega drevesa smo definirali za vsak predviden *namen* pogovora. Pogovorna drevesa smo gradili na zaznavanju *namenov* pogovora in *entitet*.

Slika 9 prikazuje vejo pogovora o maksimalnih lastnostih zlitine. Uporabili smo *namen* »maxValues« in *entiteto* »metals«. V primeru, da Watson v uporabniškem vnosu v stavku zazna oboje bo sklepal, da gre za pogovor o maksimalnih lastnostih in bo vstopil v to vejo pogovora ter podal definiran odgovor. Nato Watson čaka na nov vnos v tej veji. V našem primeru dobimo odgovor: »Do you want to enter additional parameters (limitations) for this alloy? (Yes/No)«, in nato čaka na potrditev, kar je razvidno iz slike (tretji in četrti dialog), saj se pričakuje entiteta YN, ki ima vrednost da ali ne.

Na podoben način so razvite veje pogovornega drevesa za vse možne dialoge, ki virtualni asistent »Watson Zlitine« podpira.



Slika 9: Primer veje pogovornega drevesa za "maxValues"

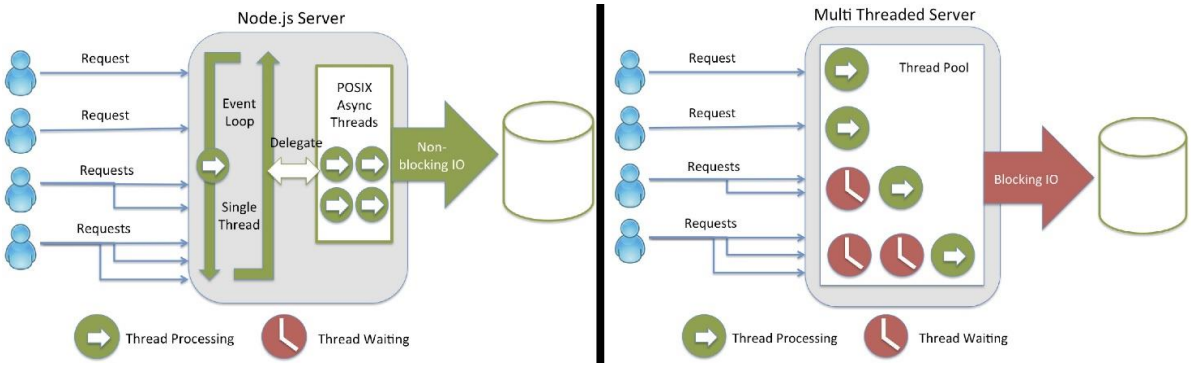
4.2. Končna implementacija

Aplikacija je sestavljena iz različnih delov, med katerimi so glavni strežnik, klient, virtualni asistent in baza podatkov, ki delujejo vzajemno ter dopolnjujejo funkcionalnost programa. Klient se uporablja za prikaz uporabniškega vmesnika ter služi za vnos uporabniških vprašanj, ki so osnova saj celotna aplikacija temelji na kognitivnem delovanju. Postavljeno vprašanje se nato prenese na lokalni server kjer se takoj za tem, preko API poizvedbe storitve Watson Conversation, pošlje povpraševanje na IBM-ov strežnik, ki odgovori z zaznanimi *namen*i ter *entitetami*. Glede na dobljene podatke API klica uporabniku takoj vrnemo želen odgovor, oziroma v kolikor je to potrebno, dodatne podatke pridobimo iz podatkovne baze ter nato strukturiramo odgovor.

4.2.1. Strežnik

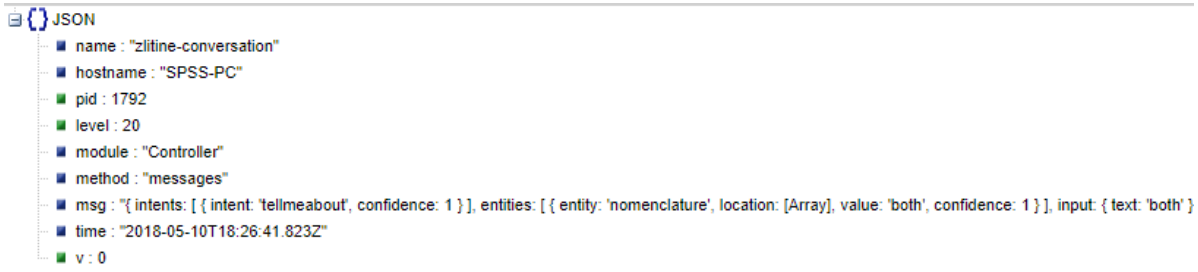
Pri izboru strežniškega okolja smo bili omejeni glede na podporo IBM-ovega API-ja - storitve Watson Conversation. V času našega razvoja so bili podprti jeziki Curl, NodeJS, Java in Python. Kot potencialna programska jezika za razvoj strežnika sta bili izbrana NodeJS, ki ga je priporočal IBM in Java v obliki Java servleta, ki je sicer v večini primerov uporabljen na produkciji.

Tako je bila prva verzija strežnika napisana v NodeJS, kasneje pa tudi v programskem jeziku Java. Ob tem se je izkazalo, da je ob primerjavi pošiljanja poizvedb na oba strežnika ter merjenju časa, strežnik napisan v NodeJS hitrejši pri dostavljanju odgovorov končnemu uporabniku. Ob tem je bila logika obeh strežnikov zelo podobna, saj je nato v primeru Java strežnika šlo le za prepis JavaScript kode v ustrezne strukture, ki jih podpira Java. Razlog za hitrejše delovanje lahko pripišemo asinhronemu delovanju NodeJS strežnika pri katerem ni blokiranja poizvedb in posledično čakanja na obdelavo zahteve.



Slika 10: Primerjava strežnika NodeJS (levo) in Java strežnika (desno)

Glavna naloga strežnika je pred-obdelava podatkov, ki se posredujejo uporabniku, sprejemanje vnesenega teksta s strani klienta, ter razni klici API-jev. Tako strežnik najprej sprejme besedilo, ga posreduje IBM-ovi storitvi Watson Conversation, ta pa strežniku nazaj vrne zaznane *namene*, *parameter prepričanosti* v pravilno detekcijo in *entitete*. Odgovori IBM-ove storitve so v notaciji JSON, ki ga vsakič ustrezno razčlenimo, da lahko izluščimo podatke, ki so nam pomembni. V kolikor je *parameter prepričanosti* dovolj velik (vsaj 70 % prepričanost, da je bil zaznan pravilen *namen*), strežnik pripravi ustrezen odgovor. V nasprotnem primeru uporabnik prejme sporočilo, da njegovega namena ni možno razbrati. Ob pravilni detekciji strežnik pogovor vodi dalje ter operira s podatki iz spleta oz. podatkovne baze IBM DB2.



Slika 11: Primer JSON odgovora IBM-ove storitve Watson Conversation. V ključu »msg« so vidni zaznani nameni, entitete ter prepričanja v pravilnost zaznave (»confidence«).

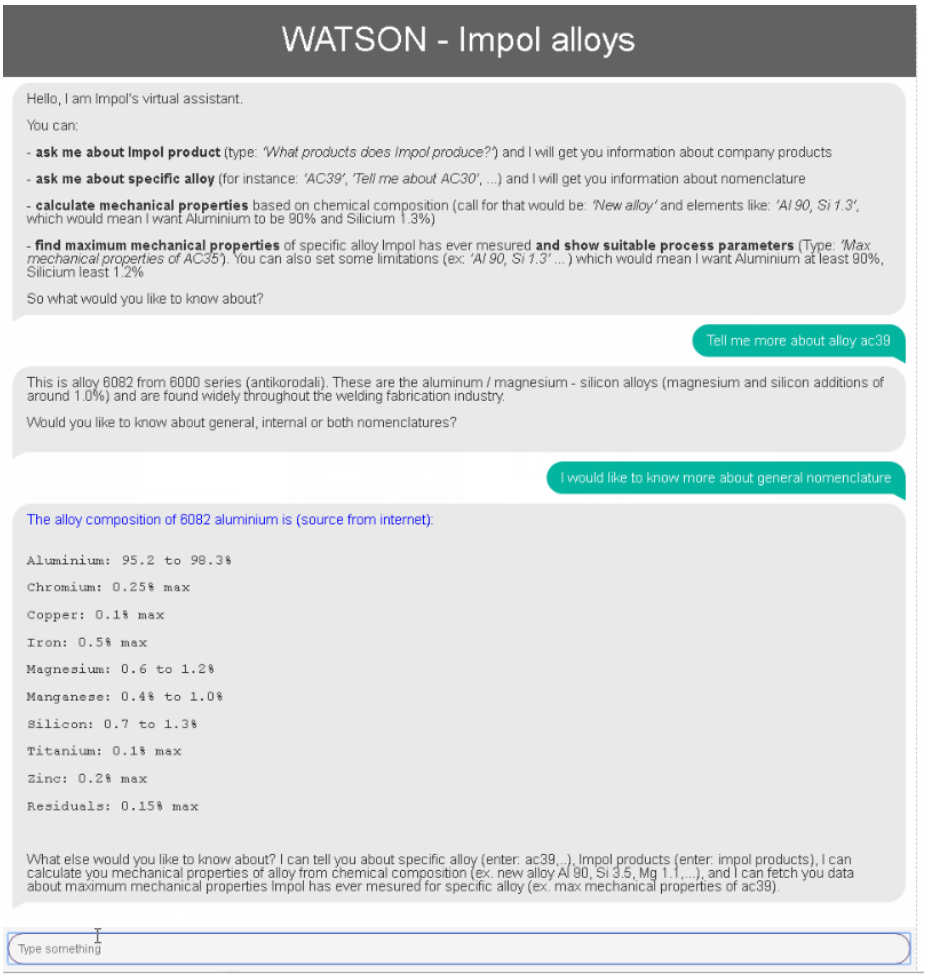
4.2.2. Klient

Za prikaz vmesnika uporabniku smo uporabili tehnologijo React, ki se vse bolj pogosto uporablja za hiter razvoj uporabniškega vmesnika. Ker je šlo za prvo različico aplikacije je bil vmesnik narejen tako, da je omogočal zahtevane funkcionalnosti, glede samega izgleda pa je možnih še veliko izboljšav. Podatkov na klientu v večini primerov nismo obdelovali, v redkih primerih smo uporabili funkcije za spremembo pisave, barve ali pa prikaz slik.

Uporabnik preko tipkovnice vnaša vprašanja in aplikacija mu podaja odgovore. Vse je vodeno kot pogovor z virtualnim asistentom. Primer uporabe in izgled klienta na sliki 12.

4.2.3. Podatkovna baza

Podatkovna baza, ki se uporablja za interne informacije je je IBM-ova relacijska baza DB2. Podatkovna baza je bila že v uporabi, njene tabele zgrajene, zato smo za pridobivanje podatkov zgolj morali napisati ustrezne poizvedbe, ter jih vključiti na ustrezna mesta v logiki strežnika. V nekaterih primerih smo morali opraviti osnovne operacije združevanje različnih tabel glede na določene parametre, tako da smo pridobili vse želene podatke.



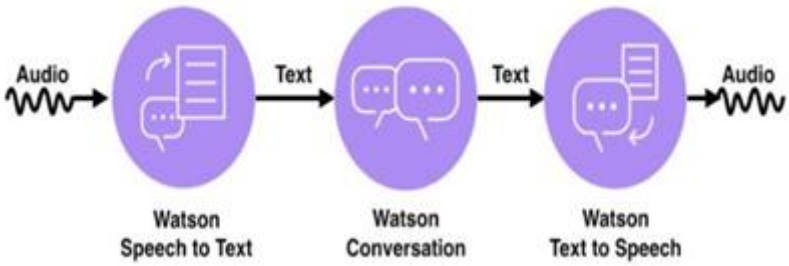
Slika 12: Primer uporabe in izgled klienta

4.3. Mobilna aplikacija

Aplikacija je podprta tudi na mobilnih napravah kjer lahko, namesto klasičnega vnosa teksta preko tipkovnice, uporabimo funkcionalnost *speech to text – text to speech*.

Za uporabniški vmesnik na mobilnih platformah IOS ter Android smo uporabili okolje IBM MobileFirst, ki omogoča razvoj poslovnih mobilnih aplikacij. Prednost uporabe MobileFirst platforme je prenosljivost kode, saj ustvarjamo hibridne mobilne aplikacije, katere nato samo izvažamo na ciljne platforme. Hibridna aplikacija predstavlja koncept pisanja kode v vmesnem programskem jeziku. Kodo aplikacije pišemo le enkrat in jo nato koristimo na več ciljnih platformah (npr. Windows phone, IOS, Android, Blackberry ipd).

Mobilno aplikacijo smo nadgradili s funkcionalnostjo govor v tekst ter tekst v govor. Pri implementaciji smo uporabili servisa IBM Watson Speech-To-Text ter IBM Watson Text-To-Speech. Na ta način koristimo prednosti virtualnega asistenta tudi preko govornih uporabniških vnosov, asistent pa nam zna odgovore zvočno podati tudi nazaj.



Slika 13:Speech to text in text to speech pretvorba

Zaradi konsistentnosti in efektivnosti prikazovanja podatkov je izgled mobilne aplikacije podoben izgledu spletne aplikacije. Mobilna aplikacija uporablja za pridobivanje podatkov in komunikacijo z Watson Conversations storitev strežnik iz poglavja 4.2.1.

Slika 14: Mobilna aplikacija Watson Zlitine

5. ZAKLJUČEK

Osnovna ideja projekta je bila raziskati novo področje kognitivnega računalništva ter se srečati z Watsonom. Cilj je bil prehoditi pot od podatkov do znanja. Tekom projekta smo spoznavali kognitivne pristope, metode in produkte, hkrati pa si zastavili smernice za naprej.

Nove tehnologije na področju računalništva povečujejo našo sposobnost za smiselno urejanje podatkov in informacij, s čimer si zagotavljamo podporo za odločitve in tako omogočimo spremembe v številnih panogah in tudi metalurgija ni izjema.

Razvili smo program Watson Zlitine, ki je uporaben kot pomoč pri sestavi zlitine. Minimizira porabo primarnih elementov in maksimizira uporabo scrapa (odpadnega aluminija), hkrati pa nam omogoča predvideti in napovedati končne mehanske lastnosti pri izbrani recepturi. S tem se že v fazi načrtovanja zlitine optimirajo stroški, optimira se poraba surovine na skladišču ter se predvidi ustreznost končnih lastnosti. Rešitev je osnova za razvoj platforme, ki se bo uporabljala pri razvoju novih zlitin, tehnologij ter obdelavi povpraševanja, kot močno analitično orodje v rokah tehnologov. Kot orodje za obdelavo in obvladovanje procesnih parametrov različnih procesov, iskanje vpliva spremembe posameznega parametra na soodvisnost do drugih parametrov procesne verigi. Takšna platforma pa podjetju omogoča tehnološko prednost pred konkurenco.

6. LITERATURA

[1] Predstavitev IBM Watson Conversation Workshop: 2. Developing Cognitive Applications with IBM Watson Services.pdf (4.7.2017)

[2] Predstavitev IBM Watson Conversation Workshop: 3. Watson Conversation - Building blocks.pdf (4.7.2017)

[3] Predstavitev IBM Watson Conversation Workshop: 4. Intents.pdf (4.7.2017)

[4] Predstavitev IBM Watson Conversation Workshop: 5. Entities.pdf (4.7.2017)

[5] https://www.ibm.com/watson/ (17.5.2018)

[6] https://watson-assistant.ng.bluemix.net (17.5.2018)

[7] https://strongloop.com/strongblog/node-js-is-faster-than-java (17.5.2018)

Univerzitetna založba Univerze v Mariboru

SPLETNE KOMPONENTE IN KNJIŽNICA X-TAG

VIKTOR TANESKI IN GREGOR JOŠT

Povzetek: JavaScript, en bolj popularnih programskih jezikov, se izvaja na več kot 90% spletnih straneh. V zadnjih nekaj letih smo priča pojavu programskega jezika JavaScript tudi na različnih platformah in napravah. Glede na obseg in kompleksnost aplikacij, ki jih lahko razvijemo s programskim jezikom JavaScript, običajno stremimo k čim večji ponovni uporabi programske kode (angl. code reuse). V ta namen imamo na voljo programska ogrodja (angl. frameworks), ki prav tako zmanjšujejo kompleksnost programske kode in posledično razvoja. Programska ogrodja so preizkušena in uporabljena v različnih scenarijih in situacijah ter običajno razvita s strani večjih podjetij, kot sta Google (Angular) in Facebook (React). Pogosta kritika programskih ogrodij JavaScript je, da so številna, se pogosto posodabljajo, razvijalci pa posledično težko sledijo spremembam in novostim. Po drugi strani spletne komponente (angl. Web components) predstavljajo drugačen pristop razvoja elementov po meri (angl. Custom elements) z ovitimi (angl. encapsulate) funkcionalnostmi, ki jih lahko večkrat uporabimo kjer koli v kodi. Za razliko od obstoječih ogrodij so spletne komponente skupek W3C specifikacij, ki se hitro pomikajo proti standardizaciji. Ker so te specifikacije v različnih stanjih podprtosti v brskalnikih, je trenutno še priporočljivo uporabiti eno izmed obstoječih knjižnic za razvoj spletnih komponent. V tem članku bomo predstavili knjižnico X-Tag, razvito s strani Mozille in trenutno pod okriljem Microsofta. X-Tag je zelo preprosta JavaScript knjižnica, ki za delovanje zahteva le eno izmed W3C specifikacij spletnih komponent. Podprta je s strani brskalnikov, kot so Internet Explorer 9+, Edge, Firefox, Chrome, Safari in Opera.

Ključne besede: • spletne komponente • X-Tag • JavaScript

NASLOV AVTORJEV: Viktor Taneski, asistent, Univerza v Mariboru, Fakulteta za elektrotehniko, računalništvo in informatiko, Smetanova ulica 17, 2000 Maribor, Slovenija, e-pošta: viktor.taneski@um.si. Gregor Jošt, asistent, Univerza v Mariboru, Fakulteta za elektrotehniko, računalništvo in informatiko, Smetanova ulica 17, 2000 Maribor, Slovenija, e-pošta: gregor.jost@um.si.

DOI https://doi.org/10.18690/978-961-286-162-9.22 ISBN 978-961-286-163-6

Dostopno na: http://press.um.si.

1. UVOD

Včasih je bil razvoj spletnih aplikacij preprost; dovolj je bila že uporaba privzetih elementov HTML (Hypertext Markup Language) z nekaj stilov CSS (Cascading Style Sheets) in osnovnih funkcionalnosti programskega jezika JavaScript. Ker so spletne aplikacije postajale vse bolj kompleksne, je bilo potrebno poskrbeti tudi za primerno odziven in privlačen uporabniški vmesnik (angl. user interface, v nadaljevanju UV). V ta namen je nastalo precej ogrodij, ki olajšajo razvoj naprednih, dinamičnih spletnih aplikacij. Večina teh ogrodij (npr. React, Angular) omogoča razvoj komponent, ki jih lahko ponovno uporabimo v različnih delih aplikacij. Takšnih ogrodij je precej, njihove posodobitve so običajno zelo pogoste in včasih spremembe vplivajo na obnašanje obstoječih aplikacij [1], [2]. Potrebno je torej ostajati v koraku z novostmi, kar je lahko časovno in finančno neugodno. Uporaba takšnih knjižnic oz. ogrodij je pri razvoju odjemalca spletnih aplikacij nujna, saj brskalniki privzeto ne ponujajo razvoja lastnih komponent [3]. Omenjeno pomanjkljivost rešujejo spletne komponente.

