ZBORNIK RADOVA SA KONFERENCIJA

KOM 2019

ELEKTRONIČKE KOMUNIKACIJSKE TEHNOLOGIJE
I NORME

CASE 2020

RAZVOJ POSLOVNIH I INFORMATIČKIH SUSTAVA

2019 / 2020

KOM 2019

ELEKTRONIČKE KOMUNIKACIJSKE TEHNOLOGIJE I NORME

25.11. - 26.11.2019

Zlatni partner

citus grupa
Internet Development Company

Brončani partneri

HAKOM

CISCO

IEEE Croatia Section

Mreže budućnosti projektiramo već danas

Projektna rješenja koja daju krila Vašoj viziji

Projektiranje

Instalacija

netiks

Održavanje

Edukacija

Prestignite konkurenciju...

NETIKS d.o.o. Sjepana Podhorskog 15 (Dugave), 10000 Zagreb, Tel: 01/6652 940, Fax: 01/6652 902 www.netiks.hr

ORGANIZATOR

CASE d.o.o.

ORGANIZACIJSKI I PROGRAMSKI ODBOR

Goran Belamarić
Ante Polonijo
Mislav Polonijo

ZLATNI PARTNER

BRONČANI PARTNERI

Izdavač, priprema i tisak:
CASE d.o.o., Rijeka

Urednik:
Mislav Polonijo

ISSN 1334-4463
UDK 007.5 : 621.39 : 681.324

Copyright ©"Case", Rijeka, 2019

Sva prava pridržana. Niti jedan dio zbornika ne smije se reproducirati u bilo kojem obliku ili na bilo koji način, niti pohranjivati u bazu podataka bez prethodnog pismenog dopuštenja izdavača, osim u slučajevima kratkih navoda u stručnim člancima. Izrada kopija bilo kojeg dijela zbornika zabranjena je.

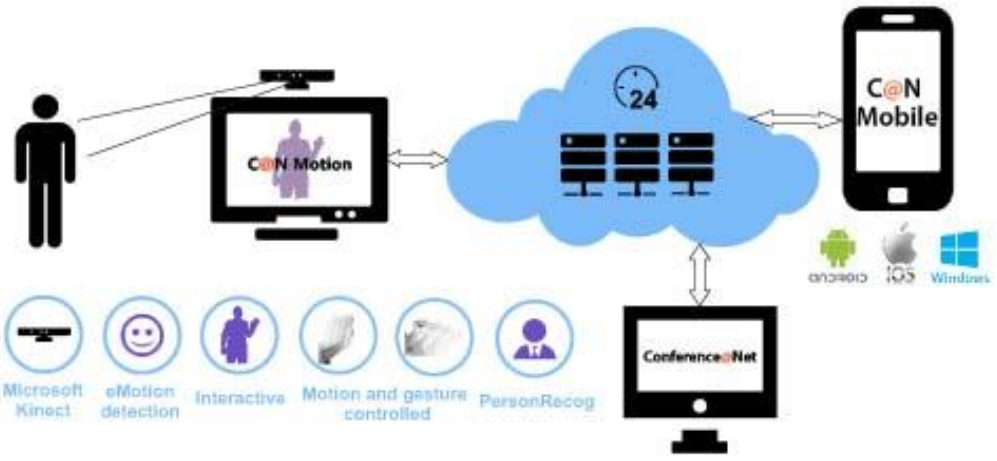
Case d.o.o., Antuna Barca 12, 51000 Rijeka
tel: 051/217-875, fax: 051/218-043, e-mail: case@case.hr, internet: www.case.hr

citus grupa
Internet Development Company

Conference@Net

inovativno rješenje za
upravljanje organizacijom konferencija
integrirano s društvenim mrežama i dostupno na uređajima upravljanim dodirom

www.conferenceatnet.com



Microsoft Partner
Silver Application Development
Silver Data Platform
Silver Midmarket Solution Provider
Silver Mobility
Silver Small Business
Cloud Accelerate

CITUS d.o.o.
Dragutina Golika 63, Zagreb
http://www.citus.hr
citus@citus.hr

SADRŽAJ

dr.sc. Winton Afrić:
ZNAČAJ RADIJSKIH VEZA U RAZVOJU ŠIROKOPOJASNE PRISTUPNE MREŽE 1.5

Roko Rogulj, Ivo Kovačević, dr.sc. Tonko Kovačević:
PRIMJENA ANT + TEHNOLOGIJE 1.13

dr. sc. Dina Šimunić
5G I ZDRAVLJE 1.23

ZNAČAJ RADIJSKIH VEZA U RAZVOJU ŠIROKOPOJASNE PRISTUPNE MREŽE

THE IMPORTANCE OF RADIO CONNECTIONS IN THE DEVELOPMENT OF BROADBAND ACCESS NETWORK

dr.sc. Winton Afrić

SAŽETAK:

U prvom djelu rada govori se o planovima Europske komisije za digitalizaciju europskog društva, te o ulogzi širokopojasne mreže kao tehnološke osnove na kojoj se proces digitalizacije ostvaruje. Nakon toga o Stanju, planovi i aktivnosti u daljnjem razvoju širokopojasne pristupne mreže u EU i u Hrvatskoj. U završnom djelu rada govori se o značaju i mjestu radijskih komunikacija u daljnjem razvoju širokopojasne mreže, diskutiraju se privremena i dugoročna razvojna rješenja koja koriste radijske tehnologije. Razvojne strategije digitalizacije Hrvatskog društva i mjesto radijskih komunikacija u njima. Na kraju je dan zaključak.

ABSTRACT:

Introduction. Plans of the European Commission for the digitization of European society, and the role of the broadband network as the technological basis on which the digitization process is being pursued. Status, plans and activities for the further development of the broadband access network in the EU and in Croatia. The importance and place of radio communications in the further development of broadband, temporary and long-term development solutions using radio technologies. Development strategies of the digitalization of the Croatian Society and the place of radio communications in them. Conclusion.

1. UVOD

Telekomunikacije kao infrastrukturna osnova digitalizacije društva u posljednjih dvadeset godina doživljavaju snažan razvoj u tehnološkom i količinskom smislu. Razvoj telekomunikacijskih tehnologija nije slučajan, on se temelji na razvojnim planovima tadašnjeg CCITT-a početkom osamdesetih godina prošlog stoljeća (Preporuka G 702 CCITT-a od 1982. godine). Tada se je odlučilo da će telekomunikacijske mreže budućnosti biti, digitalne, multiuslužne i da će se osnivati na prospajanju pakete. Sva tehnološka rješenja koja imamo danas nisu se tada mogla sagledati, te osnovne planske postavke nadograđivale su se i modificirale tijekom vremena. Krajem prošlog stoljeća već se je govorilo o informatičkom društvu budućnosti i o komunikacijskim sustavima kao tehnološkoj osnovici na kojoj će se temeljiti društvo znanja. U zadnjih četrdeset godina procesi izgradnje i trajne transformacije telekomunikacijskih sustava bio je buran i intenzivan.

Europska komisija sagledava dalji ukupni razvoj europskog društva kroz procese digitalne transformacije u svim društvenim sferama. Taj proces ne bi bio moguć bez tehnološke osnove na kojoj se temelji, a to su; širokopojasna telekomunikacijska infrastruktura i daljnji razvoj računalnih centara koje su njen integralni dio.

Sagledavajući telekomunikacijsku mrežu sadašnjosti i budućnosti u pristupnoj i transportnoj razini u posljednjem desetljeću vidljivi su procesi naglog rasta pristupnih brzina. Ovaj porast potaknut je sve zahtjevnijim aplikacijama i uslugama, naglim rastom potreba za sve većim informacijskim volumenom na transportnoj razini i vrlo snažnim i brzim razvojem informatičkih centara.

Kao rješenje za realizaciju fizičkog prijenosnog medija velikog kapaciteta nametnula se je svjetlovodna tehnologija, jedino ona može u cijelosti zadovoljiti potrebe za informacijskim volumenom kako u pristupnoj tako i u transportnoj razini mreže. Galvanska (bakrena) infrastruktura koja je još prisutna u širokoj primjeni u pristupnoj mreži dosegnula je svoj maksimum mogućnosti prijenosa i nije u stanju odgovoriti ni postojećim, a kamo li budućim zahtjevima. Totalni prijelaz sa bakra na optiku već se odavno desio u transportnoj razini mreže, a u pristupnoj se očekuje u narednim decenijama.

Pitanje na kojem želimo odgovoriti u ovom članku je kakova je uloga radijskih tehnologija u budućoj širokopojasnoj mreži, posebno u njenom pristupnom dijelu?

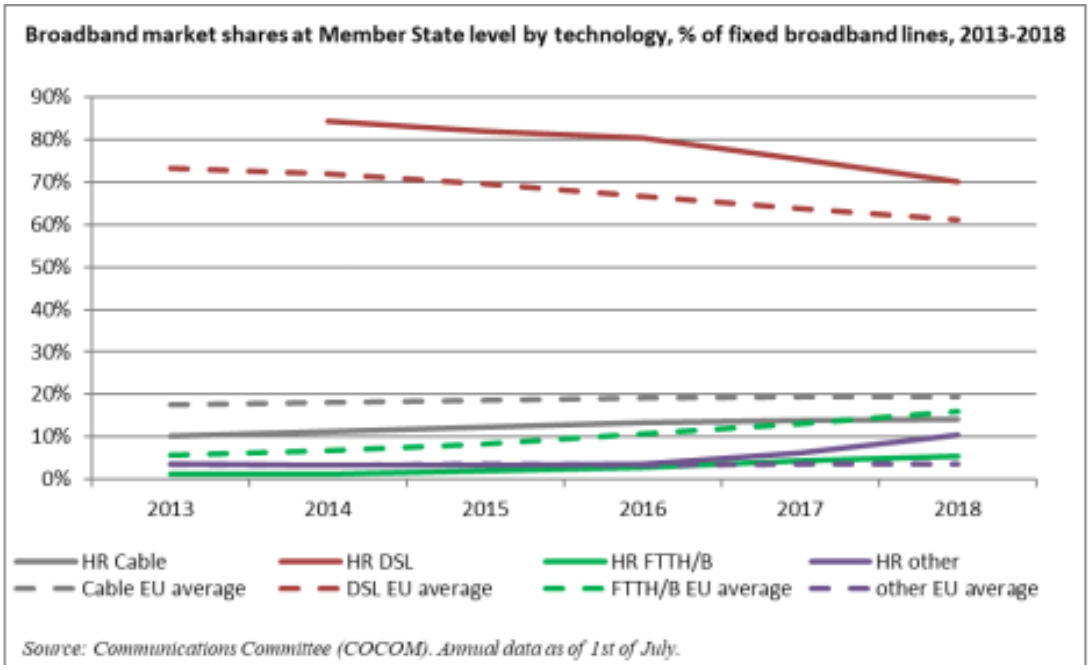
Uloga i značaj radijskih tehnologija u telekomunikacijskoj mreži mijenjala se od kad te mreže postoje i to onom brzinom kojom su se te mreže mijenjale. Prije pola stoljeća dok još nije bilo svjetlovodnih vlakana u komercijalnoj upotrebi, a sustavi bili analogni, dominantnu potrebu za prijenos informacijskog volumena određivala je analogna telefonska komunikacija. U to vrijeme sustavi usmjerenih radijskih mreža činili su okosnicu nacionalne komunikacijske mreže. Danas, mogućnosti prijenosa usmjerenim radijskim veza nisu usporedive sa onim što pruža svjetlovod, radio prijenos postupno gubi značaj na magistralnoj razini mreže. Usmjerene radijske veze ostaju kao skromna alternativa, ograničenih mogućnosti u slučaju većih katastrofa na optičkim komunikacijskim linijama. Kada govorimo o usmjerenim

radijskim vezama one još uvijek nalaze primjenu u razinama pristupne mreže, kao alternativno i privremeno rješenje dok se ne postave svjetlovodi. Radijske tehnologije punu svoju afirmaciju sve više ostvaruju u krajnjem širokopojasnom pristupu, tu razvoj ide u dva pravca koji postupno konvergiraju i u budućnosti će se stopiti u jedan. Jedan od tih smjerova je razvoj mobilnih komunikacijskih sustava koji pružaju raspoloživi kapacitet i osiguravaju globalnu mobilnost. Drugi od tih smjerova su bežični sustavi koji fiksnim korisnicima omogućavaju prostorno ograničenu pokretljivost. (Poput Wireless mreža koje omogućavaju u stanovima bežično povezivanje, laptopa, tableta, mobitela, a od nedavno i televizora na kućni fiksni priključak, eliminirajući potrebu za velikim brojem kabela razasutih po stanovima).

Područje telekomunikacija u razvojnom smislu je dinamično područje zato da bi smo pronašli odgovor na gore postavljeno pitanje prvo moramo kazati što nas očekuje u skoroj budućnosti, to možemo pročitati iz materijala Evropske komisije koji govore o daljnjim razvojnim planovima u periodu od 2020. do 2030. na prostoru EU.

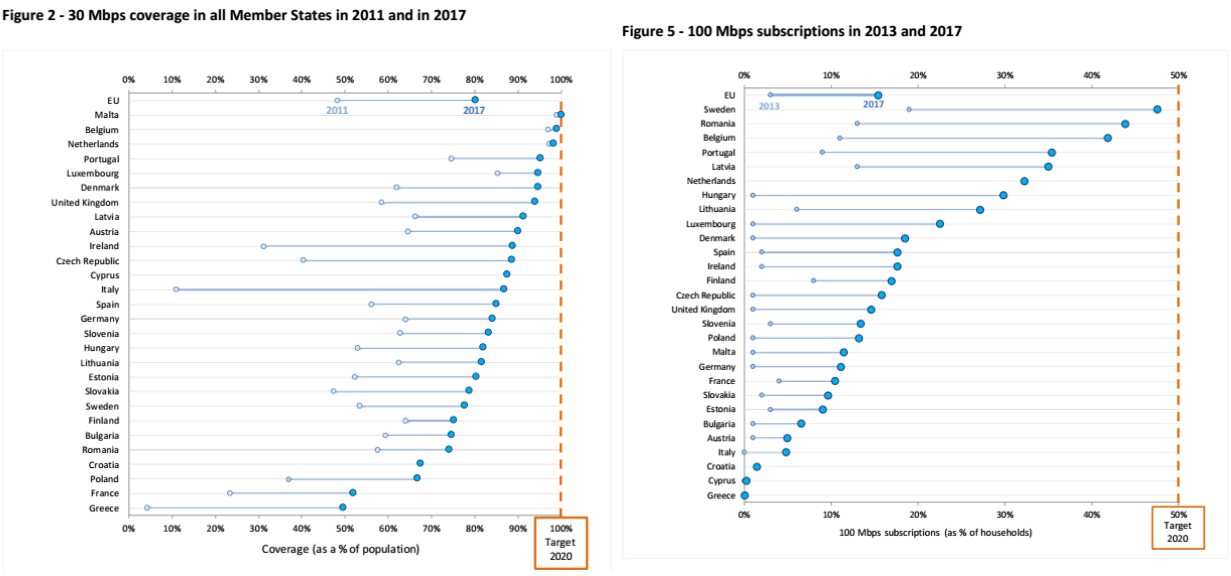
2. PLANOVI EUROPSKE KOMISIJE ZA DIGITALIZACIJU EUROPSKOG DRUŠTVA

Kroz digitalnu transformaciju Europskog društva Europska komisija sagledava daljnji gospodarski rast EU, ali i cjelokupni razvoj društvene zajednice. Digitalizacija kao proces zahvaća sve segmente Europskog društva. Digitalizacija je proces društvene transformacije koji nije moguć bez tehnološke osnove na kojoj počiva. Jedan od segmenata tehnološke osnove je i širokopojasna mreža, u svom transportnom i u svom pristupnom dijelu. Europska komisija kroz plan razvoja širokopojasne mreže za period 2010.-2020. godina predvidjela je da svaki građanin EU bez obzira na mjesto na kojem se nalazi ostvaruje minimalnu pristupnu brzinu od 100 Mbit/s a da barem 50% priključaka ostvaruju brzinu od 1 Gbit/s. Ovakve pristupne brzine moguće je ostvariti dovođenjem svjetlovoda do svakog domaćinstva ili do svake točke u kojoj nam je potrebna spoj na širokopojasnu mrežu. Kako smo na kraju ovog planskog perioda, možemo kazati da se ovako ambiciozno postavljen plan sa visokim cjelivam nije i neće niti u jednoj članici EU u cijelosti ostvariti. Neke su se zemlje realizaciji tog plana približile više, a neke manje. Republika Hrvatska se u realizaciji tog plana nalazi vrlo nisko, na pred posljednjem mjestu svih članica EU, iza Republike Hrvatske trenutačno se nalaze Cipar i Grčka. Na slici 1. prikazana je usporedba penetracije različitih pristupnih tehnologija između EU prosjeka i stanja u RH kroz vrijeme.



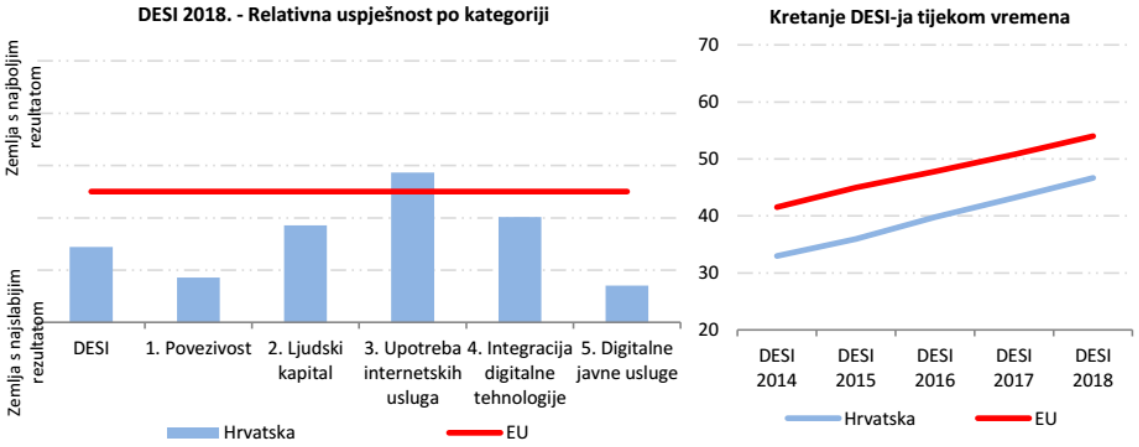
Slika 1. Usporedba penetracije pristupnih tehnologija u Republici hrvatskoj sa prosjekom EU.[1]

Iz slike 1, je razvidno da Republika Hrvatska slijedi Europske trendove, ali s značajnim zakašnjenjem. DSL pristupne tehnologije u RH zastupljenije se nego što je prosjek EU za 10%, trend umanjenja podudara se s EU trendom, čak se dešava i nešto brže što je dobar pokazatelj. Međutim, kada govorimo o optičkim pristupnim tehnologijama onda RH se nalazi na svega 5% penetracije dok je EU prosjek 15%, a trend brzine porata EU prosjeka je veći nego trend brzine porasta u RH, što nije dobar razvojni pokazatelj.



Slika 2. Usporedba stanja širokopojasnih priključaka za zemlje EU sa pristupnim brzinama od 30 i 100 Mbit/s

Po DESI pokazateljima Republika Hrvatska pokazuje kroz godine postupni rast i danas se nalazimo na 22 mjestu od 28 zemalja, što nije osobito visoko, ali je znatno bolje od 24 ili 25 mjesta. Međutim, taj postupni rast u ukupnim DESI pokazateljima ne smije nas zavarati, pogotovo kada govorimo o infrastrukturnoj podlozi koja je osnova za digitalnu transformaciju društva.



Slika 3. Poredba DESI pokazatelja za republiku Hrvatsku sa EU prosjekom [2]

Po ukupnoj „Povezivosti“ koja opisuje stanje naše širokopojasne infrastrukture nalazimo se na pretposljednjem ili 27 mjestu, a iza nas je samo Grčka[3]. Kako je infrastruktura nuždan uvjet za sve ostale pokazatelje, sasvim je logično da da se ni po ostalim DESI pokazateljima ne možemo nalaziti previše visoko. Jedan od indikativno loših DESI pokazatelja je i „Digitalne javne usluge“ iako smo tu pozicionirani na 22 mjesto među EU članicama. Međutim visoko nas pozicionira e-zdravstvo po kojem smo na 10 mjestu, a po kompletnosti ponuđenih usluga smo na 28 odnosno posljednjem mjestu [3]. Ono na što mi trebamo posebno obratiti pažnju su:

a) Podizanje pokazatelja “Connectivity” (Povezivost) kroz intenziviranje izgradnje optičke pristupne infrastrukture i
b) Podizanje pokazatelja „Digital Public Service”, kroz brže, transparentnije i učinkovitije unapređenje usluga javnih službi u Republici Hrvatskoj.

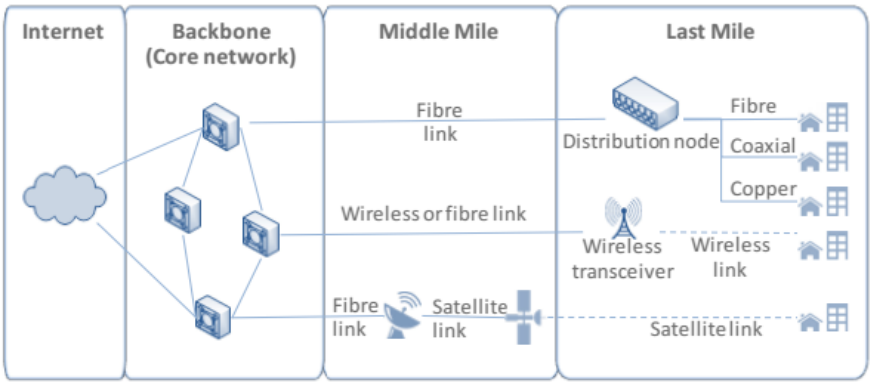
Značajnije zaostajanje po ova dva DESI pokazatelja rezultat je našeg ne sustavnog pristupa rješavanja ovog problema. Kod nas je veći problem nedostatka opće podrške i svijesti o ovom problemu nego nedostatak financijskih ili nekih drugih mogućnosti da se isti riješi. Aktivnosti za unapređenje ovih DESI pokazatelja jako su financijski podržane kroz čitav niz EU fondova. Nije razvojni problem kad nema novaca, problem je kad nema znanja, pa onda ni svijesti o postojanju problema. Također, važno je naglasit da u Republici Hrvatskoj ne nedostaje kadra koji se je u stanju nositi sa ovim problemom. Ima kadra i ima znanja, ali ono nije ni valorizirano ni distribuirano po društvenim pozicijama na adekvatan način. To je i problem iseljavanja mladih iz Hrvatske, ne iseljavaju se mladi zato što se ovdje ne može živjeti, nego zato što su pritisnuti nepravdom koja proizlazi iz klijentelizma, dakle, podobnosti, a ne sposobnosti.

Europska komisija u ovom trenutku priprema daljnje planove za Digitalnu transformaciju europskog društva za koje sagledava daljnji rast i prosperitet. Radi se o periodu 2020.-2030. godina. U nekim od prijedloga za daljnji razvoj govori se da do 2024. svaki građanin EU bez obzira gdje se nalazi mora ostvarivati pristupnu brzinu od 400 Mbit/s.

Ljudi koji nemaju viziju daljnjeg društvenog razvoja s svoje točke gledišta postavit će sasvim opravdano pitanje: „Što će nam to?“ Takove konstatacije poduprijet će zbrajanjem trenutnih potreba za informacijskim volumenom pristupne mreže, mjereći potrebni volumen pristupnih usluga. Za jedan video tok televizijske gledljivosti treba 4 Mbit/s (nakon prelaska sa MPEG-2 na MPEG – 4 trebat će oko 2 Mbit/s), za promatranje videa preko Interneta cca 1 Mbit, a sve ostalo je sitno. Dakle, četiri, pet Mbit/s je dovoljno i ništa ne treba mijenjati, jer to već imamo. Čak i ako dođe do povećanja usluga po kvalitetu i kvantitetu, više video tokova od kojih je jedan visoke rezolucije, prelasku videa sa Interneta na brzine od 4 Mbit/s još uvijek se sve može smjestiti ispod 30 Mbit/s. Ta konstatacija je jednako toliko pametna kao ideja o zatvaranju patentnog ureda jer je već sve izmišljeno što nam treba (Luj XIV Kralj Sunce prijelaz sa 17 na 18 stoljeće).

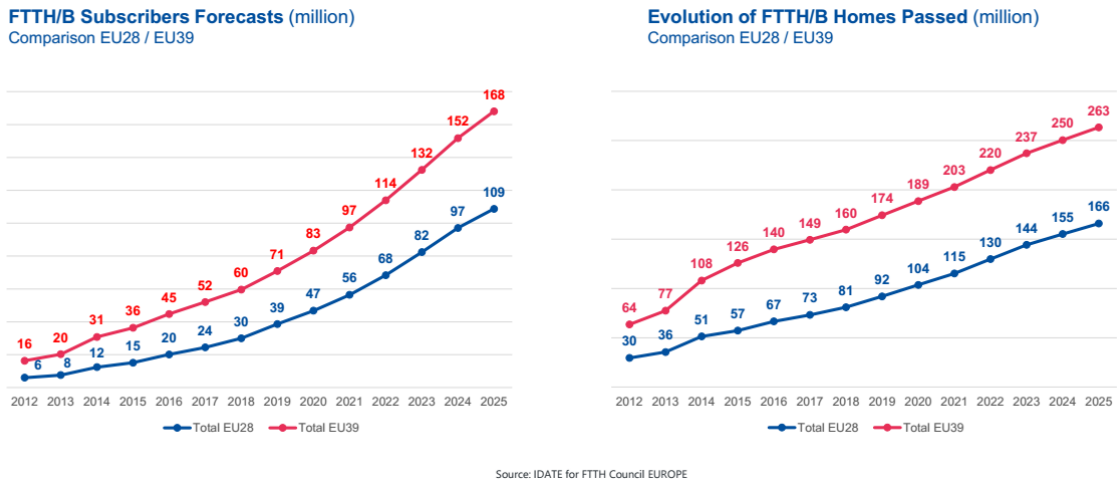
Europska Komisija sagledava daljnji gospodarski rast Europe kroz proces digitalizacije ne samo zbog znatno dostupnije informacije nego i zbog značajnih smanjenja troškova osnivanja tvrtki kao i redovnog poslovanja. U klasičnom načinu rada tvrtka mora posjedovati vlastiti prostor, mora u njemu postaviti vlastitu LAN infrastrukturu, mora kupiti serversku opremu i smjestiti je u klimatizirani prostor, mora kupiti aplikacije koje će joj služiti u radu i mora imati svoju IT službu za održavanje aplikacija i mreže. U suvremenom načinu rada tvrtka ne mora imati vlastiti prostor, ne mora graditi vlastitu LAN infrastrukturu, ne mora kupovati servere i ne mora ih smještavati u klimatizirani prostor, ne mora posjedovati svoju IT službu, ne mora kupovati aplikacije, ne treba trošiti na održavanje aplikacija i mreže. Sve što tvrtka treba, odnosno njeni djelatnici je brzi pristup internetu koji je jednake brzine kao što su brzine na LAN mrežama i terminale osrednjih ako ne i loših kvaliteta. LAN je virtualan i formira se u oblaku, aplikacije se ne kupuju nego se uzimaju u mjesečni najam. Kada su troškovi osnivanja tvrtke manji, tvrtku je lakše osnovati, a kada su troškovi poslovanja manji proizvod je konkurentniji na tržištu. Međutim, ovaj model da bi bio ostvariv traži kvalitetan i brz pristup širokopojasnoj mreži, istu onu brzina koju podržava LAN. Kako aplikacije postaju sve zahtijevanije glede informacijskog volumena tako se procjenjuje da u budućnosti neće biti dostatna brzina od 100 Mbit/s nego znatno veća. Razvoj širokopojasne mreže u cijelosti treba gledati kao razvoj gospodarske infrastrukture, a ne kao neke društvene nadogradnje za igru i razbibrigu tijekom slobodnog vremena. Zemlja koja ostane informacijski neprohodna ostat će i gospodarski zaostala jer se ne može uklopiti u suvremene gospodarske tokove. Mjesta, sela i gradovi koji ne budu imali širokopojasnu pristupnu infrastrukturu u budućnosti će se sasvim isprazniti od ljudi.

3. STANJE, PLANOVI I AKTIVNOSTI U DALJNJEM RAZVOJU ŠIROKOPOJASNE PRISTUPNE MREŽE U EU I U HRVATSKOJ



Slika 4. Elementi suvremene širokopojasne mreže

Europska komisija predviđa daljnju snažnu digitalizaciju Europskog društva i u njoj sagledava daljnji ekonomski prosperitet zemalja Europske unije. Za daljnju digitalnu transformaciju Europskog društva nužno je daljnje jačanje komunikacijske infrastrukture. Ciljevi koje je kroz digitalnu agendu Europska komisija 2010. godine postavila za vremenski period do 2020. u razvoju širokopojasne pristupne mreže neće biti ostvareni, ali neke zemlje su se tim ciljevima približile više, a neke zemlje manje. Republika Hrvatska je u približavanju ka tim ciljevima nešto uspješnija od Cipra i Grčke, ali znatno zaostaje za Rumunjskom i Bugarskom. Težište daljnjeg razvoja širokopojasne pristupne infrastrukture sagledava se u daljnjem intenzivnom razvoju FTTH/B pristupa. Sagledava se da će snažan doprinos razvoju širokopojasnog pristupa na područjima na kojima nije ekonomski isplativo postavljati svjetlovode (barem ne za sad) potrebnu pristupnu brzinu osigurati 5G mreže. Daljnji razvoj NGA rješenja temeljenih na VDSL tehnologiji sagledava sa kao rješenje koje bi moglo usporiti i koje usporava daljnju penetraciju svjetlovoda do krajnjeg korisnika.



Slika 5. Predviđanja za EU28 i EU39 ratvoja FZZH/B do 2025. [7]

Na gornjoj slici su predviđanja daljnjeg porasta FTTH/B korisnika i priključenih domaćinstava u Europi i zemljama EU za period do 2025. godine. U sljedeći šest do sedam godina predviđa se prosječeni porast FTTx infrastrukture za više 150%. Europska komisija priprema dokumente za daljnji desetogodišnji razvoj digitalizacije, te sagledava potrebna sredstva za ulaganje u taj razvoj kao i fondove iz kojih će ih osigurati. Neka predviđanja govore da će ciljevi za naredni period biti postavljeni još više pa se govori o 100% pokrivenosti sa brzinama do 400 Mbit/s.

Republika Hrvatska još nije donijela cjelovitu strategiju digitalne transformacije Hrvatskog društva. Mi više na intuitivnoj razini procjenjujemo da bi to bilo dobro, jer to i drugi tako kažu, ali nismo sagledali što je sve potrebno da bi se takova transformacija provela. Ne znamo točno što to znači za daljnji gospodarski rast, kao ni što bi osim pristupne mreže trebalo napraviti, nismo se opredijelili što s našim informacijskim centrima i kakvu strategiju razvoja trebamo imati za njih. Jedino što znamo je da skromnije postavljeni ciljevi od EU u Strategiji razvoja širokopojasnog pristupa do 2020. neće biti dosegnuti. Krajnji je trenutak da Hrvatska definira što hoće u razvojnom smislu i sagleda kako to ostvariti.

4. ZNAČAJ I MJESTO RADIJSKIH KOMUNIKACIJA U DALJNJEM RAZVOJU ŠIROKOPOJASNE MREŽE

Širokopojasna mreže kako u svom transportnom tako i u svom pristupnom dijelu odlikuje se velikim informacijskim volumenima. Postavlja se opravdano pitanje kakova je buduća uloga radio komunikacijskih sustava u razvoju širokopojasne mreže? Radio prijenos je ograničen po svom kapacitetu i frekvencijskom spektru. Kada govorimo o tehnologijama mobilnih mreža koje će podržati u budućnosti prijenos na razini gigabita onda treba kazati da kada bi i ostvarili te kapacitete njih bi dijelili svi korisnici radio pristupa, dakle stvarni kapacitet koji bi mogao ostvariti korisnik u velike bi ovisio o tome koliko na tom pristupnom čvoru ima drugih korisnika i kakove su njihove komunikacijske potrebe. U promotivnim akcijama koje ponuđači radio pristupa sprovode radi prodaje svojih usluga najčešće se navode maksimalno mogući kapaciteti koje sustav može ostvariti u idealnim ili laboratorijskim uvjetima. Stvarni kapaciteti u veliko ovise o realnim uvjetima radio linka kao i o tome da li je korisnik u mirovanju ili pokretu i kojom brzinom se kreće kroz prostor.

Tijekom čitave povijesti razvoja komunikacijskih sustava radijski linkovi su mijenjali svoju ulogu u ukupnom razvoju komunikacijskih sustava, ali nikad nisu pa ni neće biti bez značaja. Tehnologija radijskog prijenosa stalno je doživljavala svoj razvoj i napredak, a on je bio posebno intenzivan u posljednjih tridesetak godina. U domenu mapiranja signala i radio-modulacijskih postupaka razvile su se tehnologije poput OFDM-a koje su visoko-otporne na glavnu smetnju u radio komunikaciji, a to je višestruko prostiranje. Korištenje većeg broja podnosioca omogućilo je kod radio-komunikacijskih sustava dodatnu fleksibilnost u distribuiranju raspoloživa kapaciteta među korisnicima. U zadnjih dvadesetak godina široko se je razvila primjena kognitivnih radija koji se snagom, modulacijskim postupkom i drugim čimbenicima prilagođavaju kvaliteti raspoloživog radio linka. U domenu antenskih sustava snažan razvoj išao je u više pravaca koje objedinjujemo pod nazivom inteligentni antenski sustavi. Dakle, radio tehnologija je doživjela vrlo buran i intenzivan razvoj bez kojeg ne bi bile ostvarive današnje usluge kao što je DVB (digitalno emitiranje televizijskih signala) uz postavljanje jedno-frekventne mreže koja je omogućila značajnu uštedu u radio frekvencijskom spektru, te oslobađanje djela RF opsega koji se je ranije koristio za emitiranje televizijskih signala. Bez novih spoznaja ne bi bili mogući novi komunikacijski sustavi poput 4G LTE danas već široko primijenjene mobilne komunikacijske tehnologije (U Europi se koriste opsezi 800 MHz , 1800 MHz i 2600 MHz).

4.1. Radijske tehnologije u krajnjem pristupu širokopojasnog korisnika

Već iz navedenog možemo uočiti da radijske tehnologije ne gube na značaju nego mijenjaju svoju ulogu u ukupnoj komunikacijskoj mreži. U doba analognih sustava usmjerene radijske veze bile su glavna okosnica magistralnog uvezivanja sustava, imale su veliki značaj u core ili transportnoj mreži. Danas su taj značaj izgubile, ali suvremene

radijske tehnologije postaju sve značajnije u povezivanju krajnjih korisnika na mrežu, sve su značajnije u pristupnoj mreži ili krajnjem koraku do korisnika.

Predvidiva je ne tako daleka budućnost u kojoj neće biti fizičkog konektiranja krajnjih terminala, već će se ona obavljati u zadnjem kilometru ili u zadnjim metrima radijskim putem. 5G rješenja mobilnih komunikacijskih sustava kreću se upravo u tom smjeru. 5G po svojim specifikacijama može koristiti i koristi različite dijelove frekvencijskog opsega od svega 600 MHz pa sve do 50 GHz, a po nekim rješenjima i po nekim proizvođačima opreme i do 73 GHz. Niske frekvencije omogućit će pokrivanje vrlo velikih područja, ali primjena visoko frekvencija oko i preko 50 GHz znači upravo vrlo sitni ćelijski sustav ne veći od par do maksimalno stotinjak metara. Problem ovako visokih frekvencija je veliko gušenje EMV (antene izuzetno malih dimenzija), interakcija EMV sa česticama atmosfere te nemogućnost ogiba EMV. Pitanje je i utjecaja na ljudsko zdravlje kada se radi o ovako visokim frekvencijama EMV. Energija kvanta elektromagnetskog vala je Planck-ova konstanta pomnožena sa frekvencijom. Energija koja je dostatna da razori DNA molekulu je 1eV i ostvaruje se kod UV i viših zračenja. Energija na 50 GHz iznosi oko 1,85 x 10^{-4} eV odnosno pet i pol tisuća puta je manja od one koja može direktno oštetiti naš DNK. Međutim, izaziva termalne efekte u ljudskom tijelu, tako da širu upotrebu ovih frekvencija treba ipak razmotriti i s aspekta ljudskog zdravlja. Ako ćeliju ograničimo na prostor od svega stotinjak metara, dostatni kapaciteti za broj korisnika koji fizički stanu u taj prostor može se ostvariti i na znatno nižim frekvencijama.

Gušći ćelijski sustav koji zahtijevaju 5G rješenja, uz pretpostavku korištenja frekvencija ispod 11 GHz (kada još uvijek imamo ogib EMV) znači ujedno po emitiranoj snazi i tiši sustav jer radijski link do korisnika je kratak.

4.2. Radio relejne ili usmjerene radio veze u pristupnoj mreži.

Iako je izuzetno velik broj naselja u Republici Hrvatskoj povezan sa svjetlovodima ipak još uvijek postoje brojna naselja koja to nisu. Poglavito kada govorimo o mjestima na otocima, koja su danas turistički atraktivna, a koje u daljnjem razvoju nautičkog turizma sprječava upravo loša informatička povezanost.

U trenutku kada se je u Hrvatskoj vršila privatizacija Hrvatskog telekoma nije izvršena privatizacija samo usluge nego usluge i infrastrukture. To nije bio mudar potez, trebalo je podijeliti kompaniju na uslugu i infrastrukturu, te infrastrukturu zadržati u državnom vlasništvu i dati svim operaterima na korištenje pod jednakim uvjetima, a uslugu privatizirati. Međutim, sad nema dubljeg smisla raspravljati o tome jer je to prošlo vrijeme. Posljedica tog postupka dovela je i dovodi vrlo često do situacije u kojoj do velikih korisnika postoji višestruko izgrađena infrastruktura, a na mnogim drugim dijelovima je uopće nema. Državna uprava je zakazala što operaterima uz davanje koncesije na uslugu nije uvjetovala cjelovitu pokrivenost nacionalnog teritorija širokopojasnim pristupom određene brzine u odgovarajućem vremenskom periodu.

Ponekad i ima svjetlovoda postavljenih do otoka, ali on je u vlasništvu pojedinog operatera, pa drugi operateri nerado uzimaju u najam infrastrukturu konkurentskog operatera.

Ponekad se brzi širokopojasni pristup nastoji za pojedina naselja osigurati korištenjem sustava usmjerenih radijskih veza. Usmjerena radijska veza na jednom radijskom kanalu može prenijeti nešto malo više od 300 Mbit/s prometa, tako da se danas u tu svrhu nude RR konfiguracije (6+0) s mogućnošću ukupnog prijenosa informacije oko 2 Gbit/s. Konfiguracije sa šest radnih radijskih uređaja u istom RF području, ali na različitim RF kanalima. Nema potrebe postavljati rezervni uređaj u konfiguraciju jer u slučaju ispada jednog od uređaja uslijed fedinga ili kvara ne prekida se promet nego se samo smanjuje ukupni raspoloživi radni kapacitet. Radijski uređaji spregnuti su na istu antenu. Nerijetko kako bi se uštedjelo na širini RF spektra koriste se antene sa križnom polarizacijom, ali to nije preporučljivo ako su veze nisko postavljene iznad mora je hod horizontalne polarizacije reflektirani val mijenja fazu za 180^{o} stupnjeva. Refleksiju signala od velikih ravnih površina (more, polja) treba izbjeći pozicioniranjem antena. RF opsezi koji se koriste u ove svrhe mogu biti i viši poglavito na manjim udaljenostima, ali tada treba voditi računa o fedingu uslijed kiše (feding uslijed kiše postaje značajniji iznad 10 GHz). Više frekvencije omogućit će primjenu manjih antena, ali te su veze primjenjive na manjim udaljenostima. Niže frekvencije poglavito ispod 10 GHz zahtijevaju veće antene, ali su primjenjive na većim udaljenostima, dvadeset i više kilometara.

Postavljanje usmjerenih radijskih veza može se obaviti relativno brzo, dok je polaganje svjetlovodnog kabela do nekog naselja investicija koja traje znatno duže vremena. Međutim, usmjeren radijske veze ne mogu biti po kapacitetu ni približna zamjena svjetlovodnoj infrastrukturi. Ako potreba za informacijskim kapacitetom bude i dalje rasla, a predviđanja su da hoće, pogotovo što sve više bude rada u oblaku *(cloud)*, rješenja povezivanja pojedinih naselja korištenjem usmjerenih radijskih veza nisu i neće biti trajno nego privremeno rješenje.

Priče o masovnom korištenju radijskih linkova na frekvencijama većim od 50 GHz nisu vjerodostojne. Na tim frekvencijama radijski link ne može biti dug, svega nekoliko stotina metara. Valna duljina je takova da EMV već ulazi u vrlo snažnu interakciju sa atmosferom te dolazi do njegova vrlo snažnog slabljenja uslijed apsorpcije i refleksije o čestice u atmosferi [4]. Na visokim frekvencijama veliko je i slabljenje prostiranja EMV (antena je mala i ne može primiti veliku količinu energije iz zraka), nije ni lako proizvesti EMV izuzetno frekvencije i izuzetno visoke izlazne snage i t d.

5. RAZVOJNE STRATEGIJE DIGITALIZACIJE HRVATSKOG DRUŠTVA I MJESTO RADIJSKIH KOMUNIKACIJA U NJIMA

Sve razvojne strategije digitalizacije Hrvatskog društva vješto izbjegavaju napominjanje tehnoloških rješenja. Govori se o kapacitetu kojeg treba ostvariti, a pri tome ne želi se spominjati tehnološko rješenje, kako bi se izbjeglo preferiranje pojedinih tehnologija i njihovih proizvođača ili dobavljača na tržištu.

Takav pristup ne navođenja tehnologija nije dobar iako je s aspekta neutralnosti korektan. Međutim, potrebno je kazati da se tehnologije prijenosnih medija bitno razlikuju po kapacitetu. Svjetlovod iako je danas jedan od najjeftinijih prijenosnih medija po kapacitetu je zasigurno najsnažniji prijenosni medij i drugi mediji se ne mogu mjeriti s njim. U sustavima s gustom raspodjelom valnih duljina u L i C području danas na 80 valnih duljina koje su razmaknute za 0,8 nm (ekvivalentno frekvencijskom razmaku od 100 GHz) po svakoj valnoj duljini moguće je prenijeti do 10 GHz (na udaljenostima do 75 km, ograničenje je zbog kromatske disperzije na standardnom vlaknu G. 652). Postoje standardi i s većom gustoćom raspodjele među valnim duljinama kao na primjer razmak od 0,4 nm (ekvivalentno 50 GHz) ili 0,2 nm (ekvivalentno 25 GHz) međutim, tada se ne može ići sa brzinama do 10 Gbit/s po valnim duljinama, već manjima. Linijski kod NRZ kod svjetlovoda je amplitudna modulacija nosioca (infracrvenog svjetla) digitalnim signalom s indeksom modulacije m=1, te signal ima spektar koji se sastoji od nosive gornjeg i donjeg bočnog opsega. Također kod ovih sustava s više valnih duljina u gustom rasporedu mora se vodit računa o ujednačenosti razinama signala po pojedinim valnim duljinama, o tome da se modulirani signali spektrima ne prekrivaju. Dakle. današnjom tehnologijom korištenja svjetlovoda realno mogu prenijeti ukupni tok informacija od 800 Gbit/s. Povezivanje DESLAM ili drugog distribucijskog uređaja danas se nerijetko ostvaruje linkovima od 10 Gbit i više. Dakle, radio tu ne može biti nešto što će činiti dugoročnu i perspektivnu zamjenu.

Radio sve više dobiva i dobivat će ulogu u završnom povezivanju terminala širokopojasne mreže na pristupnu točku u zadnjih stotinjak metara. Tu će doći do razvoja Wi Fi pristupnih tehnologija na velikim brzinama Wirelles LAN-ova. Na primjer do jedne marine dovodeći svjetlovod nije problem dovesti veliki kapacitet, ali sadašnje Wi Fi tehnologije počinju predstavljati usko grlo. U večernjim satima na malom prostoru nalazi se nekoliko stotina brodova, a na svakome po desetak potencijalnih korisnika širokopojasnog pristupa, za vrlo kratko vrijeme treba ukupno distribuirati kapacitet od 10 i više Gbit/s. Obzirom na mali prostor nije moguće staviti veliki broj bežičnih LAN-ova jer ih se ne može adekvatno prostorno odvojiti. U tom smislu očekuju se nova i kvalitetnija rješenja.

Hrvatske razvojne strategije ne bi smjele biti popu dosadašnjih strategija razvoja širokopojasnog pristupa u kojim osim konačno željenog kapaciteta ne piše ništa. Kada govorimo o razvoju digitalne širokopojasne mreže onda je to u svrhu digitalne transformacije društva, pa i digitalne ekonomije. Treba sagledati također i tehnologije koje mogu dati dostatan kapacitet, ali i jasno definirati ulogu i poziciju informatičkih centara u Republici Hrvatskoj.

6. ZAKLJUČAK

Da bi se cjeloviti prostor Republike Hrvatske mogao staviti u gospodarsku funkciju nužno je na njemu osigurati širokopojasni pristup dostatnih brzina. Kako je veliki dio prostora nenaseljen ili rijetko naseljen, to ujedno znači i nisku gustoću potencijalnih korisnika, pa se u tom slučaju radijska rješenja koja se temelje na brzom širokopojasnom pristupu nameću kao ekonomski održivo rješenje. Takove, prostore treba pokriti 4G i 5G signalima mobilnih komunikacijskih sustava. Poglavito treba insistirati na pokrivanju 4G LTE signalima u 800 MHz području. U nižim frekvencijskim područjima 4G rješenja mogu ostvariti relativno velike ćelijske dosege do 100 km, ali optimalna rješenja makro-ćelija 4G sustava u ruralnim područjima su od 5 do 10 km, maksimalno 30 km. Relativno malim brojem ćelija može se pokriti teritorij RH bez posebno velikih infrastrukturnih ulaganja. Sustavi 4G i 5G omogućit će stavljanje u gospodarsku funkciju cjelokupni prostor RH. Nakon prelaska na DVBT2 predviđa se oslobađanje djela spektra u 700 MHz području i njegova prenamjena za mobilne komunikacijske sustave. Kada do toga dođe u ponudi koncesije mobilnim operaterima potrebno je staviti prioritet pokrivanje ruralnih prostora RH takovim signalom. Do udaljenih baznih postaja mogu se koristiti radijska rješenja usmjerenih veza Ethernet radija kako bi se osiguralo dobro pokrivanje svih područaja.

