ZBORNIK RADOVA SA KONFERENCIJA

KOM 2020

ELEKTRONIČKE KOMUNIKACIJSKE TEHNOLOGIJE I NORME

CASE 2021

RAZVOJ POSLOVNIH I INFORMATIČKIH SUSTAVA

2020 / 2021

KOM 2020

ELEKTRONIČKE KOMUNIKACIJSKE TEHNOLOGIJE I NORME

23.11. - 24.11.2020

Zlatni partner

citus grupa
Internet Development Company

Brončani partneri

HAKOM

CISCO

IEEE
Croatia Section

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, 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

MOBILNE TEHNOLOGIJE 5G I NJIHOV UTJECAJ NA LJUDSKO ZDRAVLJE

5G MOBILE TECHNOLOGIES AND THEIR IMPACT ON HUMAN HEALTH

dr.sc. Winton Afrić

SAŽETAK:

U uvodnom dijelu rada u kratkim crtama govori se o povijest razvoja mobilnih komunikacijskih sustava i o arhitekturi 5G mreža. U nastavku rada prikazani su Frekvencijski opsezi koji se predviđaju za rad 5G mreža na području EU. Način rada u pojedinim frekvencijskim opsezima, te o trenutnoj zauzetost frekvencijskog spektra namijenjenog za rad 5G mreža sa postojećim tehničkim sustavima. Nadalje se vrši usporedba izračenih snaga kod sustava koji danas koriste dijelovi frekvencijskog spektra namijenjenog za budući rad 5 G mreža, sa snagama u mobilnim mrežama. U trećem djelu rada govori se o Europskoj regulativi za zaštitu od ne ionizirajućeg zračenja, te o mehanizmima djelovanja elektromagnetskog zračenja radijskih frekvencija na ljudski organizam. Na kraju slijedi zaključak sa preporukama o ponašanju koje nas štiti od negativnog utjecaja EMV u našem okolišu.

ABSTRACT:

The introductory part of the paper briefly discusses the history of the development of mobile communication systems and the architecture of 5G networks. In the following of the paper will be presents the frequency bands that envisaged for the operation of 5G networks in the EU. The mode of operation of 5G networks in certain frequency bands, and the current occupancy of the frequency spectrum intended for the operation of 5G networks with other existing technical systems. Furthermore, the radiated power of the systems used today by parts of the frequency spectrum intended for the future operation of 5 G networks compared with the launched power in mobile networks. The third part of the paper discusses the European regulations for protection against non-ionizing radiation, and the mechanisms of action of electromagnetic radiation of radio frequencies on the human body. Finally, a conclusion follows with recommendations on behavior that protects us from the negative impact of EMV in our environment.

1. UVOD

Danas smo svjedoci vrlo snažnog razvoja svih komunikacijskih sustava, pa tako i mobilnih komunikacija. Multimedijsko komuniciranje uvodi nas u doba informatičkog društva i ono čini jedan od značajnih čimbenika daljnjeg gospodarskog rasta i razvoja. Suvremeno poslovanje nije moguće ako ne postoji širokopojasna komunikacijska prospojenost. Mobilne komunikacije čine značajni udio u ukupnim komunikacijama. Povijest mobilnih komunikacija nije duga, to je vremenski period od svega pola stoljeća, ali vremenski period koji je obilježen burnim tehnološkim razvojem i transformacijom mobilnih komunikacija od ekskluzivne usluge namijenjene poslovnim korisnicima do masovne usluge namijenjene svima. Čitav razvoj mobilnih komunikacijskih sustava uz svu dobrobit koju donosi ljudima i ljudskom društvu praćen je i sa čitavim nizom strahova od mogućih negativnih utjecaja na ljudsko zdravlje. Kako raste gustoća baznih stanica, kako se pojavljuju nove generacije mobilnih komunikacijskih sustava, kako se sve više zaposjedaju novi dijelovi frekvencijskog spektra ili se prenamijenjenu već korišteni dijelovi spektra sve više kod opće populacije produbljuje se strah od mogućih potencijalnih negativnih utjecaja na ljudsko zdravlje.

Suvremena komunikacijska sredstva, ne samo mobilna, imaju snažan utjecaj na transformaciju ljudskog društva u svim pa i u socijalnim segmentima. Često se tvrdi da se ljudi danas manje druže. Međutim, druženje je promijenilo svoj oblik i formu, često se i sve više družimo sa ljudima koji nam nisu u fizičkoj blizini, ali s kojima dijelimo iste interese i poglede. Ovo druženje odvija se preko čitavog niza socijalnih mreža i sve više postaje druženje na globalnoj razini. Gotovo svaka informacija postaje nam dohvatljiva, a mogućnosti učenja u multimedijskom okruženju čine nas sve obrazovanijima.