Spletne komponente so standard W3C in omogočajo razvoj lastnih značk HTML, ki jih lahko ponovno uporabljamo znotraj spletnih aplikacij. Sodobni brskalniki trenutno že delno podpirajo nekatere standarde, vendar se obseg podpore razlikuje. V ta namen je zato priporočljivo uporabiti enega izmed obstoječih ogrodij za razvoj spletnih komponent. Ko bodo brskalniki v celoti podprli vse standarde spletnih komponent, bo mogoče razvijati takšne komponente brez uporabe kakršnega koli ogrodja [4], [5].

V tem članku bomo najprej predstavili spletne komponente in kaj le-te omogočajo. Opisali bomo štiri specifikacije, na katerih temeljijo spletne komponente, pregledali njihovo podprtost v brskalnikih in predstavili obstoječa ogrodja, ki omogočajo razvoj spletnih komponent. Na koncu bomo predstavili eno izmed teh knjižnic, X-Tag, ki omogoča razvoj elementov po meri.

2. KAJ SO SPLETNE KOMPONENTE?

Spletne komponente so bile prvič predstavljene leta 2011 s strani konzorcija W3C (World Wide Web Consortium). Glavni namen spletnih komponent je implementacija ponovno uporabnih gradnikov (angl. widget) oz. komponent, ki jih lahko uporabimo znotraj spletnih strani in spletnih aplikacij [6]. Spletne komponente temeljijo na štirih glavnih specifikacijah, in sicer [6]–[8]:

- Specifikacija elementov po meri (angl. custom elements) predstavlja način, kako razširimo obstoječe ali ustvarimo nove elemente HTML. S tem pridobimo na bolj modularni programski kodi, ki se lahko ponovno uporabi v različnih kontekstih.
- Prikriti DOM (angl. shadow DOM) standard omogoča enkapsulacijo spletne komponente, tj., da so konstrukcija, stil CSS in obnašanje komponente ločeni od ostale kode na spletni strani.
- Predloge (angl. templates) omogočajo, da se komponenta opredeli v obliki predloge HTML, ki lahko vključuje kakršne koli elemente HTML.
- Uvoz HTML (angl. HTML imports) standard opredeli, kako uvoziti dokumente HTML znotraj drugih dokumentov HTML.

Vsako izmed naštetih specifikacij bomo v nadaljevanju podrobno predstavili.

2.1. Elementi po meri

Elementi po meri (angl. custom elements) so ključni oz. najpomembnejši del standarda spletnih komponent, saj omogočajo razvoj novih elementov (značk) HTML, obenem pa tudi omogočajo razširitev že obstoječih elementov HTML. Poznamo torej dva večja tipa elementov po meri, in sicer »avtonomni elementi po meri« (angl. autonomous custom element) in »razširitev obstoječih elementov po meri« (customized built-in element) [9]. Elementi po meri naj bi delovali povsod, kjer deluje že obstoječ HTML, kar vključuje tudi uporabo znotraj vseh ogrodij (angl. frameworks) JavaScript. Brez elementov po meri uporaba spletnih komponent ne bi bila mogoča, saj nudijo [10]:

- Definicijo novih elementov.

- Razvoj elementov, ki dedujejo od drugih elementov.
- Združevanje funkcionalnosti znotraj ene značk HTML.
- Razširjanje programskih vmesnikov programskih vmesnikov oz. API-jev (angl. Application Programing Interface) obstoječih elementov.

Nov element po meri lahko implementiramo z uporabo programskega jezika JavaScript. Globalni objekt *customElements* se uporablja za definiranje novega elementa preko metode *define()*. Definicija metode je naslednja: *customElements.define(ime, konstruktor, možnosti),* kjer posamezni parametri predstavljajo:

- ime - ime značke HTML,
- konstruktor - razred JavaScript, ki deduje od razreda *HTMLElement* in
- možnosti - (opcijsko) objekt z metodo *extends*, namenjen razširitvi obstoječih elementov po meri.

Slika 1 prikazuje preprost primer implementacije elementa po meri, ki se nato uporabi v dokumentu HTML.

```
10  <body>
11
12      <ots-copyright></ots-copyright>
13      <script>
14          customElements.define('ots-copyright', class extends HTMLElement {
15              constructor() {
16                  super();
17                  this.innerHTML = `
18                      <div id="copyright">
19                          Copyright &copy; 2018
20                          <a href="http://www.ots.si">
21                          &copy; UM FERI, Inštitut za informatiko. Vse pravice pridržane.
22                          </a>
23                      </div>
24                      `;
25              }
26          });
27      </script>
28
29  </body>
```

Slika 1: Implementacija elementa po meri

Elementi po meri so trenutno podpri v brskalnikih Chrome 54 in Safari 10.1, medtem ko so v Microsoftov Edge in Mozilli še v fazi vključevanja [11].

2.2. Prikriti DOM

Ko brskalnik naloži dokument HTML, takoj pretvori to strukturo (statično besedilo) v podatkovni model objektov in vozlišč. Brskalnik torej ohrani hierarhijo dokumenta HTML tako, da ustvari drevo vseh teh vozlišč – DOM (Document Object Model). Za razliko od statičnega dokumenta HTML, DOM struktura vsebuje lastnosti in metode. DOM torej predstavlja API, ki definira logično strukturo dokumentov in zagotavlja njihov dostop in manipulacijo [12].

Prikriti DOM (angl. shadow DOM) omogoča pripenjanje skritih DOM dreves na elemente običajne DOM strukture. Naslavlja pomemben del spletnih komponent, in sicer enkapsulacijo, tj. zmožnost skrivanja strukture, stilov CSS in obnašanja od preostale kode na spletni strani. V tem kontekstu prikriti DOM omogoča pripenjanje skritega, ločenega DOM na določen element HTML [13]. Uporaba prikritega DOM-a ni pogoj za razvoj spletnih komponent, vendar doprinese precej funkcionalnosti

(ustvarjanje ločenega dosega CSS, enkapsulacija DOM, kompozicija). Uporaba prikritega DOM in elementov po meri skupaj izjemno poveča ponovno uporabo [12].

Prikriti DOM za določen element ustvarimo z uporabo metode *attachShadow*, kot prikazuje slika spodaj (Slika 2).

```
12  <ots-content></ots-content>
13  <script>
14      customElements.define('ots-content', class extends HTMLElement {
15          constructor() {
16              super();
17
18              const shadowRoot = this.attachShadow({ mode: 'open' });
19              shadowRoot.innerHTML = `
20                  <style>
21                      #content {
22                      padding: 2em;
23                      }
24
25                      p {
26                      font-style: italic;
27                      }
28                  </style>
29                  <div id="content">
30                      <h1>Spoštovani!</h1>
31                      <p>že hiter pogled na program letošnje konference...</p>
32                  </div>
33                  `;
34          }
35      });
36  </script>
```

Slika 2: Uporaba prikritega DOM

Kot je razvidno iz slike zgoraj, prikriti DOM lahko vključuje tudi stil CSS, na elemente znotraj prikritega DOM pa se ne moremo sklicevati kasneje v JavaScript programski kodi. Za zgornji primer bi naslavljanje *document.getElementById('content')* vrnilo *null*, čeprav v DOM obstaja element z ID vrednostjo *content*.

2.3. Predloge

Večina ogrodij, ki omogočajo razvoj spletnih aplikacij, običajno uporabljajo predloge za prikaz podatkov, npr. Smarty (PHP) [14] in Razor (ASP.NET MVC). V splošnem lahko definiramo predlogo kot dokument ali datoteko, ki ima vnaprej definirano obliko. Takšne strukture posledično ni potrebno ponovno definirati vsakič, ko jo želimo uporabiti. Spletne komponente vključujejo ta koncept preko elementa HTML *<template>* [15]. Element torej definira standardni pristop, ki temelji na vmesniku DOM in je namenjen ustvarjanju predlog na strani odjemalca. Uporablja se za ustvarjanje delov dokumentov HTML, ki predstavljajo želeno predlogo. Te predloge lahko nato kloniramo in vstavimo v dokument s pomočjo programskega jezika JavaScript.

Z ovijanjem vsebine v element *<template>* pridobimo naslednje lastnosti [14]:

- Vsebina elementa ni dostopna, dokler se ne aktivira z že prej omenjenim načinom (kloniranje in vključitev preko programskega jezika JavaScript). Do takrat je vsebina elementa dejansko skrita in se ne prikazuje.
- Vsebina znotraj predloge ne bo delovala (npr., skripte se ne bodo zagnale in slike se ne bodo naložile) do eksplicitne uporabe predloge.

- Vsebina predloge ni del dokumenta HTML, zato manipulacija s programskim jezikom JavaScript ni možna (npr. uporaba *document.getElementById()* ali *querySelector()* ne bo vrnila želenega elementa znotraj predloge.
- Predloge lahko postavimo kjer koli znotraj *<head>*, *<body>* ali *<frameset>*. Vključimo jih lahko torej tudi v dele dokumenta, kamor jih razčlenjevalnik HTML (angl. HTML parser) ne dopušča.

Predlogo uporabimo tako, da jo najprej kloniramo in nato vstavimo v DOM. Implementacija v programskem jeziku JavaScript je prikazana na sliki spodaj (Slika 3).

```
10  <body>
11      <div class="container">
12          <button class="btn btn-outline-primary" onclick="showContent()">Prikaži vsebino</button>
13          <div id="content"></div>
14      </div>
15
16      <template id="konferencaOTS">
17          <h1>23. KONFERENCA OTS</h1>
18          <h3>19. in 20. JUNIJ</h3>
19          <p>
20              <video width="100%" controls="" autoplay="">
21                  <source src="https://medijske.um.si/nikazorjan/NikaZorjan-1.mp4" type="video/mp4">
22              </video>
23          </p>
24      </template>
25
26      <script>
27          const showContent = () => {
28              const template = document.getElementById('konferencaOTS');
29              const clone = template.content.cloneNode(true);
30              document.getElementById('content').appendChild(clone);
31          }
32      </script>
33  </body>
```

Slika 3. Primer uporabe predlog

2.4. Uvozi HTML

Vire lahko vključimo v spletne strani na različne načine, glede na njihov tip: v primeru datotek JavaScript je na voljo *<script src="">*, za CSS uporabljamo *<link rel="stylesheet">*, za slike *<img>*, za avdio *<audio>* itd. Kot je razvidno, ima večina vsebine na spletu preprost in deklarativni način uvoza zunanjih virov. Po drugi strani pa je uvoz dokumenta HTML (ki predstavlja najbolj osnovno vsebino spleta) še nedolgo nazaj predstavljal izziv. Uvoz dokumenta HTML se je tako reševal (in se še vedno) na enega izmed naslednjih načinov [16]:

- *<iframe>* – uporaben, a zahteven način uvoza dokumenta HTML. Vsebina elementa *<iframe>* se nahaja v ločenem kontekstu od naše aplikacije, kar lahko povzroča dodatne izzive (npr., nastavljanje velikosti ali stila okvira).
- *AJAX – Asynchronous JavaScript And XML* tehnika zahteva programski jezik JavaScript za uvoz dokumentov HTML, kar ni optimalno.
- Na voljo imamo tudi različne nekonvencionalne pristope, npr., vgrajevanje vsebine dokumenta HTML v niz znakov ali skrivanje v komentarje (npr., *<script type="text/html">*).

Prva različica specifikacij uvozov HTML (angl. HTML Imports) je bila predstavljena leta 2013, brskalnik Chrome pa je prvič podpiral to funkcionalnost leta 2014 [17].
Uvozi HTML predstavljajo način vključevanja spletnih komponent v spletno stran. Na takšen način lahko združimo skupaj (angl. bundle) HTML, CSS in JavaScript kodo, ki jo lahko uporabimo kjer koli znotraj lastne ali druge spletne aplikacije. Slednje sicer lahko dosežemo tudi z uporabo AJAX, a uvoz HTML omogoča bolj jasen in pregleden način uvoza neodvisnih komponent v spletno stran [18].

Uvozi HTML omogočajo razdelitev spletne aplikacije na več komponent, vsaka komponenta pa lahko vključuje oznake (angl. markup), oblikovanje (angl. style) in skripte. Alternativa je, da nimamo nobene komponente, kar lahko povzroči, da imamo celotno spletno stran v eni datoteki HTML. Uporabimo lahko tudi module JavaScript, ki naslavljajo sicer le skriptni vidik, kar pa ne odpravi vseh že omenjenih izzivov [17].

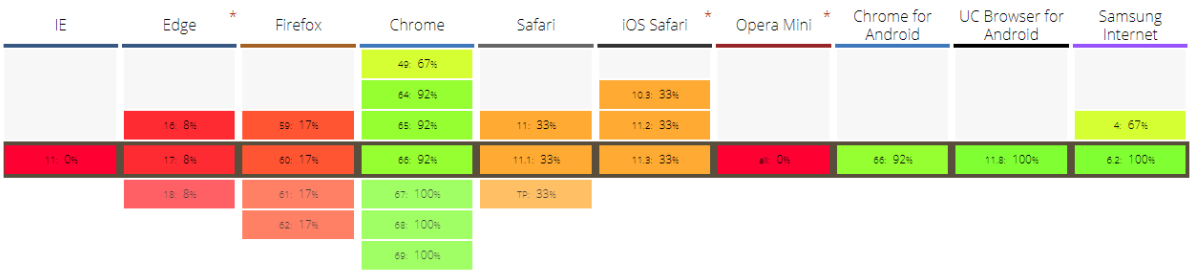
Pri tem standardu gre torej za povezovanje datotek HTML, CSS in JavaScript v zgolj eno datoteko, kar olajša razvoj predlog CSS, knjižnic JavaScript ali delitev aplikacijo v logične dele [16]. Uvoz HTML ne predstavlja zgolj povezave HTML na pripadajočo komponento, ampak je potrebno implementirati pripadajočo logiko v programskem jeziku JavaScript. Slika 4 predstavlja primer implementacije.

```
<!-- video.html -->
<p id="ots_video">
    <video width="100%">
        <source src="https://medijske.um.si/nikazorjan/NikaZorjan-1.mp4" type="video/mp4">
    </video>
</p>

<!-- index.html -->
<!DOCTYPE html>
<html>

<head>
    <meta charset="utf-8" />
    <link rel="import" href="video.html" />
</head>

<body>
    <h1>23. KONFERENCA OTS</h1>
    <h3>19. in 20. JUNIJ</h3>

    <script>
        const link = document.querySelector('link[rel=import]');
        const content = link.import.getElementById('ots_video');
        document.body.appendChild(content.cloneNode(true));
    </script>
</body>

</html>
```

Slika 4. Primer uporabe HTML uvoza

3. PRIVZETA PODPORA SPLETNIM KOMPONENTAM

Spletne komponente je prvič predstavil Alex Russell na konferenci Fronteers leta 2011 [19], nato pa je Google leta 2013 objavil programsko ogrodje za razvoj spletnih komponent, imenovan Polymer [20]. Namen ogrodja Polymer je pohitritev razvoja spletnih komponent, saj vključuje tudi dodatne funcionalnosti. Standardi spletnih komponent namreč še vedno niso podprti v vseh brskalnikih. Brskalnik Chrome trenutno podpira največ standardov spletnih komponent (Slika 5), medtem ko ostali brskalniki precej zaostajajo.



Slika 5. Podprtost spletnih komponent za posamezne brskalnike [21]

3.1. Trenutno stanje podprtosti v brskalnikih

Ko govorimo o spletnih komponentah in združljivosti z obstoječimi brskalniki je pomembno izpostaviti, da je brskalnik Chrome en izmed redkih, ki nudi domorodno (angl. native) podporo API-jev verzije 0 (v nadaljevanju v0). Ostali brskalniki so v0 podpirali s pomočjo tako imenovanih »polyfill knjižnic«, ki simulirajo manjkajoče funkcionalnosti. Situacija se je spremenila z verzijo 1 (v1), ki podpira ECMAScript sintakso razreda [22].

API-ji v1 so sprejeti s strani več sodobnih brskalnikov in rešujejo sintaktične težave različice v0. Posledično lahko razvijamo spletne komponente z manj zadržkov oz. strahu, da določene funkcionalnosti ne bi bile podprte v brskalnikih. Kot že delno prikazuje Slika 5, najnovejše različice spodaj naštetih brskalnikov podpirajo spletne komponente ali preko polyfill knjižnic ali pa že preko domorodne podpore [22]:

- Namizni: Chrome, Safari, Firefox, Opera, Edge, IE (8+ preko polyfill knjižnic)
- Mobilni: iOS, Android, FirefoxOS, KindleFire, Windows Phone, Opera Mobile, Blackberry OS, webOS

3.2. Spletne komponente in obstoječa ogrodja

Spletne komponente se lahko uporabljajo tudi znotraj ogrodij oz. knjižnic za razvoj JavaScript aplikacij, kot sta že omenjena React in Angular. Obenem pa nekatera takšna ogrodja tudi nudijo podporo za razvoj spletnih komponent. Na primer, Angular z različico 6 (izdano 3. 4. 2018) v celoti omogoča preprost način razvoja domorodnih elementov po meri z uporabo »Angular Elements« (Slika 6) [23].

```
// @ts-ignore
import { Input, Component, ViewEncapsulation, EventEmitter, Output }

@Component({
    selector: 'custom-button',
    template: `<button (click)="handleClick()">{{label}}</button>`,
    styles: [`
    button {
      border: solid 3px;
      padding: 8px 10px;
      background: #bada55;
      font-size: 20px;
    }
  `],
    encapsulation: ViewEncapsulation.Native
})
// @ts-ignore
export class ButtonComponent {
    // @ts-ignore
    @Input() label = 'default label';
    // @ts-ignore
    @Output() action = new EventEmitter<number>();
    private clicksCt = 0;

    handleClick() {
        this.clicksCt++;
        this.action.emit(this.clicksCt);
    }
}
```

Slika 6: Razvoj elementov po meri v ogrodju Angular 6 [23].

Po drugi strani pa knjižnica React omogoča vključitev že implementiranih spletnih komponent v obstoječo React kodo in obratno. React in spletne komponente ne rešujejo istih problemov, ampak se dopolnjujejo; medtem ko spletne komponente nudijo enkapsulacijo komponent, React predstavlja knjižnico, ki skrbi da je DOM skladen s podatki. Slika 7 prikazuje oba načina uporabe spletnih komponent in knjižnice React.

```
// Primer uporabe spletnih komponent v ogrodju React
class HelloMessage extends React.Component {
    render() {
        return <div>Hello <x-search>{this.props.name}</x-search>!</div>;
    }
}

// Primer uporabe ogrodja React v spletnih komponentah
class XSearch extends HTMLElement {
    connectedCallback() {
        const mountPoint = document.createElement('span');
        this.attachShadow({ mode: 'open' }).appendChild(mountPoint);

        const name = this.getAttribute('name');
        const url = 'https://www.google.com/search?q=' + encodeURIComponent(name);
        ReactDOM.render(<a href={url}>{name}</a>, mountPoint);
    }
}
customElements.define('x-search', XSearch);
```

Slika 7: Primera uporabe spletnih komponent in knjižnice React [24]

3.3. Knjižnice za razvoj spletnih komponent

Implementacija spletnih komponent je možna zaradi pripadajočih specifikacij, predvsem elementov po meri in prikritega DOM-a. Kot že rečeno, so najnovejše različice (v1) omenjenih specifikacij že vključene v nekatere brskalnike (npr. Chrome, Mozilla in Safari). Osnovna ideja trenutnih v1 specifikacij je torej definiranje novih elementov HTML in pripadajočih značk (angl. tag) v programskem jeziku JavaScript [22]. Sodobni brskalniki še vedno ne podpirajo vseh specifikacij v celoti, nekatera obstoječa ogrodja JavaScript pa nudijo podporo spletnim komponentam samo v ožjem obseg. V ta namen je na voljo več knjižnic, posvečenih razvoju spletnih komponent. Med slednje sodijo tudi [22], [25]:

- Polymer – Ena najbolj popularnih, obsežnih in zmogljivih knjižnic za razvoj spletnih komponent je Googlov Polymer, ki podpira vse specifikacije za razvoj spletnih komponent.
- X-Tag: – Razvita s strani Mozille in trenutno pod okriljem Microsofta. Je preprosta knjižnica za razširitev funkcionalnosti specifikacij elementov po meri.
- Slim.js – Knjižnica za razvoj domorodnih spletnih komponent, ki nudi naprednejše funkcionalnosti, kot sta »vezanje podatkov« (angl. data binding) in podpora ECMAScript 6 dedovanju med razredi.