Wireless rješenjima za širokopojasni pristup potrebno je intenzivnije planirati pokrivanje javnih prostora u RH (trgova, šetališta i drugih prostora na kojima se ljudi okupljaju) pogotovo u onim sredinama koje planiraju razvoj turizma. Navedeno zahtijeva napore lokalne uprave, pa dio prihoda od turizma koje ostvaruje lokalna uprava kroz boravišne pristojbe treba usmjeriti u daljnje unapređenje turističke djelatnosti kroz bolju i kvalitetniju širokopojasnu dostupnost.

Najučinkovitiji razvoj je onaj koji se odvija planski. Republika Hrvatska osim deklarativne odluke o „Digitalnoj transformaciji društva“ mora sagledati kakve će dobiti kao društvu to donijeti te u tom smislu definirati ukupnu razvojnu strategiju iz koje će proizlaziti i sagledavanje realnih potreba za svim pa i tehnološkim segmentima ove transformacije. Digitalna transformacija društva zahtijeva ulaganja. Uloženi novac u daljnji razvoj komunikacijske infrastrukture je kapital koji će omogućiti rast i razvoj domaćih kompanija ali i onih kompanija koje trebaju biti nosioci ovog posla.

Literatura:

1 DESI2019 Croatian TelecomChapters

2 Indeks digitalnoga gospodarstva i društva (DESI)1 za 2018., Izvješće za Hrvatsku file:///C:/Users/Elektrotehnika/Downloads/HR-DESI2018-Country-Profile-LANGpdf.pdf

3 DESI Conectivity https://digital-agenda-data.eu/charts/desi-components#chart={"indicator":"desi_1_conn","breakdown-group":"desi_1_conn","unit-measure":"pc_desi_1_conn","time-period":"2016"}

4 Luigi Moreno; Point- to - Point Radio Link Engineering , Torino Italy 2010. https://radiocomunicazioni.org/files/stories/files2/pdf/PPRLE_E_Book_v1_2.pdf

5 Vlada Republike Hrvatske, STRATEGIJA RAZVOJA ŠIROKOPOJASNOG PRISTUPA U REPUBLICI HRVATSKOJ U RAZDOBLJU OD 2016. DO 2020. GODINE, https://mmpi.gov.hr/UserDocsImages/arhiva/Strategija-sirokopojasni-pristup2016-2020-usvojeno%20na%20VRH.pdf

6 European Court of Auditors: Broadband in the EU Member States: despite progress, not all the Europe 2020 targets will be met, https://www.eca.europa.eu/Lists/ECADocuments/SR18_12/SR_BROADBAND_EN.pdf

7 Roland Montagne. FTTH Forecast for EUROPE. Market forecast by 2020 and 2025, https://www.ftthcouncil.eu/documents/Reports/2019/FTTH%20Council%20Europe%20-%20Forecast%20for%20EUROPE%202020-2025.pdf

Podaci o autoru:

dr.sc. Winton Afrić

e-mail: wafric@oss.unist.hr

Winton Afrić rođen je 1956. godine u Splitu. Doktorsku disertaciju obranio je 2003. godine na Fakultetu elektrotehnike i računarstva u Zagrebu, na Zavodu za radiokomunikacije. Od 11. lipnja, 2008. godine do danas u radnom odnosu na Sveučilištu u Splitu, Sveučilišnom odjelu za stručne studije. Radio je u RO PTT prometa u Splitu i pravnim slijednicima tog poduzeća na radnim mjestima planiranja, projektiranja, razvoja i nadzornog inženjera kod izgradnje telekomunikacijskih sustava i postrojenja. Član je društva ELMAR, a od 2003. i izvršnog odbora društva ELMAR (Hrvatsko društvo elektronika u pomorstvu). Član je Hrvatske Komore inženjera Elektrotehnike i ovlašteni inženjer elektrotehnike. Od 2007. godine je član suradnik Hrvatske akademije tehničkih znanosti. Ovlašteni je sudski vještak za telekomunikacije. Učesnik Domovinskog rata od 1991. do 1995. godine.

Kao istraživač sudjelovao je četiri znanstvena projekta Ministarstva znanosti i tehnologije. Aktivno se služi engleskim (B1) i talijanskim (C1) jezikom.

University of Split
University department of vocational study
Livanjska 5, 21000 Split Croatia

PRIMJENA ANT + TEHNOLOGIJE

APPLICATION OF ANT+ TECHNOLOGY

Roko Rogulj, Ivo Kovačević, dr.sc. Tonko Kovačević

SAŽETAK:

Primjena ANT+ tehnologije u aplikacijama koje se koriste u medicini, sportu i svakodnevnom življenje je sve veća. Jedan od glavnih problema prilikom implementacije bežičnih senzorskih mreža je energetska efikasnost. U tu svrhu osmišljen je ANT protokol, kao alternativa dosadašnjim bežičnim mrežama, kojemu je glavna odlika visoka energetska efikasnost. ANT+ je upravljana mreža ANT protokola, a koristi se za senzore u sportu i medicini. U ovom radu prikazana je aplikacija ANT+ mreže unutar ANT protokola za vizualizaciju mjerenja otkucaja srca pomoću platforme Node.js.

Ključne riječi: ANT+ tehnologija, ANT protokol, bežične senzorske mreže

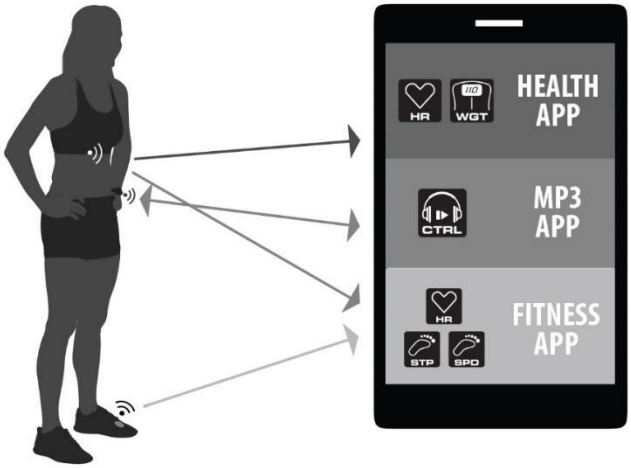
SUMMARY:

The usage of ANT+ technology in applications used in medicine, sports and daily life is increasing. One of the major problems during implementing wireless sensor networks is energy efficiency. For this purpose, the ANT protocol was designed as an alternative to wireless network technologies, whose main feature is high energy efficiency. ANT+ is a managed network of ANT protocols and it is used for sensors in sports and medicine. An application of ANT+ network within the ANT protocol for visualizing heart rate measurements using the Node.js platform is presented in this work.

Key words: ANT + technology, ANT protocol, wireless sensor networks

1. UVOD

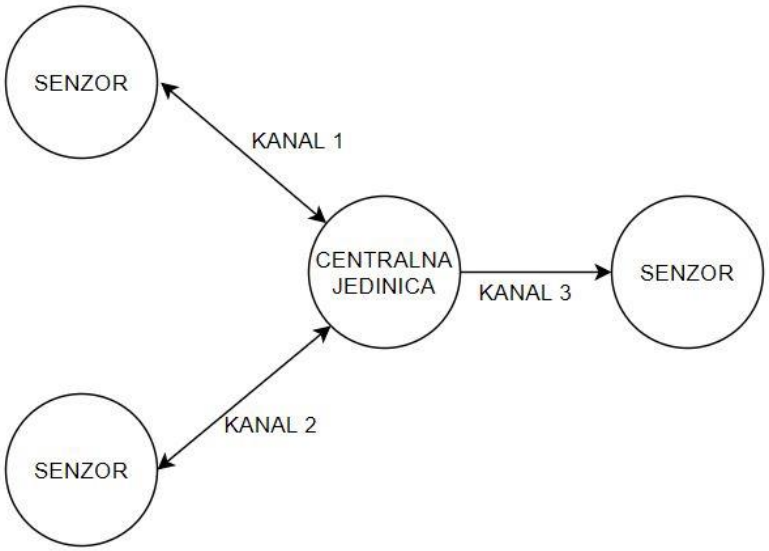
Bežične senzorske tehnologije doživjele su značajan tehnološki rast posljednjih godina. Potražnja tržišta uvjetovala je razvitak tehnologija koje se zasnivaju na uređajima visoke energetske efikasnosti te su iz tog skupa iznjedrile dva značajna protokola: *Adaptive Network Topology* (ANT) i *Bluetooth Low Energy* (BLE), gdje oba protokola rade u 2.4 GHz ISM frekvencijskom rasponu. Oba protokola su zamišljena za primjenu u okvirima WBAN - *Wireless Body Area Network*. BLE dodatno podržava, zbog velikih brzina prijenosa (2 Mbit/s), prenošenje glazbe i videa preko uređaja. Iako se velika brzina prijenosa čini kao prednost, BLE jako je ograničen kod postavljanja same mreže za razliku od ANT protokola. ANT koristi adaptivnu transmisiju podataka, koja omogućava komunikaciju velikog broja uređaja bez interferencije signala te ne zahtijeva postojanje globalnog takta razmjene podataka. ANT je u srži jednostavniji protokol od BLE te se njegova komunikacijska mreža može ostvariti kroz sve topologije, gdje se kod BLE može koristi samo zvjezdasta topologija [1].



Slika 1. ANT bežična senzorska mreža [2]

2. ANT PROTOKOL

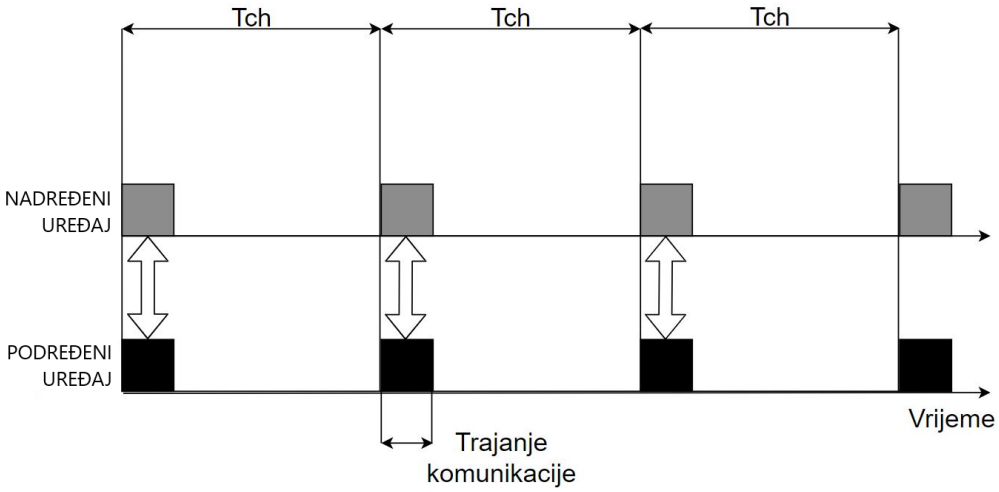
ANT jer bežični mrežni protokol koji radi u 2.4 GHz ISM frekvencijskom pojasu namijenjen za prikupljanje senzorskih podataka. Glavna odlika ovoga protokola je u energetskoj efikasnosti gdje je potrošnja električne energije višestruko manja u odnosu na slične protokole iz iste domene. Na slici 2. prikazan je primjer jednostavne ANT mreže.



Slika 2. *Jednostavna ANT mreža*

ANT mreža definirana je preko kanala. Svaki čvor mreže (prikazan kao krug na slici 1.) može se povezati pomoću kanala s drugim čvorom. Uobičajeno je da svaki kanal povezuje dva čvora, iako se mreža može konfigurirati da kanal povezuje više čvorova [2]. Svaki kanal, kao minimalni uvjet, mora imati barem jedan nadređeni čvor - *master* i jedan podređeni čvor - *slave*. Svaki čvor može biti i nadređeni i podređeni čvor istovremeno. Također, svaki čvor u mreži može se ponašati kao prijemnik, odašiljač ili oboje, slika 2.

Većina ANT čvorova koristi sinkronu, nezavisnu i dvosmjernu komunikaciju. Svaki put kada nadređeni čvor otvori sinkroni kanal, prvo provjerava da li se ta komunikacija interferira s transmisijom nekog drugog uređaja. Ako nema interferencije postavlja se određeni vremenski period unutar čvora *Tch* – vremenski period kanala (slika 3.), unutar kojega nadređeni uređaj šalje podatke. Kod dvosmjerne komunikacije, nadređeni uređaj ostavlja prijemnik upaljenim nakon slanja podataka da bi podređeni uređaj mogao poslati svoje podatke.

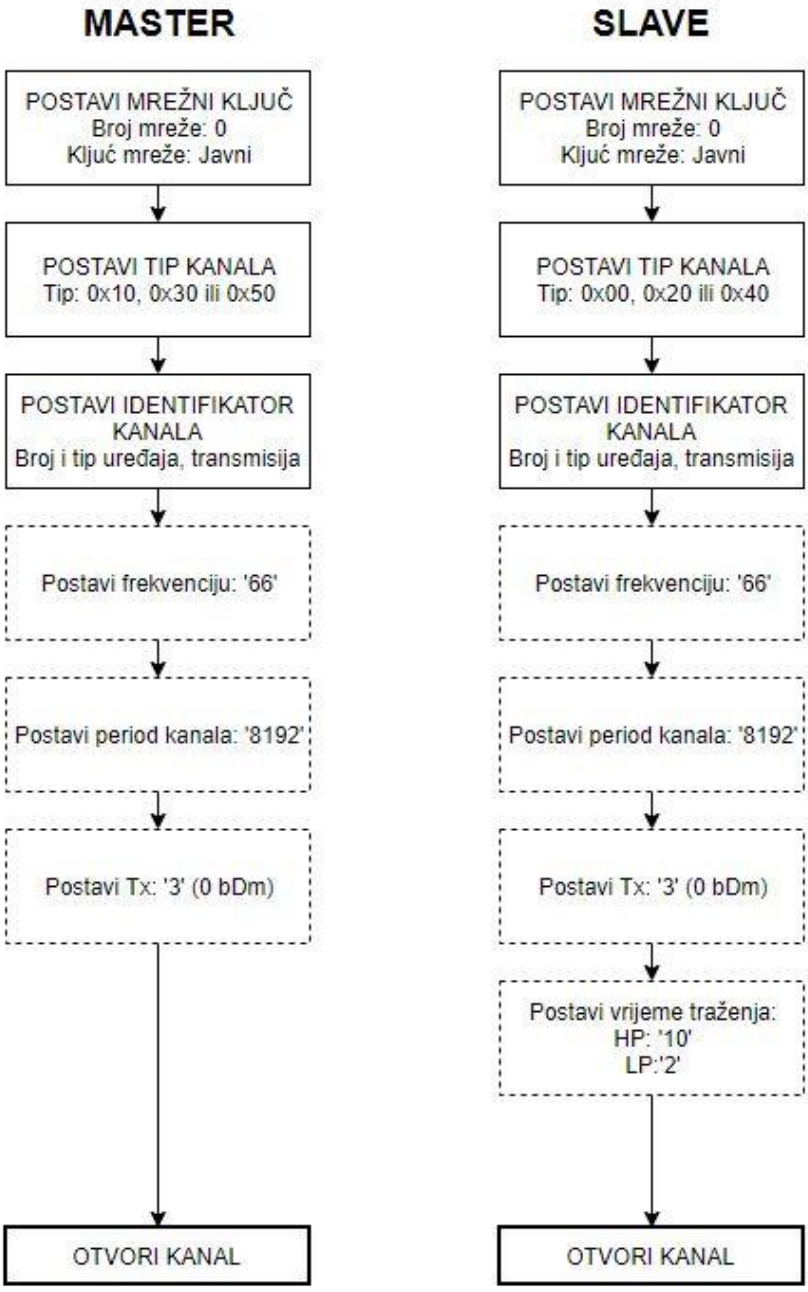


Slika 3. *Komunikacija između nadređenog i podređenog uređaja*

Za ostvarivanje komunikacije između dva ANT uređaja potrebno je definirati sljedeće parametre:

- tip kanala,
- frekvenciju kanala
- identifikator kanala (tip transmisije, tip uređaja i broj uređaja)
- vremenski period kanala,
- mreža.

Na slici 4. prikazan je primjer procesa uspostave kanala. Postavke na slici koje nemaju unaprijed definirane vrijednosti označene su punom linijom dok postavke koje imaju unaprijed definirane vrijednosti označene su iscrtanom linijom (proizvođač uređaja određuje unaprijed definirane postavke).



Slika 2. Primjer uspostave kanala

Razlikujemo četiri tipa podataka u ANT mrežama: *Broadcast*, *Acknowledged*, *Burst* i *Advanced Burst*. Tip podataka se ne definira kao parametar kanala stoga jedan kanal može koristiti sve tipove podataka. Svaki od ova četiri tipa podataka može biti poslan u *forward* ili *reverse* smjeru kod dvosmjerne komunikacije. Jedno ograničenje se postavlja kod jednosmjernih kanala gdje se može koristiti samo *forward-broadcast* tip kanala. Kratak pregled podataka prikazan je u tablici 1.

Tablica 1. *Tipovi podataka u ANT protokolu*

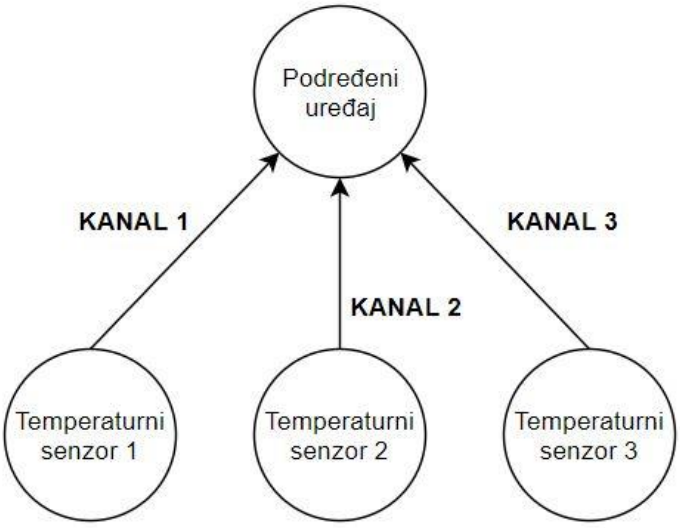
Tip podatka	***Smjer kanala***	***Opis***
Broadcast	*Forward*	Osnovni tip podataka. Šalje se u svakom periodu kanala.
	Reverse	Podaci koji se šalju samo na zahtjev podređenog uređaja.
Acknowledged	*Forward*	Šalje se samo na zahtjev.
	Reverse	Podaci koji se šalju samo na zahtjev podređenog uređaja te se ne mogu poslati ako prethodno nije bila primljena poruka.
Burst	*Forward*	Paketi koji su sinkronizirani odvojeno od glavnog takta slanja poruka. Dolazi do retransmisije paketa ako uređaj nije primio nove podatke.
	Reverse	Šalju se samo na zahtjev podređenog uređaja. Šalju se samo jednom te nema retransmisije.
Advanced Burst	*Forward*	Paketi koji su sinkronizirani odvojeno od glavnog takta slanja poruka. Dolazi do retransmisije paketa ako uređaj nije primio nove podatke. Brzine slanja su veće nego kod *Burst* podataka.
	Reverse	Šalju se samo na zahtjev podređenog uređaja. Šalju se samo jednom te nema retransmisije.

Jednostavna ANT mreža koja se sastoji od jednog podređenog uređaja i tri nadređena uređaja (temperaturne sonde) prikazana je na slici 5. Podređeni uređaj želi ostvariti trajnu vezu sa svim temperaturnim sondama. Da bi započelo uparivanje svaka temperaturna sonda mora kronološki odraditi sljedeće radnje:

1. Konfiguraciju kanala
2. Postavljanje identifikatora kanala
3. Otvaranje kanala za slanje podataka
4. Slanje podataka u svakom vremenskom periodu kanala.

Podređeni uređaj mora se pripremiti za traženje svih nadređenih uređaja odgovarajućeg tipa. On obavlja sljedeće radnje:

1. Konfiguracija kanala
2. Postavljanje identifikatora kanala
3. Otvaranje kanala za prijem podataka
4. Traženje nadređenih uređaja.



Slika 3. Primjer uparivanja u ANT mreži

Podređeni uređaj pronalazi temperaturne senzore koji imaju bit uparivanja unutar ANT poruke postavljen na '1'. Kada se uspostavi kanal, podređeni uređaj prosljeđuje svoj identifikator kanala temperaturnom senzoru gdje ga on trajno sprema u svoju memoriju. Na takav način se osigurava da se postupak uparivanja ostvaruje samo jednom kod trajne veze gdje se smanjuje opterećenje sustava. Enkripcija kanala može biti postavljena na svim nezavisnim kanalima. Kada je enkripcija aktivna, ANT kanal koristi 128-bitni AES-CTR (*Advanced Encryption Standard – Counter Mode*). Mora se naglasiti da enkripcija nije aktivna u svim mrežama. Primjerice, ANT+ mreža zabranjuje korištenje enkripcije radi kompatibilnosti uređaja između različitih proizvođača. Kriptirani ANT+ kanali ne mogu koristiti ANT+ mrežni ključ ni radio frekvencije mreže.

3. ANT+ PROTOKOL I UREĐAJI

ANT+ je upravljana ANT mreža kojoj je svrha stvaranje jedinstvenog sustava unutar kojega uređaji različitih proizvođača mogu komunicirati. ANT+ se koristi u svrhe sporta i kućne medicine. Primjeri korištenja ANT+ mreže su:

- Mjerenje otkucaja srca
- Brzina gibanja na biciklu
- Mjerenje snage bicikla
- Mjerenje mase
- Senzori opreme za tjelovježbu
- Temperaturni senzori.

Svi ANT+ uređaji imaju svoj profil. Svaki profil određuje funkciju pojedinog uređaja. Primjerice, *Environment* profil uvijek šalje iste podatke nezavisno o proizvođaču uređaja. Svi ANT+ uređaji su konfigurirani tako da ANT+ korisnička sučelja uvijek znaju koji tip podatka dobivaju [3]. Bitna razlika ANT+ i ANT mreže su postavke kanala. Primjer postavki kanala za podređeni ANT+ mreže prikazan je u tablici 2. ANT+ može koristiti bilo koji tip poruke za prijenos podataka (najčešće prošireni *flagged* tip).

Tablica 2. *Postavke podređenog kanala u ANT+ mreži*

Parametar	***Vrijednost***	***Opis***
Mrežni ključ	ANT+ mrežni ključ	ANT+ mrežni ključ se dijeli proizvođačima uređaja na službenim ANT+ internet stranicama
Frekvencija	57	ANT+ radio frekvencija
Broj uređaja	0	Traži sve nadređene uređaje
Tip prijenosa	0	Uparivanje
Tip uređaja	25	*Environment* senzor
Tip kanala	0x000	Podređeni kanal
Period kanala	65535	Takt slanja od 2 Hz

Svaki ANT+ uređaj posjeduje profil koji sadrži određena pravila mreže te podatke tog profila može poslati unutar poruke. Profili se sastoje od stranica (*eng. pages*) te unutar svakoga profila možemo pronaći više različitih stranica. Stranice su označene brojevima a broj stranica koji se šalju određen je samim profilom koji se koristi. Bitno je naglasiti da se podaci senzora šalju preko stranica te je dopušteno slanje ostalih poruka unutar ANT mreže nezavisno o stranicama (kao dijagnostički podaci ili podaci potvrde). Postoje sljedeći ANT+ profili:

- ***Bicycle Power*** – mjerenje izlazne snage biciklista
- ***Controls*** – kontrola glazbenih elektroničkih uređaja
- ***Geocache*** – mjerenje geografske lokacije
- ***Multi Sport Speed & Distance*** – mjerenje brzine i prijeđene udaljenosti
- ***Stride Based Speed & Distance*** -mjerenje brzine i prijeđene udaljenosti hodom
- ***Bicycle Speed & Cadence*** -mjerenje brzine i broja koraka
- ***Enviroment*** – mjerenje temperature
- ***Hearth Rate Monitor*** – mjerenje otkucaja srca
- ***Muscle Oxygen Monitor*** – mjerenje koncentracije kisika u mišićima
- ***Sync*** – spremanje podataka za kasniju analizu
- ***Blood Pressure*** – mjerenje krvnog tlaka
- ***Fitness Equipment*** – profil za komuniciranje s opremom za tjelovježbu
- ***Light Electric Vehicle*** – upravljanje i nadzor električnih vozila malih snaga
- ***Racquet*** – senzor za mjerenje parametara teniskih reketa
- ***Weight Scale*** – mjerenje mase
- ***Extended Display*** – komunikacija prema udaljenom zaslonu
- ***Suspension*** – mjerenje amortizirajućih sila u vozilima
- ***Dropper Seatpost*** – nadzor i podešavanje sjedišta bicikla
- ***Tracker*** – pronalaženje uređaja pomoću GPS tehnologije.

Svaki podređeni uređaj pri slanju podataka šalje određene stranice koje nadređeni uređaj poznaje (spremljene u memoriji nadređenog uređaja). Primjerice, *Environment* profil može slati tri tipa stranica: podatkovne stranice (temperaturni podaci), stranice o proizvođaču uređaja te stranice o tehničkim specifikacijama samoga podređenog uređaja. Broj stranice koji se šalje određen je prvim bajtom unutar samih korisničkih podataka.

ANT+ uređaji dolaze u mnogo oblika, funkcijskih izvedbi i tehnologija. Najčešći princip izvedbe je podređeni uređaj u nosivome obliku (narukvica ili ogrlica) dok se za nadređeni uređaj koristi ANT+ kompatibilni pametni telefon ili računalo s kompatibilnim ANT+ USB uređajem, slika 6.

Slika 6. ANT+ senzorski uređaji za mjerenje otkucaja srca

Za sve uređaje dostupna je prilagođena mobilna aplikacija proizvođača u kojoj je predstavljen prikaz trenutnih mjernih vrijednosti te dodatne mogućnosti kao praćenje lokacije i trajanje vježbe (ostvareno preko funkcija pametnog mobitela). Izgled korisničkog sučelja aplikacije proizvođača „Garmin“ prikazan je na slici 7. Korisnik nije dužan koristiti mobilnu aplikaciju proizvođača, već zbog kompatibilnosti samoga protokola može koristit bilo koju ANT+ aplikaciju.

Slika 7. *Garmin Fit mobilna aplikacija*

Radi utvrđivanja valjanosti ANT+ *Hearth Rate* profila, u okviru ovoga rada izvršen je nadzor serijskog sučelja na nadređenom uređaju. Na osobno računalo postavljen je ANT+ kompatibilni USB uređaj te je ostvarena komunikacija s mjeračem otkucaja srca proizvođača „Sosche“. Promet je stavljen pod nadzor pomoću programa „SysNucleus USBTrace“. Primjer očitane komunikacije prikazan je na slici 8.

52.100975	BULK_OR_INTERRUPT_TR...	IN	81	\Device\USBPDO-3	0x27434B...	STATUS_SUCCESS	A4 14 4E 01 04 FF 40 AE 5B B3 B2 2F E0 1D...	24
52.101022	BULK_OR_INTERRUPT_TR...	OUT	81	\Device\USBPDO-12	0x27434B...	STATUS_SUCCESS		0
52.101027	BULK_OR_INTERRUPT_TR...	OUT	81	\Device\USBPDO-3	0x27434B...	STATUS_SUCCESS		0
52.101046	BULK_OR_INTERRUPT_TR...	IN	81	\Device\USBPDO-3	0x27434B...	STATUS_PENDING		0
52.346971	BULK_OR_INTERRUPT_TR...	IN	81	\Device\USBPDO-3	0x27434B...	STATUS_SUCCESS	A4 14 4E 01 04 FF 40 AE 5B B3 B2 2F E0 1D...	24
52.347005	BULK_OR_INTERRUPT_TR...	OUT	81	\Device\USBPDO-12	0x27434B...	STATUS_SUCCESS		0
52.347009	BULK_OR_INTERRUPT_TR...	OUT	81	\Device\USBPDO-3	0x27434B...	STATUS_SUCCESS		0
52.347024	BULK_OR_INTERRUPT_TR...	IN	81	\Device\USBPDO-3	0x27434B...	STATUS_PENDING		0
52.593975	BULK_OR_INTERRUPT_TR...	IN	81	\Device\USBPDO-3	0x27434B...	STATUS_SUCCESS	A4 14 4E 01 04 FF 5B B3 76 B8 B3 2F E0 1D...	24
52.594012	BULK_OR_INTERRUPT_TR...	OUT	81	\Device\USBPDO-12	0x27434B...	STATUS_SUCCESS		0
52.594015	BULK_OR_INTERRUPT_TR...	OUT	81	\Device\USBPDO-3	0x27434B...	STATUS_SUCCESS		0
52.594025	BULK_OR_INTERRUPT_TR...	IN	81	\Device\USBPDO-3	0x27434B...	STATUS_PENDING		0
52.839988	BULK_OR_INTERRUPT_TR...	IN	81	\Device\USBPDO-3	0x27434B...	STATUS_SUCCESS	A4 14 4E 01 04 FF 5B B3 76 B8 B3 2F E0 1D...	24

Slika 8*. Nadzor USB sučelja na ANT+ protokolu*

Sve podatkovne poruke su duljine 24 bajta te dolaze u vremenskim intervalima od 0.25 sekundi (4 Hz) u heksadekadskom obliku te je jedan primjer poruke prikazan ispod:

A4 14 4E 01 04 FF 5B B3 76 B8 B3 2F E0 1D 73 78 01 10 00 68 00 2C 99 84

Može se primijetiti da poruke odgovaraju proširenom *legacy* formatu ANT protokola te se iz poruke može iščitati:

- Broj uređaja = 29 449
- Tip podataka = 4E (što odgovara *broadcast* podacima)
- Broj kanala = 01.

Payload prikazane poruke ima vrijednost *04 FF 5B B3 76 B8 B3 2F*. Iz prvoga bajta može se primijetiti da se radi o stranici '4' te da su izmjereni otkucaji srca od 47 *bpm*.

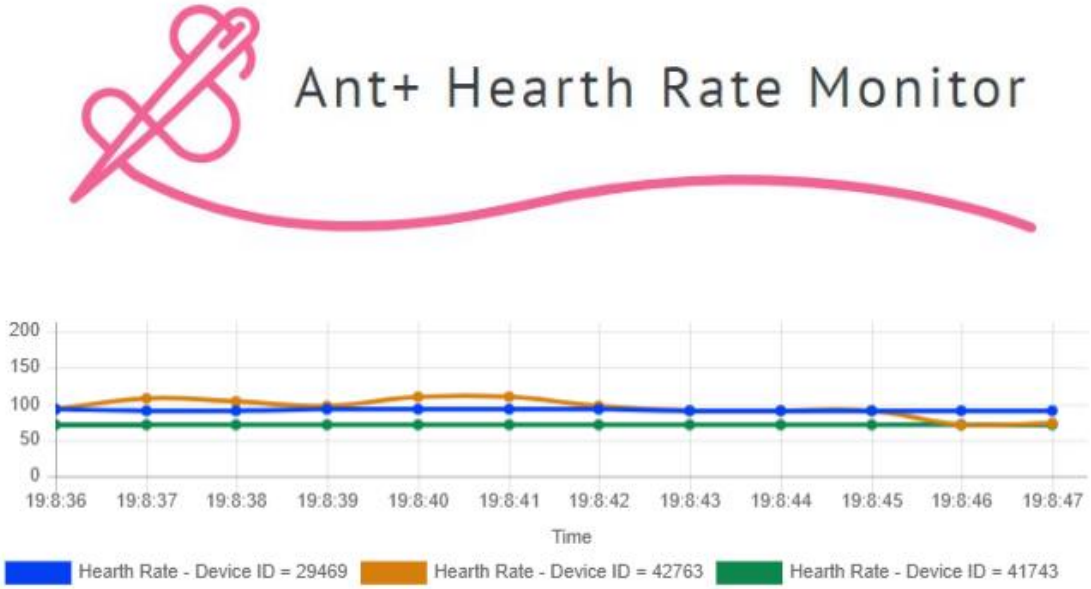
4. IMPLEMENTACIJA ANT+ PROTOKOLA POMOĆU NODE.JS

Implementacija ANT+ protokola može se izvršiti na više načina. ANT+ Alliance nudi programska rješenja kroz C++ programski jezik unutar "Microsoft Visual Studio okruženja". U slučaju kada bi htjeli napraviti Internet aplikaciju, gdje korisnici mogu spremati podatke na udaljeni server, morali bi se poslužiti drugim alatima. U tu svrhu upotrijebiti će se Node.js izvršno okruženje. Node.js je izvršno okruženje koje izvršava JavaScript kod izvan internet pretraživača. Ono omogućava korisnicima izvršavanje JavaScript programa na strani servera dok se obrađeni podaci šalju korisniku preko internet pretraživača. Ovime su reducirani zahtjevi prema performansama korisnikovog računala te kvaliteta usluge najvećim djelom ovisi o kvaliteti samoga servera [4].

Vizualizacija podataka u okviru ovog rada ostvarena je unutar Internet pretraživača primjenom grafičkog alata *Charts.js*. *Chart.js* je *JavaScript* alat za izradu grafikona s mnoštvom mogućnosti za dizajn i interakciju na samim grafikonima. Spremanje podataka obavlja se primjenom tekstualnih datoteka u JSON formatu. JSON (*JavaScript Object Notation*) je format otvorenog standarda koji koristi podatkovne objekte za spremanje podataka. Objekti se sastoje od imena i vrijednosti. Podaci senzora će se spremati u sljedećem JSON obliku:

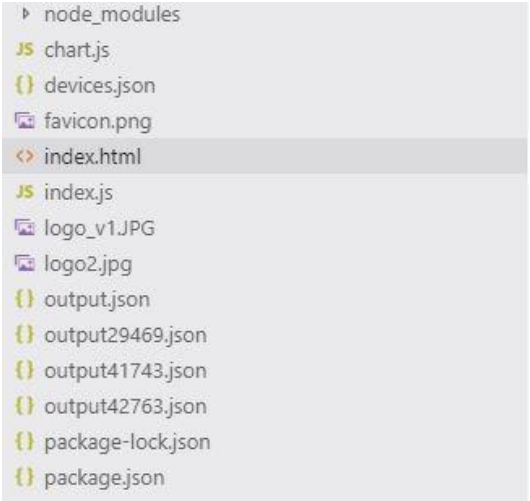
[{"hr":10,"time":"start"},{"hr":0,"time":"12:36:13"},{"hr":0,"time":"12:36:14"},{"hr":0,"time":"12:36:15"},{"hr":0,"time":"12:36:16"},{"hr":0,"time":"12:36:17"},{"hr":0,"time":"12:36:19"},{"hr":0,"time":"12:36:20"},{"hr":0,"time":"12:36:22"},{"hr":0. . . . . .

JSON datoteke svakog senzora se pišu i čitaju svake sekunde. Na slici 9. prikazan je izgled vizualizacijskog sučelja.



***Slika 9**. Izgled vizualizacijskog sučelja ANT+ Hearth Rate monitora*

Struktura programa postavljena je tako da se programi *chart.js* i *index.js* pozivaju unutar *index.html* datoteke. Program *chart.js* služi za vizualizaciju podataka dok program *index.js* služi za prikupljanje i spremanje mjerenja senzora u bazu podataka. Datotečna struktura programa prikazana je na slici 10. Program je nadopunjen s dodatnim datotekama kao što su slike i dodatne postavke okruženja. JSON datoteke se spremaju unutar glavnoga direktorija.



Slika 10. Datotečna struktura programa

Uz standardne dijelove HTML zaglavlja u ovoj datoteci se dodatno pozivaju JavaScript program chart.js te se uključuju biblioteke za obradu JSON datoteka i biblioteke za rad sa Chart.js skriptama:

```
<script src="http://ajax.googleapis.com/ajax/libs/jquery/1.7.1/jquery.min.js"
type="text/javascript"></script>
<script src="https://cdnjs.cloudflare.com/ajax/libs/Chart.js/2.1.4/Chart.min.js">
</script>
<script  src = "chart.js">
```

Unutar chart.js datoteke nalaze se funkcije za prikaz podataka na grafikonu te funkcije za čitanje JSON datoteka. Program u radu stvara dva tipa JSON datoteka a to su:

1. Podaci o uređajima – „devices.json“

Za ostvarivanje komunikacije između ANT+ uređaja i računala potrebno je da računalo poznaje identifikator uređaja. Podaci o identifikatoru uređaja mogu se pronaći na stranicama proizvođača ili se skeniraju putem aplikacije. Za tri navedena uređaja u ovom radu sadržaj JSON datoteke je sljedeći:

```
[{"device":29469},{"device":42763},{"device":41743}]
```

Podaci ove datoteke mogu se unositi ručno ili preko neke druge aplikacije. Broj uređaja zapisanih unutar datoteke određen je brojem kanala koji USB ANT+ uređaj podržava jer se svaki uređaj spaja na posebni kanal.

2. Mjerenja senzora – „output.json“

Senzorski podaci se spremaju kao uređeni par vremena mjerenja i vrijednosti mjerenja (otkucaji srca). Naziv JSON datoteke određen je identifikatorom uređaja pa primjerice sadržaj datoteke „output29469.json“ bio bi sljedeći:

```
{"hr":10,"time":"start"},{"hr":0,"time":"12:33:23"},{"hr":0,"time":"12:33:24"},{"hr":70,"time":
"12:33:25"},{"hr":70,"time":"12:33:26"},{"hr":70,"time":"12:33:27"},...
```

Vrijeme mjerenja se zapisuje u formatu *h:m:s* izuzev prvoga mjerenja koje je označeno s vrijednošću *start*. Time se označava početak rada senzora. Chart.js datoteka se sastoji od jedne globalne varijable *HRChart* i dvije funkcije: *getDevice()* i *getData().* Varijabla *HRChart* predstavlja definiciju samoga grafikona na kojemu će se vizualizirati podaci mjerenja. Osim postavki za dizajn i funkcionalnosti grafikona potrebno je definirati vrijednosti apscise i ordinate samoga grafikona preko *data* varijable. *Labels* varijabla predstavlja oznake na apscisi (vremenski trenutci mjerenja) dok *datasets* predstavlja vrijednosti mjerenja senzora. Kod inicijalizacije, vrijednosti *data* varijable ostavljene su prazne:

```
data: {
    labels: [' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',],
    datasets: []
  }
```

Funkcijom *getDevice()* se očitava sadržaj datoteke *devices.json* tako da se dinamički dodaju novi skupovi podataka unutar varijable *HRChart*. Također se za svaki novi uređaj dodaje legenda ispod grafikona radi lakšeg prepoznavanje uređaja. Ova funkcija je periodična, te se izvršava svake sekunde. Funkcijom *getData()* se čitaju mjerenja senzora te iz JSON datoteka. Podaci se zapisuju u varijablu *HRChart*. Ova funkcija se također izvršava svake sekunde. Iako se mjerenja senzora uzimaju u prosjeku svako 0.25 sekundi, ovdje se zapisuje svako četvrto mjerenje da bi se reduciralo zagušenje podataka te povećala preglednost čitanja podataka.

Unutar index.js datoteke, osim uključivanja node.js modula i postavki servera, očitavaju se podaci uređaja koji su spojeni na ANT+ USB uređaj. Prvo se mora ostvariti veza između svakoga ANT+ uređaja i računala funkcijom *sensor.attach()* i to za više senzora na sljedeći način:

```
stick.on('startup', function () {
    for(var j = 0; j < jsonDevice.length; j++){
      sensor[j].attach(j+1, jsonDevice[j].device);
    }});
```

Funkcijom *sensor.attach()* se ostvaruje dvosmjerna veza između senzora i računala tako da svaki uređaj komunicira preko zasebnoga kanala. Ostvarena veza je aktivna sve dok korisnik ne prekine rad programa. Nakon definicije korištenih uređaja potrebno je očitati mjerenja uređaja te podatke spremiti u JSON datoteku. U tu svrhu koristi se funkcija *senzor.on()* i to na sljedeći način:

```
for(var j = 0; j < jsonDevice.length; j++){
  sensor[j].on('hbData', function (data) {
    var today = new Date();
    var tim = today.getHours() + ":" + today.getMinutes() + ":" +
    today.getSeconds();

    appendObject({"hr":data.ComputedHeartRate,time":tim},'./output'+
    data.DeviceID+'.json');
});}
```

Ova funkcija posebno čita svaki senzor te zapisuje podatke u pripadajuću JSON datoteku. Vrijeme očitanja senzora uzima se unutar samoga programa u trenutku mjerenja te se skupa s vrijednosti mjerenja zapisuje u datoteku.

5. ZAKLJUČAK

Rezultati analize ANT protokola pokazali su da je ANT robusna i jednostavna. Upravljanje i nadzor ANT uređaja pokazao se kao jednostavan proces gdje je zahvaljujući doprinosu proizvođača, jako olakšana integracija ANT protokola unutar računalnih okruženja. Najveća prednost ANT protokola je u tome što su ANT uređaji jako energetski efikasni. To se i posebno pokazalo u izradi ovoga rada gdje baterije uređaja nisu ni jednom trebale biti napunjene u vremenu korištenja od tri mjeseca. Osim primjene ANT uređaja unutar sportske opreme, nudi se i prilika za korištenje senzora unutar nekritičnih industrijskih procesa zbog jake otpornosti ANT protokola na interferencije i mogućnosti integracije raznih mrežnih topologija. Primjenom tehnologija otvorenog koda prezentirana je primjena ANT+ uređaja u okviru ovog rada.

Literatura:

1 It just works! , https://www.thisisant.com/developer/, pristupljeno: lipanj, 2019.

2 Dynastream Innovations Inc., *ANT Message Protocol and Usage*, 5. izdanje, 2018.

3 Dynastream Innovations Inc., *ANT+ Common Pages*, 3. izdanje, 2018.

4 Node.js, https://nodejs.org/en/, pristupljeno: lipanj, 2019.

Podaci o autorima:

Roko Rogulj

Sveučilišni odjel za stručne studije, Sveučilište u Splitu

Kopilica 5, 21000, Split

e-mail: roko.rogulj@hotmail.com

Diplomirao na Sveučilišnom odjelu za stručne studije Sveučilišta u Splitu 2019. godine na smjeru Industrijska elektronika i stekao naziv stručni specijalist inženjer elektrotehnike. Kao student više godina radio u tvornici Cemex unutar odjela električnog održavanja te kao demonstrator na vježbama iz više stručnih predmeta iz područja elektrotehnike. Trenutno zaposle kao projektant u proizvodnji u tvrtki Brzoglas d.o.o u Splitu.

Ivo Kovačević

FESB, Split

Ruđera Boškovića 32, 21000, Split

e-mail: ikovacevic96k@gmail.com

Student 2. godine sveučilišnog diplomskog studija Računarstva na Fakultetu elektrotehnike, strojarstva i brodogradnje u Splitu.

dr.sc. Tonko Kovačević

Sveučilišni odjel za stručne studije, Sveučilište u Splitu

Kopilica 5, 21000, Split

e-mail: tkovacev@oss.unist.hr

5G I ZDRAVLJE

prof.dr.sc. Dina Šimunić

1. UVOD

Uvođenje tehnologije pete generacije pokretne telefonije, tzv. 5G, već je započelo u Europi i Republici Hrvatskoj. Budući da 5G, kao i svaka nova bežična tehnologija, donosi uvijek nove reakcije javnosti vezane uz moguće učinke elektromagnetskih polja, tako je i ovaj put iskorištena tema 5G kako bi se lažno alarmirala javnost o povezanosti 5G i COVID-19, što je rezultiralo nevjerojatnom reakcijom javnosti i uništenjem nekoliko baznih postaja u Velikoj Britaniji za vrijeme perioda karantene vezane uz COVID-19. Sjetimo se iste pojave alarmantnih vijesti koja je počela s pojavom pokretne telefonije tzv. druge generacije (2G, tj. GSM tehnologija) ranih devedesetih godina prošlog stoljeća, preko istih takvih vijesti s trećom generacijom (3G, tj. UMTS tehnologija) do početka primjene četvrte generacije (4G, tj. LTE tehnologija). Javnost u skladu s vijestima reagira, unatoč činjenici da su vijesti lažne i potpuno neutemeljene, jer se vijesti danas šire brzinom svjetlosti uslijed postojeće sveopće globalne povezanosti društvenim mrežama.

Stoga će se u ovom osvrtu kratko diskutirati elektromagnetsko zračenje, te neionizirajuća priroda elektromagnetskog zračenja koje se primjenjuje u pokretnoj telefoniji.

2. SPEKTAR ELEKTROMAGNETSKOG ZRAČENJA

Elektromagnetsko zračenje se sastoji od neionizirajućeg i ionizirajućeg dijela. Neionizirajući dio počinje sa statičkim dijelom bez oscilacija (frekvencija je nula [Hz]) i završava sa svjetlosnim područjem, iza kojega se računa da počinje ionizirajući dio elektromagnetskog spektra. Granica neionizirajućeg i ionizirajućeg zračenja nije određena frekvencijom, već dovoljnom energijom za ionizaciju atoma kisika i vodika. Naime, jedino dovoljna energija elektromagnetskog zračenja može razbiti kemijsku vezu. Ta energija iznosi 12eV. Neionizirajuće zračenje je definirano s kvantom energije elektromagnetskog zračenja manjim od 1eV. Budući da postoji najmanje jedna dekada razlike, potpuno je sigurno da ono ne može uzrokovati ionizaciju.

Striktno konzervativno neionizirajuće zračenje s energijom ispod 1eV uključuje radiofrekvencijski dio i sve frekvencijske pojase kao što su VLF (pojas vrlo niskih frekvencija), LF (pojas niskih frekvencija), MF (pojas srednjih frekvencija), HF (pojas visokih frekvencija), VHF (pojas vrlo visokih frekvencija), UHF (pojas ultra visokih frekvencija), SHF (pojas super visokih frekvencija) i EHF (pojas ekstra visokih frekvencija). Svi navedeni pojasevi uključuju sljedeće primjene: AM i FM radijski prijenos, mikrovalove (300MHz do 300GHz) s pokretnom telefonijom i drugim bežičnim prijenosom tipa Wi-Fi, Bluetooth i slično. Tu također pripada i dio infracrvenog područja. Frekvencija koja odgovara kvantu energije elektromagnetskog zračenja od 1eV je 241THz, što odgovara valnoj duljini od 1,2μm. Kratkovalno infracrveno područje obuhvaća područje od 1,0 do 3μm, tako da dio tog područja i blisko infracrveno područje (0,7-1μm) ne pripada više u striktno konzervativno neionizirajuće zračenje, kao ni ostatak optičkog dijela elektromagnetskog spektra (vidljiva svjetlost i ultraljubičasti dio). U klasično ionizirajuće zračenje pripadaju X-zrake, g-zrake i kozmičke zrake (Tablica 1).

Tablica 1. Neionizirajući i ionizirajući dio elektromagnetskog spektra

Radiofrekvencijski dio	**Optički dio**			**X-zrake**	**g-zrake**	**Kozmičke zrake**
VLF	IC	Vidljivi dio	UV			
LF						
MF						
HF						
VHF						
UHF						
SHF						
EHF						

Tablica 2 prikazuje izračunatu energiju fotona s Planckovom formulom za karakteristične frekvencije u [J] i [eV]. Frekvencija 241THz je odabrana radi izračunate energije fotona od 1eV, 384THz i 789THz, jer su to početak i kraj svjetlosnog područja, te 3PHz, jer energija fotona iznosa 12eV, što je definitivno početak ionizirajućeg dijela elektromagnetskog spektra.

Tablica 2. Izračunata energija fotona na karakterističnim frekvencijama

Frekvencija vala	**Energija fotona [J]**	**Energija fotona [eV]**
300MHz	1,8 * 10-25	1,2*10-6
3GHz	1,8 * 10-24	1,2*10 -5
30GHz	1,8 * 10-23	1,2*10-4
3THz	1,8 * 10-21	1,2*10-2
241THz	1,6 * 10-19	1
384THz	2,5 * 10-19	1,6
789THz	5,2 * 10-19	3,3
3PHz	1,8 * 10-18	12

Energija kemijskih veza u organskim molekulama je između 3 i 5eV. U trodimenzionalnoj strukturi bjelančevina važna je molekula vodika s energijom nešto manjom od 1eV. Stoga je bitno vidjeti i shvatiti Tablicu 2 iz koje je izravno razvidno da niti jedno neionizirajuće zračenje ne može imati nikakav utjecaj na kemiju živih tkiva. Jednostavno, ne postoji rizik od degeneracije stanica uslijed upotrebe bilo kakve bežične tehnologije, a time i pokretnih telefona, dok god su oni operativni u već definiranom striktno konzervativnom neionizirajućem području elektromagnetskog spektra. S druge strane, UV zračenje i zračenje na frekvencijama iznad UV zračenja u naravi doista imaju potencijal utjecati na kemiju naših stanica. Izvrstan primjer je rendgensko zračenje.

3. FREKVENCIJSKI POJASEVI I 5G

Tablica 3. prikazuje frekvencijske pojaseve za globalnu primjene 5G, koji apsolutno pripadaju u neionizirajući dio elektromagnetskog spektra.

Tablica 3. Frekvencijski pojasevi za primjenu 5G u svijetu i u Republici Hrvatskoj

Naziv frekvencijskog pojasa	**5G frekvencijski pojas svijet**	**5G frekvencijski pojas Republika Hrvatska**
Pojas niskih frekvencija	Ispod 1GHz, tipično 600/700MHz	700MHz
Pojas srednjih frekvencija	Između 3 i 5GHz	3,6GHz
Pojas visokih frekvencija	Između 20 i 100GHz	26GHz

Pojas niskih frekvencija je interesantan, jer omogućuje mnogo rjeđi raspored područja pokrivanja baznih postaja od onog na višim frekvencijama. To ujedno omogućuje pokrivanje istog zemljopisnog područja s mnogo manjim brojem baznih postaja.

Pojas srednjih frekvencija je interesantan radi većeg raspoloživog frekvencijskog spektra.

Pojas visokih frekvencija je interesantan, jer omogućuje veliku brzinu s malim kašnjenjem (temeljni zahtjevi 5G mreže), iako na vrlo malim udaljenostima (10 do 150m).

4. LEGISLATIVA REPUBLIKE HRVATSKE U PODRUČJU ZAŠTITE OD NEIONIZIRAJUĆIH ELEKTROMAGNETSKIH POLJA

U Republici Hrvatskoj se Zakonom o zaštiti od neionizirajućeg zračenja (NN 91/2010) [1] i Pravilnikom o zaštiti od elektromagnetskih polja (NN 146/14) [2] regulira ovo područje. Pravilnikom se propisuju granične razine elektromagnetskog polja, postupci njihovog provjeravanja, izvori elektromagnetskog polja (za koje je obvezna dozvola ministra zdravlja), posebni zahtjevi za uređaje, postrojenja i građevine koje su izvori elektromagnetskog polja, te uvjeti za dobivanje ovlasti za obavljanje tih postupaka.

5. ZAKLJUČAK

Prikazani su ključni momenti potrebni za razumijevanje razlike ionizirajućeg i neionizirajućeg dijela elektromagnetskog spektra. Svi bežični komunikacijski sustavi koji se primjenjuju na teritoriju Republike Hrvatske su temeljeni na neionizirajućem zračenju, koje ne može utjecati na ljudsko tijelu u vidu promjene kemijskog sastava tijela. Republika Hrvatska ima uređen sustav vezan uz kontinuiranu provjeru razina neionizirajućih elektromagnetskih polja na cjelokupnom teritoriju, te je stoga zabrinutost pojedinih građana doista nepotrebna.

Literatura:

1 Zakon o zaštiti od neionizirajućeg zračenja (NN 91/2010), br. 2571

2 Pravilnik o zaštiti od elektromagnetskih polja (NN 146/2014), br. 2740

3 Pravilnik o izmjenama i dopunama Pravilnika o zaštiti od elektromagnetskih polja (NN 31/2019), br. 627

Podaci o autoru:

Prof.dr.sc. Dina Šimunić

Sveučilište u Zagrebu

Fakultet elektrotehnike i računarstva

Unska 3, 10000 Zagreb

e-mail: Dina.Simunic@fer.hr

Dina Šimunić je rođena u Zagrebu, gdje je završila osnovnu i srednju školu. Diplomirala je i magistrirala na Elektrotehničkom fakultetu u Zagrebu 1985, odnosno 1992. Godine na smjeru Radiokomunikacije i profesionalna elektronika. Doktorsku disertaciju obranila je na Tehničkom sveučilištu u Grazu, Austrija, 1995. godine. Redoviti je profesor Fakulteta elektrotehnike i računarstva Sveučilšta u Zagrebu. Na Zavodu za radiokomunikacije nositelj je i predavač na prediplomskom diplomskom te doktorskom studiju. Profesorica Šimunić mentorica je više od 100 uspješno obranjenih diplomskih radova studenata. Predsjednica je HZN TO Telekomunikacije, a u CEN/CENELEC-u je predstavnica HZN-a za područje edukacije o normizaciji. Voditeljica je grupe “Strategy towards the development of materials for education about standardization, as well as identification of common requirements“ u Međunarodnoj telekomunikacijskoj uniji (ITU).

My profession.
My organization.
My IEEE.

Discover the benefits of IEEE membership.

Join a community of more than 365,000 innovators in over 150 countries. IEEE is the world's largest technical society, providing members with access to the latest technical information and research, global networking and career opportunities, and exclusive discounts on education and insurance products.

Join today
www.ieee.org/join

IEEE

CASE d.o.o.

RAZVOJ POSLOVNIH
I INFORMATIČKIH SUSTAVA

CASE 2020

Zlatni pokrovitelj

citus grupa
Internet Development Company

Srebrni pokrovitelj

foi

Brončani pokrovitelji

konto d.o.o.
Projektiranje informacijskih sustava

Veleučilište u Rijeci
Collegium Fluminense

TVZ
POLYTECHNICUM ZAGRABIENSE

ODJEL ZA INFORMATIKU
SVEUČILIŠTE U RIJECI

Sveučilište Jurja Dobrile u Puli

FIPU
Fakultet informatike u Puli

24.02.-25.02.2020, Zagreb

ORGANIZATOR

CASE d.o.o.

ORGANIZACIJSKI I PROGRAMSKI ODBOR

TOMISLAV BRONZIN mag. ing. el.
ANTE POLONIJO
MISLAV POLONIJO
ZLATKO SIROTIĆ univ.spec.inf.

ZLATNI PARTNER

SREBRNI PARTNER

BRONČANI PARTNER

Izdavač, priprema i tisak:
CASE d.o.o., Rijeka

Urednik:
Mislav Polonijo

ISSN 1334-448X
UDK 007.5 : 621.39 : 681.324

Copyright ©"Case", Rijeka, 2020

Sva prava pridržana. Niti jedan dio zbornika ne smije se reproducirati u bilo kojem obliku ili na bilo koji način, niti pohranjivati u bazu podataka bez prethodnog pismenog dopuštenja izdavača, osim u slučajevima kratkih navoda u stručnim člancima. Izrada kopija bilo kojeg dijela zbornika zabranjena je.
Case d.o.o., Antuna Barca 12, 51000 Rijeka
tel: 051/217-875, fax: 051/218-043, e-mail: case@case.hr, internet: www.case.hr

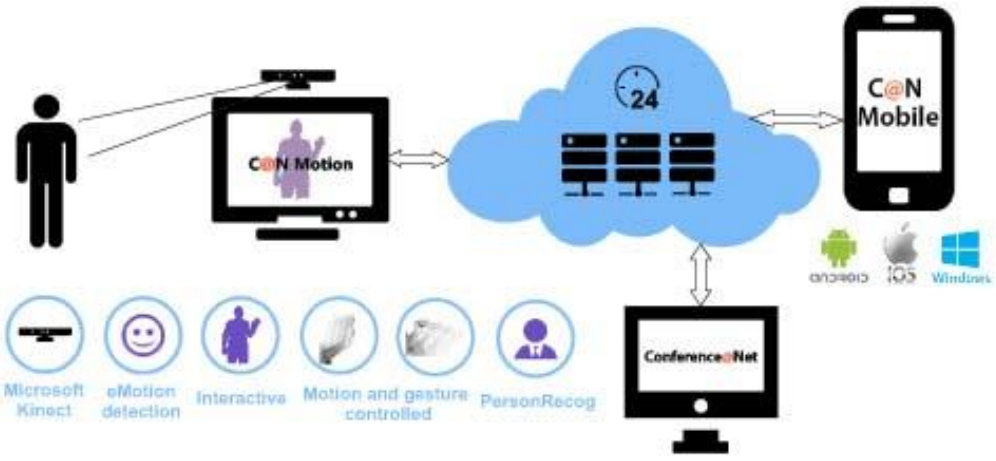
citus grupa
Internet Development Company

Conference@Net

inovativno rješenje za upravljanje organizacijom konferencija

integrirano s društvenim mrežama i dostupno na uređajima upravljanim dodirom

www.conferenceatnet.com



Microsoft Partner

Silver Application Development
Silver Data Platform
Silver Midmarket Solution Provider
Silver Mobility
Silver Small Business
Cloud Accelerate

CITUS d.o.o.
Dragutina Golika 63, Zagreb
http://www.citus.hr
citus@citus.hr

SADRŽAJ

Darjan Baričević, Božo Čulo, Matija Novak
RAZVOJ VOĐEN TESTIRANJEM U PROGRAMSKOM JEZIKU PHP 2.5

Dragutin Kermek, Matija Novak
QUO VADIS ... JAVA 2.13

doc. dr. sc. Marko Horvat
NOVE OPASNOSTI OD UMJETNE INTELIGENCIJE U ZAŠTITI OSOBNIH PODATAKA 2.21

Mirko Sršen bacc. ing. techn. inf., izv. prof. dr. sc. Tihomir Orehovački
RAČUNALNE IGRE: OD IDEJE DO IZDAVAČA 2.29

Ivan Miljančić, dr.sc. Marin Kaluža
USPOREDBA CROSS-PLATFORM FRAMEWORKA ZA IZGRADNJU MOBILNIH APLIKACIJA 2.43

Mirko Sršen bacc. ing. techn. inf., izv. prof. dr. sc. Tihomir Orehovački
SUSTAV ZA KONTROLU PROIZVODNJE U PIVARSKOJ INDUSTRIJI 2.53

Zlatko Sirotić
POSTOJI LI SAMO JEDNA "ISPRAVNA" ARHITEKTURA WEB POSLOVNIH APLIKACIJA 2.65

RAZVOJ VOĐEN TESTIRANJEM U PROGRAMSKOM JEZIKU PHP

TEST DRIVEN DEVELOPMENT IN PROGRAMMING LANGUAGE PHP

Darjan Baričević, Božo Čulo, Matija Novak

SAŽETAK

Agilni razvoj (eng. agile development) je metodologija razvoja softvera koja se sve više koristi. Temelji te metodologije bez kojih agilni razvoj nije moguć su: razvoj vođen testiranjem (eng. Test Driven Development - TDD), refaktoriranje (eng. refactoring), jednostavni dizajn (eng. simple design) i programiranje u paru (eng. pair programming). U ovom radu fokus je na razvoju vođenom testiranjem jer bez njega nije moguće ili je vrlo teško provoditi refaktoriranje, a bez refaktoriranja teško je postići jednostavni dizajn. Iako je fokus rada TDD, u radu su ukratko opisane i vezane discipline kao što su: refaktoriranje, uzorci dizajna, programiranje u paru i sl. Kako bi se programerima približilo način funkcioniranja TDD u praksi, dan je primjer korištenja TDD-a u razvoju jednog segmenta web aplikacije u programskom jeziku PHP. No bez obzira što se u primjerima koristi PHP koncepti TDD-a primjenjivi su na bilo kojem drugom programskom jeziku.

ABSTRACT

Agile development is a software development methodology that is increasingly used. The foundations of this methodology without which agile development is not possible are: Test-Driven Development (TDD), refactoring, simple design and pair programming. In this paper, the focus is on test driven development, because without it, refactoring is not possible or it is very difficult to do, and without refactoring it is difficult to achieve simple design. Although the focus is on TDD, the paper also briefly describes related disciplines such as: refactoring, design patterns, pair programming, etc. To give developers an idea of how TDD works in practice, an example is given where TDD is used to develop a segment of a web application in the programming language PHP. However, even though PHP is used in the examples, TDD concepts are applicable in any other programming language.

1. UVOD

Teško je reći kada je agilni razvoj započeo, no pojam "agile" izabran je kao najbolji pojam da opiše "novi" proces razvoja softvera 2001. godine od strane 17 programera/ki na skupu "The Light Weight Process Summit" u Snowbird, Utah, SAD. Među sudionicima bili su: Martin Fowler autor knjige "Refactoring", Kent Beck izumitelj TDD-a, Robert C. Martin autor "Clean code" serijala knjiga, Andy Hunt i Dave Thomas autori knjige "The pragmatic programmer", Ward Cunningham kreator Wiki sustava, itd. Danas agilni razvoj je metodologija koja se sve više koristi, a jedan od temelja te metodologije, prema Robert C. Martinu [1], bez koje agilni razvoj nije moguć je razvoj vođen testiranjem (eng. Test Driven Development – TDD). Naravno uz TDD tu su i drugi elementi poput: refaktoriranja (eng. refactoring), jednostavnog dizajna (eng. simple design), programiranja u paru (eng. pair programming), itd. Iako su i drugi elementi važni bez TDD nije moguće ili je vrlo teško provoditi refaktoriranje, a bez refaktoriranja teško je postići jednostavni dizajn.

Osim agilne metodologije važnost TDD-a ukazuje pokret pod nazivom „Software Craftsmanship" koji je započeo u studentom 2008. godine u Chicagu. Inicijalna ideja je bila definirati novi skup vrijednosti koje nadograđuju vrijednosti koje su definirali inicijalni osnivači agilne metodologije. [2] Potrebno je napomenuti da taj pokret ne promovira direktno nikakve discipline već se zalaže da programeri konstantno traže bolje načine za rješavanje određenih problema. Usprkos tome, u ovom trenutnku Sandro Mancuso (atuor knjige [3]) piše da je TDD odlična disciplina za rješavanje problema kako skratiti vrijeme testiranja sofrvera. Primjenom TDD-a dobivamo da na jedan klik gumba u svega nekoliko sekundi dobijemo odgovor da li je program ispravan ili ne. Stoga se može reći da je TDD disciplina koja riješava problem osiguravanja kvalitete softvera.

No neovisno o blio kojoj metodologiji „dužnost svakog programera je pisanje ispravnog koda, ali i davanje dokaza da je kod ispravan ... Ukoliko ne primjenjujemo TDD moramo naći neki drugi način da testiramo svoj kod i osiguramo njegovu ispravnost. Nikako ne bi smijalo desiti da se program isporučuje bez da je kompletno testiran." [4]

Cilj ovog rada je približiti disciplinu razvoja vođenog testiranje programerima i dati primjer korištenja TDD-a u programskom jeziku PHP. Ostatak rada strukturiran je kako slijedi. Poglavlje 2 opisuje što je i kako započeti sa razvojem vođenim testiranjem i daje opis veze prema ostalim disciplinama koje su važne, u poglavlju 3 daje se primjer korištenja TDD-a u programskom jeziku PHP počevši od instalacije i pripeme razvojnog okruženja. U poglavlju 4 poglavlje dan je zaključak.

2. RAZVOJ VOĐEN TESTIRANJEM

U prethodnom poglavlju opisana je važnost TDD-a, no nije jasno rečeno što je to. Razvoj vođen testiranjem jest metoda razvoja softvera gdje programer piše kod na način da prvo piše test za kod kojeg još nije napisao. Iako to zvuči kontra produktivno i čini se da će usporiti programera dugoročno gledano primjenom TDD-a razvoj se ubrzava. TDD se bazira na pisanju jediničnih testova, a oni daju programeru [5]: sigurnost da je program ispravan, smanjuju broj grešaka koje radi, hrabrost da mijenja kod bez straha, dokumentaciju najniže razine kako kod radi, a ponekad čak daju bolji dizajn sustava.

2.1. Opis rada

Za razliku od ostalih disciplina koje su bile aktualne u vrijeme njegova nastanka, razvoj vođen testiranjem ima vrlo jednostavan princip rada koji se temelji na vrlo kratkom razvojnom ciklusu unutar kojeg je se potrebno pridržavati nekoliko pravila. Razvojni ciklus se sastoji od samo 5 koraka [6]:

1. Napiši test
2. Pokreni sve testove i pobrini se da novi test padne
3. Napravi promjene
4. Pokreni sve testove i pobrini se da svi prođu
5. Refaktoriraj da ukloniš duplicirani kod

Razvoj svake nove funkcionalnosti unutar projekta započinje s testom, i taj je korak zapravo najteži za priviknuti se pogotovo za razvojne programere koji dolaze iz drugih razvojnih procesa. Kako napisati test za nešto što još uvijek nije izgrađeno? To zahtjeva kristalno jasno razumijevanje svih korisničkih zahtjeva sustava, a postiže se osmišljavanjem slučajeva korištenja (engl. use case) i korisničkih priča (engl. user stories), od jednostavnijih do složenijih. Ovakav pristup omogućuje razvojnim programerima da se fokusiraju na specifikaciju, a ne validaciju sustava – što je i jedan od ciljeva TDD-a.

U drugom koraku se osigurava da novi test nije defektan – odnosno mogućnost da novi test prođe unatoč tome što on treba ispitati novu funkcionalnost koja uopće nije implementirana. Uz to, produkcijski kôd je uvijek potrebno prilagođavati testovima, a ne obrnuto jer takav pristup rezultira pisanjem defektnih testova koji će uvijek prolaziti. Prve promjene u produkcijskom kodu dozvoljeno je raditi tek u trećem koraku, a u četvrtom se osigurava da nova funkcionalnost ispravno radi.

Jedna zanimljivost koju valja spomenuti je ta da je u trećem koraku, kako Beck [6] navodi, dozvoljeno raditi sve potrebne „grijehe“ samo s ciljem da test prođe. Fokus je na tome da testovi prođu pa je stoga pisanje „prljavog“ kôda sasvim nebitna stvar koja se rješava tek poslije. U posljednjem se koraku provodi refaktoriranje kako bi se počistio sav „nered“ koji je programer ostavio iza sebe u trećem koraku, a pristup refaktoriranju nešto je drugačiji nego u ostalim metodologijama (više o ovome u idućem poglavlju).

Robert C. Martin [7] je iz gore navedenih koraka izveo 3 pravila (ili zakona) kojih se svaki učenih TDD-a mora pridržavati [4]:

1. Ne smije se pisati produkcijski kod osim da zadovolji test koji ne prolazi. Drugim riječima ne smijemo pisati produkcijski kod dok nemamo test koji ne prolazi (tj. pada).
2. Smije se napisati samo toliko test koda da se demonstrira pad. Kod koji se ne kompilira se računa kao pad testa.
3. Smije se napasti samo toliko produkcijskog koda da prođe trenutno padajući test.

Na prvi pogled ova 3 zakona nemaju smisla ukoliko ih se pokušava pridržavati u potpunosti jer će razvojni programer ponekad pisati veće testove, ponekad više produkcijskog koda i sl. Ideja je da se reducira vrijeme između pisanja testova i pisanja produkcijskog koda na što manju razinu, odnosno da se smanji interval ciklusa.

2.2. Veza prema čistom kodu, uzorcima dizajna i refaktoriranju

Utemeljitelj ekstremnog programiranja (engl. extreme programming) Ron Jeffries rekao je kako TDD rezultira „čistom kôdu koji radi“[8]. Drugi dio ovog izraza je sasvim logičan s obzirom na brojna testiranja u pristupu, ali prvi dio je zvuči pomalo kontradiktorno. Za razliku od razvoja vođenog arhitekturom (engl. architecture-driven development) u kojem se čista arhitektura te samim time i čisti kod ostvaruje direktno provođenjem metodike, u TDD-u se čist kôd ostvaruje indirektno – recimo to tako. Već je rečeno da kraj svakog ciklusa završava sa refaktoriranjem i uklanjanjem dupliciranog koda, ali se također i prakticira pisanje čistih testova (engl. clean tests).

Čiste testove karakterizira čitljivost (engl. readability) , standardizirana notacija, jedna tvrdnja (engl. assert) po testu te najbitnije – ispitivanje samo jednog koncepta po testu. Ponekad je teško ispitati samo jednu funkciju ako ta funkcija poziva drugu funkciju, ta druga funkcija neku treću funkciju itd. To se događa zbog uske povezanosti (engl. tight coupling) objekata unutar koda i zahtjeva promjenu u dizajnu te primjenu uzoraka dizajna koji promoviraju slabo povezivanje objekata kao što je Dependency Injection. Isto tako, duplicirani kôd moguće je eliminirati primjenom klasnih uzoraka dizajna koji koriste nasljeđivanje kao što su Template Method i Factory Method. Postoje i uzorci koji se ne tiču produkcijskog kôda, već služe za strukturiranje testova. Takav je i Build-Operate-Check [9] koji test dijeli na 3 koraka: izgradnja, operacija, provjera. U prvom se koraku grade podaci (slanje API poziva), u drugom provode operacije nad izgrađenim podacima, a u trećem ispituje očekivani rezultat. Kada su testovi tako strukturirani, postaju čitljiviji i omogućuju da ga razumije i onaj koji ga prvi puta vidi.

Već je rečeno da se refaktoriranje u TDD-u koristi na nešto drugačiji način. Općenito, refaktoriranje se radi s ciljem poboljšanja čitljivosti i smanjenja kompleksnosti produkcijskog kôda. Kod TDD-a je naglasak na čitljivost i kompleksnost

testnog kôda, a onaj produkcijski se može refaktorirati sve dok testovi prolaze. Naravno, oni su usko vezani i uvijek će utjecati jedno na drugo ali činjenica je da razvojni programeri ne vole refaktorirati jer se boje da će „pokidati“ nešto u produkciji. U TDD-u taj strah ne postoji jer se iza promjene nalaze stotine testova koji odmah nakon promjene mogu pokazati da li nešto ne radi kako treba. Stoga, jedinični testovi su ono što čini dizajn sustava fleksibilnijim i održavanijim, a njegove komponente ponovno iskoristivima. Kod refaktoriranja se većinom koriste tehnike koje olakšavaju testiranje, pa su tako najčešće tehnike izvlačenja metoda (engl. extract method), izvlačenja sučelja (engl. extract interface) i migracije podataka (engl. migrate data).

Ono što je bitno naglasiti u svemu ovome je to da se ciklus TDD-a stalno ponavlja. Shodno tome, kôd se refaktorira neprestano, sustav postaje fleksibilniji, arhitektura čišća, pokrivenost kôda (engl. code coverage) raste, a strah od promjena – nestaje.

2.3. Kako započeti

Unatoč jasno definiranim koracima razvojnog ciklusa, početak rada s TDD-om može biti malo nezgodan. U drugim pristupima, razvoj svakog sustava započinje sa kreiranjem klase dok u TDD-u započinje sa kreiranjem testa. Prilikom kreiranja prvog testa postavlja se pitanje: što treba pokrivati test? Kada pišemo test, uvijek zamišljamo savršeno sučelje za operaciju. [6] Drugim riječima, uvijek se u testu ispituje najsavršeniji mogući slučaj kod ulaznih podataka s ciljem dobivanja očekivanog rezultata. Taj se slučaj možda i neće nikada dogoditi, ali uvijek je lakše krenuti od jednostavnih i nerealnih testova i postupno ići prema onim složenijim i realnijim.

Kako bi se učenici TDD-a lakše uhodali u cijeli proces i promijenili način razmišljanja, osmišljene su TDD kate. Kata na japanskom znači forma, a u borilačkim vještinama opisuje detaljne uzorke kretnji koje se vježbaju samostalno ili u paru s ciljem ostvarenja razine mišićne memorije (engl. muscle memory). Pojam kate u programskom inženjerstvu osmislio je Dave Thomas, a ideja je slična kao i u borilačkim vještinama – razviti mišićnu memoriju odnosno automatizirati sebe. Postoji brdo različitih kata dostupnih na webu, a svaka od njih ima svoj opis problema, korake u rješavanju i zaključak ili pouku. Kata se uči kao forma, a ne kao zaključak jer nije bitan zaključak kao takav već koraci koji su doveli do zaključka [10]. Kroz katu, učenici nauče kako razmišljati u pojedinim situacijama, a ponavljanjem kate iznova i iznova moguće je istrenirati um do te razine da automatski odgovara na probleme u dizajnu s kojima se pojedinac susreće. Eksperti TDD-a preporučuju da se jedna kata vježba svaki dan po 30 minuta i tako 1-2 tjedna, pa nakon toga opet druga kata i tako sve dok učenik ne počne instinktivno i bez razmišljanja pretvarati skup zahtjeva u dizajn odnosno rješenje. Prije nego krenemo opisivati znanja, vještine i stavove koje programer mora imati opisali bi smo nekoliko zabluda na koje smo naišli.

3. PROGRAMIRANJE U PHP-U

Kako bi se bolje objasnio razvoj vođen testiranjem u ovom će se poglavlju opisati primjena TDD-a u programskom jeziku PHP. Prvo će se opisati priprema razvojnog okruženja, zatim instalacija i konfiguracija dodtaka PHPUnit te sljedi kako pisati testove u njemu. Na kraju dan je jedan primjer koda koji je razvijen korištenjem TDD-a.

3.1. Priprema razvojnog okruženja

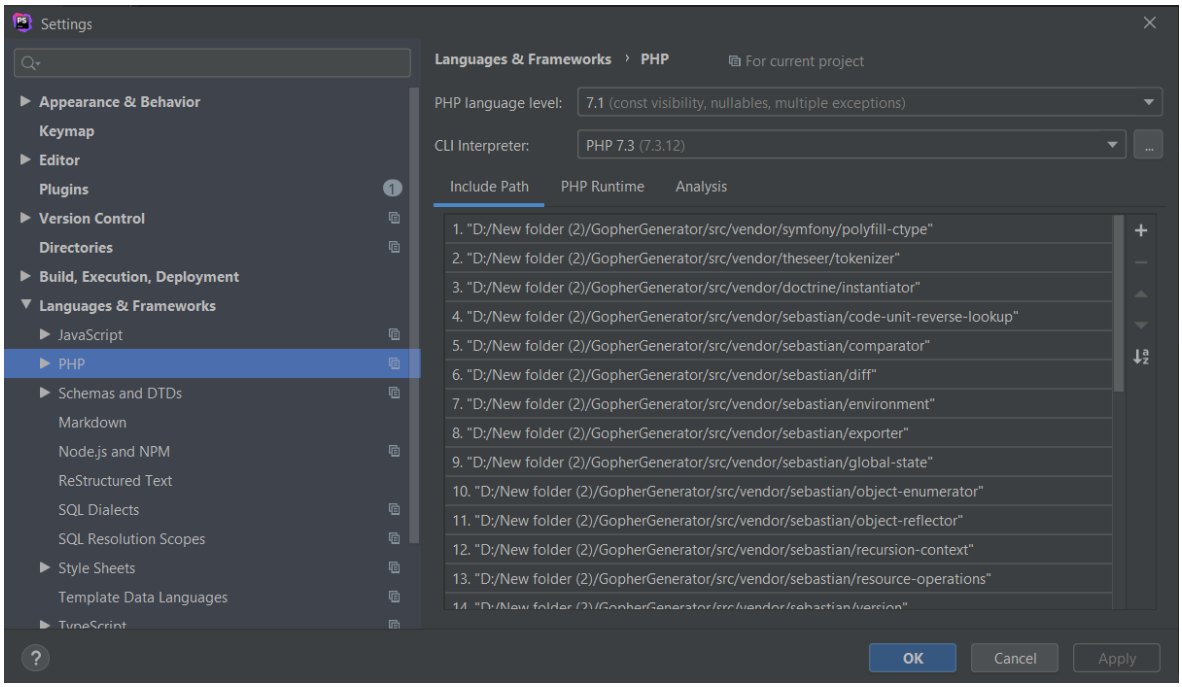
Prije provođenja bilo kakvog testiranja, u bilo kojem jeziku, pa tako i u PHP-u (engl. HyperText Preproccesor), potrebno je pripremiti razvojno okruženje. Kako je PHP popularan i često korišten jezik za razvoj Web (engl. World wide web) aplikacija, na raspolaganju stoji puno različitih razvojnih okruženja (engl. Integrated Development Environment - IDE). Kako ih je nemoguće sve navesti i testirati, ovaj rad će se isključivo fokusirati na IDE PhpStorm[1].

Prije instalacije PhpStorm, nužno je instalirati sam PHP. Kako bi postupak instalacije bio što lakši, preporuča se korištenje programskog paketa XAMPP[2] koji svojom instalacijom priprema sve nužne komponente za rad. XAMPP predstavlja Apache distribuciju koja sadrži MySQL i programske jezike PHP i Perl. Naravno, moguće je instalirati sve komponente individualno, ali time je postupak instalacije dosta dulji i teži.

PhpStorm je IDE koji je razvijen od poznate kompanije JetBrains. JetBrains uspješno posluje još od 2000. godine te osim PhpStorm-a, nudi široku paletu raznih dodataka i alata koji olakšavaju rad. Bitno je napomenuti da PhpStorm nije besplatan, ali ga je moguće testirati 30 dana besplatno. Osim toga, postoji puno alternativa koje su besplatne kao što su Visual Studio Code i NetBeans. Kako je PhpStorm isključivo fokusiran na PHP, dolazi sa svim instaliranim dodatcima koji su ključni za svakog razvojnog inženjera. PhpStorm nudi potporu za sve popularne okvire poput Symfony-a i Laravel-a, mogućnost brzog i lakog refaktoriranja koda (engl. Refactoring), traženja pogrešaka (engl. debugging), dopunjavanja koda (engl. Code completion), potpore za front-end tehnologije kao što su HTML, CSS, JavaSript itd. Postupak instalacije je vrlo jednostavan. Potrebno je preuzeti instalacijsku datoteku sa službene stranice, a zatim pokrenuti i instalirati. Prilikom instalacije potrebno je samo odabrati lokaciju instalacije i temu, odnosno izgled sučelja PhpStrom-a.

[1] https://www.jetbrains.com/phpstorm/

[2] https://www.apachefriends.org/index.html



Slika 1. Kofiguracijea PHP-a u PHPStorm razvojom okruženju

Nakon uspješne instalacije, još uvijek nije moguće pokretati PHP skripte. Naime, potrebno je podesiti verziju PHP-a i interpreter na komandnoj liniji (engl. Command Line Interpreter - CLI). To je moguće podesiti u postavkama (engl. Settings), u sekciji „jezici i okviri“ (engl. languages and frameworks) kao što je prikazano na slici 1. Kao opciju nužno je odabrati PHP, a nakon toga odabrati verziju PHP-a (engl. PHP language level) i CLI-a. Odabir verzija isključivo ovisi o tome koja verzija PHP-a je dostupna na računalu. Uvijek je poželjno odabrati najnoviju verziju, ali moguće je sav posao obaviti i na starijim verzijama. U ovom radu verzija PHP-a je 7.1, a verzija CLI-a je PHP 7.3.

3.2. Instalacija i konfiguracija PHPUnit dodatka

PHPUnit[3] je programski okvir namijenjen za testiranje u PHP-u. PHPUnit omogućuje pisanje jediničnih testova kojima testiramo napisane dijelove koda. Ovaj način je puno efikasniji od testiranja aplikacije kao cjeline, jer testiranjem aplikacije kao cjeline nemamo mogućnost testiranja željenih komponenti ili dijelova koda. Također, prilikom testiranja aplikacije kao cjeline, veća je vjerojatnost da će određena komponenta ili dio koda ostati netestiran. Na taj način, ako i dođe do pogreške, teško je odrediti koji dio koda je odgovoran za pogrešku. Osim za testiranje na pogreške, pisanje testova omogućuje lakše izmjene na postojećem kodu. Pokretanjem jediničnih testova, programer može utvrditi da li je neka promjena u kodu dovela do pogreške ili neočekivanog ponašanja. Idealno je da prilikom pisanja jediničnih testova, pišemo testove za sve moguće scenarije.

Za instalaciju PHPUnit-a, sa službene stranice[4], potrebno je preuzeti phpunit.phar datoteku koja omogućuje izvršavanje jediničnih testova u PHP-u. Preuzetu datoteku moguće je koristiti na dva načina. Prvi način je da putanju do preuzete datoteke stavimo u sistemske varijable kako bi se PHPUnit mogao koristiti na bilo kojem projektu. To se ne preporuča ako se planiraju koristiti okviri poput Symfony-a jer je PHPUnit sastavni dio tog okvira pa zbog toga može doći do raznih konflikata. Drugi način je ručno dodavanje putanje u postojeći projekt, u kojemu želimo pisati testove. To je moguće podesiti i postavkama, u sekciji „jezici i okviri“. Potrebno je odabrati opciju PHP, a unutar PHP opcije odabrati testni okviri (engl. Tests Frameworks). Nakon toga, potrebno je postaviti putanju do phpunit.phar datoteke. Ako je sve uspješno podešeno moguće je pisati i pokretati jedinične testove.

3.3. Pisanje testova

Postupak pisanja testova u PHP-u započinje kreiranjem nove PHP skripte unutar projekta. U kreiranu skriptu, dodaje se nova klasa, koja kao prefiks sadrži ime klase koju testiramo, a kao sufiks riječ Test. Kreirana klasa treba nasljeđivati PHPUnit klasu TestCase. Kako bi bilo moguće koristi TestCase klasu, potrebno ju je uvesti (engl. Import) iz prostora imena (engl. Namespace) PHPUnit/Framework.

Što se tiče imenovanja metoda kod pisanja testova, nužno je da nazivi imaju prefiks test. Svaki naziv metode mora biti čitak i jasan. Na temelju naziva metode, svaki razvojni inženjer bi trebao zaključiti koja je svrha testa. [6]

[3] https://phpunit.de/

[4] https://phpunit.de/manual/6.5/en/installation.html

Kako bi svrha testova bila što jasnija, poželjno je da su nazivi testova rečenice. Jedna vrlo korisna konvencija kod naziva metoda je korištenje riječi given, when i then kao što piše Martin Fowler [11]. Iza riječi given stavljaju se ulazni parametri metode koja se treba testirati. Nakon toga dolazi riječ when, nakon koje slijedi naziv događaja, odnosno naziv metode koja će se izvršiti. Na kraju naziva metode stavlja se riječ then zajedno sa očekivanim rezultatom testirane metode. [12]

Unutar testova pozivamo metode objekta koje želimo testirati. Za testiranje jedne metode objekta, moguće je napisati više testova. Svaki test služi za testiranje jednog slučaja. Testovi se pišu, sve dok se ne zadovolje svi mogući slučajevi.

PHPUnit na raspolaganje daje nekoliko ugrađenih metoda koje se izvršavaju prije i poslije napisanih testnih metoda. Takve metode pomažu kod ponovnog korištenja koda. Prije pokretanja svake testne metode, poziva se metoda setUp, koja daje mogućnost kreiranja objekt za testiranje. Kada testna metoda završi, moguće je pozvati metodu tearDown, koja se koristi za čišćenje i brisanje kreiranih objekata. Osim navedenih metoda, PHPUnit daje mogućnost korištenja metoda setUpBeforeClass i tearDownAfterClass. Navedene metode se koriste kod rada sa bazom podataka. Za otvaranje veze na bazu podataka, koristi se metoda setUpBeforeClass, a za zatvaranje veze tearDownAfterClass. [13]

3.4. Primjer korištenja

Kao primjer prikazat ćemo kako izgledaju testna klasa i klasa produkcijskog koda za jednostavnu klasu čiji je zadatak parsirati HTML kod. Zadatak je iz HTML koda izbaciti sve HTML oznake kako bi ostao samo čisti tekst.

Prvo se započinje sa jednostavnim testom koji služi samo da se kreira produkcijska klasa. Kod izgleda ovako:

```
<?php
declare(strict_types=1);
include_once("HTMLParser.php");
use PHPUnit\Framework\TestCase;
class HTMLParserTest extends TestCase {
   private $htmlParser;
   public function setUp(): void{
      $this->htmlParser = new HTMLParser();
   }
   public function testHasInstance(): void {
   $this->assertInstanceOf(
      HTMLParser::class, $this->htmlParser);
}}?>
```

Nakon toga test pada i kreira se produkcijska klasa koja izgleda ovako:

```
<?php class HTMLParser{ } ?>
```

Nakon toga pokušamo napisati test koji je što jednostavniji, a da testira neku stvarnu logiku u ovom slučaju je to test koji predaje samo čisti tekst bez oznaka. Te sama funkcija je sljedeća:

```
public function testGivenPureTextWhenGetParsedDataThenReturnSameText(): void { $this->assertEquals('PureTextWithNoHTML',$this-
>htmlParser-          >getParsedData("PureTextWithNoHTML")); }
```

Rezultat toga je da se taj problem može riješiti jednostavno i produkcijski kod sada izgleda ovako:

```
function getParsedData($text){ return $text; }
```

Zatim smišljamo novi najjednostavniji test kojeg se možemo sjetiti i to je ovaj koji testira da prima samo HTML:

```
public function testGivenOnlyHTMLCodeWhenGetParsedDataThenReturnEmptyText()
:void { $this->assertEquals('',$this->htmlParser-> getParsedData("<p></p>")); }
```

Riješenje koje najjednostavnije, a da oba test prolaze je ovo:

```
function getParsedData($text){
    if(!strpos($text,'<',0)) return $text;
    else return ''; }
```

I tako se postepeno dodaju testovi od jednostavnih pa prema složenijima. Rješenje se svaki puta mijenja tako da svi testovi uvijek prolaze. U rješenju uvijek pokušavamo napraviti što manje izmjena, a da testovi prođu. Kada imamo već veći kod tada se u svakoj iteraciji on čisti, refaktorira nakon što su testovi prošli. Taj korak je važan da na kraju kada smo gotovo klasom imamo klasu koja je lako razumljiva. Detaljni prikaz svakog daljnjeg koraka prelazi granice ovog članka te stoga u nastavku je prikazana samo finalna verzija.

Pa tako testovi koji su još dodani su:

```
public function testGivenNoTextWhenGetParsedDataThenReturnEmptyText():void {
     $this->assertEquals('',$this->htmlParser->getParsedData(""));    }
  public function testGivenHTMLCodeSPANWithTextWhenGetParsedDataThenReturnOnlyText():void {
     $this->assertEquals('TextInSPAN',$this->htmlParser->getParsedData("<span>TextInSPAN</ span>"));  }
  public function testGivenHTMLCodeParagraphWithTextWhenGetParsedDataThenReturnOnlyText()
   :void {$this->assertEquals('TextInParagraph',$this->htmlParser-
         >getParsedData("<p>TextInParagraph</p>"));    }
  public function testGivenHTMLCodeH1WithTextWhenGetParsedDataThenReturnOnlyText():void {
```

```
    $this->assertEquals('TextInH1',$this->htmlParser->getParsedData("<h1>TextInH1</h1>"));}
  public function testGivenMultipleHTMLElementsWithTextWhenGetParsedDataThenReturnOnlyText()  :void { $this-
>assertEquals("TextInH1\nTextInP\nTextInSPAN",$this->htmlParser-
      >getParsedData("<h1>TextInH1</h1><p>TextInP</p><span>TextInSPAN</span>"));  }

  public function testGivenNestedHTMLelementWithTextWhenGetParsedDataThenReturnOnlyText():void {

    $this->assertEquals("TextInH1\nTextInP\nTextInSPAN",$this->htmlParser-
      >getParsedData("<h1>TextInH1<p>TextInP<span>TextInSPAN</span></p></h1>"));  }

  public function testGivenNestedHTMLelementInARowWithTextWhenGetParsedDataThenReturnOnlyText():void {
    $this->assertEquals("TextInH1-TextInP-TextInSPAN",$this->htmlParser-  >getParsedData("<h1><p><span>TextInH1-TextInP-
TextInSPAN</span></p></h1>"));  }

  public function testGivenNestedAndMultipleHTMLelementWithTextWhenGetParsedDataThenReturnOnlyText():void {

    $this->assertEquals("TextInH1-TextInP-TextInSPAN\nTextInH1-TextInP-TextInSPAN",$this-  >htmlParser-
>getParsedData("<h1><p><span>TextInH1-TextInP-TextInSPAN</span></p></      h1><h1><p><span>TextInH1-TextInP-
TextInSPAN</span></p></h1>"));  }

  public function testGivenNestedAndMultipleHTMLelementAndMixedWithTextWhenGetParsedDataThenReturnOnlyText()
:void { $this->assertEquals("TextInH1-TextInP\nTextInH1-TextInP-TextInSPAN\nTextInH1-TextInP-      TextInSPAN",$this-
>htmlParser->getParsedData("<h1><p>TextInH1-TextInP<span>TextInH1-T  extInP-
TextInSPAN</span></p></h1><h1><p><span>TextInH1-TextInP-TextInSPAN</span></p></      h1>")); }

  public function testGivenHTMLCommentWithTextWhenGetParsedDataThenReturnEmptyText():void {

    $this->assertEquals("",$this->htmlParser->getParsedData('<!-- wp:heading          {\"level\":1,\"align\":\"center\"} -->'));  }

  public function testGivenMultipleHTMLCommentWithTextWhenGetParsedDataThenReturnEmptyText()  :void {$this-
>assertEquals("",$this->htmlParser->getParsedData("<!-- /wp:heading -->\    n\n<!-- wp:paragraph
{\\\"align\\\":\\\"left\\\",\\\"fontSize\\\":\\\"regular\\\"} --      >"));}
```

Finalna klasa izgleda ovako:

```
<?php class HTMLParser{
  function getParsedData($text){
    if($this->hasHTML($text)){
      $text = $this->removeAllComments($text);
      $result = $this->parseOutAllHTML($text);
    } else  return $text;
    return $result;  }

  public function removeAllComments($text){
    $text = preg_replace('<!--[^<>]*-->','',$text);
    $text = str_replace("<>","",$text);
    return trim($text);  }
  public function parseOutAllHTML($text)  {
    $result = ""; $elements = explode('>', $text);
    foreach ($elements as $k => $v) $result .= $this->extractNewHTMLline($v);
    return trim($result); }
  private function hasHTML($text){
    $start_pos=$this->getPositionOfFirstHTMLElement($text);
    if(is_int($start_pos))  return true;
    else  return false;  }
  private function parseOutOneHTMLelement($text){
    $startTextPosition=$this->getPositionOfFirstHTMLElement($text)+1;
    $lengthOfText=$this->getLengthOfTextInFirstHTMLElement($text,$startTextPosition);
    return substr($text,$startTextPosition,$lengthOfText);  }
  private function getLengthOfTextInFirstHTMLElement($text,$start_pos){
    $end_pos=strpos($text,'<',$start_pos);
    $length=$end_pos-$start_pos;
    return $length;  }
  private function getPositionOfFirstHTMLElement($text){ return strpos($text,'>');  }
  public function extractNewHTMLline($element) {
    $line = $this->parseOutOneHTMLelement(">" . $element . ">");
    $line = trim($line);
    if($this->isElementEmpty($line))  return $line . "\n";  }
  public function isElementEmpty($line)  { return strlen($line) != 0; }
}
?>
```

4. ZAKLJUČAK

Razvoj vođen testiranjem pristup je koji predstavlja temelj agilne metodologije koja se pojavila prije nešto manje od 20 godina, a također je pomogao u razvitku ostalih tadašnjih pristupa koje su karakterizirali elementi poput refaktoriranja, jednostavnog dizajna, programiranja u paru, itd.