Ker večina takšnih knjižnic temelji na že omenjenih standardih, ni potrebno skrbeti, katero knjižnico uporabiti za razvoj spletnih komponent. Vse v večini ponujajo abstrakcijo, ki omogoča hitrejši razvoj spletnih komponent z manj programske kode. Odločitev o uporabi knjižnic je običajno odvisna zgolj od tega, kateri dodatki (angl. syntactic sugar) nam najbolj ustrezajo.

V nadaljevanju bomo opisali knjižnico X-Tag, ki velja za eno bolj preprostih knjižnic za razvoj spletnih komponent. Za delovanje zahteva le eno izmed W3C specifikacij, in sicer specifikacijo elementov po meri. X-Tag je prav tako podprt s strani večine brskalnikov (Firefox, Chrome, Safari, Opera, Internet Explorer 9+ in Edge).

4. KNJIŽNICA X-TAG

4.1. Zgodovina

Knjižnica X-Tag je nastala pod okriljem Mozille z namenom podpore razvoju spletnih komponent, še preden so brskalniki podpirali te standarde [26]. Knjižnica se osredotoča na najpomembnejšega od štirih standardov spletnih komponent, in sicer standard elementov po meri. V kolikor brskalnik ne podpira tega standarda, knjižnica X-Tag zagotovi delovanje z uporabo iste polyfill knjižnice, kot Googlovo ogrodje Polymer. Trenutno je razvoj X-Tag knjižnice prevzelo podjetje Microsoft [27]. Stabilna različica knjižnice je v1, na voljo pa je že različica v2.0, ki je še v alfa fazi življenjskega cikla. Različica v2.0 omogoča uporabo ECMAScript 6 sintakse razredov [28].

4.2. Programski vmesnik (API) knjižnice X-Tag

Razvoj z uporabo knjižnice X-Tag poteka v celoti v programskem jeziku JavaScript. Knjižnica nudi skupek metod, povratnih funkcij (angl. callbacks) in lastnosti za definiranje oz. implementacijo elementov po meri.

Najpomembnejša metoda knjižnice X-Tag je *register*, ki ustvari element po meri in vključuje življenjski cikel elementa, dostop do atributov in poslušalce dogodkov. Osnovna struktura elementa po meri z uporabo knjižnice X-Tag je predstavljena na sliki spodaj (Slika 8).

```
<script>
    xtag.register('OTS-predvajalnik', {
        content: '',
        lifecycle: {
            created: function () { },
            inserted: function () { },
            removed: function () { },
            attributeChanged: function (attrName, oldValue, newValue) { }
        },
        accessors: {},
        methods: {},
        events: {}
    });
</script>
```

Slika 8: Osnovna struktura elementa po meri

Kot je razvidno iz zgornje slike, metoda *register* pričakuje dva parametra. Prvi parameter je niz znakov (angl string) in predstavlja ime elementa po meri. Drugi parameter pa predstavlja objekt, z naslednjimi lastnostmi:

- *content*,
- *lifecycle*,
- *accessors*,
- *methods* in
- *events*.

Lastnost *content* predstavlja glavni način vključevanja vsebine v komponento. Pričakuje preprost niz znakov, ki predstavlja strukturo HTML ali pa funkcijo, ki vsebinsko predstavlja strukturo HTML, zapisano v obliki komentarja. Oba primera uporabe sta predstavljena na sliki spodaj (Slika 9).

```
<script>
    xtag.register('ots-noga', {
        content: `COPYRIGHT 2018 UM FERI, INŠTITUT ZA INFORMATIKO. VSE PRAVICE PRIDRŽANE`
    });

    xtag.register('ots-glava', {
        content: function () {/*
        <header>
          <h1>23. KONFERENCA OTS</h1>
          <h2>19. IN 20. JUNIJ</h2>
        </header>
          */}
    });

</script>
```

Slika 9: Primer vključevanja vsebine v komponento (kot preprost niz in kot komentar)

Lastnost *Lifecycle* predstavlja objekt, kjer so definirane štiri metode. Vsaka izmed teh metod predstavlja povratni klici življenjskega cikla (angl. lifecycle callbacks), ti povratni klici pa se sprožijo v ustreznem trenutku življenjskega cikla komponente: *created* se sproži vsakič, ko je element HTML na novo ustvarjen, *inserted* in *removed* se sprožita, ko je ta element dodan oz. odstranjen iz DOM-a, *attributeChanged* pa se sproži vedno, ko se spremeni vrednost določenega atributa.

Knjižnica X-Tag prav tako omogoča preprost način za upravljanje z atributi, in sicer preko lastnosti *accessors*. Gre za vmesnik, ki omogoča dostop do lastno definiranih atributov elementa preko *get* in *set* metod. Slednje dosežemo tako, da objektu *accessors* podamo dodatne lastnosti, ki predstavljajo atribute elementa. Primer uporabe je prikazan na sliki spodaj (Slika 10).

```
    <ots-glava naslov="23. KONFERENCA OTS" podnaslov="19. IN 20. JUNIJ"></ots-glava>
    <script>
        xtag.register('ots-glava', {
            content: function () {/*
                <header>
                    <h1></h1>
                    <h2></h2>
                </header>
            */},
            accessors: {
                naslov: {
                    attribute: {},
                    set: function (value) {
                        const element = xtag.query(this, 'header h1')[0];
                        element.innerHTML = value;
                    }
                },
                podnaslov: {
                    attribute: {},
                    set: function (value) {
                        const element = xtag.query(this, 'header h2')[0];
                        element.innerHTML = value;
                    }
                }
            }
        });
    </script>
```

Slika 10: Atributi elementov po meri

Poslovno logiko elementov po meri lahko vključimo v sklopu lastnosti *methods*, ki predstavlja objekt, kjer je vsaka lastnost želena metoda. Po drugi strani pa lahko običajne dogodke (npr. *click*, *tap*) vključimo v lastnost *events*. Spodnja slika (Slika 11) predstavlja uporabo obeh opisanih lastnosti.

```
<video width="400" controls>
    <source src="https://medijske.um.si/nikazorjan/NikaZorjan-1.mp4" type="vid
</video>
<ots-predvajaj-video></ots-predvajaj-video>

<script>
    xtag.register('ots-predvajaj-video', {
        content: `<button>Predvajaj/ustavi</button>`,
        lifecycle: {
            created: function () {
                this.xtag.data.isPlaying = false;
                this.xtag.data.video = xtag.query(document.body, 'video')[0];
            }
        },
        methods: {
            predvajaj: function () {
                this.xtag.data.video.play();
                this.xtag.data.isPlaying = true;
            },
            ustavi: function () {
                this.xtag.data.video.pause();
                this.xtag.data.isPlaying = false;
            }
        },
        events: {
            click: function () {
                if (this.xtag.data.isPlaying) {
                    this.ustavi();
                } else {
                    this.predvajaj();
                }
            }
        }
    });
</script>
```

Slika 11: Primer uporabe metod in dogodkov znotraj elementa po meri

5. ZAKLJUČEK

V članku smo predstavili spletne komponente, pripadajoče standarde in knjižnice, ki omogočajo razvoj spletnih komponent tudi za brskalnike, ki jih še ne podpirajo (v celoti).

Spletne komponente so sestavljene iz štirih standardov, in sicer elementi po meri, prikriti DOM, predloge in uvoz HTML. Ker brskalniki trenutno še ne podpirajo vseh standardov (z izjemo brskalnika Chrome), je v večini primerov potrebno uporabiti eno izmed obstoječih knjižnic.

V tem članku smo se osredotočili na knjižnico X-Tag, saj je zelo preprosta za uporabo, za delovanje pa potrebuje podporo zgolj enega od štirih standardov, in sicer standard elementi po meri. Slednji je najpomembnejši del standarda, saj brez njega implementacija spletnih komponent ni možna. Kot smo videli iz primerov, lahko z uporabo knjižnice X-Tag zelo preprosto in hitro implementiramo element po meri.

Spletne komponente imajo svetlo prihodnost, saj omogočajo razvoj prenosljivih komponent, ki vključujejo tudi logiko in stile. Obenem pa jih lahko uporabljamo skupaj s popularnimi ogrodji JavaScript, kot sta React in Angular, saj spletne komponente ne predstavljajo konkurenco oz. alternativo obstoječim ogrodjem, temveč se dopolnjujejo.

6. LITERATURA

[1] "JavaScript Developers: Do we really need frameworks?" [Online]. Available: https://blog.revillweb.com/javascript-developers-do-we-really-need-frameworks-5cd659ec7365. [Accessed: 27-May-2018].

[2] Ian Allen, "The Brutal Lifecycle of JavaScript Frameworks," 2018. [Online]. Available: https://stackoverflow.blog/2018/01/11/brutal-lifecycle-javascript-frameworks/. [Accessed: 27-May-2018].

[3] "Polymer vs. React." [Online]. Available: https://www.upwork.com/hiring/development/polymer-vs-react/. [Accessed: 27-May-2018].

[4] "Wrapping Web Components With React - Blog | SitePen." [Online]. Available: https://www.sitepen.com/blog/2017/08/08/wrapping-web-components-with-react/. [Accessed: 27-May-2018].

[5] J. O'Neill, "Web Components VS Frameworks," *Medium*. [Online]. Available: https://medium.com/@oneeezy/frameworks-vs-web-components-9a7bd89da9d4. [Accessed: 27-May-2018].

[6] INKONIQ, "Uncomplicate the WEB Using Web Components," 2017. [Online]. Available: https://medium.com/inkoniq-blog/uncomplicate-the-web-using-web-components-4d5f7edaac05. [Accessed: 27-May-2018].

[7] "Introduction - webcomponents.org." [Online]. Available: https://www.webcomponents.org/introduction. [Accessed: 27-May-2018].

[8] "Web Fundamentals | Web | Google Developers." [Online]. Available: https://developers.google.com/web/fundamentals/. [Accessed: 27-May-2018].

[9] W. Community, "HTML Standard," 2015. [Online]. Available: https://html.spec.whatwg.org/#the-canvas-element. [Accessed: 27-May-2018].

[10] E. Bidelman, "Custom Elements: defining new elements in HTML - HTML5 Rocks," 2013. [Online]. Available: http://www.html5rocks.com/en/tutorials/webcomponents/customelements/. [Accessed: 20-May-2018].

[11] E. Bidelman, "Custom elements v1: reusable web components | Web Fundamentals - Google Developers," 2017. [Online]. Available: https://developers.google.com/web/fundamentals/primers/customelements/. [Accessed: 20-May-2018].

[12] "Shadow DOM v1: Self-Contained Web Components | Web Fundamentals | Google Developers." [Online]. Available: https://developers.google.com/web/fundamentals/web-components/shadowdom. [Accessed: 27-May-2018].

[13] J. Swisher and C. Mills, "Using shadow DOM - Web Components | MDN," 2018. [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/Web_Components/Using_shadow_DOM. [Accessed: 27-May-2018].

[14] E. Bidelman, "HTML's New Template Tag: standardizing client-side templating - HTML5 Rocks," 2013. [Online]. Available: http://www.html5rocks.com/en/tutorials/webcomponents/template/. [Accessed: 27-May-2018].

[15] "HTML Standard." [Online]. Available: https://html.spec.whatwg.org/multipage/scripting.html#the-template-element. [Accessed: 27-May-2018].

[16] E. Bidelman, "HTML Imports: #include for the web - HTML5 Rocks," 2013. [Online]. Available: http://www.html5rocks.com/en/tutorials/webcomponents/imports/. [Accessed: 27-May-2018].

[17] "HTML imports are the best web component - Ashley's blog." [Online]. Available: https://www.construct.net/si/blogs/ashleys-blog-2/html-imports-are-the-best-web-component-941. [Accessed: 27-May-2018].

[18] "An Introduction to WebSockets - Treehouse Blog." [Online]. Available:

http://blog.teamtreehouse.com/an-introduction-to-websockets. [Accessed: 27-May-2018].

[19] "The state of Web Components – Mozilla Hacks – the Web developer blog." [Online]. Available: https://hacks.mozilla.org/2015/06/the-state-of-web-components/. [Accessed: 27-May-2018].

[20] "Welcome - Polymer Project." [Online]. Available: https://www.polymer-project.org/. [Accessed: 27-May-2018].

[21] Fyrd, "Can I use... Support tables for HTML5, CSS3, etc," *Fyrd*, 2013. [Online]. Available: http://http//caniuse.com/#search=web gl. [Accessed: 27-May-2018].

[22] "An Introduction to Web Components." [Online]. Available: https://www.upwork.com/hiring/development/web-components/. [Accessed: 27-May-2018].

[23] "Building Custom Elements/Web Components with Angular 6." [Online]. Available: https://medium.com/@tomsu/building-web-components-with-angular-elements-746cd2a38d5b. [Accessed: 27-May-2018].

[24] "Web Components - React." [Online]. Available: https://reactjs.org/docs/web-components.html. [Accessed: 27-May-2018].

[25] "Libraries - webcomponents.org." [Online]. Available: https://www.webcomponents.org/libraries. [Accessed: 27-May-2018].

[26] "Web Components With X-Tag." [Online]. Available: http://www.i-programmer.info/programming/javascript/6462-web-components-with-mozillas-x-tag.html. [Accessed: 27-May-2018].

[27] Pankaj Parashar, "Building Custom Web Components with X-Tag," 2015. [Online]. Available: https://www.sitepoint.com/building-custom-web-components-with-x-tag/. [Accessed: 27-May-2018].

[28] "X-Tag - Web Components Library." [Online]. Available: https://x-tag.github.io/. [Accessed: 17-May-2018].

Univerzitetna založba Univerze v Mariboru

API KOT STIČIŠČE ANGULAR OBLIČJA IN NA PAMETNIH POGODBAH TEMELJEČEGA ZALEDJA

PATRIK REK, BLAŽ PODGORELEC, LUKA HRGAREK IN MUHAMED TURKANOVIĆ

Povzetek: Pri razvoju informacijskih sistemov se poskušamo držati načela šibke sklopljenosti posameznih komponent. Iz takšnega načela običajno izhaja delitev na oblični (ang. frontend) in zaledni (ang. backend) del sistema, ki se še najbolj pogosto pojavlja pri spletnih aplikacijah. V primerjavi z razvojem klasičnih spletnih aplikacij, temelječih na centraliziranih strežnikih, zahteva razvoj spletne aplikacije, ki temelji na pametnih pogodbah verig blokov, upoštevanje in uporabo določenih posebnosti. Predvsem je potrebno izpostaviti porazdeljeno in decentralizirano okolje zaledja ter asinhronost, ki je povezana s tehnologijo veriženja blokov. Razvijalcu mobilne ali spletne aplikacije lahko s pomočjo REST aplikacijskega vmesnika in z uporabo web3 knjižnice, izpostavimo funkcionalnosti, ki jih ponuja pametna pogodba. S tem v veliki meri olajšamo razvoj na pametnih pogodbah temelječih aplikacij, hkrati pa dodamo tudi vmesno točko, na kateri lahko preverjamo identiteto uporabnika, združujemo klice večih pametnih pogodb in dodamo morebitno transformacijo podatkov. V prispevku bomo predstavili prednosti, slabosti, izzive ter primer uporabe API-ja kot stičišča spletnega obličja ter zaledja, temelječega na verigah blokov platforme Ethereum.

Ključne besede: • veriženje blokov • pametne pogodbe • aplikacijski programski vmesnik • Ethereum • Web3.js

NASLOV AVTORJEV: Patrik Rek, Univerza v Mariboru, Fakulteta za elektrotehniko, računalništvo in informatiko, Koroška cesta 46, 2000 Maribor, Slovenija, e-pošta: patrik.rek@um.si. Blaž Podgorelec, Univerza v Mariboru, Fakulteta za elektrotehniko, računalništvo in informatiko, Koroška cesta 46, 2000 Maribor, Slovenija, e-pošta: blaz.podgorelec@um.si. Luka Hrgarek, Univerza v Mariboru, Fakulteta za elektrotehniko, računalništvo in informatiko, Koroška cesta 46, 2000 Maribor, Slovenija, e-pošta: luka.hrgarek@um.si. dr. Muhamed Turkanović, docent, Univerza v Mariboru, Fakulteta za elektrotehniko, računalništvo in informatiko, Koroška cesta 46, 2000 Maribor, Slovenija, e-pošta: muhamed.turkanovic@um.si.

DOI https://doi.org/10.18690/978-961-286-162-9.23 ISBN 978-961-286-163-6

Dostopno na: http://press.um.si.

1. UVOD

Tehnologija veriženja blokov nam omogoča razvoj decentraliziranih, šibko sklopljenih informacijskih sistemov [1]. Razvijalce je prepričala predvsem zaradi možnosti razvoja zaupanja vrednih aplikacij, kar je doseženo z decentralizacijo in porazdeljenostjo, s katero dosežemo, da nepoštenih dejanj ne more biti. Zaradi relativno enostavne možnosti gradnje zasebnih in javnih omrežij ter razvoja pametnih pogodb je trenutno najbolj priljubljena platforma verig blokov, platforma Ethereum [2].

Ethereum je osnovan na transakcijah in klicih in te se lahko izvajajo ročno ali preko decentraliziranih aplikacij. Decentralizirane aplikacije so lahko običajne oblične aplikacije, zasnovane na Javascriptu, a ne komunicirajo več s centraliziranim zaledjem, temveč s pametnimi pogodbami, ki so distribuirane med vse deležnike omrežja verige blokov.

V članku naslavljamo decentralizirane aplikacije, ki vsebujejo aplikacijski programski vmesnik, zasnovan na Node.js in Web3.js knjižnici, ki poskrbi za preprostejšo in varnejšo komunikacijo z omrežjem Ethereum in možnost prilagoditve tehnologije ter uporabo pametnih pogodb za poljubne primere [3]. Alternativa aplikacijskemu programskemu vmesniku je decentralizirana aplikacija, ki neposredno uporablja funkcije na pametni pogodbi.

Naslovili bomo tudi izzive glede asinhronih operacij in dejansko decentraliziranost aplikacij, ki so zasnovane na takšnem principu z vmesnim členom – aplikacijskim programskim vmesnikom.

2. VERIŽENJE BLOKOV

V tem poglavju bomo predstavili tehnologijo veriženja blokov, ki predstavlja infrastrukturo zaledja naše decentralizirane aplikacije.

2.1. Ethereum

Ethereum je omrežje in obenem platforma veriženja blokov, ki omogoča razvoj pametnih pogodb z visokonivojskim programskim jezikom Solidity, omejene zgolj s Turingovo polnostjo. Ponuja abstraktno plast, ki vsakomur omogoča izdelavo lastnih pravil za lastništvo, oblike transakcij in funkcije za spremembo stanj. To dosežemo z uvedbo programljivih in izvršljivih pametnih pogodb, ki predstavljajo skupek pravil, ki se avtomatizirano izvedejo le, če so izpolnjeni določeni pogoji [2].

2.1.1. Ethereum Virtual Machine

Ethereum Virtual Machine (EVM) je navidezna naprava omrežja Ethereum, ki deluje kot izvajalno okolje za pametne pogodbe zasnovane na Ethereum platformi. EVM se uporablja za samodejno izvajanje transakcij ali različnih akcij na verigi blokov, ki so na verigo shranjene v obliki zlogovne kode, prevedene iz visokonivojskega jezika Solidity. Vsako vozlišče, ki je del Ethereum omrežja poganja EVM in tako omogoča izvršljivost pametnih pogodb na vseh vozliščih [1].

2.1.2. Računi

Omrežje Ethereum vsebuje račune, kjer vsak račun predstavlja naslov in spremembe stanja. Glavno stanje so preslikave med naslovi in stanji računov [2].