Ideja pristupa je da razvojni programer piše kod na način da prvo piše test za produkcijski kod kojeg još nije napisao s ciljem ostvarivanja: sigurnosti da je program ispravan, smanjenog broja grešaka tijekom razvoja, hrabrosti da mijenja kod bez

straha, dokumentacije najniže razine te boljeg dizajna sustava. TDD ima najkraći razvojni ciklus od svih agilnih pristupa, a sastoji se od samo 5 koraka koji se ponavljaju: napiši test, pokreni ga i pobrini se da padne, napravi promjene u produkcijskom kodu, pokreni opet test i pobrini se da prođe i na kraju refaktoriraj da ukloniš duplicirani kod. Takav slijed koraka omogućuje programeru da se prvo fokusira na specifikaciju, a tek onda na validaciju sustava. Naglasak TDD-a je na pisanju čistih testova koji moraju biti čitljivi i svima razumljivi, a s tim ciljem se primjenjuju uzorci dizajna koji promoviraju slabo povezivanje objekata i oni koji smanjuju duplicirani kod korištenjem nasljeđivanja. Isto tako, produkcijski kod se refaktorira s ciljem smanjenja kompleksnosti i lakšeg testiranja jer su jedinični testovi zapravo ono što čini dizajn sustava fleksibilnijim i održavanijim.

Danas postoje brojna razvojna okruženja i alati koji olakšavaju i automatiziraju pisanje jediničnih i integracijskih testova te testova prihvatljivosti, pa je stoga danas primjena ovog pristupa znatno lakša nego što je bila prije 15 godina. U kontekstu ovog rada, za programski jezik PHP su to: PhpStorm, PHPUnit, Codeception, Selenium i drugi.

Započeti sa TDD-om nije lako i potrebno je puno vremena dok cijeli ciklus učeniku postane prirodan. Zbog toga su osmišljene TDD kate koje su koncipirane kao vježbe, a sastoje se od opisa problema, koraka u rješavanju i zaključka. Koliko god na početku djelovalo neintuitivno i obeshrabrujuće, prakticiranje TDD u konačnici rezultira većoj pokrivenosti koda i fleksibilnijem sustavu te omogućuje da se nove funkcionalnosti isporučuju u kraćim intervalima dok programera tjera da bude bolji programer – a to je ono što je najbitnije.

Literatura:

1 R. C. Martin, *Clean Agile - Back to Basics*. Pearson Education, Inc., 2020.

2 S. Mancuso, “Software Craftsmanship,” in *Clean Agile - Back to Basics*, Pearson Education, Inc., 2020.

3 S. Mancuso, *The software craftsman: professionalism, Pragmatism, Pride*. Pearson Education, 2014.

4 M. Novak and D. Kermek, “Profesional programmer: knowledge, attitude and misconceptions,” in *CASE30-Razvoj poslovnih i informatičkih sustava*, 2018, pp. 12–18.

5 R. C. Martin, *The clean coder: a code of conduct for professional programmers*. Pearson Education, 2011.

6 K. Beck, *Test-driven development: by example*. Addison-Wesley Professional, 2003.

7 R. C. Martin, “Professionalism and Test-Driven Development,” *IEEE Softw.*, vol. 24, no. 3, pp. 32–36, May 2007, doi: 10.1109/MS.2007.85.

8 R. Jeffries, “Twitter - Ron Jeffries,” 2019. [Online]. Available: https://twitter.com/RonJeffries/status/1098236863016505344. [Accessed: 12-Feb-2020].

9 R. C. Martin, *Clean code: a handbook of agile software craftsmanship*. Pearson Education, 2009.

10 R. C. Martin, “The Bowling Game Kata,” *butunclebob.com*, 2005. [Online]. Available: http://www.butunclebob.com/ArticleS.UncleBob.TheBowlingGameKata. [Accessed: 12-Feb-2020].

11 M. Fowler, “GivenWhenThen,” 2013. [Online]. Available: https://martinfowler.com/bliki/GivenWhenThen.html. [Accessed: 12-Feb-2020].

12 D. Terhorst-North, “Intoducing BDD,” *Dan North*, 2006. [Online]. Available: https://dannorth.net/introducing-bdd/. [Accessed: 12-Feb-2020].

Podaci o autorima:

Darjan Baričević

e-mail: dbaricevi@foi.unizg.hr

Darjan Baričević je student 5. godine diplomskog studija „Informacijsko i programsko inženjerstvo“ na Fakultetu Organizacije i Informatike u Varaždinu. Svoju programersku karijeru započinje u srednjoj školi kada je počeo učiti programske jezike poput Pascala i Pythona. Godine 2015. upisuje navedeni fakultet, a kroz svoj period fakultetskog obrazovanja uvijek je nastojao sudjelovati na što više projekata, radionica i drugih izvannastavnih aktivnosti. Sudjelovao je na projektima „STEM revolucija u zajednici“ 2018. godine, zatim godinu nakon „Hacklica hackaton“ i „CPSRK natjecanje za izradu prototipa aplikacije“ na kojem je u paru s kolegom osvojio 1. mjesto. Te aktivnosti nisu ostale nezapažene, i 2019. godine dobiva svoju 1. dekanovu nagradu za znanstvena i stručna postignuća. Glavno područje interesa mu je „Front-end“ web razvoj i razvoj korisničkih sučelja općenito, pa se stoga u slobodno vrijeme bavi i UX/UI dizajnom i izradom dizajna za jednostavne početne stranice.

Božo Čulo

e-mail: bculo@foi.unizg.hr

Božo Čulo je student 5. godine diplomskog studija „Informacijsko i programsko inženjerstvo“ na Fakultetu Organizacije i Informatike u Varaždinu. Sa programiranjem započinje u srednjoj školi gdje je počeo raditi s programskim jezicima C++ i C. Glavno područje interesa mu je razvoj aplikacija u .NET tehnologijama i ReactJS-u. Posjeduje certifikat iz predmeta Programsko inženjerstvo za odlično razumijevanje procesa razvoja softverskih proizvoda. Sudjelovao je kao autor na konferenciji Računalne igre 2018. U 2019. godini sudjeluje na natjecanju „Hacklica hackaton“ i na „CPSRK natjecanju za izradu prototipa aplikacije“ gdje je osvojio 2.mjesto.

Matija Novak

e-mail: matovak@foi.unizg.hr

Matija Novak 2010. stekao je akademski naziv „Magistar Informatike“ sa visokom pohvalom na Fakultetu Organizacije i Informatike. U studenom 2013. upisuje doktorski studij na Fakultetu Organizacije i Informatike i radi kao asistent na istom fakultetu na predmetima: Web dizajn i programiranje, Napredne web tehnologije i servisi i Razvoj web aplikacija. Autor je nekoliko stručnih i znanstvenih članaka iz područja detekcije plagijata izvornog koda, softverskog inženjerstva i skladišta podatka. Po završetku diplomskog studija radio je dvije godine u NTH Grupi u Varaždinu kao Voditelj razvoja za produkte i poslovni savjetnik za mobilne aplikacije za švicarsko i njemačko tržište. Nakon toga, radio je jednu godinu u tvrtki MCS d.o.o u Strahonincu kao arhitekt programskih sustava za mobilne i web platforme. U toku rada u navedene dvije tvrtke stekao je iskustvo, znanje i razumijevanje u razvoju Web i mobilnih aplikacija te samom procesu potrebnom za njihov razvoj.

Sveučilište u Zagrebu
Fakultet organizacije i informatike
Varaždin 42000, Pavlinska 2, Hrvatska

QUO VADIS ... JAVA

Dragutin Kermek, Matija Novak

SAŽETAK

Programski jezik Java kao i cijeli eko sustav oko njega zauzima važno mjesto u razvoju softvera. Tijekom njegovih 25 godina došlo je različitih promjena unutar tog sustava pri čemu su vrlo značajne zadnjih nekoliko godina. U radu se polazi od pregleda povijesti razvoja programskog jezika Java i njegovog eko sustava. Nastavlja se analizom kritičnih trenutaka koji su bitno utjecali na pojedine duguročne odluke. Slijedi analiza provedene ankete u vezi programskog jezika Java. Na kraju je kratak zaključak.

Ključne riječi: Java, Java tehnologija, eko sustav, povijest, anketa o korištenju

ABSTRACT

The Java programming language, as well as the entire eco system around it, occupy an important place in software development. Over the course of its 25 years, there have been various changes within that system, with significant changes over the last few years. The paper starts with a review of the history of Java programming language development and its eco system. It continues with an analysis of critical moments that have significantly influenced individual long-term decisions. The following is an analysis of a survey conducted regarding the Java programming language. Finally a brief conclusion is given.

1. UVOD

Razvoj softvera podrazumijeva korištenje određenog programskog jezika. Više različitih elemenata utječe na izbor programskog jezika koje će se koristiti u razvoju novog softvera. Kod pojedinca ili razvojnog tima možda je poznavanje određenog jezika prvi i najbliži element koji ima svoj utjecaj. Sigurno nije jedini a također ni odlučujući. Vrlo često vanjski elementi (npr. naručitelj) mogu dominantni prilikom definiranja programskog jezika koji će se koristiti u razvoju određenog softvera. Radoznalost i istraživački poriv može biti onaj element koji će usmjeriti prema upoznavanju novog programskog jezika, analizi njegovih mogućnosti te njegovoj primjeni u razvoju novih projekata. Strateška odluka o prijelazu s jednog na drugi programski jezik može imati vrlo značajne posljedice po tvrtku, po tim koji je obuhvaćen tom promjenom kao i po pojedince članove tima.

1. POVIJEST PROGRAMSKOG JEZIKA JAVA I NJEGOVOG EKO SUSTAVA

Rad na projektu Oak[1] kao prethodniku programskog jezika Java započeo je 1991. u tvrtki Sun Microsystems. Današnje ime Java dodijeljeno mu je 1995. godine kada je izašla prva službena verzija. Od 1996. godine počinje izdavanje razvojne opreme za Javu (eng. Java Development Kit) pod oznakom JDK 1.0. Sljedeća promjena naziva došla je 1998. s verzijom 1.2 koja je nazvana J2SE 1.2. Promjenom naziva želi se naglasiti da je to Java platforma budući da je te godina izdana Java EE – Enterprise Edition kao izdanje koje se bazira na Java platformi, a namijenjeno je za razvoj velikih, distribuiranih poslovnih sustava. Nakon toga sljedeća promjena naziva obavljena je 2004. godine kada je umjesto naziva J2SE 1.5 došao naziv J2SE 5.0. Već kod sljedeće verzije ponovno dolazi do promjene naziva u Java SE 6. Nakon toga više se nije mijenjao naziv nego se samo mijenjaju verzije. Trenutno je važeća verzija Java SE 13 od 17. rujna 2019. godine.

Dulje vrijeme postojalo je pravilo da svake parne godine izlazi nova glavna/veća verzija Jave. To pravilo prvi puta je prekinuto kod Java SE 7 koja je izašla 2011. godine nakon gotovo 5 godina od svoje prethodne verzije. Jedan od glavnih razloga za taj dugi period između dviju verzija nalazi se u preuzimanju tvrtke Sun Microsystems od strane tvrtke Oracle Corporation. To je ujedno značilo da se s novim vlasnikom mijenja pristup razvoju programskog jezika Java što je dovelo da James Gosling, jedan od kreatora Jave, napusti tvrtku. Zatim slijedi pravilo nove verzija svakih 3 godina koje završava 2017. godine s verzijom Java SE 9. Od 2018. godine s verzijom Java SE 10 počinje potpuno nova politika izdavanja nove verzije svakih 6 mjeseci.

Trenutna situacija s podrškom aktivnih verzija programskog jezika Java uključuje sljedeće verzije:

- Java SE 8 – Java SEu241 ažuriranje broj 241 za otklanjanje pogrešaka

- Java SE 11 (LTS [1]) – Java SE 11.0.6
- Java SE 13 – Java SE 13.0.2.

Do sada su spomenuta dva izdanja odnosno tehnologije koje su povezane uz programski jezik Java: Java SE i Java EE. Osim njih još postoje [2][3]:

- Java ME – pruža okolinu za aplikacije koje se izvršavaju na ugrađenim i mobilnim uređajima
- Java Embedded – za razvoj aplikacije za Internet stvari (IoT)
- Java Card – da pametne kartice i drugi sigurnosno otporni čipovi mogu udomaćiti aplikacije na bazi Java tehnologije
- Java TV – temeljena je na Java ME a služi za razvoj Java aplikacija koje se izvršavaju na TV i povezanim uređajima
- Java DB – Oracle-ova podrška Apache Derby bazi podataka otvorenog koda.

Java Enterprise Edtion predstavlja vrlo važnu tehnologiju zbog svoje primjene u razvoju složenih, distribuiranih, sigurnih, skalabilnih, podržanih web servisima, web aplikacija. Put razvoja Java EE[4] počinje 1999. godine s J2EE 1.2 te zadržava isto ime do verzije J2EE 1.4 iz 2003. godine. Od sljedeće verzije dobiva naziv Java EE 5 u 2006. godini te ga zadržava kroz sljedeće 4 verzije do Java EE 8 u 2017. godini. Nakon toga krajem 2017. godine najavljeno je da dolazi do prijenosa Java EE s tvrtke Oracle Corpration na Eclipse Foundation. Kao što se moglo očekivati zbog vlasništva registiranog znaka Java, došlo je do promjene naziva u Jakarta EE te 8. rujna 2019. godine objavljeno da se Jakarta EE 8 u potpunosti podudara s Java EE 8. Jakarta EE[5] trebala bi predstavljati najbolji put za otvaranje nove primjene programskog jezika Java u područjima kao što je računartvo u oblaku, aplikacije kritične misije itd.

3. ANALIZA POPULARNOSTI PROGRAMSKIH JEZIKA

Svaki programski jezik ima skup svojih prednosti kao i mana. Ako se polazi od vlastite želje za učenjem novog programskog jezika bez obzira na postojeće iskustvo tada jedan od korisnih izvora za odabir programskog jezika mogu biti renomirani indeksi popularnost programskih jezika. Većinom se spominju dva takva indeks: PYPL (PopularitY of Programming Language) [6] i TIOBE[7]. Kada se koriste podaci iz jednog od ta dva indeksa osobito je važno poznavati metodu prema kojoj se utvrđuje rang pojedinog jezika u određenom indeksu. PYPL indeks kreira se na bazi analize kako se često na Google pretražuju priručnici (eng. tutorials) za pojedine programske jezike. Sirovi podaci preuzimaju se iz Google Trends. TIOBE indeks ima složeniju metodu za izračunavanje koja je opisana u [8]. Ona se bazira na podacima 25 pretraživača (eng. search engines) koji ispunjavaju određeni skup uvjeta. Također postoji skup zahtjeva koje treba ispuniti pojedini programski jezik da bi bio uključen u indeks. Slijedi filtriranje podataka kako bi se isključili podaci koji nisu u stvari povezani s pojedinim programskim jezikom (eng. false positives). Na kraju se rang pojedinog programskog jezika temelji na broju zahtjeva (eng. hits).

Tablica 1 prikazuje zajedničke podatke PYPL i TIOBE indeksa za veljaču 2020. godine. Tablica sadrži prvih 10 programskih jezika iz oba indeksa. Kod PYPL indeksa Python ima naveći trend porasta u odnosu prošlu godinu (+4.15), a kod TIOBE indeksa to je C (+4.34%). Najveći trend pada kod PYPL indeksa ima Java (-1.8%), a kod TIOBE indeksa to je C++ (-1.28%). Detaljnija analiza PYPL indeksa [6] pokazuje da nema promjene u rangu prvih 9 programskih jezika u odnosu na godinu dana ranije. Gleda se samo prvih 10 po rangu. Kod TIOBE indeksa [7] nema promjena u rangu prvih 4 programskih jezika u odnosu na godinu dana ranije. Odnosno, između prvi 10 po rangu njih 6 je zadržalo svoj rang koji su imali godinu dana ranije. U oba indeks došlo je do ulaza novog programskog jezika među prvih 10.

Tablica 1. Rang programskih jezika za 02.2020.g prema PYPL[6] i TIOBE[7]

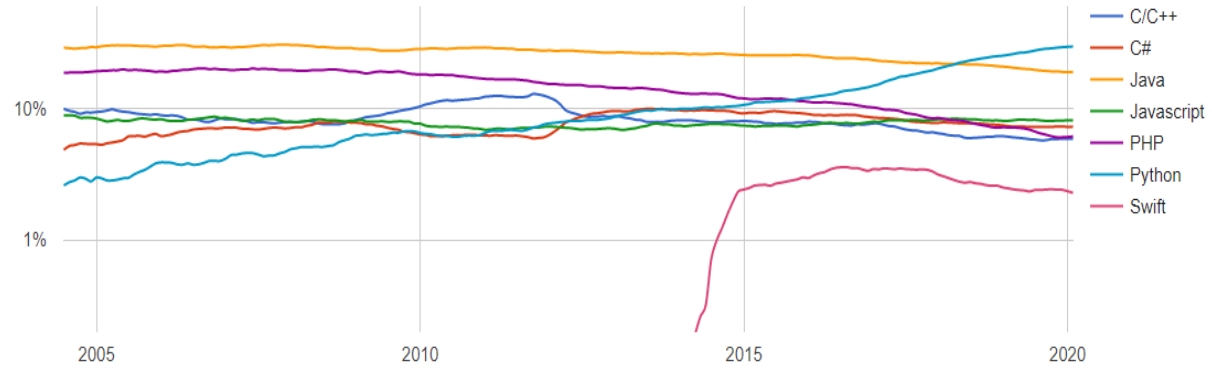
Rang	PYPL			TIOBE			Programski jezik	Prosječni rang
	Programski jezik	Udio	Trend	Programski jezik	Udio	Trend		
1	Python	29.88 %	+4.1 %	Java	17.358%	+1.48%	Java	1.5
2	Java	19.05 %	-1.8 %	C	16.766%	+4.34%	Python	2.0
3	Javascript	8.17 %	+0.1 %	Python	9.345%	+1.77%	C/C++	4.5
4	C#	7.3 %	-0.1 %	C++	6.164%	-1.28%	C#	4.5
5	PHP	6.15 %	-1.0 %	C#	5.927%	+3.08%	JavaScript	5.0
6	C/C++	5.92 %	-0.2 %	Visual Basic .NET	5.862%	-1.23%	PHP	6.5
7	R	3.74 %	-0.2 %	JavaScript	2.060%	-0.79%	Swift	9.5
8	Objective-C	2.42 %	-0.6 %	PHP	2.018%	-0.25%		
9	Swift	2.28 %	-0.2 %	SQL	1.526%	-0.37%		
10	TypeScript	1.84 %	+0.3 %	Swift	1.460%	+0.54%		

[1] LTS – Long Term Support

Zanimljivo je da se 7 programskih jezika nalazi među prvih 10 u oba indeksa. Na temelju ranga u oba indeksa izračunat je prosječan rang koji j prikazan u desnom dijelu tablice. TIOBE indeks sadrži pojedinačno progranske jezike C i C++, a PYPL indeks ih ima zajedno. Zbog toga su za TIOBE indeks uzeta zajedno oba programska jezika te je izračunat njihov prosječni rang unutar TIOBE indeks, koji je zatim korišten za izračun prosječnog ranga oba indeksa. Dobiveni podaci pokazuju da programski jezik Java ima najviši prosječni rang, a iza njega je Python. Slijede 3 programska jezika (C/C++, C#, JavaScript) sličnog prosječnog ranga.

Podaci pojedinog indeksa o rangu, udjelu i trendu za određeni programski jezik tek su gruba informacija i nisu dovoljni kako bi se mogla donijeti kvalitetna odluka o mogućoj promjeni/korištenju programskog jezika kod određenog projekta ili kao okidač da je potrebno naučiti novi programski jezik.

Jasno je da su potrebni podaci većeg broja godina kako bi se moglo potpunije sagledati trendove programskih jezika. Slika 1 prikazuje podatke zadnjih 15 godina za 7 programskih jezika koji su zajednički za PYPL i TIOBE indekse i nalaze se u desnom dijelu Tablice 1. Sam dijagram je dobiven pomoću alata koji se nalazi na [6] i temelji se na podacima iz PYPL. Brzi pogled slike mogao biti dati utisak da ne postoje velike razlike između odabranog skupa programskh jezika. Glavni razlog je logaritamska skala za podatke o popularnosti (%) jer ona sažima veće vrijednosti kako bi se bolje vidjele razlike kod manjih vrijednosti. Samo dva programska jezika (Java i Python) nalaze se iznad vrijednosti od 10%. Njihova usporedna u zadnjih 5 godina pokazuje da je Python porastao za 19% dok je Java izgubila 6.5%



Slika 1. Usporedba programskih jezika u zadnjih 15. g. prema PYPL[1] [2]

Sljedeća usporedba spomenuta dva programska jezika može se temeljiti na detaljnim podacima koji se mogu dobiti na Google Trends[9]. Za filtiranje su korišteni pojmovi „Java“ – Programming language i „Python“ – Programming language. Na slici 2 nalazi se usporedba podataka njihovog pretraživanju za zadnjih 5 godina na svjetskoj razini u svim kategorijama. Tu je jasnije vidljiv trend pada pretraživanju programskog jezika Java uz istovremeni trend porasta programskog jezika Python. Njihovo izjednačavanje dogodilo se u tjednu 24.-31.03.2019., a od tjedna 26.05.-01.06.2020. Python ima stalnu prednost koju polagano povećava.

Slika 2. Usporedba pretraživanja programskih jezika Java i Python u svijetu u zadnjih 5. g. prema Google Trends[9] – cijeli svijet i sve kategorije

Analiza podataka o pretraživanju programskih jezika na bazi svih kategorija može se smatrati relativno nepouzdanom zbog čega je bolje uzeti kategoriju(e) koje će direktnije povezana s pretraživanjem programskih jezika, a to je 'Poslovi i obrazovanje' (eng. Jobs & Education). Slika 3. prikazuje sličan trend pada pretraživanja programskog jezika Java uz istovremeni trend porasta programskog jezika Python. U ovom slučaju Java i dalje drži prednost u odnosu na Python no ona se snatno smanjila u zadnjih 5 godina.

[2] Podaci za popularnost (%) su prikazani u logaritamskoj skali.

Slika 3. Usporedba pretraživanja programskih jezika Java i Python u svijetu u zadnjih 5. g. prema Google Trends[9] – cijeli svijet i kategorija 'Poslovi i obrazovanje'

3.1.Analiza popularnosti integriranih razvojnih okolina

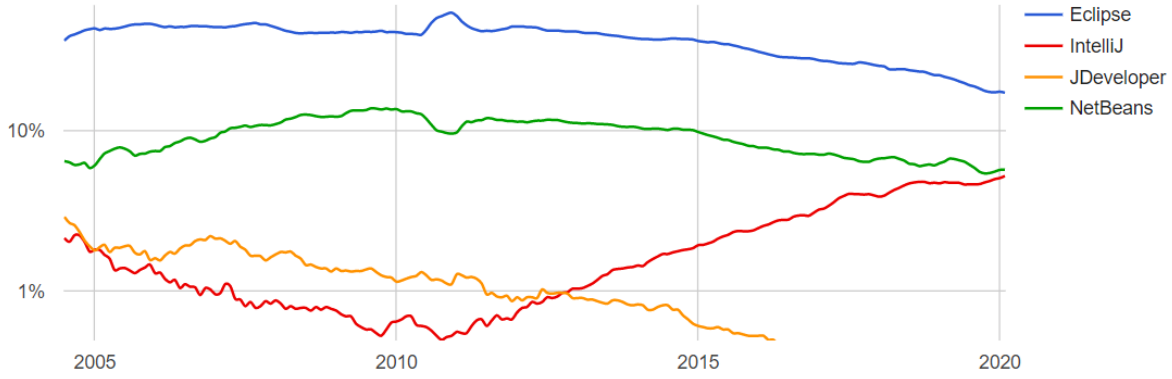
Razvoj softvera provodi se uglavnom unutar određenog integriranog razvojnog okolina (IDE). U današnje vrijeme mnoge integrirane razvojne okoline omogućavaju razvoj softvera u većem broju programskih jezika. Većina IDE ima barem jedan programski jezik koji je pretpostavljeni, a za mnoge programske jezike postoje proširenja koja se po potrebi mogu instalirati. Vrlo često postoji manji broj IDE koji se poistovjećuju s pojedinim programskim jezikom. To je samo znak da korisnici tog programskog jezika preferiraju specifične osobine koje ima određeni IDE jer su one više priklonjene potrebama projekata u kojima se primjenjuje taj programski jezik, a indirektno time su prilagođeni „mentalitetu“ korisnika, razvojnih inžinjera (eng. developers). Tablica 2 prikazuje rang IDE za veljaču 2020. godine prema PYPL[6]. Prvo mjesto drži Visual Studio koji podržava programski jezik Java no većina korisnika programskog jezika Java vjerojatno neće taj IDE identificirati kao njihov prvi izbor. Među prvih 10 IDE nalaze se 3 koja se na prvi spomen povezuju s programskih jezikom Java. To su Eclipse, NetBeans i IntelliJ. Sva tri imaju pad u rangu u odnosu na prošlu godinu. Njihov udio ima različito ponašanje u odnosuu na prošlu godinu (trend) pa tako Eclipse ima pad (-4.4%), NetBeans isto ima pad (-0.7%), a jedino IntelliJ ima rast (+0.4%). Općenito se pad interesa za spomenuta tri IDE može intuitivno povezati s padom popularnosti kod programskog jezika Java. Ipak nema dovoljno podataka kako bi se sa sigurnoštu moglo tvrditi da postoji stvarna korelacija između tih dviju pojava.

Tablica 2. Rang integriranih razvojnih okolina (IDE) za 02.2020. g. prema PYPL[6]

Rang	Promjena	IDE	Udio	Trend
1		Visual Studio	23.28 %	+0.3 %
2	↑	Android Studio	18.19 %	+1.4 %
3	↓	Eclipse	17.26 %	-4.4 %
4	↑↑	Visual Studio Code	6.15 %	+1.7 %
5	↑↑	pyCharm	5.89 %	+1.8 %
6	↓↓	NetBeans	5.72 %	-0.7 %
7	↓↓	IntelliJ	5.23 %	+0.4 %
8	↑↑	Xcode	4.57 %	+1.1 %
9	↓	Sublime Text	3.98 %	-0.1 %
10	↓	Atom	3.31 %	-0.6 %

Podaci s PYPL pružaju mogućnost da se može promatrati dugogodišnji trend pularnosti i za integrirane razvojne okolinie. Slika 5 prikazuje 15-godišnji trend popularnosti za 4 IDE koja se usko povezuju s programskom jezikom Java. Od njih 4 samo jedan IDE ima trend rasta zadnjih 9 godina, a to je IntelliJ. Ostala 3 imaju trend pada zadnjih 7 godina.

Od spomenutih 3 IDE iz tablice 2 jedan je bio, a to je NetBeans, povezan s tvrtkom Oracle Corporation vlasnikom registriranog znaka Java. Razvoj NetBeans-a započeo je 1996. godine na Charles University u Pragu u Češkoj Republici [10]. Ubrzo slijedi komercijalizacija putem novo osnovane tvrtke da bi u 1999. godini ta tvrtka bila kupljena od tvrtke Sun Microsystems, a ona 2010. od tvrtke Oracle Corporation. A time je i NetBeans prešao u njeno vlasništvo. Od Netbeans 6.5 sadrži podršku za razvoj Java EE tako da postaje referentni IDE koji se koristi u službenim priručnicima za Java EE. Pod okriljem tvrtke Oracle Corporation provodi se kontinuirani razvoj NetBeans kako bi se podržavale novo dolazeće verzije Java SE i Java EE. Oracle Corporation najavljuje u rujnu 2016. godine da će donirati NetBeans projekt Apache Software Foundation. Nakon preuzimanja u listopada 2016. godine provodi se postupak inkubacije NetBeans u sustav Apache projekata. Prva verzija pod okriljem Apache Software Foundation izdana je 29. srpnja 2018. godine pod nazivom NetBeans 9.0. Trenutno je aktivna verzija NetBeans 11.2.



Slika 4. Usporedba integriranih razvojnihi okolina (IDE) za Javu u zadnjih 15 g. prema PYPL[6] [3]

4. PRIJELOMNI DOGAĐAJI VEZANI UZ PROGRAMSKI JEZIK JAVA I NJEN EKO SUSTAV

Povijest u trajanju od 25 godina obilježena je različitim odlukama koje su utjecale na rast ili pad popularnosti programskog jezika Java. Neke od tih odluka donesene su u tvrtkama koje su bile ili jesu vlasnice trgovaškog znaka Java. A druge su došle od strane neovisnih tvrtki koje su uočile potencijal programskog jezika Java u određenoj domeni. Postoje i drugi elementi koji su bili direktno vezani uz razvoj tehnologija tako da su indirektno utjecali na programski jezik Java.

On direktnih odluka koje se pozitivno odnose na sam programski jezik Java može se izdvojiti izdavanje jezgre Java platforme u obliku besplatnog softvera i otvorenog programskog koda u 2006. godini pod nazivom OpenJDK. Slijedi prilagođavanje Java virtualnog strona (JVM - Java Virtual Machine) da može osim Jave izvršavati druge postojeće programske jezike kao što su: Javascript, Python ili Ruby. Odnosno da su dizajnirani novi programski jezici koji se mogu izvorno izvršavati u JVM kao što su: Clojure, Groovy, Kotlin i Scala. Dugi očekivani projekt Jigsaw u Java 9 uveo je sustav modula [12], što je pozitivno. Odluka da se programski jezik Java koristi za razvoj aplikacija pod operacijskim sustavom Android za pametne telefone sigurno je jedan od najvažnijih pozitivnih događaja. Sljedeći pozitivni događaji su prijenos Java EE na na Eclipse Foundation i prijenos NetBeans na Apache Software Foundation.

Osim pozitivnih događaja bilo je i negativnih koje su utjecale na pad popularnosti programskog jezika Java. Jedna od tih je iz 2012. godine odluka tvrtke Oracle Corporation da pokrene sudski spor protiv tvrtke Google [13] zbog kršenja autorskih prava za Java API. Spor još nije završen u potpunosti te se očekuje nastavak u ožujku 2020. godine na Vrhovnom sudu SAD. U programski jezik Java veže je jedna vrlo neugodna situacija koja se tiče sigurnosnog propusta za Java dodatak web preglednicima koji je otkriven 2013. godine u verziji Java 7. Zbog tog propusta došlo je blokiranja Appleta kao oblika Java aplikacije koje se izvršava unutar preglednika. Proširenje primjene weba kao platforme i otvaranje novih mogućnosti u web preglednicima kao što je Ajax utjecali su na podizanje mogućnosti web aplikacija što je utjecalo na značajan pad interesa za klasične stolne aplikacije s grafičkim sučeljem.

5. ANALIZA ANKETE

Tijekom veljače 2020. godine provedeno je anketiranje osoba iz 6 tvrtki koje se bave razvojem softvera i za koje je poznato da koriste programski jezik Java. Te tvrtke ne predstavljaju statistički značajan uzorak tako da i podaci nemaju statistčku vrijednost, a to nije bila ni namjera kod provođenja ove ankete. Ova anketa predstavlja pilot istraživanje za kasnije anketiranje u kojem će biti proširen anketni upitnik i bit će obuhvćeno znatno više tvrtki.

Nekoliko podataka o tvrtkama koje su sudjelovale u anketi. Prosječno imaju zaposleno 106 osoba od kojih prosječno 34 u razvoju koriste programski jezika Java. Postoje mnoga područja u kojima provode razvoj softvera, što uključuje bankarsko poslovanje, osiguravateljsko poslovanje, ERP, e-trgovinie, Web potale, IoT, programske komponente za tržište, a najviše po zahtjevu korisnika. Većina njih ima barem 10 aktivnih projekata u kojima se koristi programski jezik Java. A postoji i sa 100 projekata.

Četiri tvrtke koriste samo programski jezik Java u razvoju softvera. Kod ostalih se javljaju Python, Javascript, PHP, C#. Najviše ima Javascript kod dvije tvrtke po 20%. Većina tvrtki najviše se bavi se razvojem web aplikacija, a neke manjih dijelom i stolnim aplikacijama. Od integriranih razvojnih okolina većina tvrtki najviše koristi IntelliJ, a zatim Eclipse. Tvrtke većinom koriste u razvoju Java SE 8, a zatim Java SE 11. Slika 5 pokazuje raspodjelu odgovora na pitanje o politici ažuriranja manje verzije (eng. update) Java SE. Malo značajniji udio ima ažuriranje 3 do 4 puta godišnje.

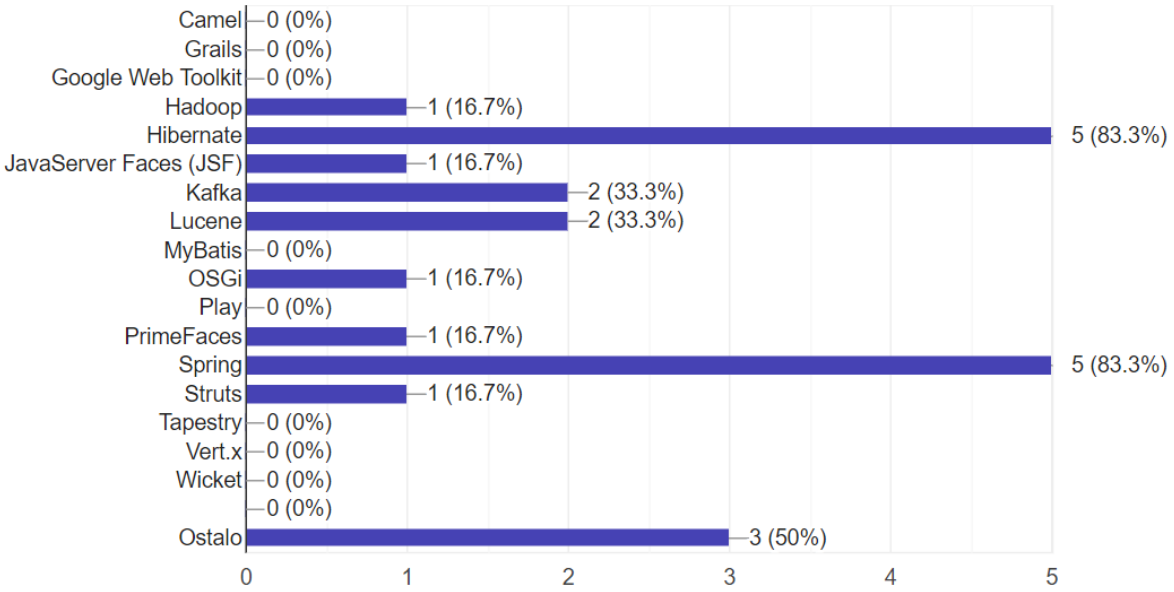
[3] Podaci za popularnost (%) su prikazani u logaritamskoj skali

Slika 5. Politika ažuriranja manje verzije Java SE

Prijelaz na višu verziju Java SE je složeniji proces nego ažuriranje verzije. Na slici 6 vidljivo je da kod pola tvrtki odlučuje vanjski element u vidu poslovnog partnera koji je naručitelj projekta.

Slika 6. Politika prijelaza na višu verziju Java SE

Većina tvrtki u svojim projektima temelji razvoj na Java EE 8, ali postoje i dvije tvrtke koje ne koriste Java EE. Aplikacijski poslužitelji su bitne komponentu u svakom sustavu. Najzastupljeniji je Tomcat, a zatim Amazon AWS. Postoji jedna tvrtka koja samo koristi WildFly. Slika 7 prikazuje koje programske okvire koriste tvrtke u svojim projektima. Najzastupljeniji su Hibernate i Spring. Nakon njih su Kafka i Lucene.



Slika 7. Korištenje programskih okvira za Javu

Kako su opisane prednosti programskoj jezika Java i njegovog eko sustava:

- Robustnost i skalabilnost, multithreading, brzina izvođenja, dobra podrška.
- Veliki community
- Multiplatformnost
- Rasirenost, podrška, solidna brzina, jako puno gotovih librarya
- Robusnost, veliki ekosustav (biblioteke, programski okviri), sigurnost
- Jednostavnost učenja i razvoja, mnogo dokumentacije i primjera, ogromna zajednica koja bilježi probleme i rješenja na probleme, univerzalnost prema različitim OS-ima, IDE-ima, brdo API-a za povezivanje s DB, WS, MQ, itd.

Kako su opisane mane programskoj jezika Java i njegovog eko sustava:

- Sporiji razvoj koda u odnosu na PHP (Symfony ili Laravel).
- Nije najbolja opcija za serverless arhitekturu
- Zamjetno sporiji u kritičnim operacijama od native koda i ovisnost o proizvođačima JVMa (rizik naplaćivanja JVM, rizik održavanja JVM)
- Jednom riječju, classloader.

6. ZAKLJUČAK

Programski jezik Java i njegova eko sustav predstavljaju zreli proizvod koji pokazuje svoje prednosti na velikom broju projekata različitih osobina. Njegova popularnost polagano pada što nije nužno znak da on više nije konkurentan u odnosu na nove dolazeće programske jezike. Sigurno je jasno da njegova raznolikost primjene (od IoT, pametnih telefona, do superačunala na bazi računarstva u oblaku) i dalje privlači nove razvojne inžinjere da ga nauče i koriste u svojim projektima. Široka lepeza aplikacijskih poslužitelja i programskih okvira ostavlja arhitektima i razvojnim inžinjerima veliku slobodu izbora. Treba se nadati da u budućnosti neće biti novih loših odluka od strane tvrtke Oracle Corporation koje bi mogle značajno utjecati da razvojni inžinjeri napuste programski jezik Java. S druge strane očekuje se da će neprofitne oganizacije Apache Software Foundation i Eclipse Foundation svojim radom i ugledom doprinjeti podizanju kvalitete projekata koje su preuzele, a preko njih i samom programskom jeziku Java.

Literatura:

1 Wikipedia, https://en.wikipedia.org/wiki/Java_(programming_language), preuzeto [16.02.2020.]

2 Oracle, https://www.oracle.com/java/, preuzeto [16.02.2020.]

3 Wikipedia, https://en.wikipedia.org/wiki/Java_(software_platform), preuzeto [16.02.2020.]

4 Wikipedia, https://en.wikipedia.org/wiki/Java_Platform,_Enterprise_Edition, preuzeto [16.02.2020.]

5 Jakarta EE, https://jakarta.ee/, preuzeto [16.02.2020.]

6 PYPL, http://pypl.github.io/PYPL.html, preuzeto [16.02.2020.]

7 TIOBE, https://www.tiobe.com/tiobe-index/, preuzeto [16.02.2020.]

8 TIOBE Programming Community Index Definition, https://www.tiobe.com/tiobe-index//programming-languages-definition/, preuzeto [16.02.2020.]

9 Google Trends, https://trends.google.com/trends/, preuzeto [16.02.2020.]

10 Wikipedia, https://en.wikipedia.org/wiki/NetBeans, preuzeto [16.02.2020.]

11 NetBeans, https://netbeans.org/, preuzeto [16.02.2020.]

12 https://www.oracle.com/corporate/features/understanding-java-9-modules.html, preuzeto [16.02.2020.]

13 https://en.wikipedia.org/wiki/Google_v._Oracle_America, preuzeto [16.02.2020.]

Podaci o autorima:

Dragutin Kermek

e-mail: dragutin.kermek@foi.hr

Dragutin Kermek redoviti je profesor u trajnom zvanju na Fakultetu organizacije i informatike Sveučilišta u Zagrebu. Nositelj je predmeta "Web dizajn i programiranje", "Napredno Web tehnologije i servisi“, „Uzorci dizajna“ te sunositelj na predmetu „Sustavi za elektroničko učenje“. Teorijsko i praktično informatičko obrazovanje nadopunjavao je nizom seminara i studijskih boravaka u inozemstvu (University of Kentucky - School of Business and Economics, SAD, Wired education in the CEENet Workshop on network technology, Hungary, Object-oriented analisys, Slovenia, itd.). Sudjelovao je na više međunarodnih i nacionalnih znanstvenih i stručnih projekata. Područja interesa su mu izgradnja sustava temeljenih na web platformi, razvoj programskih komponenti, e-učenje. Autor je većeg broja znanstvenih i stručnih radova. Trenutno je na funkciji prodekana za studijske programe na Fakultetu organizacije i informatike Sveučilišta u Zagrebu.

Matija Novak

e-mail: matija.novak@foi.hr

Matija Novak 2010. stekao je akademski naziv „Magistar Informatike“ sa visokom pohvalom na Fakultetu Organizacije i Informatike. U studenom 2013. upisuje doktorski studij na Fakultetu Organizacije i Informatike i radi kao asistent na istom fakultetu na predmetima: Web dizajn i programiranje, Napredne web tehnologije i servisi i Razvoj web aplikacija. Autor je nekoliko stručnih i znanstvenih članaka iz područja detekcije plagijata izvornog koda, softverskog inženjerstva i skladišta podatka. Po završetku diplomskog studija radio je dvije godine u NTH Grupi u Varaždinu kao Voditelj razvoja za produkte i poslovni savjetnik za mobilne aplikacije za švicarsko i njemačko tržište. Nakon toga, radio je jednu godinu u tvrtki MCS d.o.o u Strahonincu kao arhitekt programskih sustava za mobilne i web platforme. U toku rada u navedene dvije tvrtke stekao je iskustvo, znanje i razumijevanje u razvoju Web i mobilnih aplikacija te samom procesu potrebnom za njihov razvoj.

Sveučilište u Zagrebu
Fakultet organizacije i informatike
Varaždin 42000, Pavlinska 2, Hrvatska

NOVE OPASNOSTI OD UMJETNE INTELIGENCIJE U ZAŠTITI OSOBNIH PODATAKA

NEW THREATS FROM ARTIFICIAL INTELLIGENCE IN THE PROTECTION OF PERSONAL DATA

Doc. dr. sc. Marko Horvat, v. pred.

SAŽETAK

Pojam umjetne inteligencije danas je vrlo često korišten bez cjelovite spoznaje na što se zapravo odnosi. U ljudskoj je prirodi da nas plaši nepoznato. Zbog napretka i široke primjene računalnih sustava umjetne inteligencije razumljivo je da postoji opravdana zabrinutost zbog potencijalnih opasnosti koje proizlaze iz povreda prava na pristup osobnim podacima, tj. u čuvanju privatnosti. Stoga je potrebno jasno odrediti nove potencijalne opasnosti koje proizlaze iz neetičnog korištenja umjetne inteligencije u prikupljanju i obradi osobnih podataka. Također, nužno je predložiti mjere kako bi se takve opasnosti svele na najmanju moguću mjeru. Od posebne pažnje su inteligentni sustavi koje se koriste za mjerenje emocionalnih stanja, ponašanja, stavova i mišljenja, odnosno prikupljanje podataka u svrhu profiliranja osoba. U radu su navedene mogućnosti današnjih tehnologija u ovim područjima, karakteristični primjeri primjene, te koraci koji je potrebno poduzeti kako bi se smanjila mogućnost zlouporabe osobnih podataka.

ABSTRACT

Artificial intelligence as a term is nowadays commonly used without the complete understanding of what it actually refers to. Due to the advancement and widespread use of artificial intelligence computer systems, it is understandable that concerns exist about the potential dangers arising from violations of the right of access to personal data, i.e. in the protection of privacy. New potential hazards arising from the unethical use of artificial intelligence in collecting and processing of personal data must be clearly identified. Further, it is necessary to propose measures to minimize these hazards. Of particular note are intelligent systems used to estimate emotional states, behaviors, attitudes and opinions, that is, aggregation of data for the purpose of customer profiling. The paper outlines the capabilities of current technologies in these fields, gives typical application examples, and suggests steps for reduction of the possibility for personal data misuse.

1. UVOD

Makar je pojam umjetna inteligencija (engl. *artificial intelligence*, AI) danas iznimno korišten u javnoj komunikaciji često se zaboravlja da ne postoji opće prihvaćena definicija inteligencije kao niti umjetne inteligencije. Očiti problem je u izrazitoj interdisciplinarnoj širini samog područja kao i o brojnim mogućnostima primjene umjetne inteligencije. U ovom trenutku najčešće asocijacije povezane s umjetnom inteligencijom su samoupravljiva vozila, autonomne letjelice, humanoidni roboti, ili apstraktni računalni sustavi koji prikupljaju goleme količine nestrukturiranih podataka, nadziru, pa čak upravljaju osobnim životima. Zbog visoke tehnološke složenosti takvih sustava često je u društvu njihova namjena nejasna ili se čini inheretno zlonamjernom (npr. praćenje kretanja ili analiza ponašanja).

U praksi umjetna inteligencija je grana računalnih znanosti koja se bavi automatizacijom inteligentnog ponašanja, a želi objasniti i emulirati inteligentno ponašanje u smislu računalnih procesa. Možemo dodatno reći da umjetna inteligencija predstavlja niz tehnologija koje se koriste za obavljanje funkcija koje zahtijevaju inteligenciju ljudi, odnosno proučava kako učiniti da računala rade stvari u kojima su, trenutno, ljudi bolji.

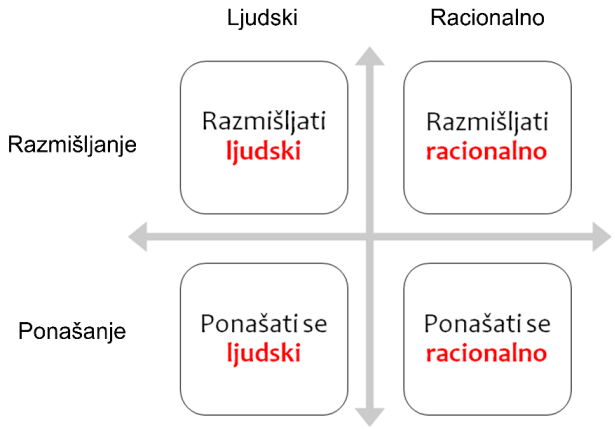
Analizirajući relevantne znanstvene izvore možemo utvrditi da je do sada objavljeno nekoliko desetaka različitih definicija pojma inteligencije. Neke od njih su većim ili manjim dijelom međusobno kontradiktorne. Pa tako najčešće definicije inteligencije koje možemo pronaći u literaturi su [1]:

(1) *Svojstvo uspješnog snalaženja jedinke u novim situacijama* (R. Pinter)
(2) *Opća sposobnost razmišljanja prilikom rješavanja problema* (Lewis Terman)

Najviše zastupljene definicije umjetna inteligencije su:

(3) *Znanstvena disciplina koja se bavi izgradnjom računalnih sustava čije se ponašanje može tumačiti kao inteligentno* (John McCarthy, 1956.)
(4) *Znanost o tome kako postići da strojevi izvode zadatke koji bi, kada bi ih radio čovjek, trebali inteligenciju* (Marvin Minsky, 1961.)

S obzirom na kompleksnost problema definirana je sistematizacija, ili podjela, definicija umjetne inteligencije kao što je prikazano na sljedećoj slici. Na jednak način podijeljeni su i sustavi, odnosno postupci i algoritmi, umjetne inteligencije s obzirom na njihovu primjenu.



Slika 1. Podjela definicija umjetne inteligencije s obzirom na namjenu.

Kao što se može vidjeti na slici definicije umjetne inteligencije podijeljene su u četiri načelne kategorije ili grupe s obzirom na njihovu namjenu ili svrhu:

1. Umjetna inteligencija kojoj je namjena razvoj sustava koji razmišljaju ljudski
2. Umjetna inteligencija kojoj je namjena razvoj sustava koji razmišljaju racionalno
3. Umjetna inteligencija kojoj je namjena razvoj sustava koji se ponašaju ljudski
4. Umjetna inteligencija kojoj je namjena razvoj sustava koji se ponašaju racionalno

Danas se razvijaju sustavi koji slijede sve četiri kategorije definicija umjetne inteligencije, ali najveći prioritet iz posve praktičnih razloga usmjeren je prema razvoju i izradi inteligentnih strojeva, odnosno nastojanju da se razvije i za dobrobit društva iskoristiti inteligentno ponašanje strojeva.

2. AKTUALNI VODEĆI STAVOVI O OPASNOSTIMA KORIŠTENJA UMJETNE INTELIGENCIJE

U posljednje vrijeme moguće je primijetiti povećanu zabrinutost mogućnih opasnosti zbog korištenja umjetne inteligencije. Ovakve stavove iznose zakonodavci, znanstvene institucije i vodeći tehnološki inovatori. Primjerice, jedan od vodećih poduzetnika i inovatora današnjice Elon Musk, nedavno je ustvrdio da bi razvoj umjetne inteligencije trebao bi biti bolje reguliran, čak i u njegovoj vlastitoj tvrtki Tesla koja proizvodi automobile sa automatskim upravljanjem [2].

S tim u svezi neki svjetski mediji izvijestili su sljedeće [3]:

Tesla CEO Elon Musk wants to see all artificial intelligence better regulated, even at his own company.

Musk has a history of expressing serious concerns about the negative potential of AI. He tweeted in 2014 that it could be "more dangerous than nukes," and told an audience at an MIT Aeronautics and Astronautics symposium that year that AI was "our biggest existential threat," and humanity needs to be extremely careful.

Musk has been floating the idea for some kind of government oversight of AI for a while… "we ought to have a government committee that starts off with insight, gaining insight. Spends a year or two gaining insight about AI or other technologies that are maybe dangerous, but especially AI." The committee would then come up with regulations to ensure the safest uses of AI, he said. Musk added at the time that he did not think such a committee would actually happen.

Ideja o nužnosti formiranja državnog, pa i međudržavnog, tijela koje će nadzirati primjenu tehnologija umjetne inteligencije nije nova. Štoviše, prisutni su i pozivi na oprez i zabranu korištenja tehnologija za prepoznavanje emocija. U tom smislu Institut za istraživanje umjetne inteligencije „AI Now" predlaže potpunu zabranu tehnologija za prepoznavanje emocija dok se u ovo područje ne uvede čvrsta zakonska regulativa [4]. Razlozi za to su, kako navode:

1) Nedovoljna znanstvena utemeljenost estimacije emocija pomoću računala
2) Primjena takve tehnologije može povećati nejednakosti, posebice temeljene na rasi i spolu

Prijedlozi Instituta utemeljeni su na nedavno objavljenim rezultatima istraživanja [5]. Tijekom spomenutog istraživanja sustavno je pregledana objavljena znanstvena literatura iz područja estimacije emocija. Istraživanje je bilo vrlo opsežno i trajalo je dvije godine, a naručila ga je međunarodna Udruga za psihološke znanosti. U istraživanju temeljito je analizirano više od tisuću znanstvenih radova o detekciji i estimaciji emocija [5]. Posebice je posvećena pažnja na postupke prepoznavanja emocija iz facijalnih ekspresija koje su povezane s određenim emocijama. Nedvosmislen zaključak ovog istraživanja jest da nije utemeljena pretpostavka da izrazi lica pouzdano odgovaraju emocijama ispitanika [6]. Postupci procjene emocionalnih stanja pomoću računala u praksi općenito nisu pouzdani i ne mogu pružiti dovoljno točne i precizne rezultate unutar heterogenih grupa ispitanika. Najučinkovitiji su kod personaliziranih pobuda, ali to nije uvijek primjenjivo.

Tehnologije estimacije emocija pomoću računala danas se već redovito koriste za procjenu podnositelja zahtjeva za posao i osoba osumnjičenih za zločine, u sklopu policijskih istraga, a ispituju se i za daljnje primjene, primjerice u računalnim igrama poput VR kaciga za procjenu emocija sudionika. Najčešće se koriste sustavi za prepoznavanje facijalnih ekspresija jer te tehnologije nemaju velike tehničke zahtjeve, najviše su dostupne, a istodobno brzo daju rezultate, pa čak i u stvarnom vremenu. Upravo ove tehnologije su najviše kritizirane u gore spomenutom istraživanju.

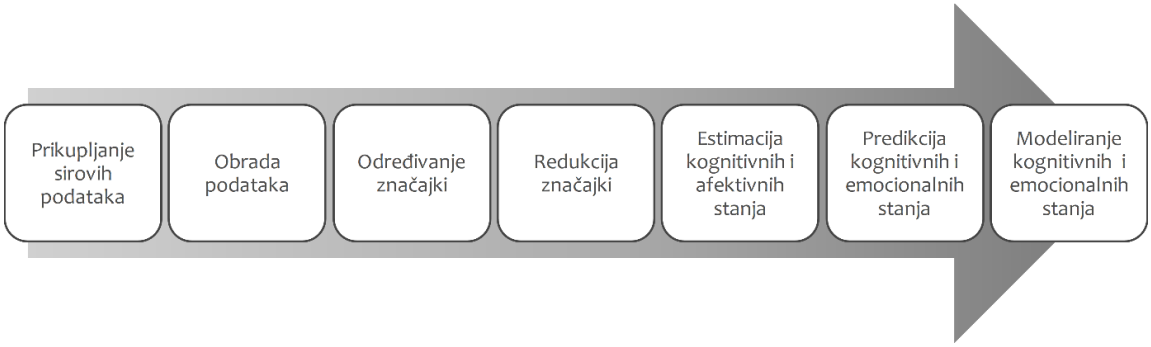
Novi elementi digitalne strategije Europske komisije objavljeni su 18. veljače 2020 [7]. U toj strategiji planirana je stroga regulacija umjetne inteligencije koju se smatra tehnologijom visokog rizika. Europa treba postati lider u pouzdanoj umjetnoj inteligenciji (engl. *trustworthy AI*), razlikujući se od slobodnijeg pristupa tim tehnologijama od Sjedinjenih Država i Kine. Komisija će izraditi nove zakone - uključujući zabranu AI sustava „crne kutije“ koje ljudi ne mogu protumačiti - za upravljanje visokorizičnim uporabama tehnologije: zdravstvenom zaštitom, transportom i kaznenim pravom. Nepristrani skupovi podataka potrebni su za izradu sustava visokog rizika kako bi oni mogli pravilno funkcionirati i osigurati poštivanja temeljnih prava, posebno nediskriminacije. Komisija također planira ponuditi novi certifikat „pouzdan AI“. Ako se za tako certificirane sustave kasnije ustanovi da su prekršili pravila oni se mogu suočiti s novčanim kaznama. Naposljetku, komisija navodi da će "pokrenuti široku europsku raspravu" o sustavima prepoznavanja lica, obliku AI koji može biometrijski identificirati ljude u maskama bez njihovog pristanka.

Također, važno je napomenuti da će prema nekim ekonomskim predviđanjima prepoznavanje emocija do 2023. god. postati komercijalna djelatnost od 23 milijarde dolara.

3. PREPOZNAVANJA EMOCIONALNIH STANJA KORIŠTENJEM POSTUPAKA UMJETNE INTELIGENCIJE

Strogo gledajući, cilj estimatora emocionalnih stanja je estimirati, odnosno procijeniti, na temelju fizioloških značajki (engl. *features*), ispitanikovo emocionalno stanje tijekom seanse, gdje je seansa definirana kao niz multimedijskih stimulacija različitog emocionalnog sadržaja i semantike tijekom kojih se prikupljaju fiziološki signali [8]. Tijekom seanse ispitanik se pobuđuje različitim stimulacijama te se kroz različite kognitivne i emocionalne procese unutar ispitanikovog mozga i tijela mijenja njegovo emocionalno stanje. Promjene u ispitanikovom emocionalnom stanju mogu se vidjeti u promjenama njegove fiziologije, izraza lica i u glasu [8]. Različitim akvizicijskim uređajima estimator emocionalnih stanja prikuplja različite fiziološke signale, poput EKG-a, EEG-a, vodljivosti kože, respiracije, temperature kože, EMG-a, itd., te korištenjem metoda dubinske analize računalno procjenjuje ispitanikovo emocionalno stanje [8, 9].

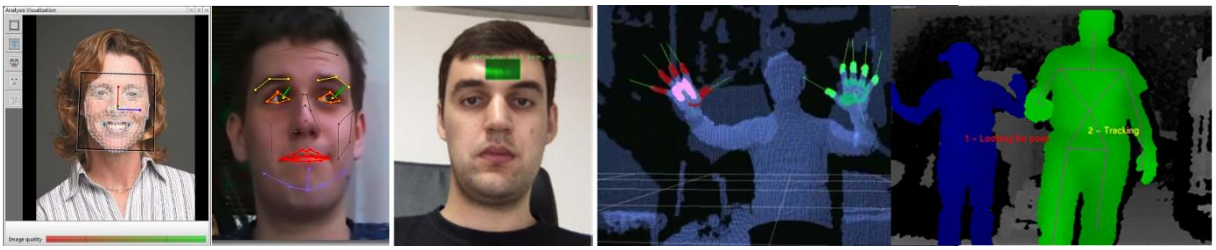
Kao što je prikazano na slici 2 postupak procjene emocija započinje s prikupljanjem sirovih podataka iz multimodalnih izvora korištenjem različitih senzora koje je potrebno obraditi. Svrha koraka obrade podataka je napraviti potrebne transformacije kako bi kasniji koraci bili uspješni. Prvenstveno to je otkloniti šum i nadomjestiti podatke koji nedostaju. Nakon toga slijedi određivanje značajki koje smanjuju kompleksnost velike količine podataka i svodi ih na nekoliko desetaka ili stotina numeričkih vrijednosti. Jednostavno rečeno, značajke opisuju model. Nakon koraka određivanja značajki slijedi odabir značajki. Ovi postupci koriste se kako bi se smanjila dimenzionalnost ulaznog skupa podataka i tako poboljšala učinkovitost postupaka strojnog učenja za izgradnju modela. Nakon odabira značajki koje nose maksimalnu informacijsku vrijednost pristupa se estimaciji, potom predikciji, a iz svega na kraju se dobiva jedinstveni model kognitivnih i afektivnih stanja [9].



Slika 2. Tijek općenitog postupak estimacije emocija [8, 9].

Izvori podataka za procjenu emocionalnih stanja mogu biti brojni (neki primjeri u Slika 3).

- izrazi lica, infracrvena (termalna) slika lica,
- smjer i dinamika pogleda,
- temperatura kože, elektrodermalne aktivnosti kože (vodljivost, galvanski odziv - GSR, …),
- srčani ritam i varijabilnost srčanog ritma, EKG,
- električna aktivnost mozga (EEG, evocirani potencijali, …),
- položaj i pokreti tijela,
- govorni signal (akustičke i lingvističke značajke) [10],
- pokreti miša i uporaba tipkovnice.



Slika 3. Primjer nekih senzora koji se koriste za estimaciju emocionalnih stanja i ponašanja.

Zbog smanjene nametljivosti i jednostavnijeg korištenja preporuča se uporaba beskontaktnih senzora umjesto kontaktnih. Procjena srčanog ritma može se i vršiti vidnim senzorima, tj. kamerom [11], bez potrebne korištenja kontaktnih senzora i na udaljenosti od nekoliko metara, a uz određena ograničenja moguće je i pametnim satovima i telefonima [12]. Tradicionalni termin za sve oblike ovakvih mjerenja je biometrija (engl. *biometrics*), ali može se koristiti i termin psihofiziologija (engl. *psychophysiology*) koji obuhvaća psihološke fenomene kao temelje fizioloških aktivnosti. Upravo ove manifestacije, odnosno aktivnosti, bilježe senzori.

4. ZAŠTITA OSOBNIH PODATAKA U KONTEKSTU PRIMJENE UMJETNE INTELIGENCIJE

Zaštita osobnih podataka, u užem smislu, je čvrsto određena hrvatskim i međunarodnim pravnim okvirom i pripadnim pravnim aktima.

Osobni podaci su svi podaci koji se odnose na pojedinca čiji je identitet utvrđen ili se može utvrditi („ispitanik“) [13]. Pravo na zaštitu osobnih podataka smatra se temeljnim ljudskim pravom.

„Svatko ima pravo na zaštitu osobnih podataka koji se na njega ili nju odnose“

Članak 8. Povelje Europske unije o temeljnim pravima (2016/C 202/02).

I također:

„Svatko ima pravo na zaštitu svojih osobnih podataka“

Članak 16. Ugovora o funkcioniranju Europske unije (2016/C 202/01).

Također, zaštita osobnih podataka utvrđena je s više pravnih akata [14]:

- Člankom 37. Ustava Republike Hrvatske (NN 85/10 – pročišćeni tekst) kojim se svakom jamči sigurnost i tajnost osobnih podataka, koji se bez privole ispitanika mogu prikupljati, obrađivati i koristiti samo uz uvjete određene zakonom, dok se zabranjuje uporaba osobnih podataka suprotna utvrđenoj svrsi njihova prikupljanja.
- Uredbom (EU) 2016/679 Europskog parlamenta i Vijeća od 27. travnja 2016. o zaštiti pojedinaca u vezi s obradom osobnih podataka i o slobodnom kretanju takvih podataka te o stavljanju izvan snage Direktive 95/46/EZ (Opća uredba o zaštiti podataka).
- Zakonom o provedbi Opće uredbe o zaštiti podataka (NN 42/18)
- Zakonom o zaštiti fizičkih osoba u vezi s obradom i razmjenom osobnih podataka u svrhe sprječavanja, istraživanja, otkrivanja ili progona kaznenih djela ili izvršavanja kaznenih sankcija (NN 68/18)

Pravo na pristup osobnim podacima je dio prava svakog ispitanika. Sukladno propisima kojima je utvrđena zaštita osobnih podataka, ispitanik ima pravo dobiti pisanu obavijesti od voditelja obrade obrađuju li se osobni podaci koji se na njega odnose, a ako se obrađuju pristup slijedećim informacijama [14]:

- svrha i pravni temelj obrade osobnih podataka;
- kategorijama osobnih podataka, ako ih ima;
- primateljima osobnih podataka;
- razdoblju pohrane;
- pravu na ispravak i brisanje;
- ograničenju prava pristupa i informacija koje se stavljaju na raspolaganje ili daju ispitaniku sukladno Zakonu o zaštiti fizičkih osoba u vezi s obradom i razmjenom osobnih podataka u svrhe sprječavanja, istraživanja, otkrivanja ili progona kaznenih djela ili izvršavanja kaznenih sankcija;
- izvoru podataka;
- pravu na podnošenje prigovora, odnosno pritužbe nacionalnom nadzornom tijelu.

Svatko tko smatra da mu je povrijeđeno neko pravo zajamčeno Općom uredbom o zaštiti podataka i Zakonom o provedbi Opće uredbe o zaštiti podataka može podnijeti zahtjev za utvrđivanje povrede prava Agenciji za zaštitu osobnih podataka.

Vrlo važno je pogledati definiciju postupka obrade osobnih podataka. Osobito je potrebno obratiti pozornost na završne odredbe koje određuju što predstavljaju „obrade osobnih podataka visokih rizika“ [15]:

„Obrada osobnih podataka obuhvaća radnje poput prikupljanja, bilježenja, čuvanja, uvida, otkrivanja, prenošenja ili uništavanja podataka. Tako primjerice možemo navesti da će Opća uredba o zaštiti podataka obuhvatiti obradu podataka zaposlenika, potrošača i klijenata, građana od strane državne administracije, pacijenata, učenika, studenata, članova udruga, korisnika društvenih mreža i svaku drugu obradu osobnih podataka koja nije u okviru gore navedenih iznimki. Također, novim

ili jačim pravilima bit će obuhvaćene one djelatnosti koje se bave obradom osobnih podataka visokog rizika za koje će biti potrebno provesti procjenu učinka. To su obrade koje se odnose na sustavnu i opsežnu procjenu osobnih aspekata pojedinaca automatiziranim putem, opsežnu obradu posebnih kategorija podataka ili podataka o kaznenim osudama ili kažnjivim djelima te sustavno praćenje javno dostupnog područja u velikoj mjeri“.

Zaključujemo da postupci estimacije emocionalnih stanja, ponašanja i kognicija zapravo predstavljaju obrade osobnih podataka visokih rizika. Pogotovo primjena estimacije emocija iz facijalnih ekspresija je jedna takva tehnologija umjetne inteligencija koja koristi osobne podatke, pokazala je nezadovoljavajuće rezultate, a vrlo često se primjenjuje u praksi.

Pri tome potrebno je voditi računa o pojedinim pravim građana, odnosno ispitanika. Ta lista je opsežna [15]:

- „transparentnost (12 - 14): pružanje informacija prilikom prikupljanja osobnih podatak kada voditelj obrade mora među ostalim informacijama obavijestiti ispitanika i o svojem identitetu i kontakt podacima, svrhama obrade i pravnoj osnovi za obradu podataka, primateljima, iznošenju u treće zemlje, razdoblju pohrane, mogućnosti povlačenja privole, itd.;
- pristup podacima (15): dobiti od voditelja obrade potvrdu obrađuju li se osobni podaci koji se odnose na njega te ako se takvi osobni podaci obrađuju, pristup osobnim podacima i informacije, među ostalim, o obrađenim osobnim podacima, o svrsi obrade, roku pohrane, iznošenju u treće zemlje itd.;
- pravo na ispravak (16): ispitanik ima pravo zahtijevati ispravak netočnih osobnih podataka koji se na njega odnose, a uzimajući u obzir svrhe obrade, ispitanik ima pravo dopuniti nepotpune osobne podatke, među ostalim i davanjem dodatne izjave;
- brisanje („pravo na zaborav“) (17): ispitanik ima pravo od voditelja obrade ishoditi brisanje osobnih podataka koji se na njega odnose bez nepotrebnog odgađanja te voditelj obrade ima obvezu obrisati osobne podatke bez nepotrebnog odgađanja ako, među ostalim, osobni podaci više nisu nužni u odnosu na svrhu obrade, ispitanik je povukao privolu za obradu, osobni podaci su nezakonito obrađeni itd., ovo pravo ima ograničenja pa tako na primjer političar ne može zatražiti brisanje informacija o sebi koje su dane u okviru svojega političkog djelovanja;
- pravo na ograničenje obrade (18): u pojedinim situacijama (na primjer kada je točnost podataka osporavana ili kada pravo na brisanju ispitanik želi da voditelj obrade zadrži njegove podatke) ispitanik ima pravo zahtijevati da se obrada ograniči uz iznimku pohrane i nekih drugih vrsta obrade;
- pravo na prenosivost (20): ispitanik ima pravo zaprimiti svoje osobne podatke, a koje je prethodno pružio voditelju obrade, u strukturiranom obliku te u uobičajeno upotrebljavanom i strojno čitljivom formatu te ima pravo prenijeti te podatke drugom voditelju obrade bez ometanja od strane voditelja obrade kojem su osobni podaci pruženi, ako se obrada provodi automatiziranim putem i temelji na privoli ili ugovoru;
- pravo na prigovor (21): ispitanik ima pravo uložiti prigovor na obradu osobnih podataka ako se ista temelji na zadaće od javnog interesa, na izvršavanje službenih ovlasti voditelja obrade ili na legitimne interesa voditelja obrade (uključujući i profiliranje), tada voditelj obrade ne smije više obrađivati osobne podatke ispitanika osim ako dokaže da njegovi legitimni razlozi za obradu nadilaze interese ispitanika te radi zaštite pravnih zahtjeva, također ako se ispitanik protivi obradi za potrebe izravnog marketinga, osobni podaci više se ne smiju obrađivati;
- pravo usprotiviti se donošenju automatiziranih pojedinačnih odluka (profiliranje) (22): ispitanik ima pravo da se na njega ne odnosi odluka koja se temelji isključivo na automatiziranoj obradi, uključujući izradu profila, koja proizvodi pravne učinke koji se na njega odnose ili na sličan način značajno na njega utječu, osim ako je takva odluka potrebna za sklapanje ili izvršenje ugovora između ispitanika i voditelja obrade podataka, ako je dopuštena pravom EU-a ili nacionalnim pravom koji se propisuju odgovarajuće mjere zaštite prava i sloboda te legitimnih interesa ispitanika ili temeljena na izričitoj privoli ispitanika“.

5. OGRANIČENJE PRISTUPA JAVNIM USLUGAMA I PRIJEDLOG RJEŠENJA

Što je javna usluga (engl. *public service*)? U svojoj definiciji svaka javna usluga obuhvaća tijela koja pružaju usluge i usluge od općeg interesa koja ta tijela pružaju. Vlasti mogu nametnuti obavezu javne usluge tijelu koje pruža tu uslugu (npr. zrakoplovne kompanije, željeznički prijevoznici, proizvođači energenata, itd.). U praksi pojam javnih usluga i koncept javnog sektora (uključujući i državnu administraciju) često se pogrešno zamjenjuju kao istoznačnice, ali ipak ova dva pojma razlikuju se po funkciji, statusu, vlasništvu i korisnicima.

U tradicionalnom razmišljanju osobni podaci nalaze se na osobnim računalima i pametnim telefonima korisnika ili sadržani njihovim mrežnim identitetima. Ali to nisu svi osobni podaci. Osobni podaci su i svi oni drugi podaci koji proizlaze iz ponašanja, stavova, razmišljanja, kognicije i emocionalnih reakcija korisnika.

Tijekom korištenja javnih usluga ove kategorije osobnih podataka svakodnevno ustupamo svojevoljno, a i besplatno. Ovaj problem se često zanemaruje ili posve negira.

Nameću se četiri važna pitanja: 1) Da li se i ti osobni podaci prikupljaju? 2) Za što se koriste? 3) Da li smo o tome, kao i kao korisnici i kao davatelji takvih osobnih podataka, obaviješteni? 4) Da li imamo mogućnost korištenja javnih servisa bez implicitnog ili eksplicitnog ustupanja naših osobnih podataka?

Stoga predlažem da se na vidljivim mjestima u prostorima gdje se vrši estimacija emocionalnih stanja nalaze standardizirane oznake koje na to upozoravaju.

Svrha nije samo informiranje već da korisnik usluge ocijeni da li mu je prihvatljiva razina obrade osobnih podataka i obuhvaćenih radnji.

Dakle, predlažem da se kao do sada ne me označavaju načini prikupljanja podatka (npr. video nadzor prostorije, snimanje razgovora, …), već je potrebno označavati primjenu prikupljenih podataka (orofiliranje, moderalianje, analiza, *customer churn analysis*, …).

Pri tome je vrlo važno omogućiti korisnicima alternativni pristup uslugama, posebice javnim servisima, makar odbiju predati svoje osobne podatke.

Mjerenje emocionalnih stanja, ponašanja, stavova i mišljenja, odnosno prikupljanje podataka u svrhu profiliranja, mora se provoditi isključivo uz privolu zaposlenika. Uzimajući u obzir prava građana, to ujedno implicira da usluga mora biti dostupna i ako privola za prikupljanje podataka ne postoji, odnosno ako korisnik nije dopustio prikupljanje osobnih podataka u svrhu profiliranja.

Davatelj usluga ne smije uskraćivati pružanje usluge korisnicima koji ne dopuste prikupljanje podataka. Drugim riječima, usluga ne smije biti metoda ucjene ili prisile. Niti jedna javna usluga, a pogotovo ne javna, ne smije biti uvjetovana

Korištenje usluga mora biti omogućeno bez obaveze prikupljanja podataka na koje korisnik ne pristaje.

6. ZAKLJUČAK

Prije svega važno je nedvosmisleno ustvrditi da se tehnologije za procjenu emocionalnih stanja, ponašanja, stavova i mišljenja ubrzano razvijaju i pronalaze široku primjenu u praksi. Osim primjene na korisnicima društvenih mreža, ove tehnologije danas se već redovito koriste i tijekom razgovora za posao ili u policijskim istragama. Brojni drugi oblici primjene su svakako mogući i potrebno je o njima voditi računa u budućnosti.

Tehnologije za procjenu emocionalnih stanja koje koriste postupke umjetne inteligencije su, kao što je nedvojbeno empirijski demonstrirano, nepouzdane i često daju poopćene rezultate koji se mogu pogrešno interpretirati, odnosno dovesti do krivih zaključaka. Rezultati mogu biti specifični za homogene skupine ispitanika koji su pobuđeni koristeći za njih prilagođen stimulacijski protokol. Takav protokol nužno obuhvaća modalitet pobude, semantiku (kontekst i sadržaj) pobude, vremenske aspekte (trajanje i pauze), individualnu psihologiju (s neurološkim temeljima), te eksperimentalnu okolinu. Svi ovi aspekti koji utječu na točnost rezultata u svakodnevnoj primjeni – van strogo nadziranog laboratorijskog okruženja – najčešće se zanemaruju, ili se na njih ne može učinkovito utjecati. Nedostatak personalizacije u pobudi i kasnijoj interpretaciji rezultata dovodi do odviše općenitih zaključaka koji mogu biti, a najčešće jesu, pogrešni kad se primijene na individualne ispitanike. Procijenjeno emocionalno stanje, ponašanje ili kognicija grupe u praksi nije identično procjeni za svakog pojedinca unutar te grupe. Jednostavno rečeno, izjednačavanje globalne s pojedinačnom estimacijom nužno će dovesti do pogrešne interpretacije emocionalnih reakcija, namjera, razmišljanja ili uzroka ponašanja korisnika.

Istodobno, postojeći zakonodavni okvir u vezi zaštite osobnih podataka, kako hrvatski tako i međunarodni, još uvijek ne prepoznaje potencijalne opasnosti koje primjena tehnologija umjetne inteligencije donosi u domeni prepoznavanja emocionalnih stanja, ponašanja i kognicije. Unatoč pozitivnim pomacima u ovom pogledu, povećanoj medijskoj pažnji i sve većem interesu zakonodavaca za pitanje regulacije umjetne inteligencije, ovo područje još nije regulirano na zadovoljavajući način. Potencijalne zloupotrebe izuzimanja i korištenja osobnih podataka su moguće. Štoviše, trenutačno korisnici ne samo da nisu informirani za što se njihovi osobni podaci koriste, već nisu niti svjesni da se vrši automatska estimacija njihovih emocionalnih stanja, ponašanja i kognicije, te da se tako dobiveni rezultati koriste za donašanje odluka s kojima su oni povezani. Takve odluke donose se djelomično ili u potpunosti automatizirano, a mogu se odnositi na – primjerice – odluku da li će dobiti traženi posao, ili da li će biti osumnjičeni tijekom policijskog postupanja.

Da bi se spriječilo neobjektivno i automatizirano donašanje zaključaka prvenstveno je potrebno osvijestiti moguće opasnosti koje primjena tehnologija umjetne inteligencije za procjenu emocionalnih stanja donosi. Potrebno je snažnije zakonski regulirati uporabu ovakvih tehnologija u bilo kojem obliku ili području primjene, a posebice nad javnim servisima koji moraju biti uvijek dostupni svim građanima, bez izuzetaka ili vanzakonskih ograničenja.

S tom svrhom u ovom radu predlaže se izrada standardiziranih znakova koji upozoravaju korisnike na prisutnost postupaka prikupljanja osobnih podataka o njihovu emocijama, ponašanju i kogniciji, te namjenu takvog postupka. Takvi znakovi moraju biti nedvosmisleni i lagano razumljivi, a njihovo razumijevanje neovisno o jeziku ili kulturi korisnika. To mogu biti vizualni simboli ili tekstualne poruke, ali obavezno se moraju nalaziti na lako zamjetljivom mjestu.

Također je vrlo važno korisnicima pružiti alternativu korištenja nekog servisa, posebice javnog, ukoliko korisnici ne želi dopustiti prikupljanje vlastitih osobnih podataka u svrhu estimacije vlastitih emocionalnih stanja, ponašanja i kognicije koji se koriste za donašanje automatiziranih odluka o njima. Bilo koji servis, tim više javni, ne smije biti uvjetovan snimanjem fizioloških podataka i profiliranjem korisnika. Ako se takvi postupci automatizirano primjenjuju, onda drugi način ostvarenja usluge servisa mora biti osiguran.

Naposljetku, možemo zaključiti da je važno kontinuirano voditi aktivnu stručnu i znanstvenu raspravu o svim aspektima razvoja i primjene tehnologija za procjenu emocionalnih stanja. U ovom području sigurno će se dogoditi brojni koraci naprijed, kako u pogledu senzora koji se koriste, količine i složenosti prikupljanih podataka, složenosti postupaka obrade podataka, tako i u pogledu točnosti i primjene dobivenih podataka.

Literatura:

1 Legg, S., & Hutter, M. (2007). A collection of definitions of intelligence. Frontiers in Artificial Intelligence and applications, 157, 17.

2 Elon Musk, https://twitter.com/elonmusk/status/1229546793811226627, pristupljeno 24. veljače 2020.

3 The Verge, Elon Musk says AI development should be better regulated, even at Tesla, https://www.theverge.com/2020/2/18/21142489/elon-musk-ai-regulation-tweets-open-ai-tesla-spacex-twitter, pristupljeno 24. veljače 2020.

4 AI Now Report 2019, https://ainowinstitute.org/AI_Now_2019_Report.pdf, pristupljeno 24. veljače 2020.

5 Barrett, L. F., Adolphs, R., Marsella, S., Martinez, A. M., & Pollak, S. D. (2019). Emotional expressions reconsidered: challenges to inferring emotion from human facial movements. Psychological Science in the Public Interest, 20(1), 1-68.

6 MIT Technology Review, https://www.technologyreview.com/2019/07/26/238782/emotion-recognition-technology-artifical-intelligence-inaccurate-psychology/, pristupljeno 24. veljače 2020.

7 Shaping Europe's digital future: Commission presents strategies for data and Artificial Intelligence, EU press release, https://ec.europa.eu/commission/presscorner/detail/en/ip_20_273, 19 February 2020.

8 Ćosić, K., Popović, S., Horvat, M., Kukolja, D., Dropuljić, B., Kovač, B., & Jakovljević, M. (2013). Computer-aided psychotherapy based on multimodal elicitation, estimation and regulation of emotion. Psychiatria Danubina, 25(3), 0-346.

9 Kukolja, D., Popović, S., Horvat, M., Kovač, B., & Ćosić, K. (2014). Comparative analysis of emotion estimation methods based on physiological measurements for real-time applications. International journal of human-computer studies, 72(10-11), 717-727.

10 Lugović, S., Dunđer, I., & Horvat, M. (2016, May). Techniques and applications of emotion recognition in speech. In 2016 39th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO), pp. 1278-1283.

11 Horvat, M., & Fodor, D. (2018). System for remote heart rate measurement using a consumer camera. Polytechnic and Design, 6(2), 128-134.

12 Pejak, I., Otočan, D., & Horvat, M. (2017). Application of Android Wear smartwatches with photoplethysmographic sensors in biofeedback therapy. Polytechnic and Design, 5(2), 133.

13 Europska komisija, Zaštita podataka, https://ec.europa.eu/info/law/law-topic/data-protection/reform/what-personal-data_hr, pristupljeno 24. veljače 2020.

14 Ministarstvo unutarnjih poslova Republike Hrvatske, Zaštita osobnih podataka, https://mup.gov.hr/zastita-osobnih-podataka/222, pristupljeno 24. veljače 2020.

15 Agencija za zaštitu osobnih podataka, Vodič kroz opću uredbu o zaštiti podataka, https://azop.hr/info-servis/detaljnije/vodic-kroz-opcu-uredbu-o-zastiti-podataka, pristupljeno 24. veljače 2020.

Podaci o autoru:

Doc. dr. sc. Marko Horvat, v. pred.

e-mail: marko.horvat3@gmail.com

Diplomirao, magistrirao i doktorirao iz znanstvenog područja tehničke znanosti znanstveno polje računarstvo na Fakultetu elektrotehnike i računarstva Sveučilišta u Zagrebu, 1999., 2007. i 2013. godine. Trenutno je prodekan za znanost, vanjsku suradnju i nove studije, te viši predavač na Tehničkom veleučilištu u Zagrebu i naslovni docent na Sveučilištu u Zagrebu u navedenom području i polju. Sudjeluje u pripremi i provedbi niza stručna i znanstvenih projekata. Kao autor ili koautor objavio je više od 100 znanstvenih i stručnih radova, sažetaka sa skupova, poglavlja u knjigama, skripti i udžbenika, a od toga 9 izvornih znanstvenih radova u CC časopisima. Održao je niz javnih i pozvanih predavanja, kao i stručnih tečajeva. Član je stručne udruge IEEE u statusu Senior Member i Hrvatskog astronomskog društva.

Tehničko veleučilište u Zagrebu
Informatičko-računarski odjel
Vrbik 8, 1000 Zagreb, Hrvatska

RAČUNALNE IGRE: OD IDEJE DO IZDAVAČA

Mirko Sršen bacc. ing. techn. inf., izv. prof. dr. sc. Tihomir Orehovački

SAŽETAK

Put od ideje do izdavača u industriji računalnih igara je jako težak i nepredvidljiv. Računalna igra pod nazivom Ominous Relict služi kao primjer pretvorbe ideje u gotov proizvod te nam omogućava uvid u probleme s kojima se proizvođači računalnih igara svakodnevno susreću. Kroz rad se iznose ideje implementacije koje su se aktivno primjenjivale u razvoju Ominous Relicta. Cijela kodna baza se zasniva na mašini stanja koja pomaže pri automatizaciji pojedinih dijelova računalne igre te dodaje određeni stupanj skalabilnosti i apstrakcije. Rad sa FMOD studijom za obradu zvuka te CG programskim jezikom za proizvodnju sjenila samo su neke od tehnologija koje Ominous Relict sadrži u sebi i koje će biti opisane u ovome radu.

ABSTRACT

Path from idea to publisher in industry of computer games is very hard and unpredictable. Computer game that goes by name Ominous Relict serves as an example of how to convert idea to finished product thus allowing us perception of problems that game developers encounter on daily basis. This paper presents ideas and their implementations that were applied when developing Ominous Relict. Whole code base is founded on state machine that helps with automatization of specific parts in computer games, and it provides certain degree of scalability and abstraction. Working with FMod studio for sound editing and CF programming language for creating shaders are just some of the technologies Ominous Relict holds and are described in this paper.

1. UVOD

Industrija računalnih igara predstavlja jednu od najbrže rastućih industrija u svijetu. Rastom potražnje porasla je i potreba za proizvodnjom sve većeg broja računalnih igara i opreme. Svaka računalna igra ima svoj stil odnosno žanr kojem pripada. Strategije, RPG-ovi, FPS-ovi, RTS-ovi, MOBA samo su neke od popularnih žanrova koji su prisutni na tržištu. U radu će biti predstavljen projekt pod nazivom Ominous Relict koji će se koristiti kako bi se utvrdili i analizirali problemi s kojima se proizvođači igara svakodnevno susreću. Koristeći se konkretnim primjerima iz projekta, rad opisuje implementaciju i korištenje mašine stanja, sustav za pohranu i učitavanje podataka, injekciju zavisnosti i način na koji su zvuk i vizualni efekti ukomponirani sa kodnom bazom.

2. OPIS PROJEKTA

U ovome radu je stavljen fokus na produkcijski proces projekta pod nazivom Ominous Relict (u nastavku OR). Radnja OR-a se događa u vrijeme druge industrijske revolucije gdje glavni junak John kreira stroj za simulaciju lucidnih snova kako bi se oporavio od gubitka oca koji je nestao tijekom rata na Tibetu. Kada John počinje gubiti osjećaj za realnost, dolazi do zanimljivog preokreta te se njegovi lucidni snovi krenu miješati sa njegovom percepcijom realnosti. Vizualni dizajn (slika 1) OR-a je baziran na 2D piksel slikama inspiriranim elementima znanstvene fantastike i žanrovima kojima računalna igra pripada su: platformer i avantura. Igra je izrađena u Unity razvojnom okruženju te sadrži par zanimljivih koncepta koji će biti opisani u poglavljima rada koji slijede.

Slika 8. Prvi elitni neprijatelj (Shaman)

3. MAŠINA STANJA

Mašina stanja nudi kvalitetan, produktivan i neovisan način pisanja koda. Objekt se promatra kroz stanja, a ne kroz funkcije te kao takav istovremeno može imati samo jedno aktivno stanje. Primjer koda 1 prikazan u nastavku definira osnovnu konstrukciju svakog stanja unutar igre.

```
1   using UnityEngine;
2
3   namespace General.State
4   {
5       /// <summary>
6       /// Core state for all actions.
7       /// </summary>
8       public abstract class State : MonoBehaviour
9       {
10          /// <summary>
11          /// Gets or sets state priority.
12          /// </summary>
13          public int Priority { get; set; }
14
15          /// <summary>
16          /// Gets or sets state controller.
17          /// </summary>
18          public StateController controller { get; protected set; }
19
20          /// <summary>
21          /// Gets or sets design controller
22          /// </summary>
23          protected DesignController designController { get; set; }
24
25          /// <summary>
26          /// Initializes components and parameters.
27          /// </summary>
28          protected virtual void Initialization_State()
29          {
30              DiContainerLibrary.DiContainer.DiContainerInitializor.RegisterObject(this);
31          }
32
33          /// <summary>
34          /// Starts on beginning of the state.
35          /// </summary>
36          public virtual void OnEnter_State()
37          {
38              if (designController != null)
39              {
40                  designController.StartTask(this.GetType().Name);
41              }
42          }
43
44          /// <summary>
45          /// Checks when is the best time for state to become active.
46          /// </summary>
47          public virtual void Update_State()
48          {
49          }
50
51          /// <summary>
52          /// While state is active.
53          /// </summary>
54          public virtual void WhileActiveFixed_State()
55          {
56          }
57
58          /// <summary>
59          /// While state is active.
60          /// </summary>
61          public virtual void WhileActive_State()
62          {
63          }
64
65          /// <summary>
66          /// Happens when state is being swapped by other state.
```

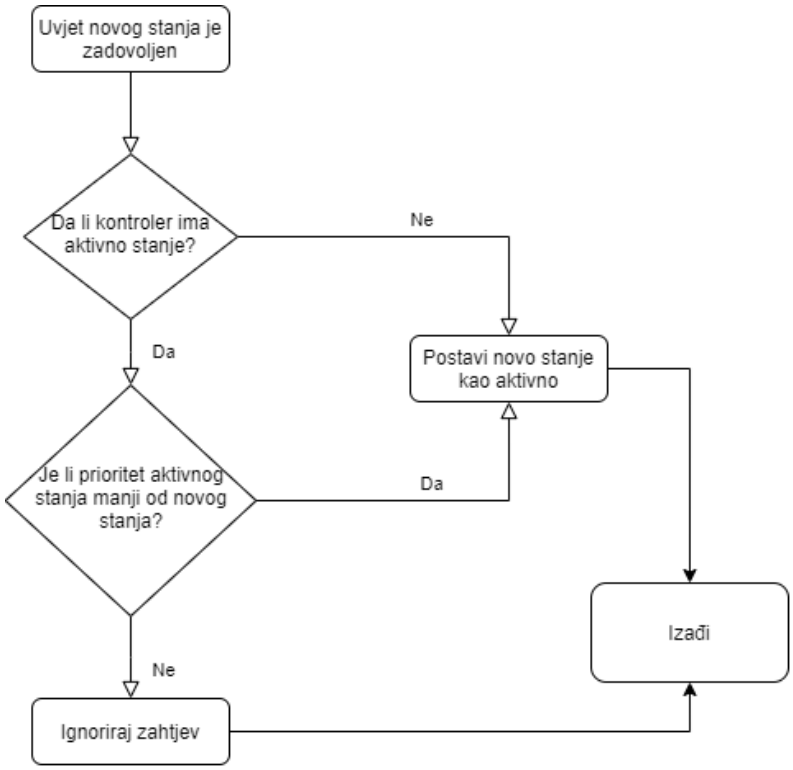
```
67          /// </summary>
68          public virtual void OnExit_State()
69          {
70              if (this.designController != null)
71              {
72                  this.designController.StopTask(this.GetType().Name);
73              }
74          }
75
76          private void Awake()
77          {
78              designController = GetComponent<DesignController>();
79              controller = GetComponent<StateController>();
80          }
81
82          // Use this for initialization
83          void Start()
84          {
85              Initialization_State();
86          }
87
88          // Update is called once per frame
89          void Update()
90          {
91              Update_State();
92          }
93      }
94  }
95
```

Primjer koda 1. Implementacija mašine stanja

Priority (linija 13) je brojčana vrijednost koja se koristi kod promjene stanja gdje se utvrđuje da li trenutno aktivno stanje ima manji prioritet od stanja koje ga želi zamijeniti. Controller (linija 18) je upravitelj stanja koji ima zadaću da radi tranzicije između stanja te izvršavanje logike trenutno aktivnog stanja. DesignController (linija 23) se sastoji od dva podkontrolera od kojih jedan ima zadaću upravljati sa zvukom, a drugi sa animacijom objekta. Funkcija ,,Initialization_State“ (linija 28) definira konstruktor stanja koji ima zadaću registrirati sve komponente i vrijednosti kao što su prioritet i injekcija ovisnosti. Funkcija ,,OnEnter_State“ (linija 36) se isključivo poziva od strane upravitelja stanja kada to stanje postaje aktivno. Razlika između ,,Initialization_State“ i ,,OnEnter_State“ funkcije je da ,,OnEnter_State“ postavlja vrijednosti određenih parametara koji nisu fiksni već se prilikom aktivacije određenog stanja ponovno postavljaju (npr. destinacija, meta, brzina trčanja, itd.). Funkcija ,,Update_State“ (linija 47) se brine o provjeri uvjeta za nastupanje tog konkretnog stanja na snagu. Funkcija ,,WhileActive_State“ i ,,WhileActiveFixed_State“ (linija 54 i 61) je logika koja se odvija dok je to stanje aktivno. Razlika između ,,WhileActive_State“ i ,,WhileActiveFixed_State“ funkcije je da se ,,WhileActiveFixed_State“ koristi za fizičke iteracije dok se ,,WhileActive_State“ koristi za sve druge potrebe. Funkcija ,,OnExit_State“ (linija 68) se poziva isključivo od upravitelja stanja kad stanje prestaje biti aktivno te uglavnom negira sve promjene koje je funkcija ,,OnEnter_State“ postavila. Ostale funkcije su isključivo vezane za Unity razvojno okruženje i služe za inicijalizaciju polja i pozivanje funkcija koje stanje definira.

4. PROMJENA STANJA

Shema funkcionalnosti upravitelja stanja (u nastavku upravitelj) je prikazana na slici 2. U slučaju da postoje dva stanja od kojih je jedno neaktivno stanje A i drugo aktivno stanje B, stanje A konstantno provjerava unutar funkcije ,,Update_State“ da li su uvjeti zadovoljeni te u slučaju da jesu šalje upravitelju zahtjev za aktivaciju. Upravitelj provjerava da li već postoji aktivno stanje te u slučaju da ne postoji, automatski postavlja stanje A kao aktivno dok u protivnom uspoređuje prioritete oba stanja. Ako B ima prioritet veći ili jednak od A, upravitelj ignorira zahtjev, dok se u suprotnom vrši izmjena stanja na način da se najprije pozove izlazna funkcija aktivnog stanja B, potom se aktivno stanje postavi na stanje A te se pozove funkcija ulaska sa aktivnog stanja A.



Slika 2. Dijagram toka stanja

5. SPREMANJE PODATAKA

Unity razvojno okruženje ne posjeduje izravni sustav s kojim možemo spremati podatke o trenutnom napretku igrača stoga je u OR-u projektu nastala potreba za brzim, jednostavnim i sigurnim načinom spremanja podataka.