Poznamo dva tipa računov – zunanje in pogodbene. Prvi so nadzorovani na podlagi zasebnih ključev, drugi s izvorno kodo pametne pogodbe. Račun je sestavljen iz štirih polj: enkratnega kriptografskega števila (angl. *nonce*), stanja Etherjev, zgoščene kode pogodbe in korena shrambe [2].

2.1.3. Transakcije in klici

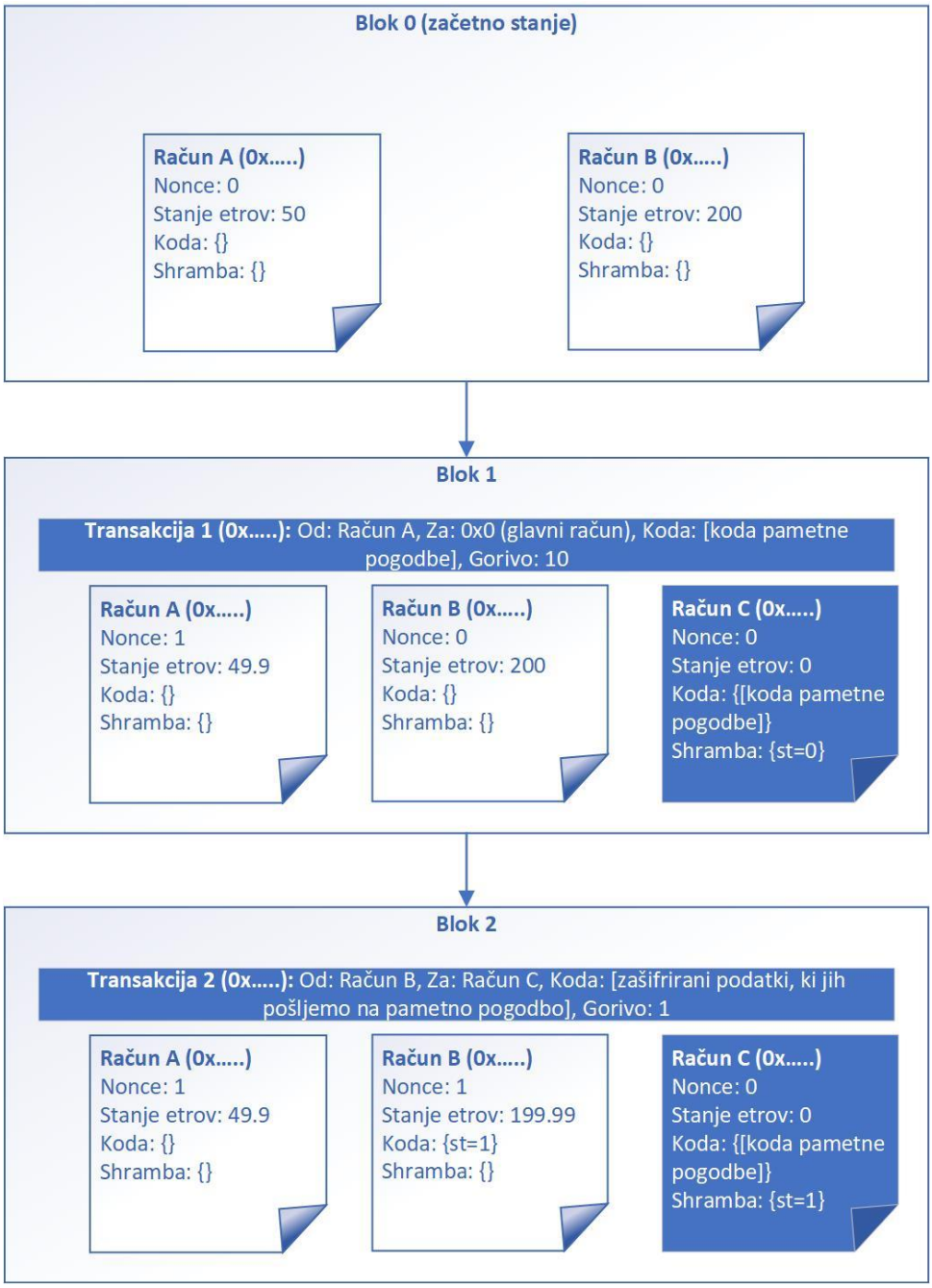
Spremembe stanja na računih predstavljajo transakcije. Vsaka transakcija je kriptografsko podpisana. Obstajata dve vrsti transakcij – sporočilni klici in transakcije, ki ustvarijo nove račune. Rezultat sporočilnih klicev je prenos sporočil ali vrednosti med pametnimi pogodbami. Vsaka transakcija vsebuje štiri pomembne podatke:

- naslov pošiljatelja,
- naslov prejemnika (ta račun je lahko zunanji ali pogodbeni),
- podatki, ki lahko vsebujejo kodo pametne pogodbe (če je transakcija, ki ustvarja nov račun) ali podatke, ki jih na pametno pogodbo pošiljamo (sporočilni klic) ter
- vrednost goriva (angl. gas), ki smo jo pripravljeni plačati.

Količina goriva je nagrada v količin, ki jo prejme tisti, ki transakcijo rudari. Za bolj zahtevne transakcije je potrebno več goriva in je tako tudi nagrada višja. Če je pošiljateljevo stanje Etherjev prenizko za dokončno izvedbo transakcije, se stanje povrne, plačilo za opravljeno rudarjenje se pa vseeno izvede. Zato je potrebno pravilno predvidevat ceno goriva glede na razpoložljivo stanje. Transakcije se potrjujejo združene v blokih in glede na zadnjo številko bloka se izvaja tudi sinhronizacija stanja med rudarji [4].

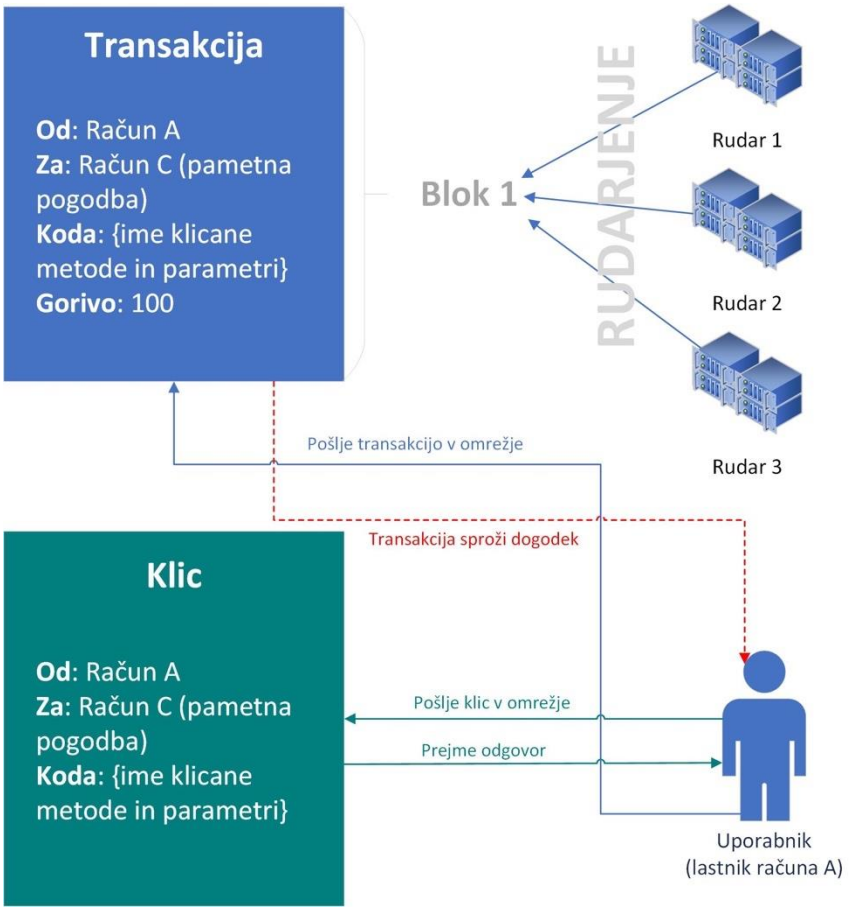
Opisana shema je prikazana na sliki 1. Na začetku imamo odprta dva računa (račun A in račun B), ki sta predstavljena z javnim ključem oz. naslovom v šestnajstiški obliki. Oba imata vrednost »nonce« na začetku nastavljeno na 0 in ta se kasneje, ko izvajamo transakcije, zvišuje. Računoma smo za prikaz nastavili tudi določeno stanje Etherjev. Začetni blok imenujemo blok 0, ker za ustvarjanje obeh računov ni bila potrebna transakcija, če predvidevamo, da je stanje Etherjev že vnaprej določeno. Ustvarjanje zunanjih računov namreč ni plačljiva dejavnost. Uporabnik si lahko denarnico, ki vsebuje podatke o njegovem računu (javni in zasebni ključ), hrani namreč tudi na svojem računalniku ali poljubni lokaciji [5].

Prva transakcija, ki se izvede v bloku 1, kreira nov račun C, ki je pogodbeni račun. Hrani namreč pametno pogodbo in shrambo ter izvaja poljubno kodo. Transakcijo izvede račun A in zanjo porabi 10 enot goriva (enota za gorivo je ponavadi wei, ki predstavlja 10^{-18} Etherja). Ta količina se odšteje od stanja računa A. Račun C dobi svoj naslov (izračunan iz javnega ključa). Računu A se po tej transakciji spremeni enkratno kriptografsko število (»*nonce*«). V bloku 2 vidimo transakcijo 2, ki pošlje zašifrirane podatke na pametno pogodbo (račun C). Ta izvede določeno kodo in spremeni stanje na shrambi. V tem konkretnem primeru poviša spremenljivko *st* na vrednost 1. Pametna pogodba lahko izvaja kompleksno kodo in tudi različne zanke. Seveda to pomeni tudi višjo ceno transakcije.



Slika 1. Shema transakcij v omrežju Ethereum

Zraven transakcij obstajajo tudi klici (angl. *contract call*), ki predstavljajo klice na pametne pogodbe, ki vračajo le vrednosti in ne spreminjajo stanja oz. izvajajo zgolj preprosto procesorsko delo prenosa in prikaza vrednosti. Takšni klici nimajo cene in se izvedejo v trenutku, ker zanje tudi ni potrebno nikakršno potrjevanje oz. rudarjenje. Zato je branje iz verige blokov nezahtevno in hitro, medtem ko je pisanje (izvajanje transakcij) zaradi rudarjenja časovno nekoliko bolj zahtevno. Kot je razvidno na sliki 2, klic vedno vrne odgovor. Ob pošiljanju transakcije uporabnik neposrednega odgovora ne prejme, ampak lahko odgovor po izvedbi (rudarjenju) dobi v obliki dogodka (ang. event), ki ga lahko sproži transakcija. Dogodke je potrebno načrtovati in dodati v kodo pametnih pogodb že v razvoju pametne pogodbe in lahko imajo različen namen. Delovanje transakcij, klicev in dogodkov prikazuje slika 2. Povezava za dogodek je drugačna, saj se proži šele, ko je transakcija potrjena s strani rudarjev v omrežju [1].



Slika 2. Delovanje transakcij in klicev v omrežju Ethereum

2.1.4. Aplikacijski binarni vmesnik (ABI)

Programska koda, ki je napisana v programskem jeziku Solidity, je ob umeščanju na verigi blokov prevedena v zlogovno kodo. To je zato, da lahko EVM, ki izvaja kodo, te programe tudi izvede. EVM procesira le zlogovno kodo, medtem ko je Solidity programski jezik - abstrakcija te kode. Ker zlogovna koda ni samo razlagalna, potrebujemo vmesnik, ki služi kot shema za prevajanje in dostopanje do pametnih pogodb tj. ABI (ang. Application Binary Interface) [6]. V ABI-ju ne shranjujemo posameznih korakov, ampak le podpise funkcij. Vhodi so opisani z imenom in tipom, ob tem pa je zabeleženo tudi, ali je metoda plačljiva in ali je konstanta. Ob morebitni spremembi delovanja pametne pogodbe ni nujna sprememba aplikacijskega binarnega vmesnika, če ni novih funkcij in če ni sprememb v podpisih funkcij, temveč le posodobitev naslovov na novo naloženo pametno pogodbo. ABI za Ethereum pametno pogodbo je JSON dokument in deluje kot je prikazano na sliki 3. Na sliki 3 vidimo primer, kako izgleda ABI za funkcijo, ki je napisana v Solidity programskem jeziku [7]. Prav tako nam slika 3 kaže tudi zlogovno koda, ki se bo izvajala znotraj EVM. Ethereum ABI prikaže tudi dogodke, ki se lahko prožijo, a se teh v tem članku ne bomo dotikali, saj za razumevanje članka takšno znanje ni potrebno.

Slika 3. Primerjava Solidity programske kode, ABI vmesnika in zlogovne kode.

3. VMESNI ČLEN – APLIKACIJSKI PROGRAMSKI VMESNIK

V tem poglavju bomo opisali razvoj vmesnega člena za dostop do verige blokov in izzive, ki se pojavijo ob tem.

3.1. Geth in Parity

Protokol Ethereum obstaja v treh različnih izvedbah – te so zasnovane na programskih jezikih C++, Python ali Go. Go Ethereum je najbolj razširjena implementacija Ethereuma in je, kot nakazuje že samo ime, zasnovana na programskem jeziku Go [5].

Go Ethereum (Geth) se lahko uporablja v Go, Android ali iOS razvojnih projektih ali kot neodvisni odjemalec Geth. Geth je ukazni vmesnik, ki ponuja zagon celotnega Ethereum vozlišča. Omogoča rudarjenje, prenos sredstev med računi, izdelavo pametnih pogodb, pošiljanje transakcij, raziskovanje zgodovine blokov idr. Geth omogoča zagon vozlišča z možnostjo JSON-RPC strežnika in interaktivne konzole s podporo Web3 Javascript knjižnici. Obe možnosti omogočata komunikacijo z vozliščem in Ethereum omrežjem iz decentraliziranih aplikacij [8].

Ob Geth je razširjen tudi odjemalec Parity, ki omogoča višjo učinkovitost in zanesljivost ter je preprost za namestitev [9]. Je popolnoma kompatibilen z JSON-RPC, kar pomeni, da je kompatibilen s knjižnicami, ki delujejo po takšnem principu [10].

3.2. JSON RPC in Ethereum aplikacijski programski vmesniki

RPC (ang. Remote Procedure Call) je protokol za oddaljene klice funkcij. Postopki JSON RPC se lahko uporabijo znotraj procesov, preko vtičnikov ali preko HTTP protokola. Podatki se prenašajo v obliki JSON dokumentov in imajo šestnajstiško obliko [11]. Primer JSON RPC klica, ki kot parameter sprejme zgoščeno kodo transakcije je viden v izseku kode 1.

```
curl -X POST --data
'{"jsonrpc":"2.0","method":"eth_getTransactionByHash","params":["0x962c3c969400
b52812e06dad741b22e1c3b3c5f45d97304155fa2414960cb2df7a"],"id":1}'
```

Izsek kode 1. Primer JSON RPC klica

Kot je razvidno v izseku kode 2, ki predstavlja del odgovora, je vsak parameter zapisan v šestnajstiški obliki.

```
"result": {
    "hash":"0xa311a679741f23f82252076c2978966348e310da1ca1c31e6c530a0791110558",
    "nonce":"0x",
    "blockHash":
"0xa50aee9e96d551c594e7a67a95ad0fdeede669b76294073d5ac06f6d04003392",
    "blockNumber": "0x4d2", // 1234
    "transactionIndex":  "0x1", // 1
    "from":"0xf3005019a966bf4103ce1c77539451fa93508bf",
    "to":"0xcc69e0d7c9c0e5140ea7eb3004b1da3c97e3658",
    "value":"0x1f4", // 500
    "gas": "0x1f4", // 500
    "gasPrice":"0x09184e72a000"
  }
```

Izsek kode 2. Primer JSON RPC odgovora

Po principu JSON RPC delujejo tudi Ethereum aplikacijski programski vmesniki, ki jih izpostavljajo odjemalci. Ker pa je takšna oblika zapisa človeku neprijazna in JSON RPC ni praktičen, saj ne podpira Javascript programskega jezika ali katerega od drugih jezikov za spletno programiranje, so Ethereum aplikacijski programski vmesniki zasnovani v HTTP obliki in se jih lahko kliče iz različnih virov. JSON RPC API lahko kličemo le preko konzole. Za omogočanje aplikacijskih programskih vmesnikov je potrebno ob zagonu odjemalca na vozlišču izrecno specificirati, katere vmesnike želimo izpostaviti. Ob zagonu moramo omogočiti HTTP RPC vmesnik, kar dosežemo z zastavico ***–rpc***. Ob tem izpostavimo tudi vmesnike z zastavico ***–ipcapi***, za dostop preko Unix vtičnika, ***--rpcapi*** za HTTP končno točko ali ***--wsapi*** za WebSocket končno točko. Z Unix in WebSocket se v tem članku ne bomo ukvarjali, saj se osredotočamo zgolj na HTTP kot končno točko. Ob zastavici moramo navesti vmesnike, ki jih želimo izpostaviti. Geth podpira vmesnike, ki so vidni v tabeli 3 [12].

Tabela 3. Vmesniki API za RPC.

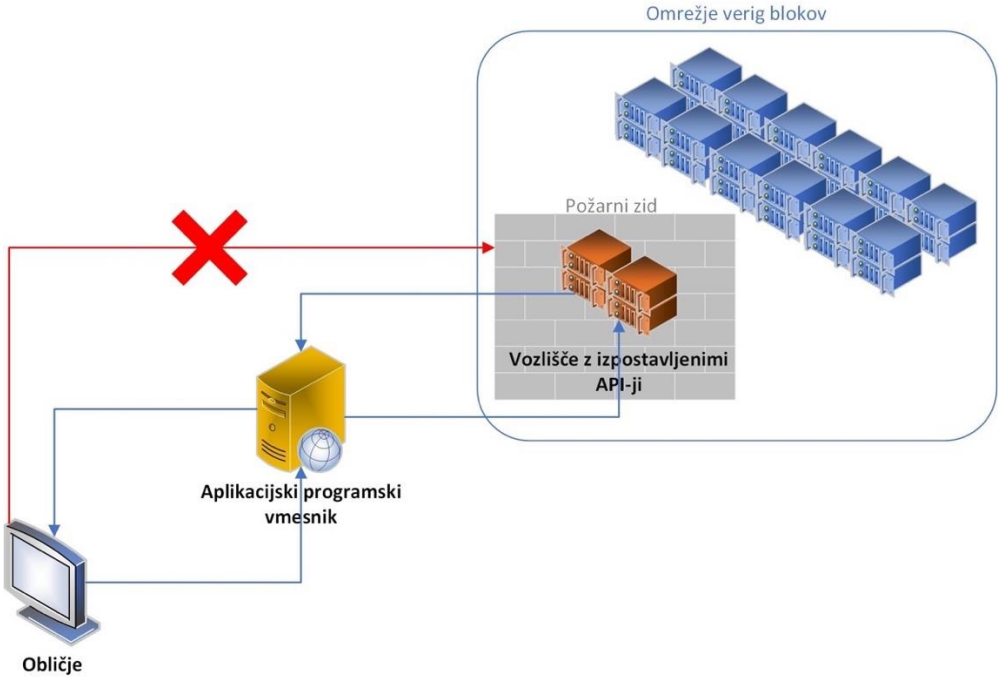
Vmesnik	**Funkcionalnosti**
db	Pridobivanje nekaterih osnovnih informacij o računih
net	Informacije o omrežju
eth	Vmesnik za delo s transakcijami, najvišji nabor funkcionalnosti
web3	Delo preko Javascript knjižnice
admin	Administracija Geth vozlišča
debug	Možnosti razhroščevanja med izvajanjem
miner	Nastavitve rudarjenja na vozlišču
personal	Delo z zasebnimi ključi v bazi ključev na vozlišču
txpool	Pregled vsebine bazena transakcij, ki vsebuje vse čakajoče transakcije

3.3. Vpeljava lastnega aplikacijskega programskega vmesnika

Izpostavljanje aplikacijskih programskih vmesnikov na Ethereum vozlišču in neposredna uporaba v Javascript spletnih aplikacijah je pogosto uporabljena možnost, ki pa prinaša tudi določene slabosti. Največja slabost, ki se pri tem pojavi je javno odpiranje Geth vozlišča. To nepridipravom odpre možnosti za napad na vozlišče in s tem na omrežje, ki lahko povzroči razvrednotenje podatkov na verigi

blokov. Druga slabost je razkritje Javascript kode na obličju, s čimer izgubimo možnost vključitve dodatne poslovne logike v decentralizirane aplikacije.

Omenjena izziva lahko naslovimo z vpeljavo lastnega aplikacijskega programskega vmesnika (API). Vmesnik tako deluje kot stičišče med zaledjem, ki je lahko zaščiteno s požarnim zidom, in obličjem, ki dostopa do metod našega aplikacijskega programskega vmesnika preko asinhronih HTTP klicev. Tako lahko v aplikacijo vključimo tudi poslovno logiko, brez da bi to izpostavili za javnost. Shemo prikazuje slika 4.



Slika 4. Shema decentralizirane aplikacije z aplikacijskim programskim vmesnikom

3.4. Node.js

Node.js je JavaScript izvajalno okolje, ki uporablja dogodkovni, asinhroni model vhodov in izhodov, kar ga naredi preprostega in učinkovitega. Node.js uporablja ekosistem paketov ***npm***, ki je največja zbirka odprtokodnih knjižnic na svetu [13]. Narejen je za gradnjo razširljivih omrežnih aplikacij. Node.js omogoča preprosto izgradnjo spletnega strežnika in zaledne aplikacije, ki v celoti temelji na jeziku JavaScript. Prednost Node.js pred ostalimi bolj konvencionalnimi strežniškimi sistemi, ki so namenjeni spletnim aplikacijam PHP, ASP.NET ali drugim, je predvsem v preprostosti in manjši porabi virov. Ob tem pa veliko prednost prinaša tudi veliko število raznolikih knjižnic, za primer veriženja blokov konkretno knjižnica Web3.js. Node.js podpira zadnjo različico JavaScript programske kode – ES6. Ta med drugim prinaša delo z asinhronimi klici funkcij in nekatere možnosti, ki jih poznamo iz tradicionalnih programskih jezikov [13].

Namestitev Node.js okolja je preprosta in je na voljo za vse namizne operacijske sisteme (Windows, macOS in Linux). Ob namestitvi se samodejno namesti tudi sistem paketov ***npm***. Za začetek gradnje Node.js aplikacije v konzolno okno vnesemo ***npm init ime_aplikacije*** in potrdimo podatke o aplikaciji. To ustvari datoteko *package.js*, ki vsebuje podatke o aplikaciji. Z ukazom ***npm install ime_knjižnice --save*** lahko nato namestimo knjižnice, ki jih potrebujemo v aplikaciji, samo jedro aplikacije pa lahko programiramo v datoteko, ki smo jo ob inicializaciji označili kot *»main«*. V vsaki datoteki moramo z ukazom *require* označiti, katere knjižnice potrebujemo za specifično datoteko. Primer, ki ga prikazuje slika 5, uporablja knjižnico ***http*** za kreiranje spletnega strežnika. Pri takšnem načinu je potrebno

obravnavati vsako zahtevo, ki jo pošlje odjemalec do katere koli datoteke na strežniku. Za zagon strežnika je v konzolno okno potrebno vnesti ukaz ***node ime_datoteke***, kjer *ime_datoteke* predstavlja ime datoteke, v kateri imamo kodo strežnika.

index.js

```
const http = require('http');

const hostname = '127.0.0.1';
const port = 3000;

const server = http.createServer((req, res) => {
  res.statusCode = 200;
  res.setHeader('Content-Type', 'text/plain');
  res.end('Pozdravljen, OTS!\n');
});

server.listen(port, hostname, () => {
  console.log(`Strežnik teče na http://${hostname}:${port}/`);
});

```

Slika 5. Primer uporabe Node.js za ustvarjanje strežnika

3.5. Express

Za poenostavitev vzpostavljanja spletnega strežnika v okolju Node.js je bil vzpostavljen projekt Express [14], ki naslavlja obravnavanje zahtev za strežnik. Knjižnica ***express*** nadomesti knjižnico ***http*** in namesto razvijalca sama obravnava različne scenarije ob zahtevkih. Ponuja usmerjanje za metode različnih tipov (GET, POST, PUT, DELETE) in tako omogoča pripravo vmesnika za vse CRUD (»Create, Read, Update, Delete) operacije [14].

Kot je vidno na sliki 6, je izvorna koda za spletni strežnik z uporabo knjižnice Express nekoliko poenostavljena. Omogoča tudi upravljanje parametrov v naslovu ali jedru. Ob tem lahko preprosto vračamo tudi JSON objekt, kar je pri aplikacijskem programskem vmesniku najboljša praksa, saj je JSON odprti standard, ki ga lahko prebere kateri koli jezik, v katerem je napisano obličje spletne aplikacije.

index.js

```
const express = require('express')
const app = express()

// GET zahtevek s parametrom v naslovu
app.get('/prvaStran/:id', (req, res) => res.send(req.params.id+', pozdravljeni na GET metodi!'));
// POST zahtevek s parametrom v jedru
app.post('/drugaStran', (req, res) => res.send(req.body.id+', pozdravljeni na POST metodi!'));

app.get('/json', (req, res) => res.json({naslov: "OTS2018"}));

app.listen(3000, () => console.log('Strežnik teče na vratih 3000!'))
```

Slika 6. Primer uporabe knjižnice Express za strežnik

3.6. Web3

Za povezovanje aplikacijskega programskega vmesnika z izpostavljenim Ethereum vozliščem se uporabi knjižnica Web3. Poznamo več izvedb knjižnice prilagojene za različne programske jezike, a v

našem članku bomo naslovili Node.js knjižnico Web3.js. Web3.js knjižnica je zbirka modulov, ki vsebujejo različne funkcionalnosti za Ethereum ekosistem in omogočajo povezovanje na vozlišče, ki ima izpostavljen web3 vmesnik [15].

Za vzpostavitev *web3* knjižnice uporabimo enak postopek kot pri vseh ostalih Node.js knjižnicah. Z vozliščem se povežemo s pomočjo ***HttpProvider***, kot prikazuje slika 7. Nato sledi priprava vmesnika za pametno pogodbo. Zato potrebujemo prej opisan ABI, s pomočjo katerega ustvarimo primerek vmesnika. Ko je vmesnik pripravljen, lahko uporabljamo vse metode kot klice ali transakcije pametne pogodbe znotraj verige blokov, ob tem pa lahko lovimo tudi vse dogodke, ki smo jih definirali v pametni pogodbi.

Klici in transakcije se izvajajo na način, ki je prikazan na sliki 7. Pomemben podatek je tukaj, da se za klice uporablja metoda ***call***, ki ne potrebuje ničesar razen imena klicane funkcije in morebitnih parametrov. Kot smo poudarili, prejmemo odgovor takoj in ga zato lahko izpišemo. Pri izvajanju transakcij je postopek nekoliko bolj kompleksen. Najprej potrebujemo šifriran ABI za transakcijo, ki jo pošiljamo. Definirati moramo tudi naslov pošiljatelja (*racun*), omejitev in ceno goriva (*gas* in *gasPrice*) ter naslov pametne pogodbe, kamor pošiljamo transakcijo. V spremenljivki *tran_res*, ki jo v zgornjem primeru izpisujemo, se izpiše naslov transakcije, na podlagi katerega lahko na različne načine sledimo stanju transakcije.

index.js

```
const express = require('express')
const app = express()
const web3 = require('web3')
// ustvarjanje povezave z vozliščem
var web3Provider = new web3(new web3.providers.HttpProvider('http://localhost:8545'));
// inicializacija pametne pogodbe glede na ABI
var web3Abi = require('./abi.js'); // v tej datoteki se nahaja ABI
var web3Contract = new web3.eth.Contract(web3Abi,'0x9b2d312eeed7a7c821a77416ce820262b0e9cae5');

// GET zahtevek s klicem metode
app.get('/klic/:x', async (req, res) => {
  let x = req.params.x;
  await web3Contract.methods.preberi(x).call(async (error, result) => {
    if (error)
      res.status(500);
    else {
      res.json({rezultat: result});
    }
  });
});

// POST zahtevek s pošiljanjem transakcije
app.post('/transakcija', async (req, res) =>
  let y = req.body.y;
  let geslo = req.body.geslo;
  let racun = req.body.racun;

  let abi_encoded = web3Contract.methods.poslji(y).encodeABI();
  let tx = {
    from: racun,
    gas: 1000,
    gasPrice: 100,
    to: web3Contract.options.address,
    data: abi_encoded
  };
  let tran = web3.eth.personal.sendTransaction(tx, geslo).then((tran_res) => {
    res.json({rezultat: tran_res});
  }).catch((error) => res.status(500));
});

app.listen(3000, () => console.log('Strežnik teče na vratih 3000!'))
```

Slika 7. Primer aplikacijskega programskega vmesnika za pametno pogodbo Ethereum

Večina funkcij je opisana v uradni dokumentaciji knjižnice Web3.js [3]. Nekatere funkcionalnosti, ki jih knjižnica sicer podpira so slabše dokumentirane. Zato v naslednjem podpoglavju izpostavimo nekaj uporabnih metod, ki so pomanjkljivo dokumentirane ali niso dovolj izpostavljene in so uporabne v praktični uporabi knjižnice.

3.6.1. Izpostavljene metode

Izpostavljene metode in njihova uporabnost so izpisane v tabeli 4.

Tabela 4. Izpostavljene metode Web3

Metoda	**Funkcionalnost**
web3.eth.getTransactionReceipt(hash)	Iz naslova transakcije pridobimo podatke o njeni izvedbi, vključujoč uporabljeno gorivo in stanje. Funkcija vrne *null*, če transakcija še ni bila potrjena.
web3Contract.events.MyEvent([options], callback)	Naročimo se na dogodek MyEvent. V nastavitvah nastavimo filtre za dogodek, v »callback-u« pa pridobimo podatke, ki jih dogodek pošlje.
web3.eth.personal.newAccount(password)	Na vozlišču ustvarimo nov uporabniški račun (zasebni ključ) in ga zaklenemo z izbranim geslom.
web3.eth.personal.sendTransaction(tx, geslo)	Pošiljanje podpisane transakcije iz uporabniškega računa, ki je zaklenjen z geslom (za odklepanje je potrebno pravilno *geslo*). Glej primer na sliki 7.

Pri metodah, ki se nahajajo v domenskem prostoru web3.eth.personal je potrebna opomba, da delujejo le, če je na vozlišču izpostavljen vmesnik ***personal***. Računi, ki so ustvarjeni na tak način, se shranijo v bazi ključev znotraj vozlišča, ki ima izpostavljen aplikacijski programski vmesnik. Tukaj pride do izraza izziv varnosti pri takšnem načinu, medtem ko pridobimo ograjevanje celotne decentralizirane aplikacije, kar končnemu uporabniku prinese dodano vrednost.

4. OBLIČJE ANGULAR

Metode, ki jih lahko izpostavimo preko aplikacijskega programskega vmesnika, omogočajo razvoj različnih uporabniških vmesnikov, od namiznih, preko spletnih ter vse do mobilnih aplikacij. Če bi želeli izdelati grafični uporabniški vmesnik za dostop do metod znotraj pametnih pogodb verig blokov platforme Ethereum, bi ena izmed možnosti bila tudi izdelava prototipne "decentralizirane" spletne aplikacije, kot je predstavljeno v poglavju 2.

Danes je na voljo kopica različnih knjižnic in ogrodij, ki poenostavljajo razvoj spletnih aplikacij, med drugimi tudi AngularJS. To je odprtokodno Javascript MVC-ogrodje, ki ga vzdržuje Google, razvil pa ga je Miško Hevery pri podjetju Brat Tech LLC leta 2009 [16]. V uradni dokumentaciji [17] je definiran kot strukturno ogrodje za dinamične spletne aplikacije. Njegov namen je razširitev označevalnega jezika HTML s posebnimi značkami in atributi, ki mu omogočajo realnočasovno sinhronizacijo z JavaScript kodo. To razvijalcu omogoča, da več časa posveti razvoju aplikacijske logike, namesto, da bi se ukvarjal s posodabljanjem prikaza podatkov oziroma MVC komponente View [18]. Med najpomembnejše lastnosti ogrodja AngularJS lahko uvrstimo dvosmerno vezavo podatkov (angl. two-way data binding), predloge (angl. templates), vstavljanje odvisnosti (angl. dependency injection) ter ukaze (angl. directives) [18].

S pomočjo ogrodja je možno pripraviti enostransko (angl. single-page) decentralizirano spletno aplikacijo v kateri se lahko na posameznih zavihkih omogoči dostop do metod znotraj pametnih pogodb verig blokov platforme Ethereum. Če so metode na aplikacijskem programskem vmesniku izpostavljene v obliki REST, se jih lahko brez težav vključi v oblični del aplikacije.

Pri takšnem razvoju je potrebno razmisliti o načinu implementacije asinhronih metod, ki jih ni možno implementirati v obliki REST. Vpeljava tehnologije WebSocket ponuja največ možnosti za odpravo

tovrstnih težav in zagotavljanje najbolj učinkovitega načina sodelovanja obličnega dela aplikacije z zaledjem.

5. ZAKLJUČEK

V prispevku smo predstavili Ethereum platformo, njeno delovanje in decentralizirane aplikacije. Te lahko oblikujemoneposredno ali s pomočjo aplikacijskega programskega vmesnika. Izpostavili smo pomembnosti Geth in Ethereum aplikacijskih programskih vmesnikov ter s tem povezane izzive. Osrednji del prispevka je API, ki je vmesni člen med obličjem in zaledjem. Dotaknili smo se tudi knjižnice Web3. Knjižnica Web3 nam omogoča klice funkcij pametnih pogodb verig blokov posredno preko izpostavljenega aplikacijskega programskega vmesnika. Knjižnico lahko uporabimo neposredno za razvoj obličja, s čimer pa nastanejo novi izzivi v obliki javnega izpostavljanja metod na vozlišču in izpostavljanja poslovne logike preko obličja. S temi izzivi se lahko spopademo z uvedbo vmesnega člena, ki je zasnovan na Node.js platformi. Na tak način pridobimo na varnosti, odpre se pa nov izziv, ki se tiče shranjevanja baze ključev in decentraliziranosti aplikacij. V prihodnje bomo ta izziv lahko odpravili z orodji za decentralizacijo spletnih aplikacij, ki že obstajajo (IPFS [19]). V članku opisan postopek bomo v prihodnosti še nadgradili z asinhronimi metodami za prikaz stanja transakcij, kar bomo dosegli s pomočjo tehnologije WebSocket.

6. LITERATURA

[1] G. Wood, „ETHEREUM: A SECURE DECENTRALISED GENERALISED TRANSACTION LEDGER“.

[2] D. Vujicic, D. Jagodic, in S. Randic, „Blockchain technology, bitcoin, and Ethereum: A brief overview“, v *2018 17th International Symposium INFOTEH-JAHORINA (INFOTEH)*, 2018, str. 1–6.

[3] Ethereum, „web3.js - Ethereum JavaScript API — web3.js 1.0.0 documentation“, 2016. [Na spletu]. Dostopno: https://web3js.readthedocs.io/en/1.0/. [Dostopano: 16. maj 2018].

[4] ConsenSys, „Ethereum: Bitcoin Plus Everything – ConsenSys – Medium“, 2016. [Na spletu]. Dostopno: https://medium.com/@ConsenSys/ethereum-bitcoin-plus-everything-a506dc780106. [Dostopano: 15. maj 2018].

[5] go-ethereum, „Go Ethereum“, 2016. [Na spletu]. Dostopno: https://ethereum.github.io/go-ethereum/. [Dostopano: 16. maj 2018].

[6] Curlysemi, „Ethereum and Solidity: Application Binary Interfaces (and Function Signatures)“, 2017. [Na spletu]. Dostopno: http://www.curlysemi.com/ethereum-and-solidity-application-binary-interfaces-and-function-signatures/. [Dostopano: 15. maj 2018].

[7] Ethereum, „Application Binary Interface Specification — Solidity 0.4.24 documentation“, 2018. [Na spletu]. Dostopno: http://solidity.readthedocs.io/en/develop/abi-spec.html. [Dostopano: 15. maj 2018].

[8] Ethereum, „Geth“, 2017. [Na spletu]. Dostopno: https://github.com/ethereum/go-ethereum/wiki/geth. [Dostopano: 16. maj 2018].

[9] S. Rouhani in R. Deters, „Performance analysis of ethereum transactions in private blockchain“, v *2017 8th IEEE International Conference on Software Engineering and Service Science (ICSESS)*, 2017, str. 70–74.

[10] Parity Technologies, „Parity“, 2018. [Na spletu]. Dostopno: https://www.parity.io/. [Dostopano: 21. maj 2018].

[11] Ethereum, „JSON RPC“, 2018. [Na spletu]. Dostopno: https://github.com/ethereum/wiki/wiki/JSON-RPC. [Dostopano: 16. maj 2018].

[12] Ethereum, „Management APIs“, 2017. [Na spletu]. Dostopno: https://github.com/ethereum/go-ethereum/wiki/Management-APIs. [Dostopano: 16. maj 2018].

[13] Node.js Foundation, „Node.js“, 2017. [Na spletu]. Dostopno: https://nodejs.org/en/. [Dostopano: 21. jul 2017].

[14] StrongLoop, „Express - Node.js web application framework“, 2017. [Na spletu]. Dostopno: https://expressjs.com/. [Dostopano: 16. maj 2018].

[15] M. Pustišek in A. Kos, „Approaches to Front-End IoT Application Development for the Ethereum Blockchain“, *Procedia Comput. Sci.*, let. 129, str. 410–419, 2018.

[16] M. Hevery, „Hello World, <angular/> is here“, 2009. [Na spletu]. Dostopno: http://misko.hevery.com/ 2009/09/28/hello-world-angular-is-here/. [Dostopano: 18. maj 2018].

[17] AngularJS, „What Is AngularJS?“, 2017. [Na spletu]. Dostopno: https://docs.angularjs.org/guide/introduction. [Dostopano: 18. maj 2018].

[18] N. Jain, P. Mangal, in D. Mehta, „Mandsaur Institue of Technology AngularJS: A Modern MVC Framework in JavaScript“, *J. Glob. Res. Comput. Sci.*, let. 5, št. 12, 2014.

[19] Protocol Labs, „IPFS is the Distributed Web“, 2018. [Na spletu]. Dostopno: https://ipfs.io/. [Dostopano: 21. maj 2018].

Univerzitetna založba Univerze v Mariboru

1 LETO ČASA – 20 LET ZGODOVINE – 300X10^4 VRSTIC KODE. TRANSFORMIRAJ ZDAJ!

ROBERT KRISTANC IN MARJAN KALIGARO

Povzetek: Investicija v razvoj programske opreme je praviloma precejšnja. To seveda pomeni, da želimo aplikacijo (če ustrezno funkcionalno deluje) uporabljati čim dlje časa. Dodaten razlog proti menjavi (poslovnih) aplikacij je tudi migracija, ki za sabo prinese cel kup potencialnih nevšečnosti (ki izhajajo bodisi iz nepoznavanja vsebin bodisi iz nekompatibilnosti virnih in ciljnih podatkovnih struktur). V podjetju SRC d.o.o. smo se omenjenih dveh problemov lotili na način, ki ne zahteva migracije in/ali zamenjave aplikacije. Vzpostavili smo mehanizem REST spletnih storitev, ki povezujejo obstoječe zaledne sisteme z modernim spletnim vmesnikom.