Svi podatci unutar OR-a se spremaju i dohvaćaju pomoću serijaliziranih objekata. Serijalizacija je postupak pretvaranja objekta u niz bitova za potrebe slanja kroz mrežu ili zapisivanja na fizičku memoriju. Specifični objekti se pronalaze pomoću jedinstvenog identifikatora kojeg svaki objekt mora sadržavati u sebi. Primjer koda 2 prikazuje sučelje jedinstvenog identifikatora.

```
namespace SaveData
{
    public interface IUniqueIndex
    {
        /// <summary>
        /// Gets or sets unique indexer.
        /// </summary>
        string Id { get; set; }
    }
}
```

Primjer koda 2. Definicija sučelja jedinstvenog identifikatora

Svaki serijalizirani objekt posjeduje odgovarajuće sučelje s kojim je definirana jasna struktura podataka koja će biti pohranjena u memoriju. Definicijom jasnog sučelja spriječena je mogućnost da se polja koja su unutar klase, a nisu implementirana od strane sučelja, spremaju u memoriju. „DbContextConfiguration“ atribut je zadužen za definiranje putanje datoteke u koju će se objekti koji implementiraju to sučelje zapisivati dok atribut „Serializable“ označava da će objekt bili serijaliziran. Valja naglasiti da ovim pristupom nije moguće serijalizirati objekte koji su definirani u Unity-jevoj biblioteci (primjerice GameObject, Transform) te u slučaju kad je u OR-u nastala potreba za tim, spremala se putanja do tih objekata te su se oni učitavali pomoću funkcije učitavanja podataka iz memorije „Resource.Load“. Primjer koda 3 prikazuje način na koji se definiraju objekti korišteni od strane sustava za spremanje podataka.

```
namespace SaveData
{
    [DbContextConfiguration("SlotData")]
    public interface ISlotData : IUniqueIndex
    {
        /// <summary>
        /// Gets or sets slot capacity. <code>null if slot is empty.</code>
        /// </summary>
        int CurrentCapacity { get; set; }

        /// <summary>
        /// Gets or sets items in slot.
        /// </summary>
        string ItemsResource { get; set; }
    }

    [Serializable]
    public class SlotData : ISlotData
    {
        /// <inheritdoc />
        public int CurrentCapacity { get; set; }

        /// <inheritdoc />
        public string ItemsResource { get; set; }

        /// <inheritdoc />
        public string Id { get; set; }
    }
}
```

Primjer koda 3. Definicija sučelja jedinstvenog identifikatora

Primjer koda 4 prikazuje klasu „SaveAndLoadData“ koja služi za pisanje i čitanje podataka. „SaveAndLoadData<TData>“ kao generički parametar prima objekt unutar kojeg mora biti implementirano „IUniqueIndex“ sučelje. U funkciji ,,LoadSpecificData“ (linija 12) kao ulazni parametar primamo jedinstveni identifikator tog objekta. U prvom koraku funkcija ,,GetAttribute“ (linija 61) dohvaća „DbContextConfiguration“ te saznaje putanju gdje se datoteka sa podatcima nalazi. U slučaju da datoteka nije pronađena, funkcija završava izvršavanje te vraća praznu (eng. null) vrijednost dok se u protivnom pronađena datoteka učitava u memoriju te se deserijalizira u listu objekata (objekti odgovaraju generičkom parametru klase). Funkcija vraća prvi objekt sa podudarajućim identifikatorom ili praznu vrijednost u slučaju da objekt s tim identifikatorom nije pronađen. Funkcija ,,LoadAllData“ (linija 28) funkcionira na isti način kao i ,,LoadSpecificData“, samo što ona vraća cijelu datoteku u obliku liste. Funkcija ,,SaveData“ (linija 42) serijalizira podatke u datoteku koju prethodno stvori u slučaju da ista nije postojala. U slučaju kada se radi o velikim količinama podataka, ovaj pristup nije prikladan iz razloga što se ovim postupkom cijela datoteka učitava u memoriju. Datoteke koje su se koristile unutar OR-a isključivo su bile veličine nekoliko kilobajta.

```
1  using System.Collections.Generic;
2  using System.IO;
3  using System.Linq;
4  using System.Runtime.Serialization.Formatters.Binary;
5
6  namespace SaveData
7  {
8      public static class SaveAndLoadData<TData> where TData : class, IUniqueIndex
9      {
10         private static IList<TData> dataContext = new List<TData>();
11
12         public static TData LoadSpecificData(string id)
13         {
14             if (!File.Exists(GetAttribute()))
15             {
16                 return null;
17             }
18
19             using (var stream = File.OpenRead(GetAttribute()))
20             {
21                 BinaryFormatter formatter = new BinaryFormatter();
22                 var result = (IList<TData>)formatter.Deserialize(stream);
23
24                 return result.FirstOrDefault(x => x.Id == id);
25             }
26         }
27
```

```
28          public static IList<TData> LoadAllData()
29          {
30              if(!File.Exists(GetAttribute()))
31              {
32                  return dataContext;
33              }
34
35              using (var stream = File.OpenRead(GetAttribute()))
36              {
37                  BinaryFormatter formatter = new BinaryFormatter();
38                  return (IList<TData>)formatter.Deserialize(stream);
39              }
40          }
41
42          public static void SaveData(TData data)
43          {
44              dataContext = LoadAllData();
45
46              var exist = dataContext.FirstOrDefault(x => x.Id == data.Id);
47
48              if (exist != null)
49              {
50                  dataContext.Remove(exist);
51              }
52
53              dataContext.Add(data);
54              using (var stream = File.OpenWrite(GetAttribute()))
55              {
56                  BinaryFormatter formatter = new BinaryFormatter();
57                  formatter.Serialize(stream, dataContext);
58              }
59          }
60
61          private static string GetAttribute()
62          {
63              Var dbAttribute = typeof(TData)
64              .GetCustomAttributes(typeof(DbContextConfiguration), true)
65              .FirstOrDefault() as DbContextConfiguration;
66              if (dbAttribute != null)
67              {
68                  return dbAttribute.FilePath;
69              }
70
71                throw new FileNotFoundException(string.Concat("Could not load attribute from the class"));
72          }
73      }
74  }
```

Primjer koda 4. Implementacija sustava za spremanje podataka

6. FMOD STUDIO ZA OBRADU ZVUKA

FMod je studio namijenjen obradi i implementaciji zvuka unutar računalnih igara. U kontekstu funkcionalnosti koje su bile potrebne OR-u, FMod se pokazao kao puno bolji izbor nego Unity-jeva standardna implementacija zvuka. Glavne prednost Fmod-a su vezane uz napredan izgled korisničkog sučelja (slika 3) i mogućnost slanja parametara na osnovu kojih svi zvukovi unutar OR-a mogu varirati u pozadinskim zvukovima, tempu, glasnoći i mnogim drugim aspektima. FMod posjeduje 2D i 3D postavke zvuka. 2D zvuk se koristi za ambijentalnu glazbu dok se 3D upotrebljava za prostorne zvukove (one zvukove koji se nalaze u sceni i glasnoća im ovisi o tome koliko je igrač daleko od njih). Ovim postupkom se rasteretio razvoj posebnog sustava za upravljanje zvukom te se povećala sama kvaliteta finalnog proizvoda.

Slika 3. Sučelje FMod studija

Način na koji pozivamo FMod studio jako je jednostavan, a cijela implementacija i sinkronizacija sa mašinom stanja je prikazana u primjeru koda 5. Klasa „StateSoundData“ sadrži 1:1 vezu između stanja i zvuka. Kolekcija „soundData“ (linija 18) ima zadaću učitati iz djece objekta sve instance StateSoundData klase. Sve instance su spremljene u jednu kolekciju radi lakšeg i bržeg dohvaćanja tokom rada igre. ,,PlaySound“ (linija 30) je funkcija koja se poziva u trenutku kad stanje postane aktivno. Zvuk se pokreće pozivom funkcije ,,Start“ (linija 45). Funkcija ,,StopSound“ (linija 53) se poziva u trenutku kad stanje prestaje biti aktivno, a funkcija ,,SetParameter“ (linija 69) postavlja specifične parametre unutar FModa da bi se pokrenuli specifični zvukovi koji ovise o njima (primjerice, što je više neprijatelja na sceni, to je ritam glazbe intenzivniji).

```
1   using System.Collections.Generic;
2   using System.Linq;
3   using FMOD.Studio;
4   using UnityEngine;
5
6   namespace General.State
7   {
8       public class FModSoundController : MonoBehaviour
9       {
10          /// <summary>
11          /// Gets or sets
12          /// </summary>
13          private Dictionary<string, EventInstance> stateSoundData { get; set; }
14
15          /// <summary>
16          /// Gets ot sets all sound data from object.
17          /// </summary>
18          private List<StateSoundData> soundData { get; set; }
19
20          private void Awake()
21          {
22              soundData = GetComponentsInChildren<StateSoundData>().ToList();
23              stateSoundData = new Dictionary<string, EventInstance>();
24          }
25
26          /// <summary>
27          /// Plays the sound with given key.
28          /// </summary>
29          /// <param name="key">Key of the audio clips from <see cref="ISoundData"/></param>
30          public void PlaySound(string stateName)
31          {
32              var selectedSound = soundData.FirstOrDefault(x => x.StateName == stateName);
33
34                if (selectedSound == null || string.IsNullOrEmpty(selectedSound.FModInputSound)) return;
```

```
35
36              if (!stateSoundData.ContainsKey(stateName))
37              {
38                  stateSoundData.Add(stateName,
39                  FMODUnity.RuntimeManager.CreateInstance(selectedSound.FModInputSound));
40              }
41
42              stateSoundData[stateName]
43                  .set3DAttributes(FMODUnity.RuntimeUtils.To3DAttributes(gameObject));
44              stateSoundData[stateName].start();
45          }
46
47          /// <summary>
48          /// Stops the selected Fmod sound.
49          /// </summary>
50          /// <param name="stateName">The name of the state to stop.</param>
51          /// <param name="stopMode">The mode on which it will be stoped.</param>
52          public void StopSound(string stateName, FMOD.Studio.STOP_MODE stopMode)
53          {
54              var selectedSound = soundData.FirstOrDefault(x => x.StateName == stateName);
55
56                if (selectedSound == null || string.IsNullOrEmpty(selectedSound.FModInputSound)) return;
57
58              stateSoundData[stateName].stop(stopMode);
59          }
60
61          /// <summary>
62          /// Sets specific parameter in fmod studio.
63          /// </summary>
64          /// <param name="stateName">Name of the state.</param>
65          /// <param name="paramterName">The parameter name.</param>
66          /// <param name="numericValue">The numeric value (can only be float).</param>
67          public void SetParameter(string stateName, string paramterName, float      numericValue)
68          {
69              var selectedSound = soundData.FirstOrDefault(x => x.StateName == stateName);
70
71                if (selectedSound == null || string.IsNullOrEmpty(selectedSound.FModInputSound)) return;
72
73              if (!stateSoundData.ContainsKey(stateName))
74              {
75                  return;
76              }
77
78                FMODUnity.RuntimeManager.StudioSystem.setParameterByName(paramterName, numericValue);
79          }
80      }
81  }
```

Primjer koda 5. Implementacija sustava za upravljanje zvukom

7. ANIMACIJE

Animacije unutar OR-a su rađene ručno u programu za piksel crtanje pod nazivom Aseprite. Kod projekta OR se vizualno težilo prema retro izgledu računalnih igara kako bi se kod pojedinih igrača stvorio osjećaj nostalgije. Osnovne animacije su se sastojale od 32x32 piksela te su imale od 6-8 slika po animaciji. Određeni objekti poput specijalnih neprijatelja su u pogledu rezolucije imali 128x128 piksela dok su pozadine u prosjeku bile sačinjene 720 x 360 piksela. Na slici 4 su prikazani neki od likova koji su korišteni unutar igre. Problem sa pikselom nastaje kod skaliranja gdje se moraju koristiti veličine sa bazom 2 jer će se u protivnom dogoditi loša reorganizacija piksela što će rezultirati smanjenjem kvalitete animacije.

Slika 4. Likovi unutar igre OR

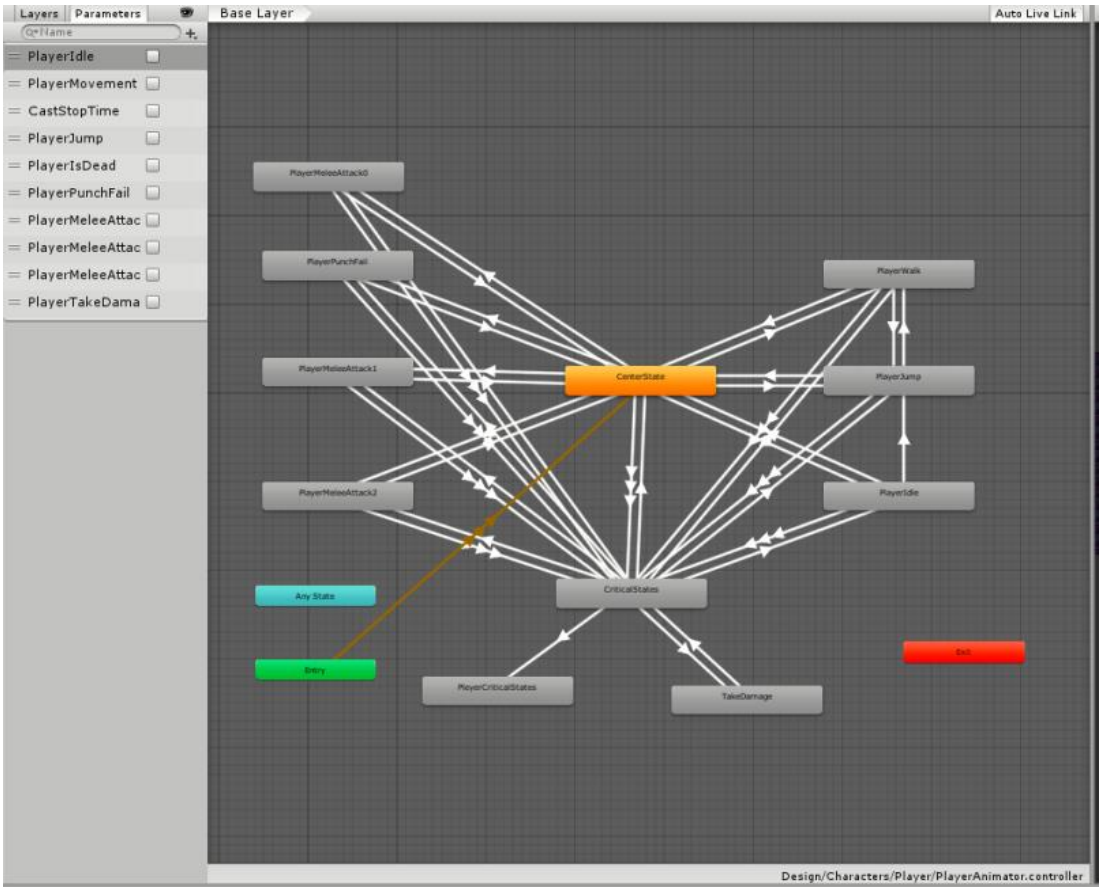
U primjeru koda 6 je prikazana sinkronizacija animatora sa mašinom stanja. Polje „Anima“ (linija 11) dohvaća Unity-jevu animacijsku komponentu unutar djeteta objekta. Funkcijom „StartAnimation“ (linija 18) započinjemo specifičnu animaciju. Parametar kojeg funkcija prima je ime stanja koje postaje aktivno. Funkcija „StopAnimation“ (linija 24) zaustavlja animaciju za stanje koje prestaje biti aktivno. Funkcija „SetStateAnimation“ (linija 34) funkcionira kao prekidač koji pali i gasi animaciju po potrebi. Funkcija „IsAnimationOver“ (linije 44 i 57) provjerava da li je animacija prošla odgovarajući pomak (0 = početak, 1 = kraj animacije) te ukoliko jest vraća istinu, a u protivnom vraća laž.

```
1  using System.Collections;
2  using UnityEngine;
3
4  namespace General.State
5  {
6      public class AnimationController:MonoBehaviour
7      {
8          /// <summary>
9          /// Gets or sets animator;
10         /// </summary>
11         public Animator Anima { get; set; }
12
13         private void Awake()
14         {
15             Anima = GetComponentInChildren<Animator>();
16         }
17
18         public void StartAnimation(string animationName)
19         {
20             if (Anima == null) return;
21             Anima.SetBool(animationName, true);
22         }
23
24         public void StopAnimation(string animationName)
25         {
26             if (Anima == null) return;
27             Anima.SetBool(animationName, false);
28         }
29
30         /// <summary>
31         /// If on turns off if off turns on.
32         /// </summary>
33         /// <param name="stateAnimationName"></param>
34         public void SetStateAnimation(string stateAnimationName)
35         {
36             if (Anima == null) return;
37             if (Anima.GetBool(stateAnimationName)) Anima.SetBool(stateAnimationName, false);
38             else
39             {
40                 Anima.SetBool(stateAnimationName, true);
41             }
42         }
43
44         public bool IsAnimationOver(State state, float offset = 1)
45         {
46             if (Anima == null) return true;
47
48               if               (Anima.GetCurrentAnimatorStateInfo(0).IsName(state.GetType().Name)               &&
49              Anima.GetCurrentAnimatorStateInfo(0).normalizedTime >= offset)
50             {
51                 return true;
52             }
53
54             return false;
55         }
56
57         public bool IsAnimationOver(string animationName, float offset = 1)
58         {
59             if (Anima == null) return true;
60
61               if                   (Anima.GetCurrentAnimatorStateInfo(0).IsName(animationName)                   &&
62              Anima.GetCurrentAnimatorStateInfo(0).normalizedTime >= offset)
63             {
64                 return true;
65             }
```

```
66
67                 return false;
68             }
69         }
70     }
```

Primjer koda 6. Implementacija sustava za upravljanje animacijama

Upravitelj animacijama (prikazan na slici 5) je poprimio oblik zvijezde iz razloga što je rijetko, ali ne i nemoguće, da objekt napravi tranziciju iz jednog specifičnog stanja u drugo. Direktne tranzicije se isključivo događaju između normalnih i kritičnih stanja kada ne želimo da igrač ima kontrolu nad svojim likom ili pri inicijalizaciji određene sekvence događaja.



Slika 5. Izgled animacijskog kontrolera glavnog karaktera

8. INJEKCIJA ZAVISNOSTI

Koristeći Zanject injekciju zavisnosti unutar OR-a stvorio se problem da je kod unutar OR-a postao previše ovisan o biblioteci koja nema sigurnu budućnost održavanja te je nepraktična za korištenje. OR je trebao mogućnost zamjene implementacije pojedinih komponenti ovisno o sceni u kojoj se igrač trenutno nalazi. Zanject također nije podržavao direktno injekciju u objekte koji se stvaraju tokom izvođenja računalne igre već je zahtijevao par međukoraka sa kojim bi se forsirala registracija parametara iz kontejnera injekcije. Prednost injekcije zavisnosti je ta da od scene do scene možemo mijenjati različite načine implementacije sučelja bez potrebe za dodatnim modifikacijama koda. Primjer koda 7 prikazuje primjer injekcije zavisnosti za gravitaciju u dvije različite scene. Jedna scena implementira sučelje „IGravityManager“ te koristi normalnu gravitaciju dok druga scena implementira isto sučelje i koristi obrnutu gravitaciju. „AsSingle()“ funkcija označava da će za svaki zahtjev instance kontejner dati istu instancu (singleton) objekta, dok će funkcija „AsTransient()“ za svaki zahtjev vratiti novu instancu objekta.

```
1  DiContainerInitializor.Container = new System.Collections.Generic.Dictionary<System.Type,ContainerData>();
2  DiContainerInitializor.Container.BindInstance<IGravityManager, GravityNormal>().AsSingle();
3  DiContainerInitializor.Container.BindInstance<IGravityManager,                    GravitzReverse>().AsSingle();
```

Primjer koda 7. Primjer injekcije gravitacije u dvije različite scene

Injekcija se izvodi u funkciji „Initialization_State“, a poziva se na sljedeći način:

```
DiContainerLibrary.DiContainer.DiContainerInitializor.RegisterObject(this);
```

Funkcija „RegisterObject(this)“ za parametar prima objekt iz kojeg pomoću refleksije dohvaća sva polja koja sadrže atribut „InjectDiContainer“. Zatim se kroz kontejner injekcije registriraju sa odgovarajućom instancom objekta kojeg to polje definira. Injekcija podržava sve tipove pristupa polju. Polje koje će biti injektirano mora posjedovati atribut „InjectDiContainer“. Kod u nastavku prikazuje izgled polja sa „InjectDiContainer“ atributom:

```
[InjectDiContainter]
protected IMainManager mainManager { get; set; }
```

9. PIKSEL SJENILA

Zadaća piksel sjenila (eng. shader) je da sudjelovanjem u procesu grafičke obrade slike (eng. graphic pipeline) direktno utječe na konačnu boju piksela koji se prikazuju na ekranu. Sva sjenila se isključivo izvršavaju direktno na grafičkoj kartici što nam pomaže pri optimizaciji i povećanju performansi igre. Prema Franciscu Tufru (2018), programski jezik za pisanje sjenila baziran je na programskom jeziku C, a postoji više vrsta kompajlera i API-ja s kojima se mogu pisati sjenila od kojih su najpoznatiji: DirectX, OpenGL i NVIDIA CG. Za potrebe OR-a korišten je CG kompajler radi svoje jednostavnosti i kompatibilnosti sa DirectX-om i OpenGL-om. CG će prepoznati koji je sustav instaliran na hardveru kojeg koristimo te će prilagoditi sjenilo tim specifikacijama. OR posjeduje par jednostavnih vrsta sjenila kojima je glavna uloga poboljšati efekt svjetla u scenama. Slika 6 prikazuje izgled sjenila koji ima za cilj učiniti površine u OR-u prilagodljivima na izvor svijetlosti.

Slika 6. Shader za ambijentalnu svijetlost unutar OR-a

10. MARKETING I IZDAVAČ

Kako bi se privukao interes javnosti, OR je od samog početka imao plan kampanje i promoviranja. Prisutnost na društvenim mrežama (slika 7) je bilo potrebno uspostaviti u što ranijoj fazi produkcije. Koristeći se društvenim mrežama, ciljalo se prema predstavljanju projekta OR-a široj javnosti kako bi se promovirali materijali i mehanike koje su se koristile prilikom izrade same računalne igre. OR je bio predstavljen na svim velikim konferencijama u Hrvatskoj kako bi se pokušavalo doći do povratnih informacija o dobrim i lošim stvarima unutar OR-a. Cijela marketinška kampanja je postigla vrhunac na Reboot Develop Blue 2019 konferenciji kada je prodajni menadžer japanskog izdavača Nippon Ichia Software, Shunsuke Nishida, prepoznao potencijal u igri te ostavio svoj kontakt u nadi će OR-a postati ekskluzivni naslov za Nintendo Switch.

Slika 7 Promoviranje OR-a na Twitteru (Ominous Relict Twitter)

11. ZAKLJUČAK

Proizvodni ciklus igre je jako težak i pun nepredvidljivosti. Za kvalitetan i uspješan proizvod uvijek je potrebno planirati i unaprijed predviđati probleme. S obzirom da na OR-u nije sudjelovalo dovoljno ljudi, bilo je potrebno osmisliti bazu koja će biti lako primjenjiva u svim područjima projekta. Kreiranjem mašine stanja nastojalo se automatizirati i rasteretiti dizajnere slike i zvuka te kreirati generalnu strukturu koda unutar OR-a. Kod spremanja podataka, cilj je stvoriti sustav koji ima jednostavnu implementaciju i praktičnu uporabu. Uvođenjem injekcije zavisnosti omogućena je promjena konfiguracije zavisno o potrebama dizajna za specifične scene. FMod je inovativna tehnologija na tržištu koja je podigla stupanj kvalitete zvuka unutar OR-a te omogućila distinkciju od konkurentnih naslova.

Literatura:

1 Tufro, F. (2018), 2D Shader Development: Foundations: (Make your game unique in a world full of lookalikes), Hidden People Club.

2 Ominous Relict (@OminousRelict) Status ID: “1077282942047698944”. 24.12. 2019, 20:20. Tweet.

Podaci o autorima:

Mirko Srsen, bacc. ing. techn. inf.

Email: mirko@rawfury.com

Mirko Sršen je 2017. godine stekao akademski naziv prvostupnika informatike na Tehničkom veleučilištu u Zagrebu. Na Fakultetu informatike Sveučilišta Jurja Dobrile u Puli trenutno pohađa 2. godinu diplomskog studija informatike. Sa 20 godina postao je certificirani C# developer. Glavna područja interesa su mu razvoj aplikacija za automatizaciju proizvodnje i razvoj računalnih igara u programskim jezicima C#, TypeScript, C++ i C.

Izv. prof. dr. sc. Tihomir Orehovački

Email: tihomir.orehovacki@unipu.hr

Tihomir Orehovački diplomirao je i doktorirao 2005. i 2013. godine, respektivno, na Fakultetu organizacije i informatike Sveučilišta u Zagrebu. Pedagoško-psihološko-didaktičko-metodičku naobrazbu stekao je tijekom akademske godine 2005./2006. na Visokoj učiteljskoj školi u Čakovcu. Zaposlen je na radnom mjestu izvanrednog profesora na Fakultetu informatike Sveučilišta Jurja Dobrile u Puli gdje je nositelj kolegija vezanih uz programiranje. Član je Povjerenstva za akademsko priznavanje inozemnih visokoškolskih kvalifikacija i razdoblja studija te Odbora za znanstveni i umjetnički rad Sveučilišta Jurja Dobile u Puli. Autor je 94 znanstvena rada objavljena u zbornicima međunarodnih konferencija, časopisima i knjigama te 16 stručnih radova objavljenih u zbornicima domaćih skupova. Znanstveno se i stručno usavršavao na brojnim radionicama i institucijama u zemlji i inozemstvu. Bio je voditelj projekta financiranog sredstvima HRZZ. Osim toga, sudjelovao je u istraživačkim aktivnostima projekata financiranih sredstvima Europske Unije, HRZZ, Zaklade Adris,

Ministarstva znanosti i obrazovanja te Sveučilišta u Zagrebu. Bio je suorganizator i predavač na 2 radionice vezane uz upravljanje međunarodnim projektima i 7 radionica vezanih uz primjenu društvenih Web aplikacija u obrazovanju. Aktivni je recenzent za 18 međunarodnih znanstvenih časopisa, 2 znanstvene knjige i 13 međunarodnih znanstvenih konferencija. Vršio je dužnost člana Programskog odbora, voditelja tematskih cjelina i sekcija sljedećih međunarodnih znanstvenih konferencija: International Conference on Information Systems Development (ISD), ACM International Conference on Intelligent User Interfaces (IUI), International Conference on the Quality of Information and Communications Technology (QUATIC) - Track on Quality in Web Engineering, International Conference on Information Technology (ICIT), International Conference on Software and Information Engineering (ICSIE), International Conference on Network Technology (ICNT) te International Conference on Human and Social Analytics (HUSO). Bio je član Organizacijskog odbora, urednik knjige sažetaka i tehnički urednik zbornika radova međunarodne znanstvene konferencije ISD 2014. Član je sljedećih međunarodnih strukovnih udruženja: Association for Computing Machinery, European University Information Systems E-Learning Task Force, Institute of Electrical and Electronics Engineers i International Society for Web Engineering. Za svoj znanstveni, nastavni i stručni rad primio je nekoliko međunarodnih i domaćih nagrada i priznanja. Dosad je pod njegovim mentorstvom 105 studenata obranilo završne i diplomske radove.

Fakultet informatike
Sveučilište Jurja Dobrile u Puli
Rovinjska 14, 52100 Pula
fipu.unipu.hr

USPOREDBA CROSS-PLATFORM FRAMEWORKA ZA IZGRADNJU MOBILNIH APLIKACIJA

Ivan Miljančić, Marin Kaluža, Veleučilište u Rijeci

SAŽETAK

Usporedba cross-platform framework-a za izgradnju mobilnih aplikacija je rad u kojem se vrši usporedba sljedećih framework-a (FW): React Native, Flutter te Ionic Framework. Usporedba se temelji na mjerenju performansi izvođenja mobilne aplikacije izgrađenoj u sva tri FW-a te same produktivnosti programera tijekom njihove izgradnje. Cilj ovoga rada je razmotriti razliku između performansa koje postiže svaka pojedina aplikacija te uočiti koji od FW-a ima najbolje performanse te omogućuje najbolju produktivnost programera. Za mjerenje performansi aplikacije koristi se Android Profiler koji dolazi uz Android Studio IDE te mjeri resurse koje aplikacije troši kao što su: korištenje procesora, korištenje memorije, mrežnih resursa te količina potrošnje energije. Primijenjena metodologija u ovome radu je metoda analize i sinteze gdje su se prvo izgradile mobilne aplikacije nad kojima se kasnije vršila analiza performansi te u konačnici sinteza konačnih rezultata.

ABSTRACT

Comparison of cross-platform frameworks for building mobile applications is a paper that covers a comparison of the following frameworks (FWs): React Native, Flutter and Ionic Framework. The comparison is based on measuring the performance of applications build in all three FWs and the productivity of the developer while building them. The goal of this paper is to find the difference in performance between each individual application and to find out which of the FWs has the best performance and enables the developer to work in the most productive manner. The tool used for measuring the performance of each application is Android Profiler that comes with the Android Studio IDE and it measures the following data: usage of processor, usage of memory, usage of network resources and the amount of energy consumption. The used methodology in this paper is the method of analysis and synthesis, where the first step was developing mobile applications over which the performance analysis could have been performed later and finally the synthesis of the final measured results.

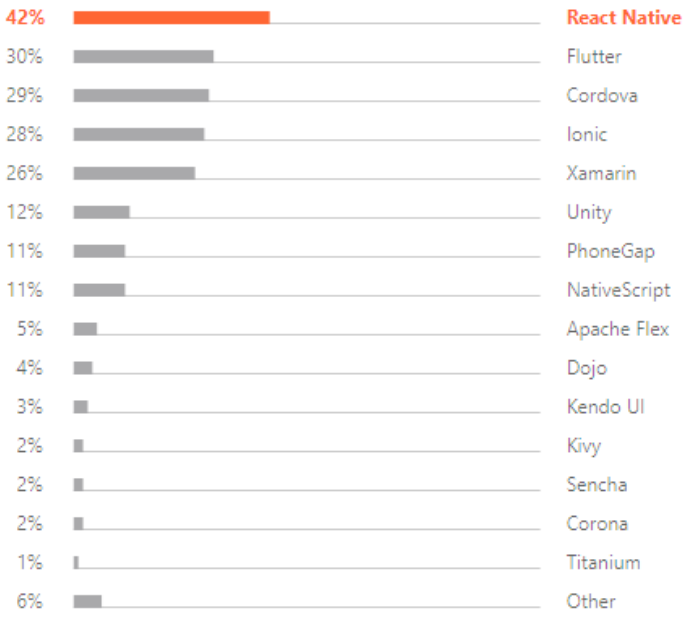
1. UVOD

U svijetu programskog inženjerstva svakog dana sve je više raznih Framework-a (FW) koji omogućuje brz i jednostavan način razvoja mobilnih aplikacija za razne operacijske sustave (Android, iOS) s jednakom kodnom bazom. Takvi FW-ci se nazivaju cross-platform FW-ci, te će se u ovome radu obratiti pažnja na sljedeće: React Native, Flutter, Ionic. Cilj cross-platform FW-a je omogućiti programeru da korištenjem jednog programskog jezika izgradi aplikaciju za razne operacijske sustave. Predmet istraživanja u ovome radu je testiranje navedenih FW-a odnosno testiranje njihove brzine izvođenja programskog koda i produktivnosti koju programer ostvaruje njihovim korištenjem. Problem koji je uočen i kojeg je potrebno riješiti je prevelik izbor FW-a na tržištu, te time je svrha i sam cilj ovoga istraživanja je čitatelju predočiti prednosti i nedostatke pojedinog FW-a, te mu olakšati odabir tehnologije koju će koristiti u sljedećem projektu. Nad navedenim FW-cima izvršavat će se usporedba koja će se bazirati na dva primarna mjerenja: brzina izvođenja aplikacije i produktivnost programera izgradnje aplikacije. Za mjerenje produktivnosti programera uspoređivat će se dokumentacija, količina unaprijed gotovih modula, gotova rješenja za izgradnju korisničkih sučelja, te potrebno vrijeme za postavljanje FW-a i izrade novoga projekta. Brzina izvođenja aplikacije mjerit će se pomoću Android Profiler aplikacije razvijene od kompanije Google. Parametri koji ću se mjeriti bit ću korištenje procesorske snage, količina korištenja memorije uređaja, korištenje mrežnih resursa te konačno korištenje energije baterije. Nakon izvršavanja mjerenje i analize produktivnosti i brzine izvođenja aplikacije usporedit ću se rezultati kako bi se dobila jasnija slika koje su prednosti i nedostatci pojedinog FW-a.

2. KRITERIJI I METODE ZA ANALIZU

Kao što je navedeno u uvodnom poglavlju, cilj ovoga rada je usporedba FW-a za izgradnju hibridnih aplikacija, te je prethodno tome potrebno postaviti kriterije po kojima će se FW-ci uspoređivati.

FW-ci koji ću biti analizirani su: ReactNative, Flutter, Ionic. Sami FW-ci su izabrani po top 3 FW-a koji se koriste prema zadnjoj anketi developera kompanije JetBrains koja razvija napredne IDE alate. Ionic FW-a je odabran iznad Cordova-e zato što je Cordova FW-a koji omogućuje povezivanje JavaScript programskog jezika s native-nim metodama operacijskog sustava, te se koristi u Ionic FW-u kako bi se izgradila mobilna aplikacija.



Slika 9 - Anketa FW-a za razvoj mobitlnih aplikacija (Izvor: https://www.jetbrains.com/lp/devecosystem-2019/, 24.12.2019)

2.1. BRZINA IZVOĐENJA MOBILNE APLIKACIJE

Mjerenje brzine izvođenja mobilne aplikacije dijeli se na sljedeće parametre: korištenje procesorske snage, memorije, mreže te količina baterije koju aplikacija troši. Za testiranja biti korišten Android mobilni uređaj, te ću sve aplikacije biti testiranja na njemu kako bi se bile jednake hardverske specifikacije za svako pojedino testiranje, a za mjerenje će se koristiti Google-ov alat Android Profiler. Android Profiler omogućuje nadgledanje aplikacija na Android mobilnim uređajima u realnom vremenu, te prikazuje korištene resurse u obliku grafa.

U svakom FW-u bit će izgrađena Todo aplikacija koja sa stranice „JSONPlaceholder“ prikuplja lažne podatke u JSON obliku te ih prikazuje na zaslon. JSONPlaceholder je besplatan online REST API koji se može koristiti kada su potrebni lažni podatci kao primjerice za izgradnju uputstva, testiranje novih knjižnica i slično, te ćemo u ovom radu koristit ćemo lažne *Todo* zadatke (https://jsonplaceholder.typicode.com/, 24.12.2019).

2.2. PRODUKTIVNOST PROGRAMERA

Produktivnost programera koristeći FW može se mjeriti tako da se promatraju mogućnosti koje sam FW nudi kod inicijalizacija projekta. Takve mogućnosti mogu biti od Hot-Reload opcije koja automatski prilikom promjene izvornog koda ažurira mobilnu aplikaciju, mogućnosti dohvata resursa na mreži kao što je REST API end-point bez dodane instalacije paketa. Također važan aspekt je količina i kvaliteta gotovo generiranog programskog koda prilikom inicijalizacije projekta.

2.3. CROSS-PLATFORM FW-OVI

2.3.1 React Native

React Native je FW koji pomaže pri izgradnji mobilnih aplikacija uz pomoć JavaScript programskog jezika, koji je podržan od Android i iOS platforma za mobitele što uštređuje vremena u razvoju samih aplikacija. Zbog ogromne popularnosti i podrške Facebook-a, React Native ima danas ogromnu podršku zajednice. React Native je izgrađen je povrh ReactJS koji je pružao veliku konkurenciju dugogodišnjem favoritu AngularJS. React Native je FW izgrađen na hijerarhiji UI komponenata do JavaScript programskog koda. Ima set gotovih komponenata za iOS i Android platforme kako bi se izgradile aplikacije s native-nim izgledom i „osjećajem“. (Thinkwik, 2018.)

2.3.2 Flutter

Flutter je FW otvorenog programskog koda kojim se izgrađuju mobilnih aplikacija za Android i iOS s native-nim izgledom i „osjećajem“. Flutter kao FW postoji već od 2015. godine kada ga je Google predstavio i ostao je u beta fazi sve do prosinaca 2018 godine, te je od počeo rasti više i više u popularnosti. Flutter je trenutno u top 11 repozitorija po količini GitHub zvijezda, te je nekoliko Flutter aplikacije izdanih na oba dvije platforme od kojih je najbolji primjer Xianyu aplikacija od Alibaba tima koja ima 50 milijuna preuzimanja. (Concise Software, 2019.)

2.3.3 Ionic

Ionic je FW koji služi za izgradnju aplikacija visokih performansi i visoke kvalitete za mobitele i stolna računala korištenjem *WEB* tehnologija (HTML, CSS, JavaScript). Originalna verzija ovoga FW izašla je 2013. godine te je bila izgrađena povrh AngularJS i Apache Cordova FW-a. Današnja verzija Ionic 3 ili jednostavno „Ionic“ je izgrađen povrh Angular FW-a. Ionic je bio jedan od prvih hibrid FW-a za izgradnju mobilnih aplikacija. (Javier Ramos, 2019).

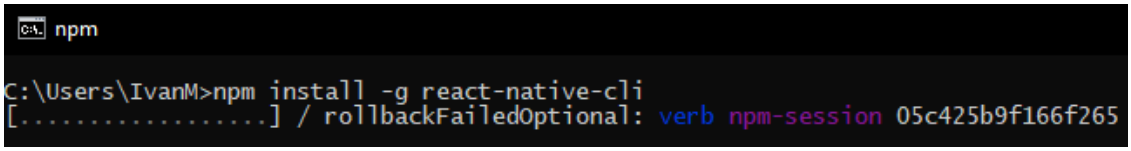
3. ANALIZA FRAMEWORK-A

3.1. IZGRADNJA TODO APLIKACIJA

Za stvaranje Todo aplikacije potrebno je u svakom FW-u inicijalizirati projekt korištenjem radnog dijagrama. Radni dijagram se izgrađuje korištenjem komandnih linija unutar konzole, te svaki od navedenih FW kreira svoju strukturu foldera i datoteka. Kako bi se kreirali projekti potrebno je instalirati Node.js (dostupan na stranici: https://nodejs.org/en/), koji je potreban za sva tri FW-a i čija instalacija se izvodi vrlo jednostavno kroz nekoliko klikova, te iz tih razloga neće biti prikazano u ovome radu. Također kako se sve aplikacije izvode na Android mobilnom uređaju potrebno je instalirati Android Studio (dostupan na stranici: https://developer.android.com/studio/0) kako bi se sve potrebne datoteke kao što je primjerice Android SDK (engl. Sotfware development kit) instalirale. (Android Studio, 2019.)

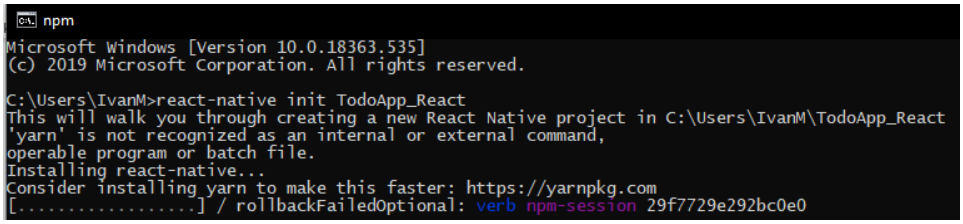
3.1.1 Todo - React Native

Prvi korak pri izgradnji mobilne aplikacije korištenjem React Native FW-a je instalacije React Native CLI (engl. Command Line Interface) koji nam daje komande za izgradnju, kompiliranje i izgradnju aplikacije za ciljani uređaj. Za instalaciju potrebno je unutar komande naprednog retka (engl. Command prompt) unijeti komandu „npm install -g react-native-cli“ kao što je prikazano u slici 2. (React Native – Getting started, 2019.)

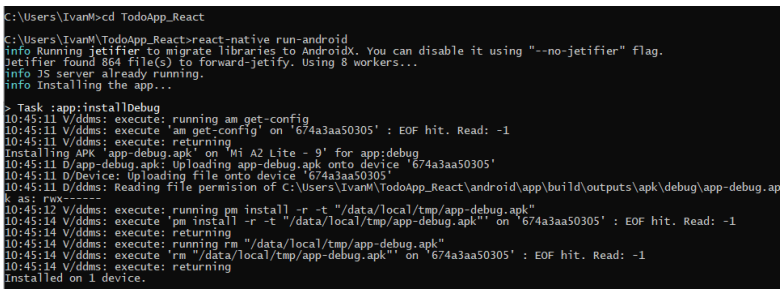


Slika 10 - React Native CLI instalacija (Izvor: obrada autora)

Nakon instalacije, naredbom „react-native init Ime_Projekta“ kreirati će se novi projekt kao što je prikazano slikom 3. Nakon što se izgradi aplikacija potrebno ju je pokrenuti na fizičkom uređaju ili na mobilnom imitatoru. Za pokretanje na fizičkom uređaju pokreće se korištenjem naredbe „react-native run-android“ za Android mobilne uređaje (prikazano slikom 4) ili „react-native run-ios“ za iOS mobilne uređaje. (React Native – Getting started, 2019.)



Slika 11 - Stvaranje novoga React Native projekta (Izvor: obrada autora)



Slika 12 - Pokretanje aplikacija na Android mobilnom uređaju (Izvor: obrada autora)

Nakon kreiranja početnog projekta izgrađena je mobilna aplikacija koja je prikazana slikom 5. Crvenom bojom su prikazani dovršeni zadatci, dok crvenom bojom ne dovršeni. U samoj aplikaciji je prikazano 200 Todo zadataka, što je sasvim dovoljno za izmjeriti performanse i produktivnost programera u sljedećim poglavljima. Za pokretanje projekta na Android mobilnom uređaju potrebno je unijeti komandu „react-native run-android“. (React Native, Running On Device, 2019.)

Slika 13 - React Native Aplikacija (Izvor: obrada autora)

3.1.2 Todo - Flutter

Za izgradnju Flutter mobilne aplikacija potrebno je preuzeti njihov SDK (dostupno na: https://flutter.dev/docs/get-started/install/windows) nakon preuzimanja potrebno ga je otpakirati korištenjem alata kao što su: WinRar ili 7zip u neki folder na računalu na kojem će se izvršavati razvoj. Kod korištenja Windows operacijskog sustava potrebno je postaviti putanju do binarnih datoteka Flutter-a unutar postavka za varijabilne okoline (engl. environment variables).

Kako bi se provjerilo da li je sve postavljeno kako treba može se pokrenuti komanda Flutter doctor unutar komandne linije. Ovaj korak je prikazan slikom 6. (Flutter – Get Started, 2019.)

```
C:\Users\IvanM>flutter doctor
Doctor summary (to see all details, run flutter doctor -v):
[√] Flutter (Channel stable, v1.12.13+hotfix.5, on Microsoft Windows [Version 10.0.18363.535], locale en-GB)

[√] Android toolchain - develop for Android devices (Android SDK version 29.0.2)
[!] Android Studio (version 3.5)
    X Flutter plugin not installed; this adds Flutter specific functionality.
    X Dart plugin not installed; this adds Dart specific functionality.
[√] VS Code (version 1.41.1)
[√] Connected device (1 available)

! Doctor found issues in 1 category.
```

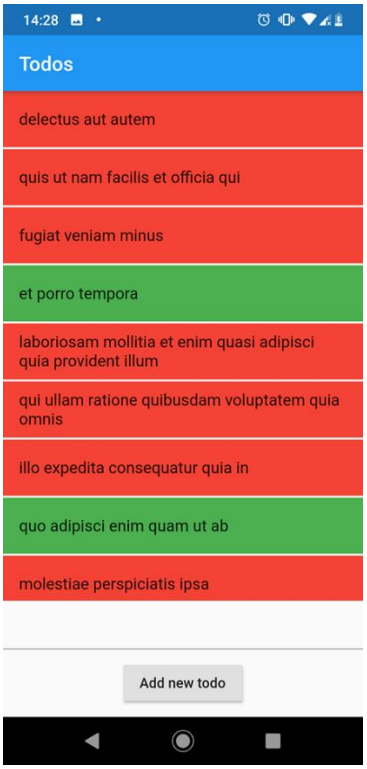
Slika 14 - Provjera Flutter FW-a (Izvor: obrada autora)

Stvaranje projekta pokreće se komandom „flutter create Ime_projekta“ te je ovaj korak prikazan slikom 7.

```
D:\>flutter create todo_app
Creating project D:\todo_app... androidx: true
  D:\todo_app\.gitignore (created)
  D:\todo_app\.idea\libraries\Dart_SDK.xml (created)
  D:\todo_app\.idea\libraries\Flutter_for_Android.xml (created)
  D:\todo_app\.idea\libraries\KotlinJavaRuntime.xml (created)
  D:\todo_app\.idea\modules.xml (created)
  D:\todo_app\.idea\runConfigurations\main_dart.xml (created)
  D:\todo_app\.idea\workspace.xml (created)
  D:\todo_app\.metadata (created)
  D:\todo_app\android\app\build.gradle (created)
  D:\todo_app\android\app\src\main\kotlin\com\example\todo_app\MainActivity.kt (created)
  D:\todo_app\android\build.gradle (created)
  D:\todo_app\android\todo_app_android.iml (created)
  D:\todo_app\android\.gitignore (created)
  D:\todo_app\android\app\src\debug\AndroidManifest.xml (created)
  D:\todo_app\android\app\src\main\AndroidManifest.xml (created)
  D:\todo_app\android\app\src\main\res\drawable\launch_background.xml (created)
  D:\todo_app\android\app\src\main\res\mipmap-hdpi\ic_launcher.png (created)
  D:\todo_app\android\app\src\main\res\mipmap-mdpi\ic_launcher.png (created)
  D:\todo_app\android\app\src\main\res\mipmap-xhdpi\ic_launcher.png (created)
  D:\todo_app\android\app\src\main\res\mipmap-xxhdpi\ic_launcher.png (created)
  D:\todo_app\android\app\src\main\res\mipmap-xxxhdpi\ic_launcher.png (created)
  D:\todo_app\android\app\src\main\res\values\styles.xml (created)
```

Slika 15- Stvaranje novoga Flutter projekta (Izvor: obrada autora)

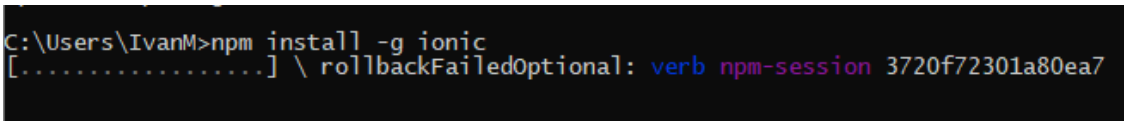
Nakon stvaranja početnog projekta izgrađena je mobilna aplikacija prikazana slikom 8. Kao i u React Native projektu crvenom bojom su prikazani ne dovršeni zadatci, dok zelenom dovršeni. Projekt za Android mobilni uređaj pokreće s naredbom „flutter run“.