Ključne besede: • digitalizacija poslovnih procesov • API ekonomija • REST spletne storitve • legacy sistemi in podatki

NASLOV AVTORJEV: Robert Kristanc, svetovalec, SRC d.o.o., Tržaška cesta 116, 1000 Ljubljana, Slovenija, e-pošta: robert.kristanc@src.si. mag. Marjan Kaligaro, pomočnik izvršnega direktorja, SRC d.o.o., Tržaška cesta 116, 1000 Ljubljana, Slovenija, e-pošta: marjan.kaligaro@src.si.

DOI https://doi.org/10.18690/978-961-286-162-9.24 ISBN 978-961-286-163-6

Dostopno na: http://press.um.si.

1. 1 LETO ČASA – 20 LET ZGODOVINE – 300×10^4 VRSTIC KODE. TRANSFORMIRAJ ZDAJ!

V tem prispevku bomo obravnavali poslovni problem, s katerim se soočajo številne organizacije, ki so v preteklosti veliko vlagale v informacijsko podporo, prikrojeno po meri, danes pa želijo poslovanje digitalizirati, prilagoditi sodobni API ekonomiji, podpreti uporabo mobilnih naprav in podobno. Pretekle investicije so bile namenjene predvsem realizaciji projektov, katerih namen je bil v prilagajanju platform in naročanju programskih rešitev za podporo specifičnim poslovnim procesom, ki jih generična programska oprema včasih ni mogla podpreti. Razvoj takšnih rešitev je bil zaradi nižje cene pogosto vezan na pripravo rešitev s pomočjo pred desetletji popularnih RAD orodij, kar v kombinaciji z velikimi projekti, katerih rezultati so preživeli desetletja, danes povzroča še toliko večje probleme pri digitalizaciji poslovanja. Podobno velja tudi za podatke v bazah, ki po obsegu daleč presegajo kakršnekoli možnosti za opustitev in nov začetek, struktura podatkov pa nočno moro za vsakogar, ki bi želel te podatke migrirati.

V nadaljevanju bomo videli, da je pogosto lažje začeti kot novinec na trgu, »disruptor«, kot nekdo, ki več deset let svojim strankam omogoča prilagajanje aktualnim tehnološkim trendom in novim vsebinskim zahtevam. Kakorkoli že – vsak »disruptor« na področju informacijske tehnologije v nekaj desetletjih postane »dinozaver«, a pomembno je, da nekje globoko v sebi še vedno skriva to osnovno sposobnost za generiranje dobrih idej, za naivno pobalinsko razvijanje alternativnih produktov in za vizijo, ki daleč presega lastne produkte.

2. VČASIH JE LEPŠE BITI START:UP

Start-up podjetja so po eni strani zelo popularna, saj lahko ponudijo iz »nič« nek inovativen, disruptiven, nov pristop k izvajanju določene storitve. Včasih z rezultati svojega dela celo uvedejo nove storitve. Vendar na tem mestu postavljam predrzno trditev: start-up podjetjem je z določenega vidika prav lepo in udobno!

Jasno je, da ne govorimo o obstoječih pogodbah in poslih, pridobljenih v preteklih letih in včasih celo v preteklih desetletjih. Prav tako ne govorimo o zaslužkih, saj so start-up podjetja res privlačna in za mlade pogosto predstavljajo nekakšno avanturo, ki sicer utegne za seboj pustiti finančno brezno ali vsaj praznino, kljub temu pa sodelavcem da neprecenljive izkušnje in kompetence, ki so zanimive za trg.

Tu govorimo o dobrih idejah, ki dejansko prodrejo na trg in osvojijo publiko. Ideje na videz nastanejo iz »nič« - čeprav gre praviloma prej za leta in leta načrtovanj, poskusov in popravkov – in zablestijo v neki tržnici aplikacij, na cesti, v gospodinjstvu v tovarni… Te uspešne inovativne in prevratniške ideje kljub vsemu praviloma nimajo neke zgodovine. Aplikacije se v ozadju povezujejo z relativno praznimi bazami, kakšna funkcionalnost ali storitev še manjka, a tega niti ne vemo, ker je doslej nismo poznali, zakonodajalci so »presenečeni« in »brez odgovorov«. Ko drzno govorimo o tem, da je včasih udobno biti »start-up«, nas predvsem muči zavist ob pogledu na mladega, do tedaj neopaženega konkurenta, ki s svojo spolirano bleščečo rešitvijo osvaja trg kot za šalo, mi pa se zavedamo, da sedimo v grdi stari zarjaveli škatli, ki z malim, kadečim se motorčkom za seboj vleče celo kompozicijo težkega tovora.

Problem, s katerim se soočamo v industriji »stari« ponudniki storitev na področju IKT, je naša lastna dediščina. To so rešitve, ki so za naročnike nastajale po meri, po kroju njihovih lastnih pravil in zahtev. To so podatki, ki so se nakopičili v letih in desetletjih uporabe, terabajti vsebin, vmesniki za specifična okolja, ki jih je svoje čase zahteval naročnik in podoba, ki v primerjavi z novimi, svežimi, disruptivnimi rešitvami seveda ni prav nič sodobna in prav nič uporabniku prijazna. Naročniki seveda želijo preskočiti na nivo sodobnega koncepta in uporabniške izkušnje, želijo uporabljati pametne naprave, IoT naprave in ostale popularne odjemalce in vire podatkov, a so ujeti v zanko zastarelega koncepta in nepregledne mase podatkov.

Videti je kot brezizhodna situacija, videti je kot odločitev o popolnem rušenju starih hiš, a kot vidimo v vsakdanjem življenju, se tudi stare vile pogosto prelevijo v sodobne, varčne, pametne in na pogled lepe stavbe, če se le v njih skriva preveč zgodovine in minulega truda, da bi enostavno vzeli dinamit in začeli graditi vse na novo. Ta prispevek govori o preobrazbi takšnih informacijskih rešitev, ki jih enostavno ne moremo in ne smemo podreti, ampak moramo na njihovih temeljih zgraditi nekaj sodobnega, prijaznega in morda celo rahlo inovativnega.

Iz zavisti tako nastajajo novi koncepti, nove strategije, nove platforme, ki so bile še pred nekaj leti zgolj utopija in rezervirane za tiste, ki sanjajo pri belem dnevu.

2.1. Kaj početi z vsemi temi podatki?

Baze morajo biti tehnološko standardne in zagotavljati dobro podporo proizvajalca ali skupnosti, mar ne? Če bi danes začeli nov projekt, bi se verjetno odločili za Oracle, DB2, MS SQL, Mongo, MySQL oziroma MariaDB, SQLite ali še kaj drugega?

Vsi vemo, kaj bi bilo prav za določene poslovne potrebe strank. Vendar žal ni vsak projekt novost. Nekateri projekti se morajo soočiti z deset, dvajset ali celo več let trajajočo zgodovino. Takšni projekti potem naletijo na stotine milijonov in milijarde podatkov v bazah, kot so Lotus NSF, Paradox ali dBase. Organizacija, ki na primer v bazo vnaša le po 1.000.000 podatkov letno in za vnos posamičnega podatka porabi povprečno samo sekundo delovnega časa, po 20 letih uporabe v bazi poleg podatkov skriva tudi 5.555 ur dela oziroma ob utopičnem pričakovanju neprekinjenega 8-urnega dela ob vseh delovnih dneh za 694 človek x dan opravljenega dela.

Realno je opravljenega dela, ki ga je treba prišteti k vrednosti samih podatkov, mnogo več. Ena od slovenskih organizacij, ki smo ji v preteklosti zagotovili podporo poslovnim procesom, ima distribuirano rešitev, ki obsega okrog 150 milijonov zapisov, kjer vsak zapis vsebuje okrog 200 podatkov. Kreiranje takega zapisa ob vseh avtomatizacijah postopkov in podpori raznih šifrantov in integracij z drugimi podatkovnimi viri celo za izkušenega uporabnika pomeni vsaj 45 sekund dela. Preračunano v delo, so zgolj za evidenco zapisov porabili do zdaj okrog četrt milijona delovnih dni.

Vse te obsežne »legacy« baze, grde in danes skrajno nezaželene, so seveda mnogo več kot le baze. So vir za izvajanje predvidenih poslovnih postopkov, so tudi vir za stotine integracij v druge poslovne postopke, ki jih sicer podpirajo druge rešitve in druge baze. Prepletenost starih sistemov je enormna in zdravo-razumskemu človeku z jasno sliko o tem, kaj bi se zgodilo, če bi vso to zgodovino zavrgli, je v hipu jasno, da je treba iskati take rešitve, ki bodo ohranile opravljeno delo in tiste postopke, ki za digitalizacijo niso pomembni, ali pa so morda celo že avtomatizirani.

Nove stvari so lepe, privlačijo nas s svojo enostavnostjo in prijaznostjo, a dela, ki smo ga vložili v stare rešitve, pač ne moremo kar odpeljati na smetišče zgodovine. To je tako, kot da bi v Rimu podrli Kolosej in na njegovo mesto postavili McDonald's. Ob iskanju sodobnih rešitev moramo upoštevati in spoštovati tudi zgodovino in pretekle investicije.

2.2. Kaj je narobe z migracijo podatkov?

Nič. Migracija podatkov je super enostaven in stroškovno nezahteven projekt, če ga le lahko izvedemo nad konsistentno podatkovno strukturo. Pri 20 in več let starih bazah pa ni vedno tako, še posebej če gre za tako imenovane »dokumentno orientirane« baze, ki hranijo zapise v natanko takšnem naboru in strukturi, kakršni so bili ob nastanku. Predstavljajte si podatek o tem, koliko časa naj bi nek dokument hranili, ki je v letih 1995 do 2004 nastopal kot navadno besedilo (na primer »40 let«), od 2004 do 2008 kot letnica, ko se dokument lahko uniči (na primer »2036«) in od leta 2009 naprej kot običajen numerični podatek (na primer »40«). Da bi bila ta zmeda še malo večja, v končni fazi ugotovite, da ne veste točno, kdaj točno se je struktura tega podatka na katerem od strežnikov zamenjala.

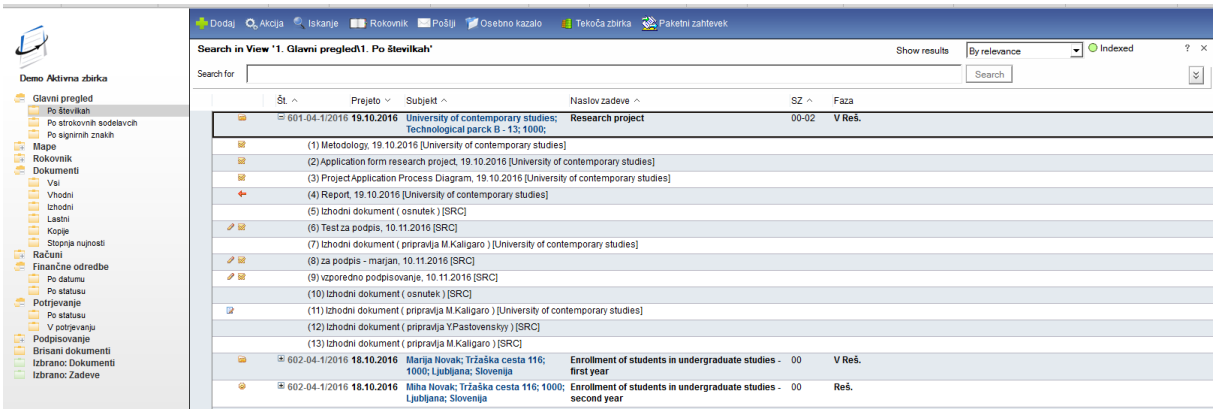
Potem pogledate podatke in ugotovite, da je precej enostavno ločiti obvezne podatke od opcijskih. Super – sedaj natančno veste, katere podatke boste zanesljivo prenesli v ciljno bazo in na katere podatke lahko računate le pogojno. Napišete kodo, ki bo poskrbela za migracijo v ciljno bazo, vzpostavite ustrezne tabele in poženete. V naslednjem trenutku vas migracijski mehanizem zasuje z napakami, čeprav ste tehnično naredili vse po pravilih in je koda napisana zgledno. Kaj se je zgodilo? Zgodila se je zgodovina. Struktura podatkov – v eni in isti bazi – se je z leti pogosto spreminjala, nabor obveznih podatkov je bil enkrat tak, drugič drugačen. Ob migraciji ste se zanašali na varljiv občutek konstantne podatkovne strukture, ki pa v zgodovini zbiranja podatkov nima nobene prave osnove. Zato je migracijski mehanizem, ki ste ga ustvarili, deloval po modelu, katerega zanesljivost limitira proti čisti ničli.

Kaj torej lahko naredimo, če imamo pred seboj baze in procedure, ki jih je treba prestaviti v sodoben čas in če te vsebujejo za 250.000 človek-dni dela, nujno potrebne podatke in vsebine, obenem pa vidimo, da se vsakič, ko poskušamo te podatke in vsebine migrirati v eno od trenutno modernih platform, zaletimo v zid? Eno od uresničljivih in ne pretirano zahtevnih metod bomo spoznali v nadaljevanju.

3. KAKŠEN JE TOREJ NAČRT, KI PRESEGA ENKRATNO UPORABO IZDELKA

Aplikativno stanje, ki smo ga želeli transformirati je v našem specifičnem primeru obsegal (poleg že omenjene količine podatkov) tudi večjo količino različnih aplikativnih modulov – tj. ločenih aplikacij na isti platformi IBM Domino.

Skladno z »zgodovinskimi dejstvi« so bile seveda omenjene ločene aplikacije razvite z izrazito monolitnim pristopom, kjer so se tudi (izrazito) mešali koncepti podatkovnega modela, poslovne logike ter uporabniškega vmesnika. Primer ene izmed teh aplikacij lahko prikažemo spodnjo sliko. Poleg že omenjenih arhitekturnih dejstev je razvidna tudi za današnji čas neprimerna uporabniška izkušnja.



Slika 1: Ena izmed obstoječih IBM Domino aplikacij

Cilji, ki smo jih naslovili z vzpostavitvijo nove platforme lahko strnemo v:

- izogniti se nepotrebni migraciji podatkov, ki ima v praksi večinoma manj kot 100% natančnost,
- obdržati obstoječo poslovno logiko v čim večjem obsegu (oz. v celoti),
- vzporedno delovanje obstoječih aplikacij ter nove platforme,
- izboljšati uporabniško izkušnjo.

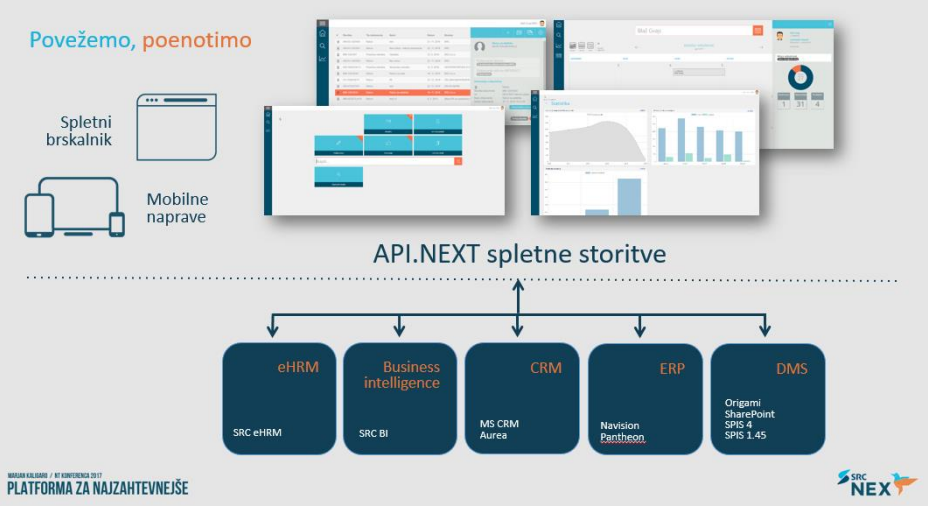
Pri doseganju ciljev pa seveda brez izzivov ne gre. Pri našem primeru smo tako naleteli na naslednja vprašanja, ki jih je bilo potrebno ustrezno nasloviti:

- zagotavljanje istega nivoja varnosti z vidika dostopa do podatkov,
- združevanje večih diverzificiranih virov podatkov v isti aplikaciji (npr. združevanje eHRM podatkov s podatki v dokumentnem sistemu, …),

- uporaba modernih tehnologij, ki omogočajo omni channel pristop na temelju starih/tradicionalnih pristopov k razvoju programske opreme.

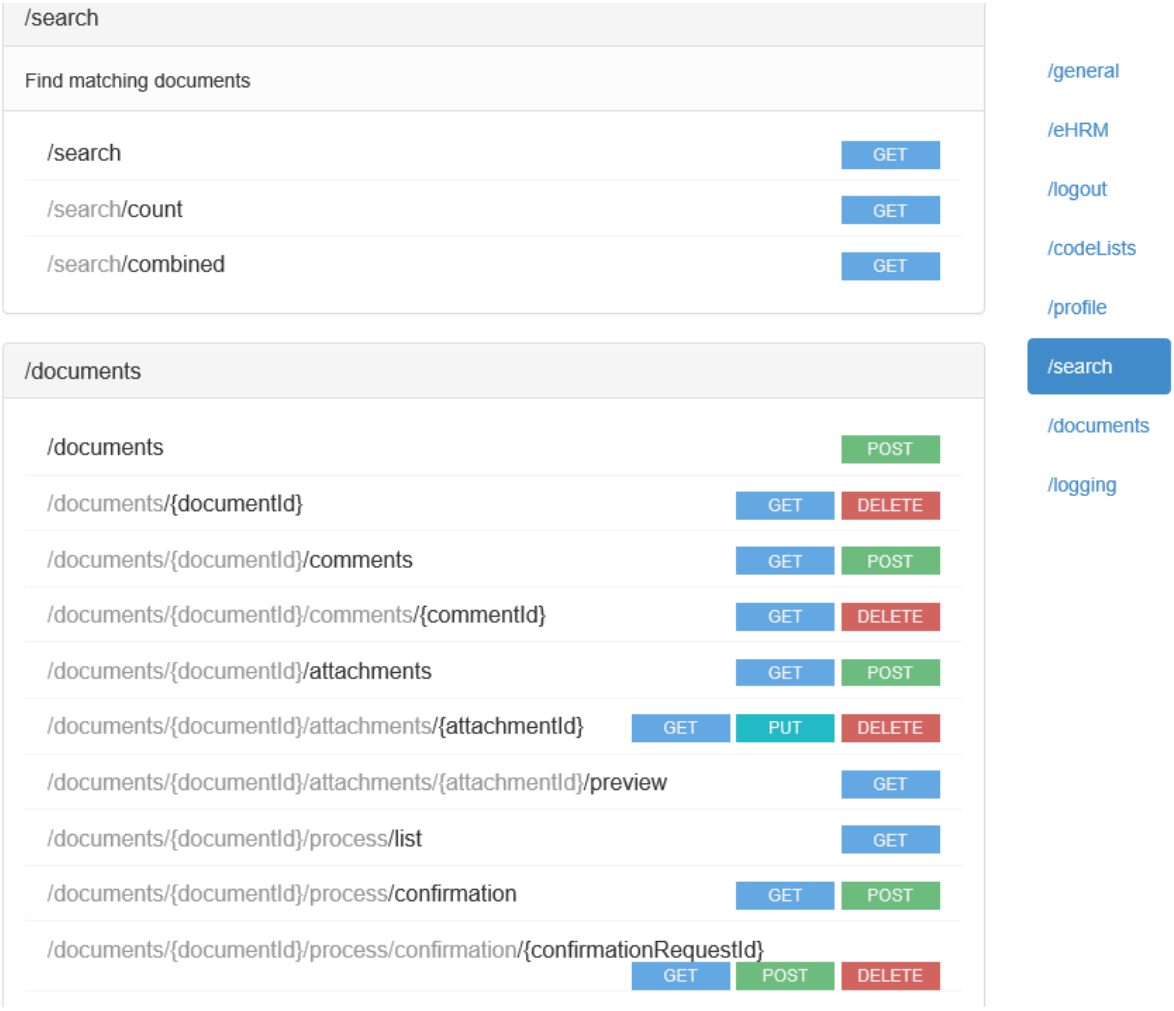
3.1. Kako smo digitalno transformirali poslovne procese, podprte s 3.000.000 vrstic kode

Z vidika arhitekture je tako nastala nova platforma, ki vsebuje tri nivoje (prikazano na sliki 2). Za nivo uporabniškega vmesnika smo uporabili Angular s čimer smo pridobili ustrezen nivo fleksibilnosti in zmožnost hitrega razvoja uporabniku prijaznega vmesnika.



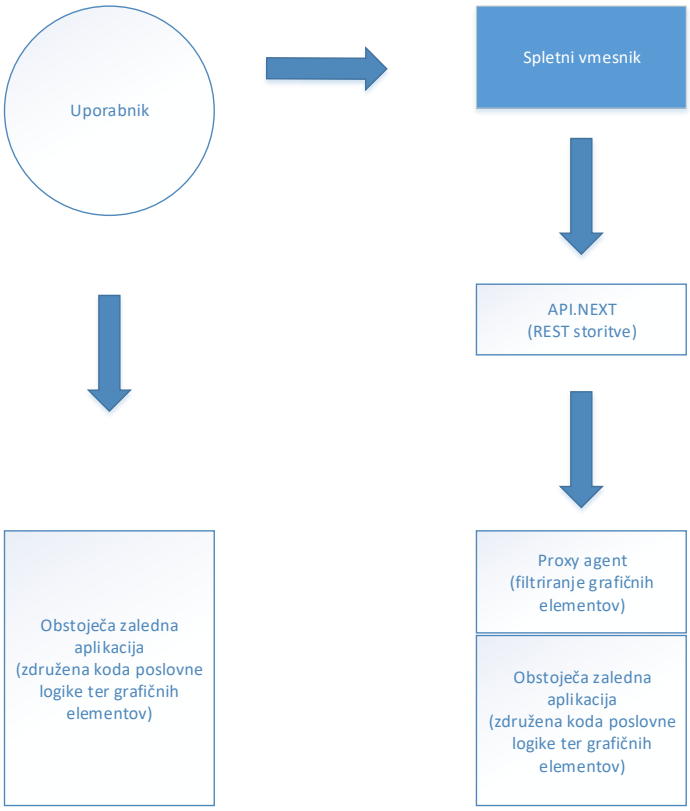
Slika 2: Ločitev obstoječe poslovne logike od uporabniškega vmesnika

Pod uporabniškem nivojem smo vzpostavili enotni REST nivo (slika 3), ki združuje storitve, ki komunicirajo s posameznimi zalednimi sistemi. Pri temu smo upoštevali dejstvo, da poslovni procesi lahko presegajo posamezno zaledno aplikacijo oz. bolje rečeno združujejo več teh (bodisi z vidika podatkov bodisi z vidika dejanskih akcij). S takšno ekstrakcijo smo zagotovili ustrezno enovitost akcij v uporabniškem vmesniku prav tako pa (vsaj s tehničnega vidika) dosegli še eno pomembno prednost – namreč na tak način lahko omogočimo relativno nebolečo migracijo s posamezne aplikacije za katero npr. proizvajalec nima ustreznega roadmap-a tehnološkega in/ali vsebinskega razvoja.



Slika 3: Izsek API.NEXT platforme (REST storitve)

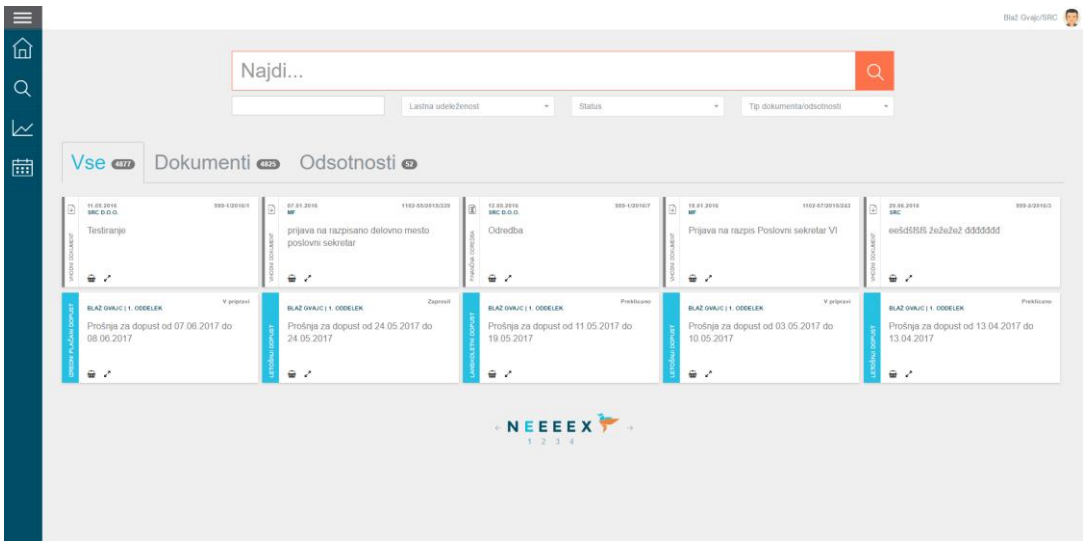
Dodaten izziv pri vzpostavitvi opisane arhitekture je bilo dejstvo, da posamezne aplikacije v obstoječi obliki vsebujejo »mešanico« programske logike in uporabniškega vmesnika (npr. pridobivanje odziva s strani uporabnika, prikaz raznih informacij v obliki pojavnih oken ipd.) kar seveda v primeru klicev teh funkcionalnosti preko spletne aplikacije ni najbolj primerno. Posledično smo na najnižjem nivoju (tj. v segmentu zalednih aplikacij) naredili dodaten abstrakcijski nivo (ti. proxy agenti), ki so klicani s strani REST storitev v API.NEXT nivoju ter nadalje ustrezno kličejo zaledne funkcije in obvladujejo morebitne grafične elemente, ki jih generirajo te zaledne funkcije (slika 4).



Slika 4: Abstrakcija zalednih funkcionalnosti

4. REZULTAT

V sklopu potrditve koncepta (ter posledičnega trženja rešitve) smo uspešno povezali v skupno uporabniško izkušnjo preko 5 različnih poslovnih aplikacij (s področja HRM, dokumentnega poslovanja, statistike ipd. – primer izgleda združenega modula HRM ter dokumentnega poslovanja je na sliki 5). S tem smo dokazali, da je možno »aplikacije starejšega datuma« ustrezno prenoviti in omogočiti uporabnikom modernejši in še pomembneje enostavnejši način rabe. Z ekonomskega vidika tako podaljšujemo življenjsko dobo investiciji v obstoječo aplikativno programsko opremo in dodatno omogočamo obstoj obstoječih podatkovnih zbirk neglede na njihovo obliko.



Slika 5: Prikaz vsebin iz HRM sistema, ki so združene s podatkovnim virom v DMS sistemu

5. LITERATURA

[1] Antikainen, J., & Pekkola, S. (2009). Factors influencing the alignment of SOA development with business objectives. *Proceedings of the 7th European Conference on Information Systems (ECIS 2009)* (str. 2579–2590). Verona: ECIS Standing Committee.

[2] Bell, M. (2008). *Service-Oriented Modeling (SOA): Service Analysis, Design, and Architecture*. Hoboken: Wiley & Sons.

[3] Chang, V., Kuo Y.-H., & Ramachandran, M. (2016). Cloud computing adoption framework: A security framework for business clouds. *Future Generation Computer Systems, 57*, 24–41.

[4] Cisco Systems, Inc. (2010). *White Paper: Planning the Migration of Enterprise Applications to the Cloud.* Najdeno 20. marca 2016 na spletnem naslovu https://www.cisco.com/en/US/services/ps2961/ps10364/ps10370/ps11104/Migration_of_Enterprise_Apps_toCloud_White_Paper.pdf

[5] Čiarniene, R., & Stankevičiüte, G. (2015). Theoretical Framework of E-Business Competitiveness. *Procedia Social and Behavioral Sciences, 213*(2015), 734–739.

[6] Haile, N., & Altmann, J. (2016). Value creation in software service platforms. *Future Generation Computer Systems, 55*, 495–509.

[7] Hustad, E., & De Lange, C. (2014). Service-oriented architecture projects in practice: A study of a shared document service implementation. *Procedia Technology, 16*(2014), 684–693.

[8] Kao, C. H., & Liu, S. T. (2013). Development of a Document Management System for Private Cloud Environment. *Procedia Social and Behavioral Sciences, 73*(2013), 424–429.

[9] SRC d.o.o. (2012). *SPIS 4 navodila* (interno gradivo). Ljubljana: SRC d.o.o.

[10] SRC d.o.o. (2015). Storitveno orientirana arhitektura (interno gradivo). Ljubljana: SRC d.o.o.

[11] SRC d.o.o. (2016). *SPIS NEXT* (interno gradivo). Ljubljana: SRC d.o.o.

Blockchain svetovanje in rešitve

- strokovna podpora pri načrtovanju in vrednotenju inovativnih poslovnih modelov
- prilagojene enodnevne ali večdnevne delavnice in usposabljanja
- strokovna podpora pri izvajanju pilotnih rešitev in dokazov o konceptu
- opredelitev in vrednotenje kakovosti inovativnih storitev temelječih na BC
- strokovna podpora pri presoji, odločanju in uvajanju strategije sprejemanja tehnologije BC
- strokovna pomoč pri analizi in izbiri najprimernejših BC tehnologij in platform

www blockchain-lab.um.si

02 220 7351

info@blockchain-lab.um.si

Blockchain Lab:UM

Part of European Blockchain Hub

All-in-One Blockchain Solutions

P O I N T . C L I C K . B L O C K C H A I N .

ARK provides users, developers, and startups with innovative blockchain technologies. We aim to create an entire ecosystem of linked chains and a virtual spiderweb of endless use-cases that make ARK highly flexible, adaptable, and scalable.

ARK is a secure platform designed for mass adoption and will deliver the services that consumers want and developers need.

FAST

Tired of staring endlessly at your wallet waiting for your transaction to clear?

With 8 second block times, ARK's network is one of the fastest in the industry.

OPEN SOURCE

Looking for a fully robust blockchain for your business or project? ARK is your solution for rapid blockchain development.

Our ecosystem and projects are completely open-source to help facilitate your needs to launch your very own SmartBridge compatible blockchain. Everything is hosted on GitHub for your ease and convenience.

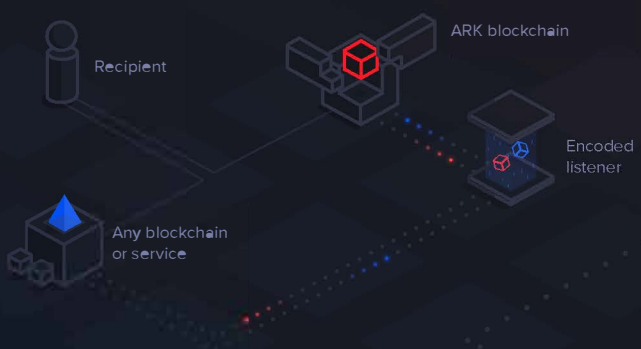
COLLABORATIVE

ARK isn't being built by one man, one team, or even in one country.

ARK is a truly global effort with 15 core team members from 11 different countries and an ever growing population of hardcore dedicated community developers, ARK is a collaborative effort in the true sense of the word.

BRIDGING

ARK bridges well known blockchains through the use of our custom SmartBridge technology, making an ecosystem of interconnected blockchains possible.



DECENTRALIZED

ARK utilizes a modified Delegated-Proof-Of-Stake (DPoS) consensus mechanism featuring 51 delegates.

These delegates are tasked with running the network and are rewarded with block rewards, much like miners in Bitcoin.

SCALABLE

Our primary goal with ARK is to keep the core blockchain lean and blazing fast.

Through the use of our custom built SmartBridge functionality we are able to off-load non-essential functions to hundreds of clone-chains. This allows for great scalability while keeping the main ARK blockchain lean and fast.

Alcad

Čas kognitivnega računalništva in umetne inteligence

INDUSTRIJA 4.0

Alcad d.o.o.

Zgornja Bistrica 4
2310 Slovenska Bistrica

T: 028055655
F: 028055665
E: info@alcad.si
S: www.alcad.si

Easy to work with

We have a positive can-do attitude, readily adapting to needs and circumstances – working with people, not just for them.

www.bearingpoint.com

BearingPoint®

bintegra

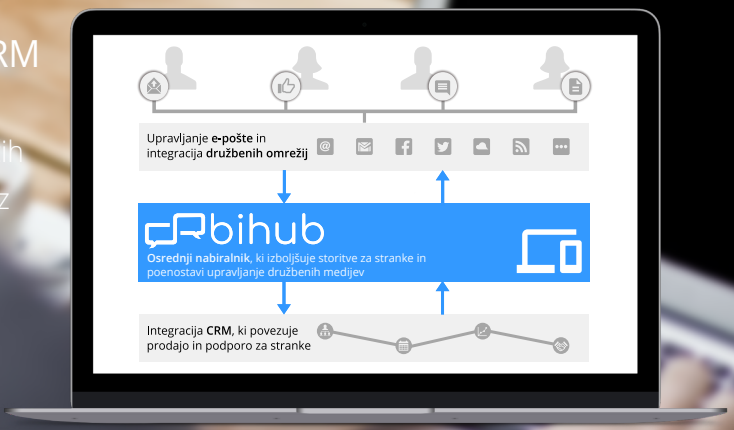
Svetovanje in IT rešitve

Integracija poslovnih aplikacij, podatkov in informacij

Pomagamo vam pri integraciji aplikacij, podatkov in informacij z namenom avtomatizacije procesov in izboljšanja zadovoljstva strank. Imamo bogate reference pri razvoju integracijskih rešitev tako s platformami proizvajalca TIBCO, kot z ogrodji temelječimi na odprtokodni Java arhitekturi.

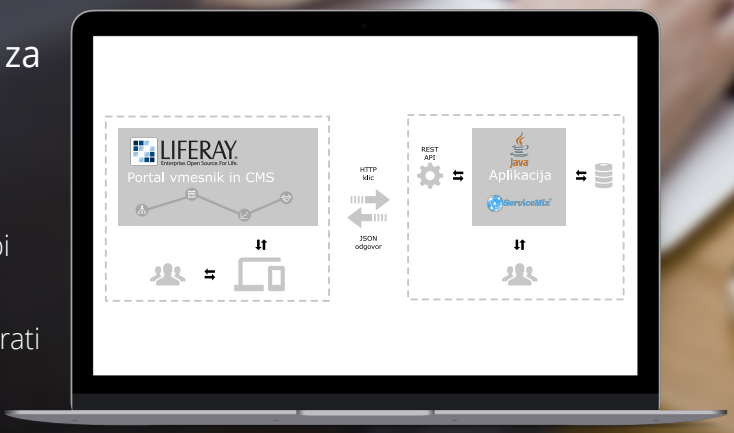
Upravljanje z družbenimi omrežji in CRM integracija

Družbena omrežja so eden izmed najhitreje rastočih kanalov, ki jih stranke uporabljajo za komunikacijo z znamkami. Z našo rešitvijo lahko poenostavite upravljanje z vašimi družbenimi omrežji, lažje sodelujete z vašo marketinško agencijo in se z njo integrirate z vašim CRM-jem.



Uporabniški portali in portalne rešitve za vaše stranke

Vaši uporabniki zdaj uporabljajo različne digitalne kanale in od vas pričakujejo štiriindvajseturno dosegljivost za interakcijo, sedem dni v tednu. Da bi izboljšali storitve za vaše stranke in integrirali marketinške iniciative, vam pomagamo implementirati spletne in mobilne rešitve.



Kontakt:

info@bintegra.com

+386 (0) 2 620 4861 /2

+386 (0) 2 620 4863

Glavna pisarna:

Železnikova 4

2000 Maribor

Slovenia

Pisarna v Ljubljani:

Stegne 23A

1000 Ljubljana

Slovenia

Game-Changing Technology

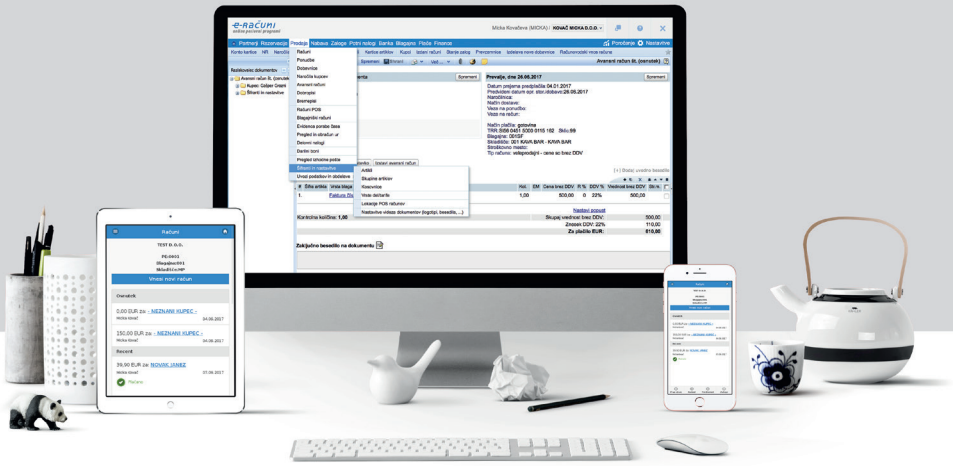
Behind every industry leader is an underlying technology that fuels its growth. Comtrade Gaming's open software is not just a solution, but a customized response to specific online and land-based business needs. With a focus on operators, vendors and regulators, its objective is more freedom with less complexity.

Your business. Ahead of the game.

COMTRADE GAMING

www.comtradegaming.com

Platforms. Systems. Games.