Slika 16 - Flutter aplikacija (Izvor: obrada autora)

3.1.3Todo- Ionic

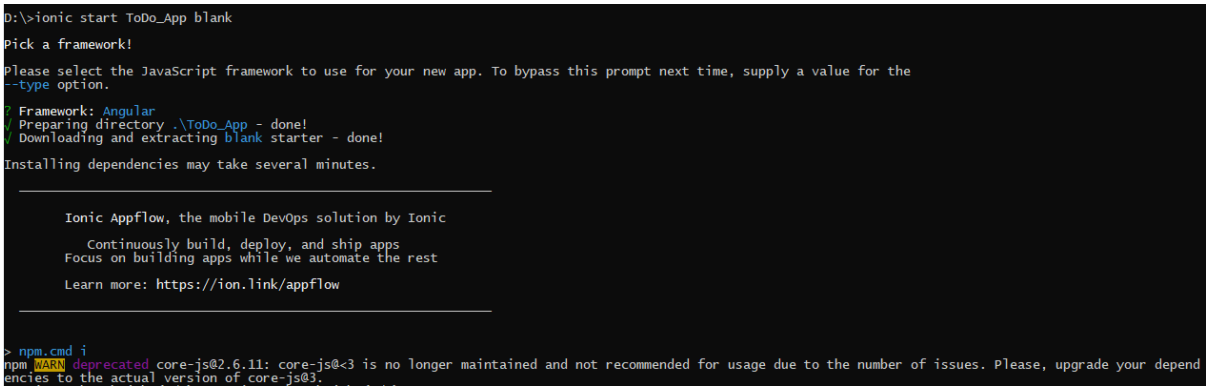
Kod izgradnje Ionic mobilne aplikacije potrebno je instalirati Ionic CLI naredbom „npm install -g ionic“, te je ovaj korak prikazan slikom 9. (Ionic – Installation, 2019)



Slika 17 - Instalacija Ionic CLI-a (Izvor: obrada autora)

Nakon uspješne instalacije potrebno je kreirati novi projekt korištenjem instaliranog CLI-a. Kako bi se kreirao novi projekt potrebno je koristiti naredbu „ionic start Ime_aplikacija ime_predloška“, Ionic nudi tri početna predloška za aplikaciju „blank“, „tabs“, „sidemenu“ (za izgradnju Todo aplikcije koristiti će se „blank“ predložak).

Kod stvaranja projekta ponude se opcije za odabirom JavaScript FW-a koji će se koristiti, ponuđene mogućnosti su Angular i React, te u ovom slučaju odabran je Angular FW ovaj korak prikazan je slikom 10. (Ionic – Installation, 2019)



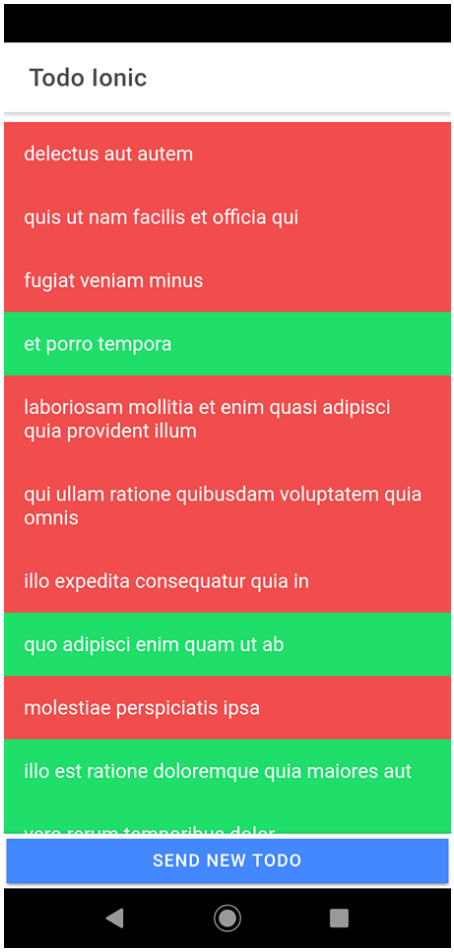
Slika 18 - Stvaranje novog Ionic projekta (Izvor: obrada autora)

Nakon uspješno stvorenog projekta potrebno je postaviti radno okruženje za izvršavanje deploy-a na Android mobilni uređaj. Kako bi se izvršila ova radnje potrebno je unutar stvorenog Ionic projekta pokrenuti naredbu „ionic cordova prepare android“ (prethodno tome potrebno je obraditi pažnju da li je Cordova instalirana na računalu). Navedeni korak prikazan je slikom 11. (Ionic – Installation (Android), 2019)

```
D:\ToDo_App>ionic cordova prepare android
> ionic integrations enable cordova
[INFO] Downloading integration cordova
[INFO] Copying integrations files to project
CREATE resources
CREATE resources\splash.png
CREATE resources\ios
CREATE resources\ios\splash
CREATE resources\ios\splash\Default~iphone.png
CREATE resources\ios\splash\Default@2x~universal~anyany.png
CREATE resources\ios\splash\Default@2x~iphone.png
CREATE resources\ios\splash\Default-Portrait~ipad.png
CREATE resources\ios\splash\Default-Portrait@~ipadpro.png
CREATE resources\ios\splash\Default-Portrait@2x~ipad.png
CREATE resources\ios\splash\Default-Landscape~ipad.png
CREATE resources\ios\splash\Default-Landscape@~ipadpro.png
CREATE resources\ios\splash\Default-Landscape@2x~ipad.png
CREATE resources\ios\splash\Default-Landscape-736h.png
CREATE resources\ios\splash\Default-Landscape-2436h.png
CREATE resources\ios\splash\Default-736h.png
CREATE resources\ios\splash\Default-667h.png
CREATE resources\ios\splash\Default-568h@2x~iphone.png
CREATE resources\ios\splash\Default-2436h.png
CREATE resources\ios\icon
CREATE resources\ios\icon\icon@2x.png
CREATE resources\ios\icon\icon.png
CREATE resources\ios\icon\icon-small@3x.png
CREATE resources\ios\icon\icon-small@2x.png
```

Slika 19 - Postavljanje radnog okruženja za Android (Izvor: obrada autora)

Nakon konačne instalacije Cordova-e unutar Ionic projekta možemo pokrenuti aplikaciju korištenjem naredbe „ionic cordova run android -l“. Izgrađena Todo aplikacija prikazana je slikom 12. (Ionic – Installation (Android), 2019)



Slika 20 - Ionic aplikacija (Izvor: obrada autora)

3.2. ANALIZA PRODUKTIVNOST PROGRAMERA

Kod izgradnje pojedinih aplikacija na navedenim FW-cima primijetila se razlika u količini komanda i vremena kod postavljanja radnog okruženja što uključuje mobilni uređaj i sam projekt.

3.2.1 Dokumentacija

React Native FW koristi JSX (JavaScript XML) sintaksu. Prilikom prvog susreta s ovakvom sintaksom potrebno je nešto vremena kako bi se razumjela srž. Za dodatnu pomoć programeru nudi se React Native dokumentacija (dostupna na: https://facebook.github.io/react-native/docs/getting-started). Dokumentacija je detaljna i obuhvaća sve dijelove potrebne za izgradnju mobilne aplikacije kao što su primjerice: mreže (engl. Networking), stilovi (engl. Style) i slično. Za svaki pojedini dio FW-a nudi se i primjer programskog koda kojeg je moguće pokrenuti u pregledniku te modificirati kako bi se vidjele i testirale promjene, te su moguće postaviti na različite mobilne uređaje odnosno Android i iOS kako bi se moglo provjeriti da nema razlike između sučelja na oba uređaja čime se uklanjanju potrebe za testiranje na pravim uređajima. Konačno se za svaku pojedinu platformu nudi se zasebna dokumentacija kako koristiti native funkcionalnosti uređaja. (React – Getting started, 2020.)

Flutter je noviji FW koji dolazi s novim programskim jezik Dart od kompanije Google. Kako se radi o novome programskom jeziku za kojeg bi se moglo reći da je sintaksu baziranu na JavaScript-u. Krivulja učenja za ovaj programski jezik je „glatka“, te nije potrebno uložiti puno vremena kako bi se naučio. Flutter je popraćen svojom vlastitom dokumentacijom (dostupno na: https://flutter.dev/docs) koja obuhvaća sve dijelove razvoja mobilne aplikacije što obuhvaća izgradnju korisničkih sučelja, dodavanja novih paketa i dodataka, optimizaciju i performanse aplikacije i slično, te gotove primjere programskog koda za izgradnju pojedinih komponenti. (Flutter – Docs, 2020.)

Ionic bogatu dokumentaciju kako omogućuje rad s dva FW-a: Angular i React, te u budućnosti će ima podršku za VueJs (dostupno na: https://ionicframework.com/docs/intro). Tijekom izrade Todo aplikacija dokumentacija je služila kao odličan izvor za informacije kako se koriste komponente za izgradnju grafičkih sučelja, ali nedostatak je bio što nema detaljne dokumentacije za korištenje mrežnih resursa kao što je korištenje Http zahtijeva na vanjske REST API resurse te je bilo potrebno pretraživati druge resurse (*StackOverflow*) kako bi se povezala aplikacija s mrežnim resursom. (Ionic – Docs, 2020.)

3.2.2 Korištenje modula i komponenata

Kako bi se omogućio brži razvoj aplikacija FW-ci često nude mogućnost korištenja gotovih modula koji izvršavaju neku radnju, primjerice komunikacija za vanjskim REST API resursima. Dodatne module moguće je instalirati korištenjem naredba „npm install ime_modula“ kod korištenja Ionic i React Native FW-a, dok za instalaciju u Flutter-u potrebno je dodati potreban modul u „pubspec.yaml“ datoteku pod „dependencies“ nakon čega će Android Studio IDE ili VS Code primijetiti promjene te instalirati novi modul. Moduli se u svakom FW-u moraju uvesti korištenjem odgovarajuće sintakse u komponente i dijelove gdje ću se koristiti.

U ovome radu korišten je modul za vršenje Http GET i POST poziva prema resursu „https://jsonplaceholder.typicode.com/todos“. Vršenje zahtjeva prema navedenom resursu najjednostavnije je bilo unutar React Native FW-a gdje se može koristiti gotova JavaScript metoda *„fetch“* koja može izvršavati sve potrebne Http metode. Ovime se olakšava razvoj programerima koji imaju znanja iz korištenja JavaScript-a u izgradnji *WEB* aplikacija u FW-cima kao što su: React, Angular, VueJs. Kod korištenja druga dva FW-a bilo je potrebno instalirati vanjske module prije nego se mogao napraviti zahtjev prema prethodno navedenom resursu gdje je za Ionic bilo potrebno instalirati „angular/http“ modul dok za Flutter „http“ modul. Zbog potrebe za instalacijom novih modula se usporio razvoj aplikacija na Ionic i Flutter FW-u.

Osim modula za komunikaciju s vanjskim resursima FW-ci nude i gotove komponente za izgradnju korisničkih sučelja od kojih Flutter ima najnaprednije mogućnosti za izgradnju brzih i korisniku lijepih sučelja bez potrebne za dodatnim CSS-om kao u ostalim FW-cima. Flutter omogućuje izgradnju grafičkom sučelja korištenjem gotovih Material komponenta koji se povezuju pomoću parent child veza. (Flutter – Widgets, 2020). Ionic također nudi mogućnosti za izgradnju korisničkih sučelja korištenjem Ionic komponenata koje su prepoznatljive po predznaku „ion“ (npr. ion-footer) (Ionic – Components, 2020). React Native također ima gotove komponente za izgradnju sučelja, ali jedna stvar koja nedostaje je gotov dizajn komponenti, te kako bi se komponenta dao neki stil potrebno je znanje CSS-a što u konačnici usporava brzi razvoj mobilne aplikacije. U konačnici izgradnja grafičkog sučelja najbrže će se postići korištenjem Flutter FW-a.

Mogućnosti koju svaki FW nudi je automatska obnova aplikacije na mobilnom uređaju prilikom promjene programskog koda. Time se omogućuje puno brže testiranje promjena koje se rade u razvoju aplikacije. Važno je za naglasiti da Flutter ima najnapredniju mogućnost zvanu „hot-reload“ koja umjesto da obnovi cijelu aplikaciju obnovi samo komponentu u kojoj je promjena napravljena što znatno bolje što se tiče samih performansi.

3.2.3 Postavljanje FW-a i izrada novoga projekta

Tijekom poglavlja gdje se izgrađivala Todo aplikacija prošlo se kroz nekolicinu naredbi koje su potrebne kako bi se postavio početni projekt u pojedinom FW-u. Postavljanje FW-a nije zahtjevan posao ako osoba ima nekakvo predznanje u korištenju sličnog, ali ako se osoba prvi put susreće s ovakvim načinom instalacije primijeti se razlika u vremenu postavljanja. Iako je svi FW popraćen s kvalitetnom dokumentacijom potrebno je nešto vremena da se razumije što je zapravo potrebno napraviti.

Od korištenih FW-a prilikom njegova postavljanja i izrade projekta najjednostavniji je bio React Native. Kako je za njega preduvjet imati instaliran NodeJS i NPM njega se u jednostavnim koracima instalirati korištenjem NPM naredbe, te je kreiranje je za kreiranje projekta dovoljno koristiti njegovu naredbu „react-native init“, te dodatne postavke za izradu aplikacije na

Android uređaju nije potrebno, dok je za izradu iOS aplikacije potrebno koristiti računalo s MacOS-om. (React Native – Getting Started, 2019.)

Flutter je sljedeći bio po jednostavnosti korištenja kako je za njega potrebno preuzeti arhiviranu datoteku koju je negdje potrebno otpakirati kako bi se postavila unutar varijabli sustava Windows. Nakon što se postavi korištenjem naredbe „flutter create“ kreiraju se projekti bez potrebe za dodatnim postavljanjem za oba operacijska sustava. (Flutter- Get Started, 2019.)

Ionic FW je bio najzahtjevniji i uzeo je najviše vremena za postavljanje iz same činjenice da ne dolazi unaprijed postavljen za pokretanje aplikacija na Android i iOS operacijskom sustavu. Osim njegove instalacije korištenjem NPM-a potrebno je instalirati i Cordova-u koje pretvara izvorni kod Ionic-a u Android ili iOS prihvatljiv oblik. (Ionic – Building, 2019.)

3.3. ANALIZA BRZINE IZVOĐENJA MOBILNE APLIKACIJE

Aplikacija se smatra sporom ako su odgovori na određene akcije unutar nje spore (npr. promjena prozora), animacije su ispucane, aplikacije se smrzne ili sruši te koristi veliku količinu energije. Kako bi se izbjegli takvi problemi potrebno je koristiti alate za profiliranje i vrednovanje kojima se vrši provjera ako je naša aplikacija ne efikasna u korištenju resursa kao što su: procesor, memorija, grafički resursi, mreža te korištenje baterije mobitela. Sve kreirane mobilne aplikacije bit će testirane koristeći već naveden Android Profiler alat od kompanije Google. Kod očitavanja rezultata testiranje uzimat će se maksimalni rezultati koji su postignuti testiranjem.

Analiza brzine izvođenja aplikacije izvršavat će se sljedećim testiranjima:

Sporim pomicanjem po aplikaciji prilikom gledanja zadataka (engl. Scrolling)
Naglim pomicanjem po aplikaciji prilikom gledanja zadataka
Unosom novoga zadataka
Označavanje zadataka kao završenog ili ne završenog
Korištenje resursa u stanju mirovanja

Prilikom izvedbe testiranje mijenjat ću se korišteni resursi te će se njima kasnije vršiti usporedba FW-a. Prvo testiranje izvedeno je koristeći aplikaciju izrađenu u React Native FW-u, te su rezultati testiranja prikazani tablicom 1.

Tablica 3 - React Native - Rezultati testiranja (Izvor: obrada autora)

Test/Resurs	Procesor	Memorija	Mreža	Energija	Thread #
1.	20%	125.2 MB	0 B/s – 0 B/s	Medium	55
2.	29%	130 MB	0 B/s – 0 B/s	Medium	56
3.	1.1.%	132 MB	3.1 KB/s - 2.2.KB/s	Light	46
4.	8%	114.6 MB	0 B/s – 0 B/s	Light	50
5.	0%	113.7 MB	0 B/s – 0 B/s	None	46

Iz prikazane tablice vidljivo je da najveće opterećenje se izvršavalo kroz pomicanje kroz aplikaciju. Gdje se testom 1 koristilo 20% procesorske snage, ali korištenje memorije se nije promijenilo puno u usporedbi sa stanjem resursa dok je aplikacija bila u stanju mirovanja. Kroz cijelo testiranje mobilne aplikacije potrošnja memorije nije se previše promijenila te je razlika bila svega maksimalnih 20 MB-a. Korištenje procesorske snage znatno se razlikovalo na svakom testiranju gdje je najznačajnija razlika bila na testu br. 2 gdje je se potrošnja procesorske porasla na 29% što je za skoro 10% veća vrijednost od testa br. 1. Energija koju je aplikacije trošila kroz nije prelazila razinu „Medium“ i u procesoru najzahtjevnijem testiranju.

Sljedeća aplikacija koja će se testirati bit će aplikacija izrađena Flutter FW-om, rezultati testiranja prikazani su tablicom 2.

Tablica 4 – Flutter - Rezultati testiranja (Izvor: obrada autora)

Test/Resurs	Procesor	Memorija	Mreža	Energija	Thread #
1.	8%	132 MB	0 B/s – 0 B/s	Light	39
2.	9%	142 MB	0 B/s – 0 B/s	Light	39
3.	2%	130 MB	3.0 KB – 2.1 KB	Light	39
4.	9%	138 MB	0 B/s – 0 B/s	Light	38
5.	0%	105.9MB	0 B/s – 0 B/s	None	37

Iz prikazane tablice vidljivo je da korištenje procesora poprilično niska na svim testovima koji su se izvršili, te korištenje nije prelazilo 10%. Kod korištenja memorija koju je aplikacija zahtijevala nije prelazila 150 MB, ali je razlika između aplikacije u stanju mirovanja i najveće moguće vrijednosti bila ~ 40 MB. Što je poprilična razlika no to nije utjecalo na sam rad aplikacije. Korištenje Energije nije prelazilo razinu „Light“ no to je zbog toga što procesor nije bio pretjerano korišten.

Zadnje testiranje koje se izvelo je testiranje Todo aplikacije korištenjem Ionic FW-a, te su rezultati prikazani tablicom 3.

Tablica 5 - Ionic - Rezultati testiranja (Izvor: obrada autora)

Test/Resurs	Procesor	Memorija	Mreža	Energija	Thread #
1.	9%	157.3 MB	0 B/s – 0 B/s	Light	49
2.	9%	160 MB	0 B/s – 0 B/s	Light	46
3.	9%	171.3 MB	5.1 KB/s – 18.8 KB/s	Light	46
4.	12%	173 MB	0 B/s – 0 B/s	Light	46
5.	0%	150.6 MB	0 B/s – 0 B/s	None	45

Iz prikazane tablice vidljivo je da je korištenje procesora poprilično nisko na svim testovima koji su se izvršili osim na testu broj 4. Tijekom označavanja zadatak dovršenim ili ne dovršenim povećala se potrošnja procesorske snage na 12%, te korištena memorija na 173 MB. Potrošnja energije bila je jednaka kroz sva testiranja. Prilikom unosa novoga zadataka korištenje prometa je bilo 5.1 KB/s sekundi tijekom slanja zadataka, a 18.8 KB/s prilikom obnove liste zadataka.

Osim navedenih testiranja izvršena je analizira potrošnje memorije dok je aplikacija u stanju mirovanja. Prema Android Profiler-u sljedeće stavke troše memoriju uređaja tijekom izvedbe: Java (Java i Kotlin kod), Native (C i C++ kod), Graphics (Grafičke komponente), Code (Fontovi i ostali resursi). Optimizacija potrošnje radne memorije može utjecati na rad same aplikacije (Memory Profiler, 2019). Potrošnja je prikazana u tablici 4.

Tablica 6 - Potrošnja memorije (Izvor: obrada autora)

Dio memorije/FW	React Native	Flutter	Ionic
Java	14.3 MB	10.2 MB	11.7 MB
Native	37.6 MB	28.1 MB	35.3 MB
Graphics	32.6 MB	45.7 MB	50.5 MB
Code	31.9 MB	24.1 MB	39.2 MB

4. ZAKLJUČAK

Provedeno istraživanje je obuhvatilo testiranje mobilnih aplikacija izrađenih u tri različita FW-a (React Native, Flutter i Ionic). Testiranje je izvedeno korištenjem Android Profiler alata koji mjeri sljedeće resurse na mobilnom uređaju: procesor, memorija, mreža i korištenje baterije s time da se dodatno mjerilo i broj dretvi (engl. Thread) koje je aplikacija koristila tijekom izvođenja. Aplikacija nad kojom su se izvršavala testiranja bila je jednostavna ToDo aplikacija koja se sastojala od mogućnosti kao što su: unos novih zadataka, označavanje zadataka završenim ili ne završenim te njihov prikaz. U analizi je pokazano da je React Native bio najzahtjevniji što se tiče potrebne procesorske snage prilikom izvršavanja procesa, dok je aplikacija izvedena u Ionic FW trošila najviše radne memorije. Usporedbom rezultata analize vidljivo je da je aplikacija koja je izrađene u Flutter FW-u koristila najmanje resursa, pa se time može reći da je prema performansama ovaj FW najbolji od analiziranih FW-ova.

Literatura:

1 React Native: What is it? and, Why is it used?, Thinkwik, Medium, 2018, https://medium.com/@thinkwik/react-native-what-is-it-and-why-is-it-used-b132c3581df, Pristupljeno: 24.12.2019.

2 What is Flutter? Here is everything you should know, Concise Software, 2019, https://medium.com/@concisesoftware/what-is-flutter-here-is-everything-you-should-know-faed3836253f, Pristupljeno: 24.12.2019.

3 https://jsonplaceholder.typicode.com/, Pristupljeno: 24.12.2019.

4 Ionic 4: All you need to know, Javier Ramos, 2019, https://medium.com/@javier.ramos1/ionic-4-all-you-need-to-know-d2b9627aaf03, Pristupljeno: 24.12.2019.

5 Android Studio, https://developer.android.com/studio, Pristupljeno: 26.12.2019.

6 React Native – Getting started, https://facebook.github.io/react-native/docs/getting-started, Pristupljeno: 28.12.2019

7 React Native – Running On Device, https://facebook.github.io/react-native/docs/running-on-device, Pristupljeno: 28.12.2019

8 Flutter – Get Started, https://flutter.dev/docs/get-started/install, Pristupljeno: 30.12.2019

9 Ionic – Installation, https://ionicframework.com/docs/installation/cli, Pristupljeno: 30.12.2019

10 Ionic – Installation (Android), https://ionicframework.com/docs/installation/android, Pristupljeno: 2.1.2020

11 Flutter – Docs, https://flutter.dev/docs, Pristupljeno: 2.1.2020

12 Ionic – Docs, https://ionicframework.com/docs, Pristupljeno: 2.1.2020

13 Flutter – Widgets, https://flutter.dev/docs/development/ui/widgets/material, Pristupljeno: 3.1.2020

14 Ionic – Components, https://ionicframework.com/docs/components, 3.1.2020

15 Memory Profiler, https://developer.android.com/studio/profile/memory-profiler, 5.1.2020

16 S.Bergmann, “PHPUnit Manual,” 2020. [Online]. Available: https://phpunit.readthedocs.io/en/8.5/.

Podaci o autorima:

Ivan Miljančić

dr.sc. Marin Kaluža

e-mail: mkaluza@veleri.hr

Marin Kaluža je viši predavač na Veleučilištu u Rijeci i nositelj nekoliko kolegija iz područja razvoja softvera i informacijskih sustava, te upravljanja i razvoja baza podataka. Doktorirao je na Filozofskom fakultetu u Zagrebu na problemima mjerenja i analize složenosti poslovnih sustava. Područje njegovog istraživanja je brzi, rapidni i agilni razvoj softvera, standardizacija i automatizacija u razvoju softvera i korisničkih sučelja. Vodio je projekte razvoja većeg broja informacijskih sustava iz područja školstva, industrijske proizvodnje, zaštite na radu, ekonomije i medicine.

Veleučilište u Rijeci
Trpimirova 2/V, 51 000 Rijeka
tel. 051/321-300
fax. 051/211-270,
www.veleri.hr

SUSTAV ZA KONTROLU PROIZVODNJE U PIVARSKOJ INDUSTRIJI

Mirko Sršen bacc. ing. techn. inf., izv. prof. dr. sc. Tihomir Orehovački

SAŽETAK

Brewing control system (u nastavku BCS) je sustav namijenjen nadzoru i automatizaciji tvornica u pivarskoj industriji. Ovaj rad opisuje niz servisa i funkcionalnosti koje se koriste unutar BCS-a. Uz primjere koda, opisan je način na koji su implementirani sustav za spremanje podataka, produkcijski servis koji se koristi kao poveznica između mikroupravljačkih jedinica i poslužitelja te obavještajni servis koji služi za slanje obavijesti korisnicima u kojima ih informira o kritičnim događajima koji se manifestiraju unutar tvornice. Osim toga, opisane se funkcionalnosti web poslužitelja i korisničkog sučelja koji zajedno imaju zadaću pružiti korisniku uvid u podatke o proizvodnji te omogućiti određeni stupanj kontrole nad proizvodnim procesom.

ABSTRACT

Brewing control system (BCS) is a system that allows tracking and automatization of factories in beer industry. This paper describes several services and functionalities that are used inside of BCS. Through the examples of code, implementation of a system for data storage, production service that is used as a link between microprocessor and server as well as notification service that is used for sending messages to users in order to inform them about critical event that are happening inside the factory, have been described. Functionalities of web server and user interface that have a job to offer users an insight into the production data and allow them to have certain level of control over production cycle are also described.

1. UVOD

BCS je sustav namijenjen manjim pivovarama koji nastoji ukloniti što veći broj pogrešaka uzrokovanih ljudskim faktorom i neispravnom opremom. Sustav se sastoji od većeg broja servisa koji se odnose na proizvodni proces i kontrolu. Mikroupravljačke jedinice koje se nalaze u sklopu tvornice šalju informacije o proizvodnji na osnovu kojih BCS odlučuje o sljedećim potrebnim koracima. Obavještajni servis je zadužen za slanje poruka koje sadrže ključne informacije o kritičnim događajima unutar tvornice kako bi korisnici mogli pravodobno reagirati na potencionalne probleme unutar tvornice. Svi servisi posjeduju određenu razinu konfiguracije koja se uz pristup podatcima preko web poslužitelja i korisničkog sučelja pruža korisniku na raspolaganje.

2. O INDUSTRIJI PIVA

Pivarska industrija je u konstantnom porastu. Ove godine predviđa se rast od oko 5% što će rezultirati ukupnim prihodima pivarske industrije u iznosu od 615 milijardi dolara. Kao posljedica navedenog, sve više dolazi do potrebe za hardverskim i softverskim rješenjima za unaprjeđenje poslovanja. Tvrtke koje danas aktivno rade na rješenjima za pivarske industrije posjeduju softver koji je u osnovi prenamijenjen iz neke duge industrije (aluminijske) i prilagođen potrebama pivovara. Osim toga, proizvodnja softverskog rješenja je skupa te se funkcionalnosti sustava teško mogu testirati u pravim uvjetima iz razloga što je potreban i hardverski dio koji se nalazi u tvornici. Hardverski dio je drugi problem sa kojim se suočavaju pivarski softveri. Danas je vrlo rijetko da jedna tvrtka pruža kvalitetno hardversko i softversko rješenje, tako da su za potpunu implementaciju potrebne dvije tvrtke od kojih će svaka ispostaviti svoj račun za projekt što dovodi do sljedećeg problema, a to je cijena. Tržište na kojem BCS želi biti prisutan nije povezano sa masivnim tvornicama koje svoje zarade mjere u milijunima ili milijardama, već malim pivovarama koje uglavnom djeluju na lokalnoj razini. Osnovne verzije sustava koji se danas nudi na tržištu koštaju više nego godišnja zarada jedne takve pivovare te su pune nepotrebnih informacija i funkcionalnosti koje su možda potrebne inženjerima unutar velikih tvorničkih postrojenja, ali ne i kuharima (eng. brewer) unutar malih tvornica koji žele znati samo kritične informacije poput temperature unutar bačve i vremena kraja procesa fermentacije. Stoga BCS koji će biti opisan u ovom radu nudi hardversko i softversko rješenje koje je svojom cijenom i više nego pristupačno malim pivovarama.

2.1. PROBLEM FERMENTACIJE

Fermentacija je proces alkoholnog vrenja u kojem se pomoću vode, kvasca, šećera i određene temperature dobiva alkoholna smjesa. Ista ovisno o vrsti piva može trajati od nekoliko dana do nekoliko tjedana. Iznimno je važno održavati temperaturu

piva tijekom fermentacije u dopuštenim okvirima jer će inače doći do ispuštanja nepoželjnih proteina iz kvasca koji će narušiti kvalitetu finalnog proizvoda. Budući da radnici imaju svoje fiksno radno vrijeme te nisu nazočni tijekom cijelog procesa fermentacije, potrebno je imati oblik automatskog rada rashladnog sustava koji će regulirati temperaturu piva tijekom cijelog dana. Jedan od načina je da motor rashladnog sustava konstantno radi i održava pritisak u cijevi dok se putanje ventila do određene bačve po potrebi otvaraju i zatvaraju. Već sad se može primijetiti da ovakav rad nije optimalan te često dolazi do problema kao što su: pucanje cijevi zbog konstantnog pritiska, pregrijavanje rashladnog motora zbog neprestanog rada, pad napona u tvornici, skupa proizvodnja i slično. Kvar opreme glavni je problem manjih pivovara iz razloga što je ista uglavnom prethodno bila rabljena (nabavljena je kao polovna) te ne postoji način oporavka bez ljudske intervencije.

2.2. DEFINICIJA PROJEKTA

BCS-ova prva implementacija i testiranje biti će obavljena unutar tvornice za proizvodnju piva u Dubrovniku (Dubrovnik beer company) koja je prikazana na slici 1. Sustav bi trebao biti zamjena za trenutno implementirani sustav u tvornici koji se pokazao nepouzdan, nepraktičan i preskup za održavanje. Glavno svojstvo koje BCS mora pružiti je regulacija temperature i obavještavanje u slučaju da dođe do problema koje sustav ne može riješiti. Cilj sustava je da smanji potrošnju energije i vijek trajanja opreme, donese faktor pouzdanosti i sigurnosti u proizvodni ciklus te samim time poveća kvalitetu i konkurentnost proizvoda na tržištu.

Slika 21. Tvornica Dubrovnik beer company

3. PROGRAMIRANJE UGRADBENIH SUSTAVA

Zadatak programera ugradbenih sustava je dizajn i konstrukcija prve razine mreže za kontrolu proizvodnog procesa koja podrazumijeva niz prostorno odvojenih modula za mjerenje, pohranu i distribuciju parametara koji uvjetuju kvalitetu proizvoda te njihovo prosljeđivanje najbližoj radnoj stanici koja je povezana sa središnjim sustavom regulacije. Svaki je modul fizički sastavljen od kućišta, članaka za napajanje, prilagodnika napajanja, mikroupravljačke jedinice, jedinice za pohranu, čipa za bežičnu komunikaciju i mjerne sonde.

Mikroupravljačka jedinica programirana je kao RTOS (eng. real time operating system) koji nadzire aktivaciju modula, provjeru ispravnosti njegovih sastavnica, proces mjerenja, pohrane i prosljeđivanja izmjerenih rezultata. Svaki umreženi modul čini čvor mjerenja (eng. meshnet) te ovisno o međusobnoj kvaliteti signala kao cjelina pronalaze najkraći put do radne stanice kako bi se izbjegla mogućnost narušavanja podataka uslijed nepredviđenih događaja, ljudske pogreške ili fizičkog kvara na pojedinom modulu. Podatci su do radne stanice proslijeđeni u obliku skripte formatiranih podataka aplikacijskim protokolom. Modul je prema potrebi moguće iskoristiti kao upravljački prekidač (relej) za kontrolu sklopnog podsustava u sustavu rashlađivanja.

Glavna regulacija odvija se kontrolom temperature spremnika za fermentaciju pri čemu se koriste sonde različitih duljina kako bi se regulirala temperatura u različitim točkama spremnika. Sonde su obložene spiralama od nehrđajućeg prehrambenog inoksa kako se ne bi ugrozila tehnološka ispravnost proizvoda.

4. BAZA PODATAKA

Za potrebe implementacije BCS-a korištena je SQL baza podataka te paket pod nazivom Entity Framework (u nastavku EF) pomoću kojeg se vrši spajanje i upravljanje podatcima u tablici. Sustav zabranjuje korištenje naredbi za definiranje modela podataka (eng. data definition language, DDL) dok su neke naredbe za manipulaciju sa podatcima (eng. data manipulation language, DML) poput brisanja omogućene samo nad administrativnim tablicama (primjerice dodavanja i brisanje korisnika). Projekt ima za cilj da sve naredbe funkcioniraju preko transakcija sa svojstvima atomarnosti, dosljednosti, izolacije i izdržljivosti. EF omogućuje korištenje C# klasa za definiciju entiteta i veza unutar baze podataka. Ovakav pristup kreiranja entiteta unutar baze podataka se još zove Model prvi (eng. Model first) pristup. omogućuje korištenje klase za definiciju tablice unutar baze podataka. Primjer koda 1 prikazuje primjer „Equipment“ klase koja služi za definiciju sve opreme unutar tvornice. Klasa koju „Equipment“ implementira je „HasId“ (linija 8) koja proširuje model sa jedinstvenim ključem (eng. primary key) unutar baze te jedinstveni naziv (eng. identifier). Kolekcija „ProductionResponses“ (linija 14) definira vezu 1:N gdje jedan „Equipment“ entitet može imati N „ProductionResponses“ entiteta.

```
using Kernel.Database;
using System;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;

namespace DatabaseModels
{
    public class Equipment : HasId
    {
        public string State { get; set; }
        public int CurrentCapacity { get; set; }
        public int MaxCapacity { get; set; }
        public DateTime? LastInspectionDate { get; set; }
        public List<ProductionResponse> ProductionResponses { get; set; }
    }
```

Primjer koda 1. Definicija modela tablice

Proces pretvaranja modela klase unutar projekta u tablice baze podataka se naziva migracija. Migracija EF-a nam omogućuje automatizirano ažuriranje modela tablica, a potrebni koraci su sljedeći:

Instalacija EF paketa pozivanjem naredbe u upravitelju paketa (eng. package manager):

Install-Package EntityFramework -Version 6.4.0

Omogućavanje migracija naredbom:

Enable-migrations

Dodavanje nove migracije:

Add-migration *proizvoljni naziv migracije*

Dodavanjem nove migracije generira se klasa unutar datoteke Migrations u kojoj su striktno definirane sve funkcije koje će se odvijati nad bazom prilikom izvršavanja migracije. Unutar klase migracije nalaze se dvije osnovne funkcije pod nazivom: Up i Down. Funkcija Up služi za dodavanje dok funkcija Down služi za uklanjanje elemenata i veza u bazi podataka.

Pokretanje migracije nad bazom se izvršava naredbom:

Update-database

Za uspješno izvršavanje migracije potrebno je posjedovati i pravilnu konfiguraciju unutar App.config datoteke. U slučaju da takva datoteka ne postoji prilikom instalacije EF paketa, EF će sam dodati svoju zadanu (eng. default) konfiguraciju. Primjer koda 2 prikazuje izgled konfiguracije koja će kreirati SQL bazu na lokalnom računalu gdje će naziv baze podataka biti „database“, a koristiti će integriranu sigurnost prilikom pristupa i manipulacije sa podatcima korisnika koji je trenutno prijavljen na računalo.

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
  <configSections>
        <section    name="entityFramework"    type="System.Data.Entity.Internal.ConfigFile.EntityFrameworkSection,
EntityFramework, Version=6.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089" requirePermission="false" />
    </configSections>
  <entityFramework>
    <providers>
      <provider    invariantName="System.Data.SqlClient"    type="System.Data.Entity.SqlServer.SqlProviderServices,
EntityFramework.SqlServer" />
    </providers>
  </entityFramework>
  <connectionStrings>
    <add       name="defaultConnection"       connectionString="Data       Source=localhost\SQLEXPRESS;Integrated
Security=true;Initial Catalog=database" providerName="System.Data.SqlClient" />
  </connectionStrings>
</configuration>
```

Primjer koda 2. Definicija konfiguracije EF-a

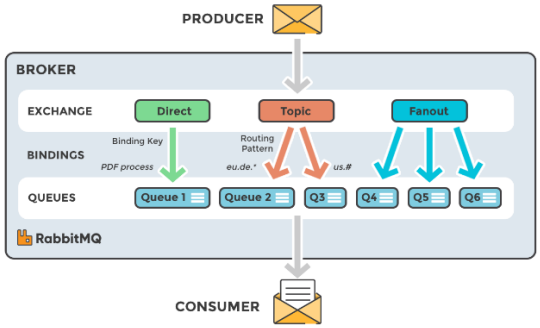
5. POSLUŽITELJSKI SERVISI I KOMUNIKACIJA

BCS se ne sastoji od jednog centralnog servisa već od više njih koji su međusobno nezavisni. Servisi mogu, ali i ne moraju međusobno komunicirati. Navedeno daje temeljni stupanj apstrakcije gdje je omogućena modifikacija (primjerice dodavanje, micanje i proširivanje) pojedinih servisa od projekta do projekta. Komunikacija se dijeli na dva osnovna tipa:

- preko Rabbit MQ-a (u nastavku RMQ)
- preko baze podataka.

Slika 2 prikazuje način na koji se vrši komunikacija pomoću razmjene poruka. BCS koristi servis pod nazivom RMQ (eng. Rabbit message queue system) koji se bazira na konfiguraciji pomoću teme (eng. topic), a funkcionira na sljedeći način:

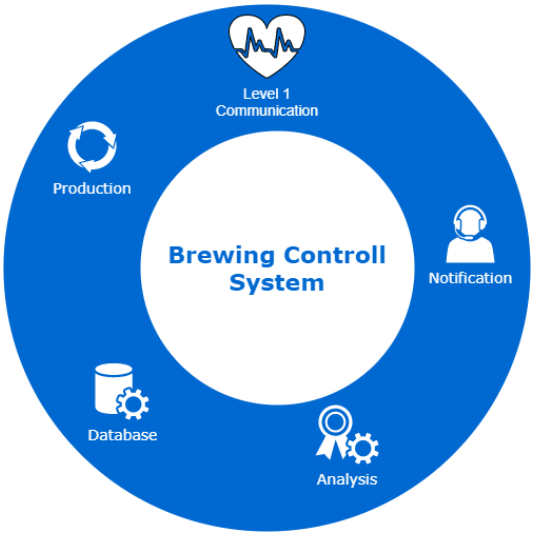
1 Generirana poruka se šalje na RMQ servis te u sebi sadrži odgovarajuće podatke i ključ pomoću kojeg se definiraju odredišni redovi.
2 Poruka se gura (eng. push) u sve redove koji su pretplaćeni na odgovarajuću temu te je servis koji je povezan sa tim redom preuzima. Ukoliko je više servisa povezano na isti red, poruku će primiti samo jedan servis po načelu FCFS (eng. first come, first served).
3 Servis obavlja logiku koja je predviđena u slučaju da se poruka takvog tipa primi.



Slika 2. Shema RMQ razmjene (Anik, 2019)

Ovakav tip interakcije se uglavnom koristi za uspostavljanje komunikacije između mikroupravljačkih jedinica i produkcijskog servisa, te se može koristiti za razmjenu podataka između više aplikacija unutar iste mreže (sinkronizacija baze između više tvornica preko VPN-a). Komunikacija preko baze podataka se isključivo koristi za servise koji ne moraju pratiti tok informacija u stvarnom vremenu (eng. real time) već rade sa skupom podataka u određenom vremenskom intervalu. Primjer toga bi bio Analitički servis koji na kraju svakog dana pokreće niz kalkulacija nad bazom kao što su prosječna potrošnja/proizvodnja piva u danu/tjednu/mjesecu/godini/ukupna, najviša/najniža temperatura po bačvi, količina nevaljanog piva, prihodi/rashodi proizvodnje, status skladišta i resursa na kraju smjene, itd. Drugi primjer bi bila treća razina sustava odnosno korisničko sučelje. Sve podatke koje korisnik vidi na sučelju i sve podatke koje korisnik mijenja pružaju se isključivo iz baze podataka.

Servisi koje sustav posjeduje su: Obavještajni servis, Produkcijski Servis i Web servis. U planu je još uvesti Analitički servis koji bi vršio kalkulacije podataka i pripremao izvještaje te servis koji će se brinuti o popisu materijala u skladištu. Slika 3 vizualno predstavlja servise od kojih se BCS sastoji.



Slika 3. Usluge koje pruža BCS

6. PRODUKCIJSKI SERVIS

Ovaj servis predstavlja komunikaciju između mikroupravljačkih jedinica i našeg sustava koristeći se RMQ servisom opisanim u prethodnim poglavljima. Poruka je serijalizirani objekt čiji je ključ jednak punom imenu klase koja taj objekt definira (primjer: ProductionService.Messages.EquipmentMessage). Budući da C# koristi generičku provjeru tijekom rada programa (eng. runtime generics) dok jezici kao C++ koriste generičku provjeru tijekom prevođenja programa (eng. compile time generics), cijeli sustav je postao više generičan te je pomoću refleksije moguće podesiti sustav na terenu bez potrebe za novom verzijom cijelog projekta. Refleksijom se također saznaje koja funkcija prima određeni objekt kao parametar što uvelike

olakšava rad produkcijskom servisu da odredi koja funkcija je zadužena za određenu poruku. Ovakvim pristupom BCS je uveo dodatni sloj između poslane poruke i baze podataka u kojem se može dodatno obavljati provjera ispravnosti poruka, zapisivanje važnih događaje o radu aplikacije te se mogu pozivati dodatni servise u slučaju da je to potrebno.

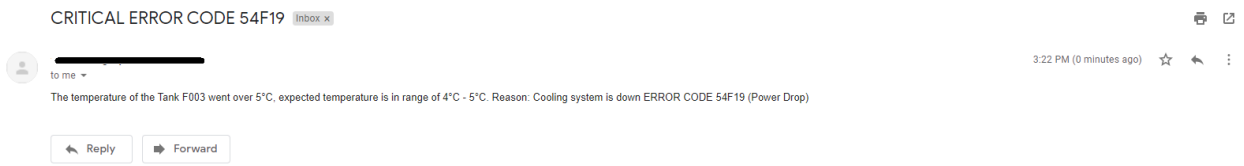
Primjer koda 3 prikazuje primjer implementacije primatelja poruka. Klasa „NetworkServiceMessage-Handler“ implementira generičko sučelje „IHandle<T>” (linija 9) sa kojim definira tip poruke na koji će se klasa pretplatiti. Implementacijom sučelja omogućuje se pristup funkciji zaduženoj za primanje poruka (linija 15) u kojoj se vrši mapiranje poruke i spremanje entiteta u bazu podataka.

```
1  using DatabaseModels;
2  using NetworkService.Messages;
3  using NetworkService.Rmq;
4  using System;
5  using System.Linq;
6
7  namespace NetworkService
8  {
9      public class NetworkServiceMessageHandler : IHandle<EquipmentMessage>
10     {
11         public NetworkServiceMessageHandler()
12         {
13         }
14
15         public void Handle(EquipmentMessage message)
16         {
17             using(var context = new MainContext())
18             {
19                 context.Database.Log = Console.Write;
20                 var first = context.Equipment.First();
21                 context.Set<Equipment>().Add(MapToModel(message));
22                 context.SaveChanges();
23             }
24         }
25
26         private Equipment MapToModel(EquipmentMessage message)
27         {
28             return new Equipment()
29             {
30                 Id = message.Id,
31                 State = message.State,
32                 CurrentCapacity = message.CurrentCapacity,
33                 LastInspectionDate = message.LastInspectionDate,
34                 MaxCapacity = message.MaxCapacity
35             };
36         }
37     }
38 }
```

Primjer koda 3. Definicija rukovoditelja poruka u produkcijskom servisu

7. OBAVJEŠTAJNI SERVIS

Ovaj servis je zadužen za obavještavanje korisnika o kritičnim porukama unutar tvornice. Ponekad u tvornicama dođe do zakazivanja opreme ili neispravne konfiguracije za koju je odgovoran ljudski faktor u proizvodnji. U praksi bi produkcijski servis trebao biti zadužen za regulaciju sve opreme u tvornici, ali postoje slučajevi kada je nastali problem izvan područja njegovog djelovanja te je potreban ljudski faktor da ispravi pogrešku (primjerice pad napona, mehanička oštećenja opreme i sl.). Obavještajni servis prima šifru pogreške koja se dogodila i obavještava sve pretplaćene korisnike do kakvog je problema došlo te predlaže korake koje je potrebno poduzeti da se sustav ponovno vrati u funkcionalno stanje. Poruke odnosno obavijesti se šalju preko SMTP servisa. Model Obavještajnog servisa sadrži kod pogreške, primatelje, pošiljatelja, opis problema, vrijeme nastanka problema i prijedlog popravka. Neke tvornice ne rade vikendima tako da ako se problem u radu u najgorem slučaju dogodi u petak nakon kraja radne smjene, nitko neće biti u mogućnosti djelovati do ponedjeljka ujutro. Obavještajni servis pomaže da se ovakvi problemi pravovremeno primijete i spriječe. Slika 4 prikazuje primjer izgleda poruke koju šalje Obavještajni servis.



Slika 4. Izgled poruke koju pretplatnik dobiva

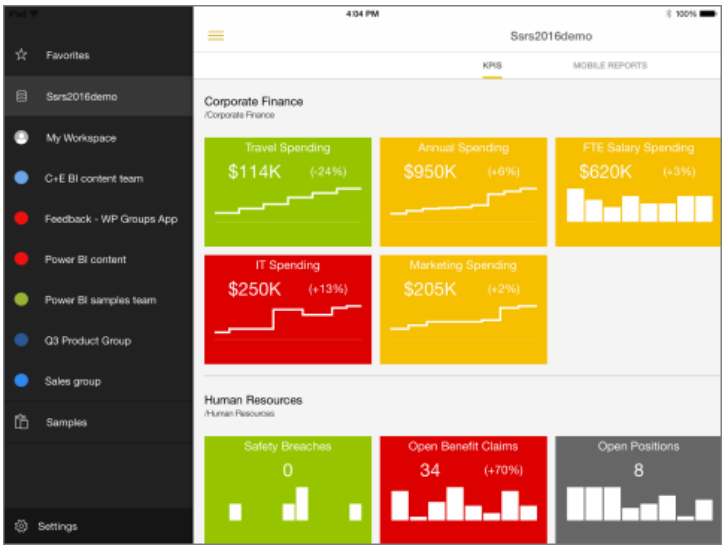
Primjer koda 4 prikazuje implementaciju slanja e-mail poruke sa Obavještajnog servisa. Funkcija „SendNotificationToUser“ (linija 14) prima model koji sadrži podatke o pošiljateljevim podatcima koji će se koristiti za slanje poruka, e-mail adresu primatelja, naslov poruke i poruku koju treba poslati. Za slanje se koristi SMTP servis kojeg pruža Gmail (linija 17). Također je potrebno isključiti sve dodatne sigurnosne provjere pošiljatelja na e-mail-u kako bi servis mogao automatski poslati poruku.