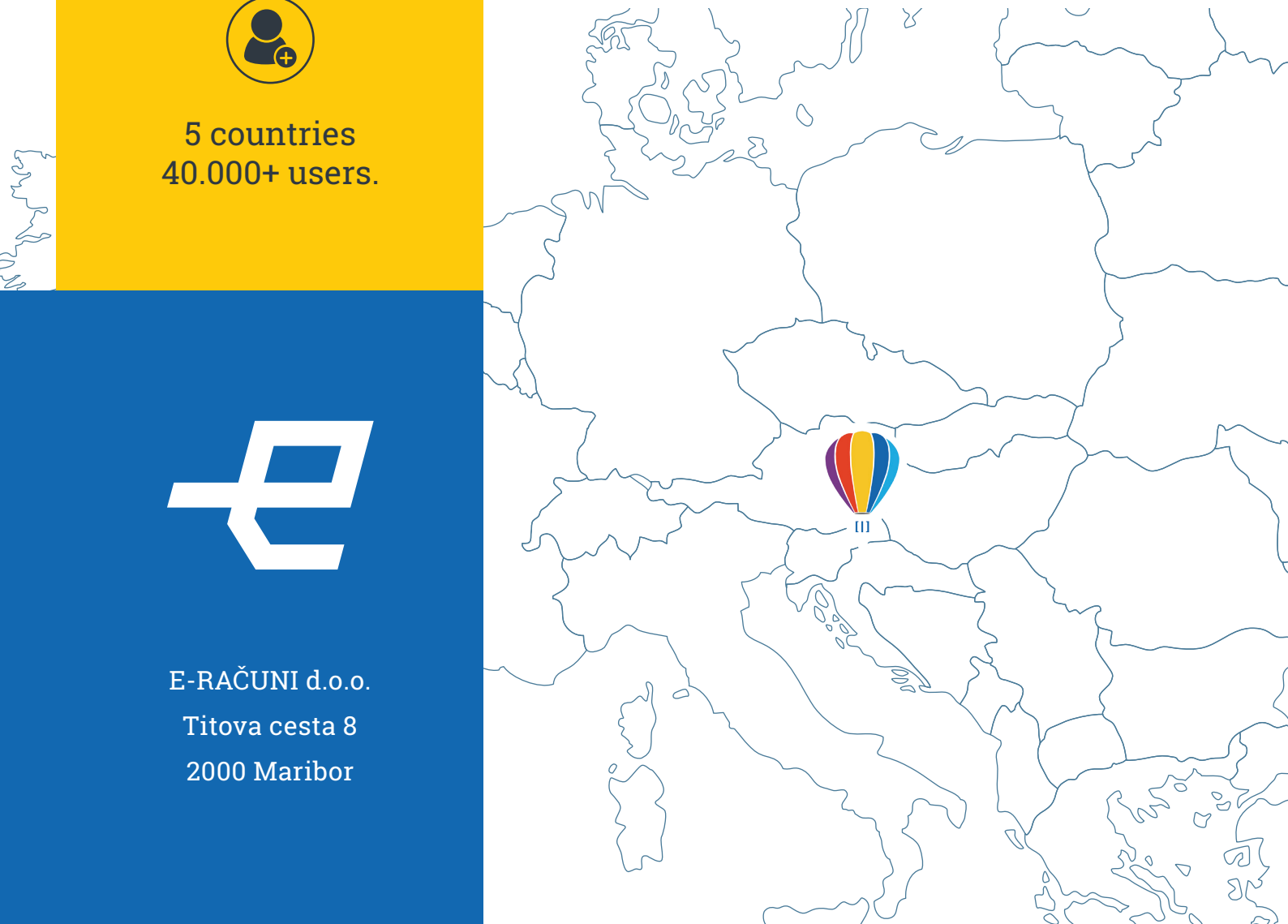


Cloud based ERP system
for Central Europe.

www.e-racuni.com
www.eurofaktura.com

Powered by Smalltalk objects since 2002.

5 countries
40.000+ users.



E-RAČUNI d.o.o.
Titova cesta 8
2000 Maribor

RC IRC Celje, d.o.o.

Ul. XIV. divizije 14, 3000 Celje

5. DESETLETJE SOUSTVARJAMO INFORMACIJSKO DOBO

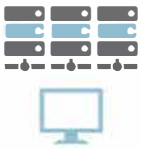
- Informacijski sistemi za zdravstvene ustanove
- Informacijski sistemi za celovito podporo poslovnih procesov
- Prenova poslovnih procesov in racionalizacija poslovanja
- Informacijska podpora za poslovno odločanje in upravljanje
- Certifikata kakovosti informacijske tehnologije in informacijske varnosti
- Storitve svetovanja pri uvedbi standardov kakovosti in informacijske varnosti
- Poslovno in informacijsko svetovanje
- Celovito vzdrževanje informacijskih sistemov

Poslovna področja

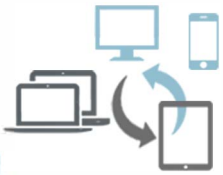
- Zdravstvo
- Televizija
- Visoko šolstvo
- Telekomunikacije
- Industrija
- Banke

Informacijska cesta, ki povezuje slovenske zgodbe o uspehu.

Programske rešitve



Razvoj programske opreme



Vzdrževanje in svetovanje

Informacijska varnost

Inovativnost v službi uspešnosti

RC IRC d.o.o.
Ulica XIV. divizije 14
3000 Celje

T. +386(0)3 427 42 00
F. +386(0)3 427 41 98
E. info@rcc-irc-si
W. www.rcc-irc.si

ISO 9001/TickIT BUREAU VERITAS Certification

ISO 27001 BUREAU VERITAS Certification

Stopi v svet digitalnih rešitev

Si želiš **soustvarjati** uspešne tehnološke zgodbe?

Razvoj spletnih rešitev

- Načrtovanje spletnih in mobilnih aplikacij
- Avtomatizacija kompleksnih procesov
- Reševanje na videz nerešljivih problemov
- Izboljševanje procesov z blockchain tehnologijo

Z mlado ekipo in edinstvenim znanjem pospremi stranke **v korak s časom** in jih postavi ob bok vodilnim podjetjem.

Z nami kompleksno postane učinkovito.
Že vse od leta 1993.

Kivi Com d.o.o.
Slovenj Gradec, Ljubljana

www.kivi.si

02 88 39 400

info@kivi.si

40 LET BISTRIH REŠITEV

KOPA

Razvoj in rešitve za digitalno transformacijo

Učinkovito koriščenje virov in obvladovanje stroškov podjetja

Planiranje, izvedba, spremljanje in ukrepanje.
KOPA ERP, KOPA HRM, PREACTOR, MAXIMO, UTRIP.PRO, ASOS, KOPA BI

Poslovno obveščanje

IKT infrastruktura

Napredni kadrovski management

Zagotavljanje kompetenc v celotni verigi poslovanja. Brezpapirna kadrovska.
KOPA HRM in PLAČE

Varnost in zanesljivost

Brezpapirno poslovanje

Digitalizacija dokumentov in procesov.
UREJENO.DOK, ODIP

Vzdrževanje sredstev

Računalništvo v oblaku

Rešitve kot storitev.
CLOUD DMS, SAS Utrip.PRO

Razvoj za konkurenčno prednost

Svetovanje in pomoč pri razvoju na področju ORACLE tehnologij.
ORASTOR

KOPA d. d.,
Kidričeva 14, 2380 Slovenj Gradec
t: 02 88 39 700, e: info@kopa.si, s: www.kopa.si

.expertise .innovation .partnership

msg life

Symass

informacijske rešitve za zavarovalnice in zavarovalne agente

upravljanje prodaje in zalednih poslovnih procesov

rešitve za mobilne naprave

spletne storitve

agilen razvoj

tradicija in kakovost

Kontakt:
msg life odateam d.o.o.
Titova cesta 8
SI-2000 Maribor
tel.: +386 (2) 2356 200
e-pošta: andrej.kline@msg-life.com

www.msg-life.si

Find your perfect place to work!

#WEARE

CashbackWorld
CashbackSolutions
GreenfinityFoundation
ChildandFamilyFoundation

myworld-solutions.com

NOVUM

www.novum-online.si

ENTERPRISE JAVA APPLICATION DEVELOPMENT

AGILE SOFTWARE DEVELOPMENT

INTERNATIONAL PROJECTS

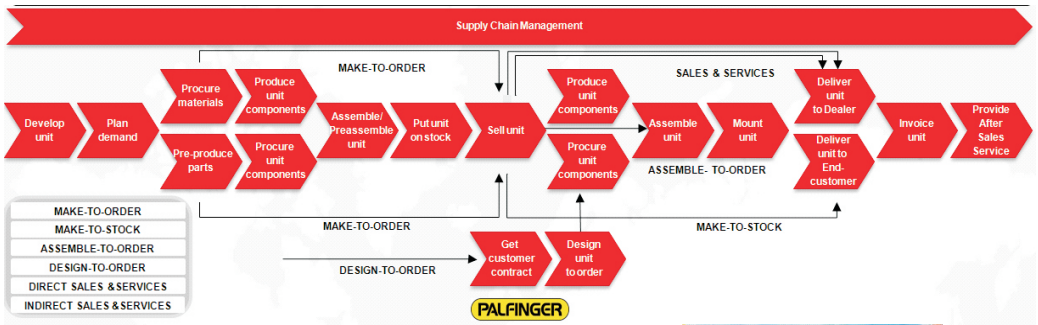
IT SOLUTIONS FOR INSURANCE BUSINESS

CLOUD SERVICES FOR INSURERS

Illnau Köln Maribor Nürnberg Salzburg Wien

RAZVOJNI ODDELEK V MARIBORU

VAS ZANIMA DELO V MEDNARODNEM OKOLJU?

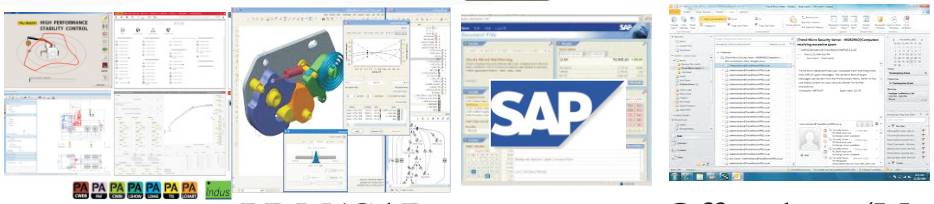


PALFINGER

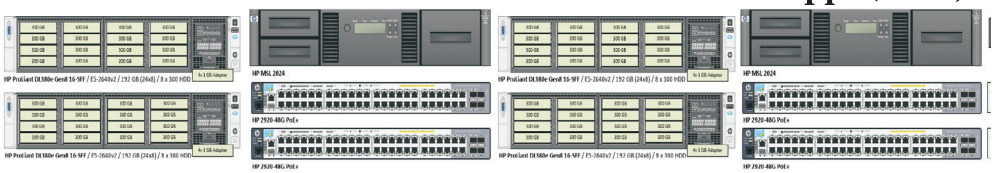
Business Model

Procesess

Aplications/ Systems

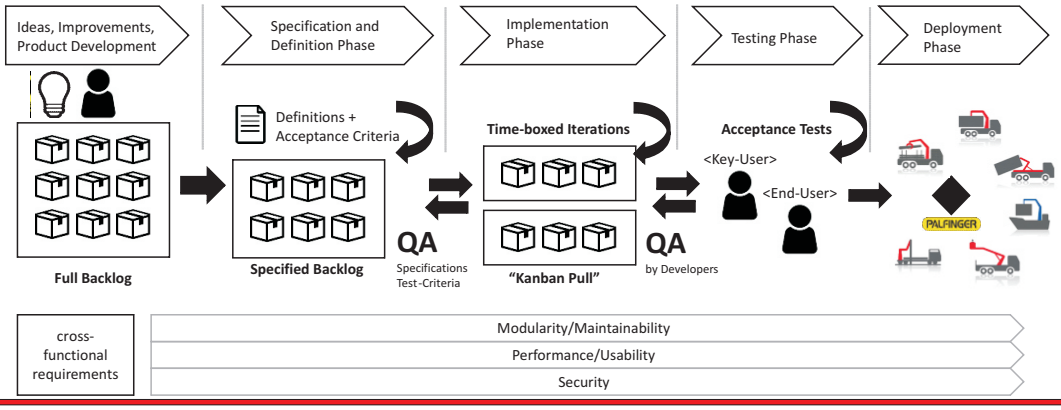


PALSoft PDM/CAD ERP Office Apps (Mail, Lync,...)



Infrastructure

- **Organization**
  Software development at PALFINGER is centrally managed as Competence Center to bundle and concentrate required expert Know-How for the whole group. We develop Software supporting PALFINGER products to gain and secure competitive advantages based on customer needs
- **Methodology**
  Modern methods of Software Development utilizing agile principles like Scrum and Kanban
  Achieve quality and maintainability by "test-driven development"
  Business impact through usable software; limit WIP of unfinished/untested software components
  View of the whole Software Development Lifecycle along the process chain



- **PLM/CAX**
  Rollout of PDM system to all locations with own Engineering departments based on best practice processes
  CAD standardization as long term goal to lever synergies of joint development projects overall the Business Unit boundaries based on best practice processes
  Provide global support structure (1st level, 2nd level, 3rd level) – local Keyuser structure as prerequisite
  System of choice for PLM for the whole group is PDM Link; for 3D CAD is Creo; for 2D CAD is Medusa
- **PALSoft**
  Software Suite enlargement to all relevant product groups / Rollout to all relevant Business Areas / Business Units
  Software Suite enrichment along with Mechatronics/Product Development/Engineering
  Develop new business functions through extensive data gathering (e.g.: predictive maintenance, fleet management, …)
  Provide global support structure (1st level, 2nd level, 3rd level) – external clients to be considered!
  Technology platform of choice is Microsoft .net
  System of choice for diagnostics and configuration is Paldiag; for mounting calculations is Pacwin; for engineering calculation is Indus
- **PALDesk**
  Consolidation/Integration of existing applications within a common, role-based user interface
  Develop new applications to support new business processes (direct sales, Leasing, …)
  Standardization and Optimization of sales processes through product configurator, order management and CRM
  Rollout to all relevant Business Areas and Business Units – including dealers and sub-dealers
  Provide global support structure (1st level, 3rd level) – 2nd level to be provided by.(local) Service/Marketing/Sales; external clients to be considered!
  Internal information, collaboration and workflow creation using Microsoft technology

Kontakt: a.varda@palfinger.com

TriDens

INNOVATIVE IT SOLUTIONS

Corporate

Zdravo in **srečno podjetje**

Učinkovit načrt promocije zdravja pri delu

Ozaveščanje

12 navad zelo zdravih ljudi®

Celovit program, namenjen trajni usvojitvi pomembnih navad za vsa življenjska obdobja. Program se osredotoča promocijo vedenj, ki so povezana s zdravim in kvalitetnim življenjem, kot so telesna aktivnosti, odpuščanje, smeh, soočanje z razvadami, zadostno spanje.

Višanje odpornosti na stres

Mayo Clinic Healthy Living Resilient Mind

Zvišajte psihično trdnost vaših zaposlenih v le nekaj minutah na dan z zabavnimi in znanstveno dokazanimi strokovnimi tehnikami. Resilient Mind program klinike Mayo je strokoven program podprt z več kot 20 raziskavami in dokazano zmanjša stres, okrepi čuječnost, izboljša dobro počutje in poveča občutek sreče.

Obvladovanje primerne telesne mase

Mayo Clinic Diet

Skupino zaposlenih, ki se sooča s prekomerno telesno maso, debelostjo ali nezdravimi prehranskimi navadami, lahko vključimo v program Mayo Clinic Diet. Le-ta uči, kako oblikovati zdrave prehranske navade, kako postopoma izgubiti odvečno telesno težo ter kako trajnostno vzdrževati primerno telesno maso in se s tem izogniti tveganju za razvoj kroničnih bolezni povezanih z debelostjo.

35%
Zmanjšanje stresa

25%
Izboljšanje psihične odpornosti

39%
Zmanjšanje izgorelosti

Za več nformacij nas kontaktirajte na:
corporate@24alife.com

Developed in collaboration with MAYO CLINIC

www.24alife.com

3fs DIGITAL PRODUCT STUDIO

We launch independent ventures, consult established companies and build the community that makes it all possible.

3fs.si
3fs.si/jobs

Računalniške novice

www.racunalniske-novice.com

Posebna ponudba za obiskovalce konference OTS 2018

12 ŠTEVILK revije Računalniške novice BREZPLAČNO!

Naročite lahko na: narocnine@stromboli.si 01 620 88 00

Želim prejeti 12 številk Računalniških novic BREZPLAČNO.

* Plačam le stroške pošiljanja, ki znašajo 9,70 € (z DDV).

Poštnina plačana po pogodbi št.: 1039/1/S

Priimek in ime: _____

Naslov: _____

Pošta in poštna številka: _____

Telefon/GSM: _____

E-pošta: _____

Računalniške novice
Cesta komandanta Staneta 4A

1215 Medvode

Ob prejemu tega naročila vam bomo izdali ponudbo za stroške pošiljanja, ki znašajo 9,70 € in vas vnesli v bazo naročnikov. Ponudba NE velja za podaljšanje naročnine oz. naročnike, ki so bili naročeni na revijo v zadnjih 2 letih.

Po preteku 12ih številk, vam bomo poslali ponudbo za podaljšanje naročnine (1 leto).
V primeru, da naročnine ne želite podaljšati, jo lahko pisno odjavite.

Za dodatna vprašanja se lahko obrnete na telefon: 01 620 88 00 ali po e pošti: narocnine@stromboli.si.

Ponudba velja do 31. 7. 2018.

S5tehnika.net

vizualne komunikacije
in založba d.o.o.

Sostrska cesta 43C
1261 Ljubljana Dobrunje
t. 059 010 952 • 040 423 302
www.s5tehnika.net • stik@s5tehnika.net

Podjetje S5TEHNIKA.net d.o.o. je založba tehnične literature, predvsem preiodičnih revij AVTOMATIKA in ELEKTRONIKA ter HAMtech Report.

Druga dejavnost je spletna prodaja ter razvoj in servis elektronske opreme, reševanje problemov tam, kjer klasični industrijski moduli odpovedo ali ne obstajajo. Specializirani smo tudi za razvoj programske opreme na Linux operacijskem sistemu na vseh nivojih!

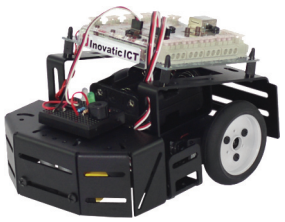
Revija avtomatika
moderne tehnologije • sodobni pristopi
učinkovite rešitve • profesionalna elektronika
+ELEKTRONIKA

HAMtech Report
Revija za popularno elektroniko in radijske komunikacije

HW-SW
ELEKTRONIKA
elektronika.tehnocenter.eu

HAMtech Video Nadzor

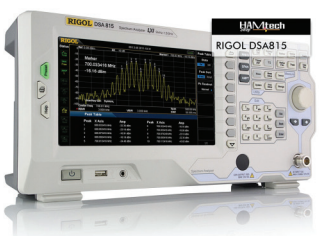
HAMtech Communications



EMoRo creative learning

Spletna trgovina **HAMtech.eu** podjetja S5TEHNIKA.net d.o.o. ima na voljo:

- vse vrste komunikacijske opreme za službene komunikacije, navtiko, letalstvo in radioamaterje
- izobraževalne elektronske sestavljanke (popularno **KIT kompleti**) z vajami za kreativno učenje skozi igro,
- učne robote **EMoRo** (emoro.hamtech.eu) z malo šolo robotike. Učni roboti omogočajo tako učenje programiranja mikrokrmilnikov (flow chart, Basic, AVR basic, C,...) kot preučevanje mobilnih robotov, izvajanje praktičnih nalog, kar ga uvršča med didaktične učne pripomočke!
- merilno opremo **RIGOL**, **RigExpert**,...



shop.hamtech.eu

ZNANOST IN ZGODOVINA ZA VSAKOGAR

LETNA NAROČNINA
CENA ZA 12 ŠTEVILK
~~54 €~~
-20 % POPUST
43,20€

Naročite na:

www.revija-history.si | www.sil.si
02 23 53 326 | revije@vecer.com

Večer Revije, d.o.o., Ulica slovenske osamosvojitve 2, 2000 Maribor

RADIO CITY

MARIBOR 100,6 | LJUBLJANA 99,5 | CELJE 100,8
VELENJE 100,8 | M. SOBOTA 100,6 | KRANJ 99,5

Ni nam lahko

www.radiocity.si

GENERALNI POKROVITELJ

inova iT

POKROVITELJI

ARK.io

Alcad

BearingPoint

bintegra
Svetovanje in IT rešitve

COMTRADE GAMING

e-racuni.com

irC

KIVI

40 LET BISTRIH REŠITEV KOPA

HOLDING MIKROpis

msg life ODATEAM

myWorld

NOVUM

PALFINGER

PS POSITA

RAZVOJNI CENTER IKTS ŽALEC

TRIDENS
INNOVATIVE IT SOLUTIONS

3fs

MEDIJSKI POKROVITELJI

Računalniške novice
www.racunalniske-novice.com

Revija avtomatika
moderne tehnologije • sodobni pristopi • učinkovite rešitve

VEČER

Radio CITY 100,8 99,5 100,8 RADIO CITY

ISBN 978-961-286-162-9
9 789612 861629

INŠTITUT ZA INFORMATIKO