```
1  using SendGrid;
2  using SendGrid.Helpers.Mail;
3  using SendGrid.SmtpApi;
4  using System;
5  using System.Collections.Generic;
6  using System.Net;
7  using System.Net.Mail;
8  using System.Threading.Tasks;
9
10 namespace NotificationService
11 {
12     public static class SendNotification
13     {
14         public static void SendNotificationToUser(NotificationData info)
15         {
16             // Command-line argument must be the SMTP host.
17             SmtpClient client = new SmtpClient("smtp.gmail.com", 587);
18             client.EnableSsl = true;
19             client.Timeout = 10000;
20             client.DeliveryMethod = SmtpDeliveryMethod.Network;
21             client.UseDefaultCredentials = false;
22              client.Credentials = new NetworkCredential(info.senderEmail, info.senderPassoword);
23
24             var message = new MailMessage();
25
26             message.To.Add(info.userEmail);
27             message.From = new MailAddress(address: info.senderEmail);
28             message.Subject = info.titleOfEmail;
29             message.Body = info.bodyOfEmail;
30
31             client.Send(message);
32         }
33     }
34 }
```

Primjer koda 4. Slanje poruka obavještajnog servisa

8. ANALITIČKI SERVIS

Za menadžere i vlasnike pivovara ovo je vjerojatno i najkorisniji sustav. Prikaz proizvodnje i svakodnevni uvid pomaže proizvođaču da prati promjene u proizvodnji te unaprijed planira svoju proizvodnju imajući uvid u stvarni kapacitet i djelotvornost svoga pogona. Za ovaj servis koristit će se SQL Reporting Servis koji bi trebao obavljati pozive direktno nad bazom i kreirati izjave o stanju tvornice u određenim vremenskim periodima. Ovaj servis je opcionalan unutar BCS-a zbog toga što ovisi o potrebama korisnika odnosno prilagođava se onim informacijama koje korisnik želi vidjeti. Slika 5 prikazuje izgled izvještaja urađenog u SQL Reporting Servisu.



Slika 5. Primjer SQL izvješća (Microsoft, 2020)

9. WEB POSLUŽITELJ

Za korištenje REST API-ja od strane korisničkog sučelja, BCS koristi web poslužitelj koji je instaliran na IIS (eng. Internet information service) lokalnom poslužitelju. Primjer koda 5 definira implementaciju kontrolera koji pruža API za segmente produkcijskog odgovora (eng. production segment response).

RoutePrefix (linija 10) definira prefiks API-ja unutar upravitelja. Adapter (linija 13) definira logiku specifičnu za produkcijske odgovore koja će se pozivati prilikom pojedinih poziva unutar upravitelja. Adapter je injektiran kroz konstruktor, a kontejner injekcije se definira u WebApiApplication prije registracije svih upravitelja. Ovaj upravitelj sadrži isključivo naredbe dohvaćanja podataka iz razloga što u niti jednom trenutku ne želimo omogućiti korisniku da radi izmjene produkcijskih podataka. Atribut „Route“ (linija 29) služi za dohvaćanje jedinstvenog podatka pomoću identifikatora koji se šalje kroz link na web poslužitelja. Svi podatci koji se primaju i šalju su u JSON formatu.

```
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Web;
5  using System.Web.Http;
6  using WebHost.Models;
7
8  namespace WebHost.Controllers
9  {
10     [RoutePrefix("ProductionResponses")]
11     public class ProductionSegmentResponseController : ApiController
12     {
13         private readonly ProductionSegmentResponseAdapter adapter;
14
15           public ProductionSegmentResponseController(ProductionSegmentResponseAdapter    adapter)
16         {
17             this.adapter = adapter;
18         }
19
20         [HttpGet]
21         [Route("")]
22         public IHttpActionResult getAll()
23         {
24             return this.Ok(this.adapter.GetAll());
25         }
26
27         [HttpGet]
28         [Route("GetByOid/{oid:int}")]
29         public IHttpActionResult getByOid(long oid)
30         {
31             return this.Ok(this.adapter.GetByOid(oid));
32         }
33     }
34 }
```

Primjer koda 5. Definicija „Production segment response“ kontrolera

10. KORISNIČKO SUČELJE

Korisničko sučelje (u nastavku UI) ima za cilj pružiti korisniku uvid u proizvodnju te omogućiti određeni stupanj kontrole nad podatcima. Korištenjem Angular okvira omogućena je bolja responzivnost stranice u produkciji jer promjenom pregleda stranice (eng. view) podatci se ne učitavaju iznova već se spremaju u privremenoj memoriji (Angular, 2019).

Velika prednost Angulara je mogućnost korištenja TypeScripta koji omogućuje određenu razinu apstrakcije na UI-u. Primjer koda 6 opisuje jedan primjer takve apstrakcije u klasi „Extensions“. Atribut „providedIn: 'root'“ (Linija 4) govori da će ova klasa biti omogućena u samom korijenu aplikacije (biti će dostupna svima na korištenje). Funkcija „sortByStartDate“ (linija 8) omogućuje sortiranje kolekcija datuma uzlazno i silazno, dok funkcija „performFilter“(linija 23) filtrira sve podatke koji odgovaraju zadanom tekstu.

```
1  import { Injectable } from '@angular/core';
2
3  @Injectable({
4    providedIn: 'root'
5  })
6  export class Extensions
7  {
8    public sortByStartDate<T extends object>(collection: T[], propertyToSort: string, sortAscending)
9    {
10     if(sortAscending)
11     {
12         collection.sort((a, b) =>{return Reflect.get(b, propertyToSort) < Reflect.get(a, propertyToSort) ? 1 : -
13 1});
14     }
15     else
16     {
17         collection.sort((a, b) =>{return Reflect.get(b, propertyToSort) > Reflect.get(a, propertyToSort) ? 1 : -
18 1});
19     }
20   }
21
22   public performFilter<T extends object>(value: string, collection: T[], property:string): T[] {
23     value = value.toLocaleLowerCase();
24     return collection.filter((single: T) =>
25     Reflect.get(single, property).toLocaleLowerCase().indexOf(value) !== -1);
26   }
27 }
```

Primjer koda 6. Definicija Ekstenzijskih funkcija unutar typescripta

Primjer koda 7 prikazuje pozivanje „Extensions“ klase kroz komponentu segmenta produkcijskog odgovora gdje se kroz funkcije „equipmentIdFilter“, „typeOfBeerFilter“ i „sort“ (linija 1) filtriraju podatci po nazivu opreme, vrsti piva i datumu.

```
1  set equipmentIdFilter(value: string)
2    {
3      this._equipmentIdFilter = value;
4      this.filteredProdSegResponse = this._equipmentIdFilter
5      ? this.extensions.performFilter(value, this.prodSegResponses, "equipmentId")
6      :  this.prodSegResponses;
7    }
8  set typeOfBeerFilter(value: string)
9    {
10     this._beerInProductionFilter = value;
11     this.filteredProdSegResponse = this._beerInProductionFilter
12     ? this.extensions.performFilter(value, this.prodSegResponses, "beerInProduction")
13     :  this.prodSegResponses;
14   }
15
16   sort<T extends object>(collection: T[], propertyToSort: string, sortAscending)
17   {
18     this.sortAscending = !sortAscending;
19     this.extensions.sortByStartDate(collection, propertyToSort, sortAscending);
20
```

Primjer koda 7. Korištenje ekstenzijskih funkcija unutar klase segmenta produkcijskih odgovora

Funkcije navedene u primjeru koda 8 se povezuju sa HTML kodom stranice pomoću Angularovih NG naredbi koje su prikazane u primjeru koda 7. NgModel (linija 5 i 19) postavlja vrijednost unesenog teksta za filter stupaca dok funkcija sortiranja (linija 10 i 15) sortira datume silazno i uzlazno.

```
          <thead>
            <tr>
              <th>
                  <p>Equipment</p>
                  <input type='text' [(ngModel)]='equipmentIdFilter'/>
              </th>
              <th>
                  <p>Start Time</p>
                  <button class='btn btn-
primary' (click) = 'sort(filteredProdSegResponse, startTime)'>Sort</button>
              </th>
              <th>
                <p>Expected end time</p>
                <button class='btn btn-
primary' (click) = 'sort(filteredProdSegResponse, expectedEndTime)'>Sort</button>
              </th>
              <th>
                  <p>Beer in production</p>
                  <input type='text' [(ngModel)]='typeOfBeerFilter'/></th>
            </tr>
          </thead>
```

Primjer koda 8. Primjer pozivanja NG naredbi unutar HTML koda stranice.

Slika 6 prikazuje izgled spomenutih funkcija na UI-u:

Equipment	Start Time	Expected end time	Beer in production
	Sort	Sort	
1f18f016-1ee5-42dc-a110-48e1e4646760	Dec 11, 2019, 5:41:24 PM	Dec 11, 2019, 8:41:24 PM	Lager
2a77f172-db5f-48ee-9f8c-6e1cbd5ecf62	Jan 1, 1, 12:00:00 AM	Jan 1, 1, 3:00:00 AM	Lager

Slika 6. Prikaz funkcija sortiranja i filtriranja

11. ANGULAR SERVISI I KOMUNIKACIJA SA WEB POSLUŽITELJEM

Angularovi servisi se koriste za uspostavu komunikacije sa web poslužiteljem. Za komunikaciju sa poslužiteljem potrebno je definirati skup linkova preko kojih možemo doći do određene funkcionalnosti. Primjer koda 9 prikazuje definiciju linkova koji će Angularovi servisi koristiti.

```
export const address = 'https://localhost:44369/';

/*Equipment */
    export const equipmentGet = address + 'Equipments';
    export const equipmentGetByOid = address + 'ProductionResponse/GetByOid/';

/*Production Response */
    export const prodRespGet = address + 'ProductionResponses';
    export const prodRespGetByOid = address + 'ProductionResponse/GetByOid/';
```

Primjer koda 9. Definicija linkova web poslužitelja.

Primjer koda 10 prikazuje implementaciju produkcijskog servisa gdje kroz injektiranog HTTP klijenta (injekcija se vrši u liniji 5) pozivamo linkove sa slike 6 u odgovarajućim funkcijama (linija 8 i 15) te dobivene podatke mapiramo u odgovarajuće modele. Funkcija „handleError“ (linija 22) ima zadaću uhvatiti i ispisati sve pogreške nestale prilikom poziva specifičnog API-ja.

```
@Injectable({
  providedIn: 'root'
})
export class ProductionResponseService {
  constructor(private http: HttpClient) { }

  getAll() : Observable<IProductionResponse[]>{
    return this.http.get<IProductionResponse[]>(prodRespGet).pipe(
      tap(data=> console.log('All' + JSON.stringify(data))),
      catchError(this.handleError)
    )
  }

  getByOid(oid:number) : Observable<IProductionResponse>{
```

```
    return this.http.get<IProductionResponse>(prodRespGetByOid + oid.toString()).pipe(
      tap(data=> console.log('All' + JSON.stringify(data))),
      catchError(this.handleError)
    )
  }

  private handleError(err: HttpErrorResponse){
    let errorMessage = '';
    if(err.error instanceof ErrorEvent)
    {
      errorMessage = `An error occurred: ${err.error.message}`;
    }
    else
    {
      errorMessage = `Server returned code: ${err.status}, error message is: ${err.message}`
    }

    console.error(errorMessage);
    return throwError(errorMessage);
  }
}
```

Primjer koda 10. Definicija produkcijskog servisa.

Servis koji se želi koristiti unutar klase potrebno je prvo injektirati kroz konstruktor te se zatim pretplatiti na odgovarajuću metodu. Primjer koda 11 prikazuje način na koji se pretplaćujemo na funkciju „getAll“(primjer koda 8 linija 7) iz servisa produkcijskih odgovora.

```
    this.productionSegmentResponseService.getAll().subscribe({
      next: prodSegResponses =>
      {
        this.prodSegResponses = prodSegResponses;
        this.filteredProdSegResponse = this.prodSegResponses;
      },
      error: err => this.errMessage = err
    })
```

Primjer koda 11. Pretplaćivanje na funkcije unutar servisa.

12. ZAKLJUČAK

BCS posjeduje set servisa koji olakšavaju i stabiliziraju rad tvornica piva. Svojim mikroupravljačkim jedinicama pruža potrebne informacije o stanju opreme i materijala te obavlja zadatke koje mu BCS određuje. Produkcijski servis se koristi kao poveznica između prve i druge razine sustava u svrhu spremanja podataka te pravodobno reagiranje na poruke upozorenja. Obavještajni servis služi za slanje poruka korisnicima preko SMTP-a kako bi ih obavijestili o kritičkim informacijama unutar tvornice kao što su zakazivanja opreme te završetci proizvodnog ciklusa. Web poslužitelj definira funkcionalnosti koje upotrebljava korisničko sučelje. Isto je zaduženo da korisniku pruži sve potrebne informacije o tijeku proizvodnje te mu omogući određeni stupanj kontrole nad proizvodnim procesom i administracijom. BSC svojom jeftinom cijenom, jednostavnom implementacijom i pouzdanošću pokušava osvojiti tržište malih i lokalnih pivovara. U daljnjim verzijama BSC-a planiraju se dodati servisi koji će povećati vrijednost proizvoda i konkurentnost sustava na tržištu.

Literatura:

1 Angular University (2019), Single Page Application (SPA): What are the Benefits? preuzeto 10.2.2020. sa <https://blog.angular-university.io/why-a-single-page-application-what-are-the-benefits-what-is-a-spa/>

2 Anik, S.S.I. (2019), RabbitMQ for PHP Developers, preuzeto 10.2.2020. sa <https://medium.com/@sirajul.anik/rabbitmq-for-php-developers-c17cd019a90>

3 Microsoft (2020), The web portal of a report server (SSRS Native Mode), preuzeto 9.2.2020. sa <https://docs.microsoft.com/en-us/sql/reporting-services/web-portal-ssrs-native-mode?view=sql-server-ver15>

Podaci o autorima:

Mirko Sršen, bacc. ing. techn. inf.

Email: mirko@rawfury.com

Mirko Sršen je 2017. godine stekao akademski naziv prvostupnika informatike na Tehničkom veleučilištu u Zagrebu. Na Fakultetu informatike Sveučilišta Jurja Dobrile u Puli trenutno pohađa 2. godinu diplomskog studija informatike. Sa 20 godina postao je certificirani C# developer. Glavna područja interesa su mu razvoj aplikacija za automatizaciju proizvodnje i razvoj računalnih igara u programskim jezicima C#, TypeScript, C++ i C.

Izv. prof. dr. sc. Tihomir Orehovački

Email: tihomir.orehovacki@unipu.hr

Tihomir Orehovački diplomirao je i doktorirao 2005. i 2013. godine, respektivno, na Fakultetu organizacije i informatike Sveučilišta u Zagrebu. Pedagoško-psihološko-didaktičko-metodičku naobrazbu stekao je tijekom akademske godine 2005./2006. na Visokoj učiteljskoj školi u Čakovcu. Zaposlen je na radnom mjestu izvanrednog profesora na Fakultetu informatike Sveučilišta Jurja Dobrile u Puli gdje je nositelj kolegija vezanih uz programiranje. Član je Povjerenstva za akademsko priznavanje inozemnih visokoškolskih kvalifikacija i razdoblja studija te Odbora za znanstveni i umjetnički rad Sveučilišta Jurja Dobile u Puli. Autor je 94 znanstvena rada objavljena u zbornicima međunarodnih konferencija, časopisima i knjigama te 16 stručnih radova objavljenih u zbornicima domaćih skupova. Znanstveno se i stručno usavršavao na brojnim radionicama i institucijama u zemlji i inozemstvu. Bio je voditelj projekta financiranog sredstvima HRZZ. Osim toga, sudjelovao je u istraživačkim aktivnostima projekata financiranih sredstvima Europske Unije, HRZZ, Zaklade Adris, Ministarstva znanosti i obrazovanja te Sveučilišta u Zagrebu. Bio je suorganizator i predavač na 2 radionice vezane uz upravljanje međunarodnim projektima i 7 radionica vezanih uz primjenu društvenih Web aplikacija u obrazovanju. Aktivni je recenzent za 18 međunarodnih znanstvenih časopisa, 2 znanstvene knjige i 13 međunarodnih znanstvenih konferencija. Vršio je dužnost člana Programskog odbora, voditelja tematskih cjelina i sekcija sljedećih međunarodnih znanstvenih konferencija: International Conference on Information Systems Development (ISD), ACM International Conference on Intelligent User Interfaces (IUI), International Conference on the Quality of Information and Communications Technology (QUATIC) - Track on Quality in Web Engineering, International Conference on Information Technology (ICIT), International Conference on Software and Information Engineering (ICSIE), International Conference on Network Technology (ICNT) te International Conference on Human and Social Analytics (HUSO). Bio je član Organizacijskog odbora, urednik knjige sažetaka i tehnički urednik zbornika radova međunarodne znanstvene konferencije ISD 2014. Član je sljedećih međunarodnih strukovnih udruženja: Association for Computing Machinery, European University Information Systems E-Learning Task Force, Institute of Electrical and Electronics Engineers i International Society for Web Engineering. Za svoj znanstveni, nastavni i stručni rad primio je nekoliko međunarodnih i domaćih nagrada i priznanja. Dosad je pod njegovim mentorstvom 105 studenata obranilo završne i diplomske radove.

Fakultet informatike
Sveučilište Jurja Dobrile u Puli
Rovinjska 14, 52100 Pula
fipu.unipu.hr

POSTOJI LI SAMO JEDNA "ISPRAVNA" ARHITEKTURA WEB POSLOVNIH APLIKACIJA

Zlatko Sirotić

SAŽETAK

Često postoji razlika u razmišljanjima između informatičara koji su radili poslovne aplikacije nad bazama podataka u klijent-server arhitekturi i informatičara koji su preskočili klijent-server arhitekturu i počeli raditi s web aplikacijama.

Prvi su obično (ali ne uvijek) skloni pisanju koda (i) u bazi, a drugi bazu podataka često (ali ne uvijek) gledaju kao "crnu kutiju", koja treba poslužiti samo za spremanje podataka, dok sva poslovna logika treba biti na strani aplikacijskog servera.

Ima i puno drugih pitanja. Autor je mišljenja da treba razmotriti konkretne potrebe. Ova će se pitanja razmatrati kroz Oracle Forms i ADF alate.

ABSTRACT

There is often a difference in thinking between IT professionals who have done business applications on databases in a client-server architecture and IT professionals who have skipped the client-server architecture and started working with web applications.

The former are usually (but not always) prone to writing code in the database, and the latter often (but not always) view the database as a "black box", which should only be used to store data, while all business logic should be on the side application server.

There are a lot of other issues as well. The author is of the opinion that specific needs should be considered. These issues will be addressed through the Oracle Forms and ADF tools.

1. UVOD

U ovom radu pokušat ćemo dati kratki odgovor na ova pitanja:

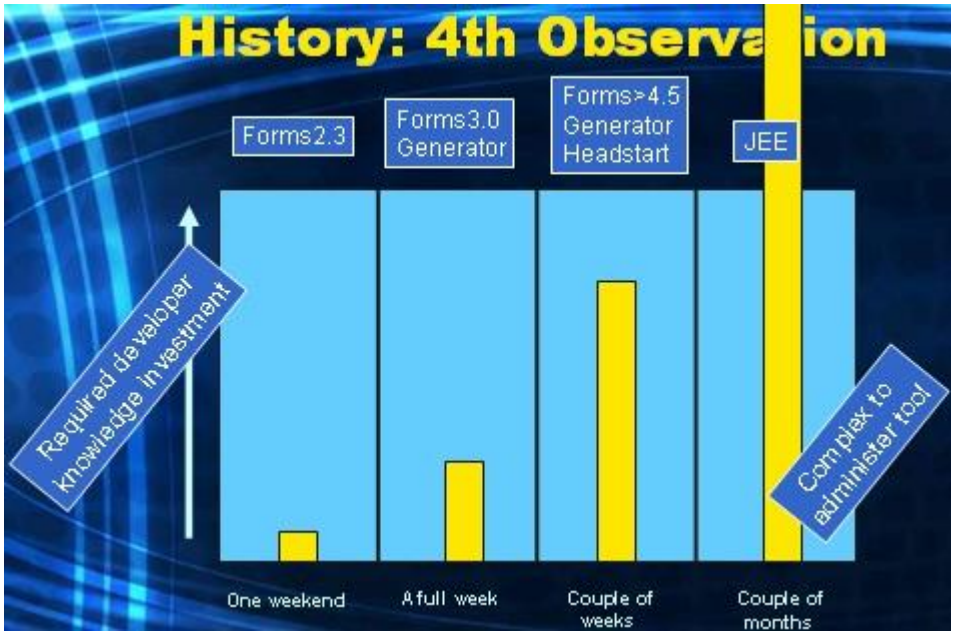
- Je li izrada web poslovnih aplikacija postala previše kompleksna?
- Treba li pisati programski kod za integritet podataka i kod za poslovnu logiku samo u bazi, samo na aplikacijskom serveru, samo na klijentu, ili nekom kombinacijom toga?
- Treba li u bazi uvijek imati samo jednog usera za pristup iz web aplikacije, ili je u redu "stari" način rada: jedna osoba = jedan user na bazi?
- Treba li konekcija između baze i aplikacijskog modula biti stalna ili privremena?
- Treba li veza između korisničke sesije u web pregledniku i aplikacijskog modula biti stalna ili privremena?

2. KAKO JE BILO NEKADA, NE TAKO DAVNO (PRIJE DVADESETAK GODINA)

Kako prikazuje slika 1. iz prezentacije Toon Koppelaars-a (slika 1.), mogućnosti DBMS sustava se stalno povećavaju (plavo). No, malo poslije 2000. godine, te se mogućnosti sve manje koriste (crveno), pa se sve više koda (koji bi mogao biti u bazi podataka) piše izvan baze podataka.

Slika 1. Mogućnosti DBMS sustava i njihovo korištenje kroz vrijeme; Izvor: [1]

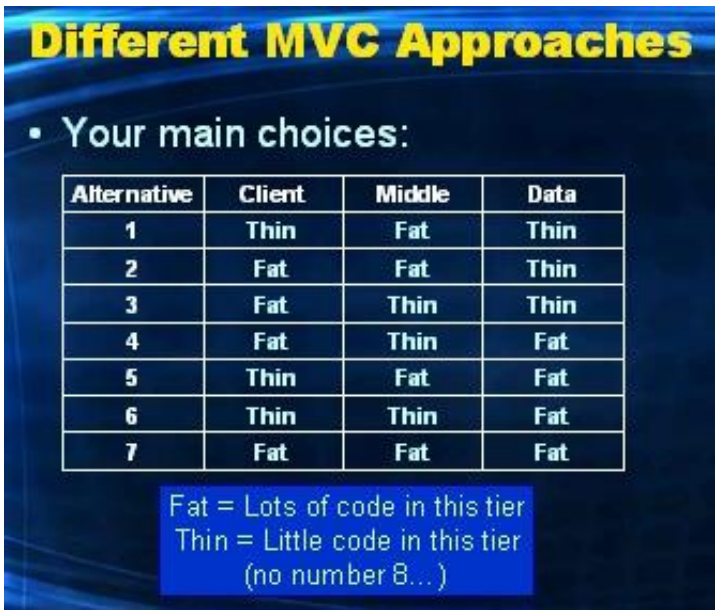
Prosječan developer nekada je mogao naučiti alat za izradu poslovnih aplikacija za par tjedana. Danas mu treba i par godina (slika 2.).



Slika 2. Alati za izradu poslovnih aplikacija su sve složeniji za učenje; Izvor: [1]

3. KAMO STAVITI ODREĐENI PROGRAMSKI KOD?

Postoje različiti pristupi pisanju "tankog" ili "debelog" koda u različitim arhitekturnim slojevima (slika 3.). Prva varijanta je najčešća. Osma varijanta (sve tanko) ne postoji – ipak treba nešto (i) programirati ☺.



Slika 3. Kamo staviti programski kod, i u kojim količinama; Izvor: [1]

Može se reći da poslovne aplikacije imaju tri vrste programskog koda:

- **Kod za rad s korisničkim sučeljem**.
- **Kod za poslovnu logiku**. On se može podijeliti na kod za čitanje podataka iz baze i kod za ažuriranje podataka u bazi. Kod za ažuriranje podataka u bazi vrlo često koristi (i) čitanje podataka iz baze.
- **Kod za osiguravanje integriteta podataka u bazi**. Često se ovaj kod brka s kodom za poslovnu logiku, pa nije čudno da se u praksi često isprepliću kod za poslovnu logiku i kod za osiguravanje integriteta podataka.

Programski kod za rad s korisničkim sučeljem (kod web aplikacija) može se nalaziti:

- **Na strani aplikacijskog servera**, što je najčešće. U JEE arhitekturi riječ je o Java servletima (ili nadogradnji servleta, kao štu su JSP, JSF i dr.).
- **Na strani klijenta**. Najčešće se danas takav kod piše u JavaScriptu, a rijetko u Javi kao Java applet (više i ne može).
- **Na strani baze**, što je dosta rijetko. Takvu arhitekturu ima npr. Oracle APEX alat, kod kojeg je HTML stranice dinamički generiraju pomoću APEX-ovih PL/SQL paketa na bazi

Programski kod za poslovnu logiku (kod web aplikacija) može se nalaziti:

- **Na strani aplikacijskog servera**, što je vrlo često. U JEE arhitekturi često je riječ o EJB-ovima.
- **Na strani baze**. Riječ je o tzv. pohranjenim (stored) procedurama / funkcijama i paketima na bazi. Vrlo često se na strani baze sprema programski kod za tzv. batch obradu. Činjenica je da se najčešće najbrže izvršava upravo kod (za poslovnu logiku) koji se nalazi na bazi.
- **Na strani klijenta** - gotovo nikad, čak niti kad se na klijent strani nalaze Java appleti.

Programski kod za osiguravanje integriteta podataka u bazi (kod web aplikacija) može se nalaziti:

- **Na strani baze**. Najčešće se koriste deklarativna integritetna ograničenja baze, kao što su integritetna ograničenja za jedinstveni ključ (UK), vanjski ključ (FK) i check constraint (CK). Iako je SQL standard još od 1992. uključio CREATE ASSERTION naredbu, praktički niti jedan SQL DBMS sustav ju ne podržava (nedavno su se pojavili ne-SQL relacijski DBMS sustavi koji ju podržavaju). Zbog toga, kada se programski kod za osiguravanje integriteta podataka želi u cijelosti pisati na strani baze, mora se pribjeći korištenju okidača baze (proceduralni pristup).
- **Na strani aplikacijskog servera**. Ovo je vrlo čest pristup u praksi. Želja da se izbjegne (dosta mukotrpna) realizacija integriteta podataka pomoću okidača baze često se navodi kao dovoljan razlog za ovakav izbor. No, time se omogućava da baza bude nezaštićena (u smislu integriteta, ne u smislu sigurnosti podataka općenito). Naime, jedna aplikacija može savršeno čuvati integritet podataka u bazi, dok, nažalost, druga aplikacija može biti tako pisana da to ne osigurava.
- **Na strani klijenta**. Vrlo često se tako rade samo jednostavnije provjere integriteta podataka, koje su istovremeno realizirane i na strani aplikacijskog servera ili/i baze.

4. KOLIKO USERA IMATI NA BAZI?

U klasičnim klijent-server aplikacijama, obično se radilo tako da je svaki korisnik (osoba) imao svoj vlastiti pristup u bazu, tj. svoj vlastiti korisnički račun (user, shemu) na bazi. Broj korisnika (osoba) obično je bio reda par stotina (ili manje), i svi korisnici su bili poznati (neanonimni).

Vrlo često web aplikacije rade s puno većim brojem korisnika (osoba), koji su često i anonimni. Tada izgleda logično da se napušta nekadašnji pristup jedan korisnik (osoba) = jedan korisnički račun (user) na bazi. Najčešće se uz aplikacijsku shemu (kojih može biti i više, npr. jedna za tablice, a druga za pakete na bazi) radi samo jedan korisnički račun (user). **No, time se neminovno smanjuje sigurnost baze.**

Nije loše koristiti srednji pristup, kod kojeg se odrede različite vrste korisnika (osoba), a onda se za svaku vrstu korisnika napravi poseban korisnički račun (korisnička shema) na bazi. Na taj način, ako netko provali u bazu kroz korisnički račun koji ima manja prava, ne može raditi ono što bi mogao da je provalio kroz korisnički račun sa većim pravima.

No, ponekad se i kod web aplikacija može primijeniti "stari pristup", ako su svi korisnici poznati (neanonimni) i ako ih nema više od nekoliko stotina. Tada se kao prepreka pojavljuje činjenica da se kod web aplikacija često koristi pool konekcija na bazu (**connection pool**), pa se ne želi da svaki korisnik ima svoj poseban pool. No, tome se može doskočiti primjenom tzv. **proxy korisničkog računa**.

5. ORACLE FORMS I ORACLE ADF ALATI

Oracle Forms je Rapid Application Development (RAD) alat, koji je Oracle napravio početkom 80-ih, nedugo nakon nastanka Oracle baze verzije 2 (verzija 1 nije nikada postojala), i radio je kao znakovno orijentirana (character mode) aplikacija na serveru.

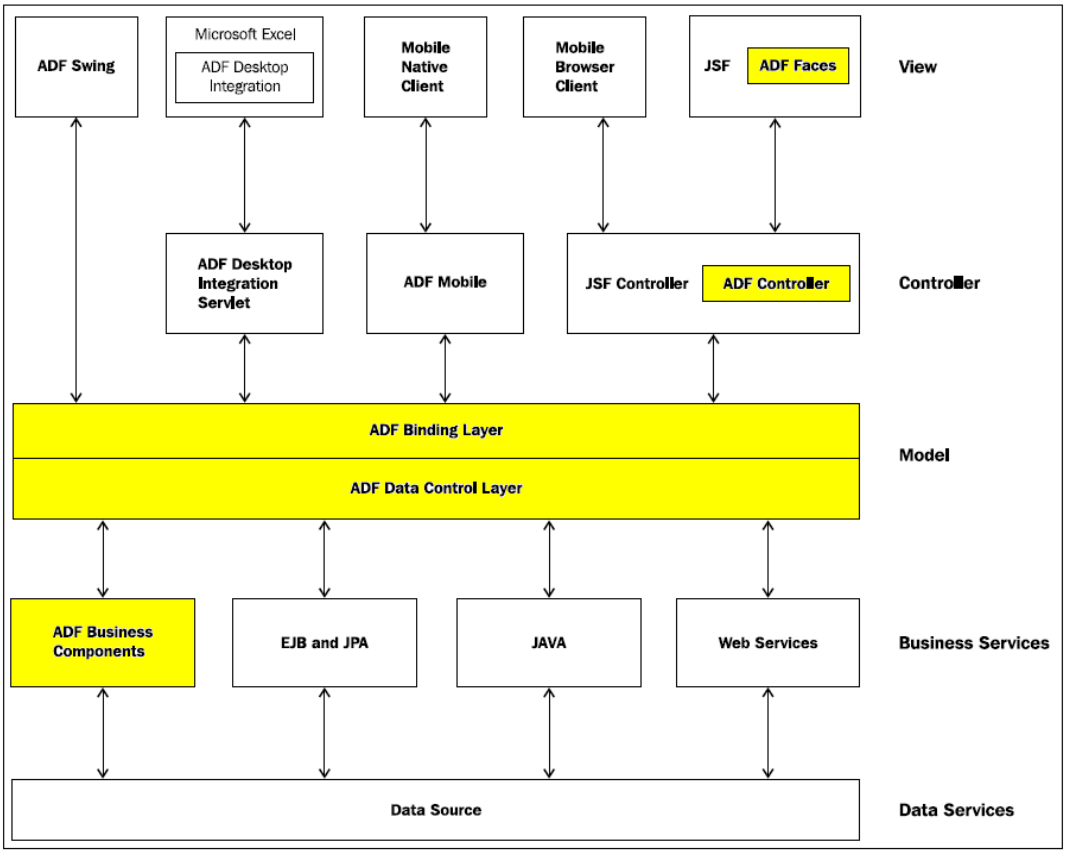
Sredinom 90-ih napravljena je klijent-server GUI varijanta. Klijent-server varijanta pratila je baze 6, 7 i 8, a Forms verzije bile su 4, 4.5, 5, 6 i 6i. U Forms verziji 6 pojavila se paralelno i web varijanta - Web Forms. Nakon verzije 6i, klijent-server varijanta više ne postoji, tj. verzije od 9i do 12.2 (verzije 7 i 8 nikad nisu postojale)rade isključivo kao web Forms aplikacija (koja nije baš jeftina).

Web varijanta Forms alata radi tako da Java applet (ili samostalni Java kod) na klijentu (koji se brine samo za kreiranje korisničkog sučelja) surađuje sa Forms servisom na aplikacijskom serveru. Forms servis manje-više čini onaj isti kod (pisan u C-u) kao i u klijent-server varijanti. **I u ovoj varijanti Forms servis drži stalnu konekciju s bazom, a korisnik stalno drži Forms module s kojima radi,sve dok ne završi rad.** U suštini, ovaj način rada nema nekih značajnih razlika u odnosu na klijent-server rad.

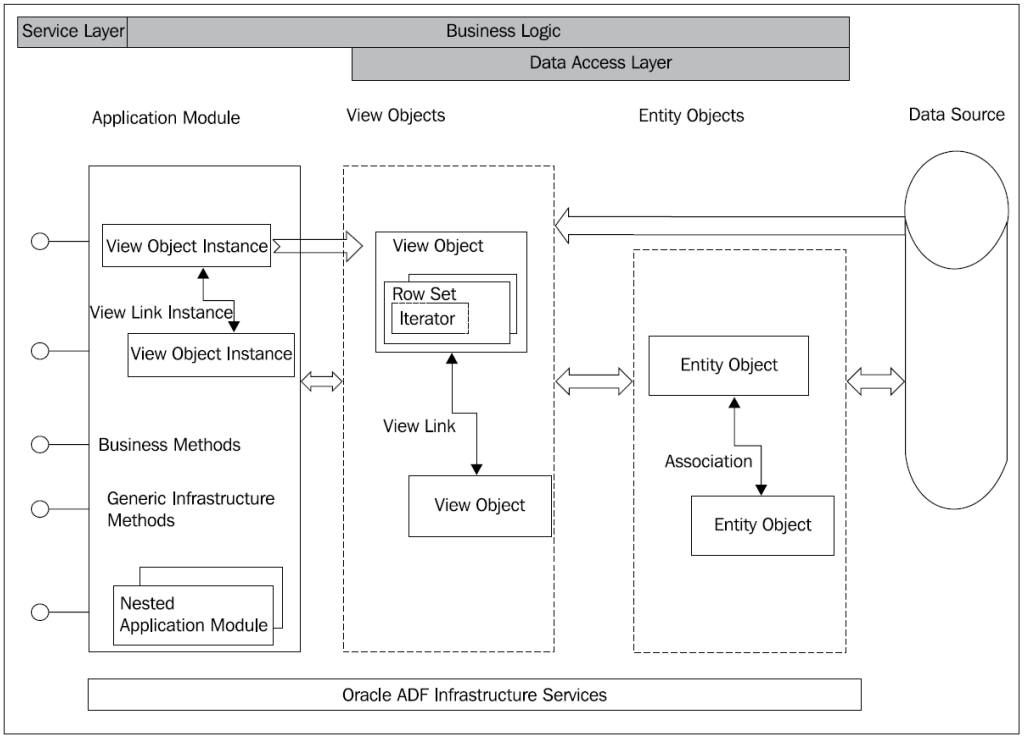
Šest mjeseci nakon što je Sun izdao verziju Jave 1.0, u Oracleu su odlučili raditi RAD alat temeljen na jeziku Java. Prvo izdanje frameworka, koji se tada nije zvao ADF već JBO (Java Business Objects), uslijedilo je 1999. godine. Ubrzo mu je ime promijenjeno u BC4J (Business Components for Java). BC4J je pokrivao onaj dio koji danas pokriva ADF BC. Uz BC4J, Oracle je počeo razvijati i odgovarajući IDE JDeveloper, licencirajući 1998. tadašnji Borlandov alat JBuilder. Oracle je 2001. temeljito preradio JDeveloper, pri čemu ga je u potpunosti "prepisao" u Java kod. Ubrzo je termin BC4J zamijenjen sa ADF. Od 2005. godine Oracle JDeveloper IDE je besplatan. **Verzija ADF Essentials je besplatna od ljeta 2012.**

Na slici 4. žutom bojom označena je najčešća varijanta korištenja dijelova ADF alata.

Na slici 5. prikazana je struktura ADF aplikacijskog modula (AM), koji je najvažniji dio ADF Business Components. AM predstavlja vezu između izvora podataka (najčešće DBMS) i korisničkog sučelja.



Slika 4. Struktura ADF frameworka; Izvor: [4]



Slika 5. Oracle ADF Application Module (AM); Izvor: [4]

6. ORACLE ADF APPLICATION MODULE POOL I CONNECTION POOL

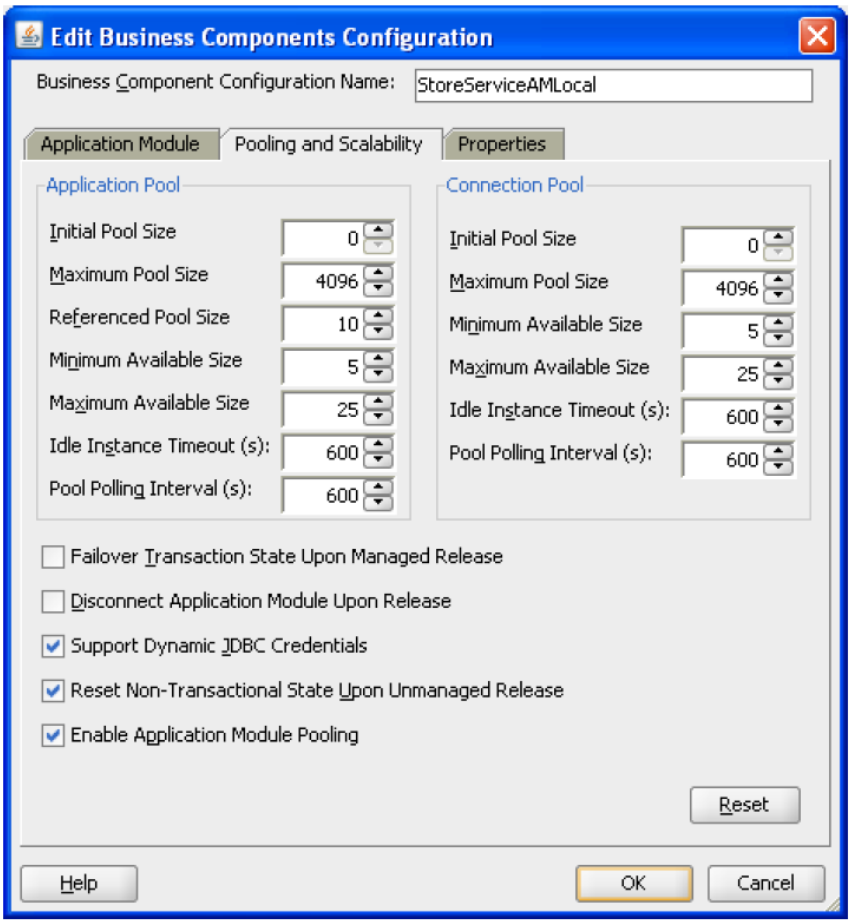
ADF ima (uobičajeni) connection pool, ali i Application Module (AM) pool.

Postoje dvije vrste connection poolova, koje se koriste u ovisnosti o tome konfiguriraju li se konekcije kao **JDBC URL konekcije**, ili **JNDI name for a data source konekcije**. Ako se koriste JDBC URL konekcije, samo tada se koristi ADF connection pool, a inače se koristi connection pool AS-a. Osnovno pravilo za ADF connection pool je: po jedan connection pool (dakle, skup konekcija, a ne jedna konekcija) se kreira za svaki par <JDBCURL, Username> na svakom JVM-u, pri čemu konekciju dobiva samo root AM instanca (ugniježđene AM instance koriste tu istu konekciju).

AM pool je kolekcija AM instanci iste vrste. AM pool omogućava da veći broj korisnika može (kvazi) istovremeno raditi na manjem broju AM instanci. AM instanca u poolu može biti u jednom od tri stanja:

- **bezuvjetno slobodna** za korištenje bilo kom korisniku;
  slobodna za korištenje, ali referencirana na aplikacijsku sesiju koja ju je prethodno koristila i koja još nije završila; u ovom slučaju AM instanca može se ipak predati drugom korisniku, ovisno o tzv. AM State Management Release Levelu, **pri čemu će standardno doći do tzv. pasivizacije (a kasnije aktivacije)**
- **AM instance;**
- **zauzeta**, kad neki korisnik (odnosno, njegova Java dretva
  na AS-u) trenutačno koristi tu AM instancu.

Veza između AM instance i konekcije (iz connection poola) je po defaultu stalna, ali se može postaviti da nije.



Slika 6. Application Module pool i Connection pool parametri; Izvor: [4]

Kod vraćanja AM instance u AM pool, postoje tri varijante otpuštanja (release levels):

- **Managed** (default): AM pool preferira zadržati istu AM instancu za istog korisnika, ako je to moguće; **ako nije moguće, radi se pasivizacija AM instance (podaci se smještaju u bazu, rjeđe u datoteku)**, a kasnije se radi aktivacija (druge) AM instance;
- **Unmanaged**: nikakvo stanje se ne pamti;
- **Reserved**: veza 1 : 1 između AM instance i korisnika; podsjeća na Forms način rada; **uglavnom nije preporučljiva za web aplikacije, iako je najjednostavnija!**

Za kraj, u tablici 1. prikazujemo primjere veza: klijent - aplikacijski modul - baza podataka.

Tablica 1. Primjeri veza: klijent - aplikacijski modul - baza podataka.

Alat i varijanta	Veza između klijenta i aplikacijskog modula	Veza između aplikacijskog modula i baze podataka	Broj potrebnih aplikacijskih modula i konekcija na bazu za 1000 klijenata
Oracle Forms	stalna	stalna	1000 modula 1000 konekcija na bazu
Oracle ADF varijanta 1	Reserved (stalna)	stalna - default	1000 modula 1000 konekcija na bazu
Oracle ADF varijanta 2	Managed - default (relativno stalna)	stalna - default	npr. 100 do 1000 modula npr. 100 do 1000 konekcija na bazu
Oracle ADF varijanta 3	Managed - default (relativno stalna)	nestalna	npr. 100 do 1000 modula npr. 10 konekcija na bazu
Oracle ADF varijanta 4	Unmanaged (nestalna)	nestalna	npr. 100 modula npr. 10 konekcija na bazu

7. ZAKLJUČAK

Izrada web poslovnih aplikacija postala je jako kompleksna. No ponekad je moguće nešto pojednostaviti. Npr. nije isto radimo li web aplikaciju za manji broj poznatih korisnika, ili ogroman broj nepoznatih korisnika. Barem ponekad, dobro je pisati programski kod za integritet podataka i kod za poslovnu logiku (uglavnom) u bazi, a ne (uglavnom) na aplikacijskom serveru (što je uobičajeno). Zbog (ne)sigurnosti, nije dobro u bazi uvijek imati samo jednog usera za pristup iz web aplikacije.Dobro je uvesti više razina usera, pa čak i "stari" način rada jedna osoba = jedan user na bazi (uz upotrebu proxy usera).

Nažalost, često developeri gledaju na DBMS sustav kao na "crnu kutiju". Nemaju vremena za dublje upoznavanje s mogućnostima konkretnog DBMS-a, drže da su svi DBMS-ovi vrlo slični, žele pisati generički kod (neovisan o DBMS-u) itd.

Literatura:

1 Koppelaars T. (2009): The Helsinki Declaration (IT-version), blog 2009–2017, http://thehelsinkideclaration.blogspot.com/2009/03/helsinki-declaration-observation-1.html (veljača 2020.)

2 Kyte, T. (2009): Expert Oracle Database Architecture, Apress

3 Oracle priručnik (2012): Forms Services Deployment Guide 11g Release 2, E24477-03

4 Oracle priručnik (2013):Developing Fusion Web Applications with Oracle Application Development Framework 12c, E23132-01

5 Purushothaman, J., (2012): Oracle ADF Real World Developer's Guide, Packt Publishing

Podaci o autoru:

Zlatko Sirotić, univ.spec.inf.
ISTRA TECH d.o.o., Pula
e-mail: zlatko.sirotic@istratech.hr

Autor radi oko 35 godina na informatičkim poslovima, uglavnom u poduzeću ISTRA TECH d.o.o., Pula (ISTRA TECH je novo ime poduzeća Istra informatički inženjering, osnovanog prije 28 godina). Oracle softverske alate (baza, Designer CASE, Forms 4GL, Reports, JDeveloper IDE, Java) koristi više od 20 godina. Objavljivao je stručne radove na kongresima / konferencijama CASE, KOM, HrOUG, JavaCro, "Hotelska kuća", u časopisima "Mreža", "InfoTrend" i "Ugostiteljstvo i turizam", a neka njegova programska rješenja objavljivana su na web stranicama firmi Oracle i Quest (danas dio firme Dell).
Na Fakultetu informatike u Puli sudjeluje (od početka osnivanja, 2011.) kao vanjski suradnik, uglavnom na kolegijima Baze podataka 2 i Informatički praktikum 1.