Kada govorimo o negativnom utjecaju novih komunikacijskih tehnologija na ljudsko zdravlje mogli bi smo te utjecaje podijeliti na psihološke i fiziološke. Pod psihološkim negativnim utjecajima sigurno spada i razvoj psihološke ovisnosti o komunikacijskim sredstvima koji može ići do patološke razine. Dobar broj pripadnika suvremene populacije doslovno spava sa mobitelima, tabletama i drugim komunikacijskim sredstvima. Ekran sve više postaje nešto što se prije spavanja posljednje gasi i prije ustajanja prvo pali.

Međutim ovaj članak se ne bavi negativnim aspektima korištenja mobilnih telefona koji se temelje na psihološkim aspektima različitih stupnjeva ovisnosti. U ovom članku se bavimo sa mogućim negativnim utjecajima mobilnih komunikacija sa fiziološkog stanovišta. Interakcija našeg tijela sa mobilnim komunikacijskim sustavima odvija se preko elektromagnetskih valova koje ovi sustavi koriste u svom radu. Postoje i prirodni izvori elektromagnetskih valova, jer je svaka zvijezda u svemiru izvor čitavog spektra elektromagnetskih valova. Prirodni izvori EMV mogu biti i na zemlji, to su oblaci, gromovi i slično. Kakav je spektar kod zvijezda ovisi o veličini i starosti zvijezda. Dakle,

elektromagnetski valovi su nešto što postoji i u našem prirodnom okruženju. Atmosfera nas štiti od EMV izuzetno visokih frekvencija jer dijelom apsorbira i reflektira njihovu energiju.

Međutim, pojava komunikacijskih sredstava koja u svom radu koriste EMV iz domena radijskih frekvencija zasigurno donekle mijenja naše prirodno okruženje. Odatle proizlazi i stah da bi ta promjena u okruženju mogla imati negativne utjecaje na ljudsko zdravlje jer se ne radi o sredini u kojoj je evolucijski razvijeno ljudsko tijelo, a ni drugi organizmi iz našeg okoliša. Dakle, brzim mijenjanjem našeg okoliša stvaramo ambijent na koji evolucijski nismo prilagođeni i to bi onda moglo imati veći ili manji negativan utjecaj na ljudsko zdravlje.

Elektromagnetski valovi u komunikacijske i radio lokacijske svrhe se koriste već nešto malo više od stotinjak godina, od slanja televizijskih i radijskih signala, preko različitih vrsta radarskih uređaja. Upotrebljivi dio radio frekvencijskog spektra za radijske komunikacije poglavito danas nije slobodan, već je dugo zauzet sa nekim drugim možda već i zastarjelim sustavima. Kod uvođenja novih generacija mobilne telefonije danas najčešće govorimo o prenamjeni dijelova RF spektra. Dakle, u nekim dijelovima RF spektar prije smo koristili jedne sustave koje gasimo i onda taj dio spektra prenamjenjujemo u neku novu svrhu i za neke nove sustave. Laički rečeno ne radi se o pojavi novih frekvencija u našem okolišu nego o promjeni namjene postojećih radio frekvencija. Druga namjena znači i druge tehnologije modulacije ali i druge razine snaga. Ako su izlazne snage nekih starih sustava koji se napuštaju bile veće ili znatno veće od onih koje će koristiti sustavi mobilnog komuniciranja, onda ne možemo govoriti o dodatnom zagađenju našeg okoliša sa EMV nego smanjenju postojeće razine zagađenja. Dakle, ako je postojao negativni utjecaj sa novim sustavima nižih razina emitiranih snaga taj utjecaj će biti manji.

Danas je aktuelno postavljanje sustava mobilnih komunikacija 5G. [1] U ovome članku analiziraju se radio frekvencijski opsezi koji su predviđeni za rad 5G sustava u Europi. Napravit ćemo usporedbu između izlaznih snaga koji sadašnji sustavi koriste, sa izlaznim snagama koje će koristiti sustavi 5G, kako bi vidjeli da li će s pojavom 5G doći do povećanja ili smanjenja energije pojedinih EMV u našem okolišu. Zatim ćemo opisati istraživanja koja se sprovode kako bi se ustvrdili negativni utjecaji EMV na ljudsko zdravlje. Dakle, što se zna i što je mjerljivo, a što se ne zna pa zbog toga nije ni mjerljivo o utjecaju EMV na ljudsko zdravlje. Nadalje se opisuju postojeće norme za zaštitu ljudskog zdravlja od EMV i na kojim spoznajama se te norme temelje.

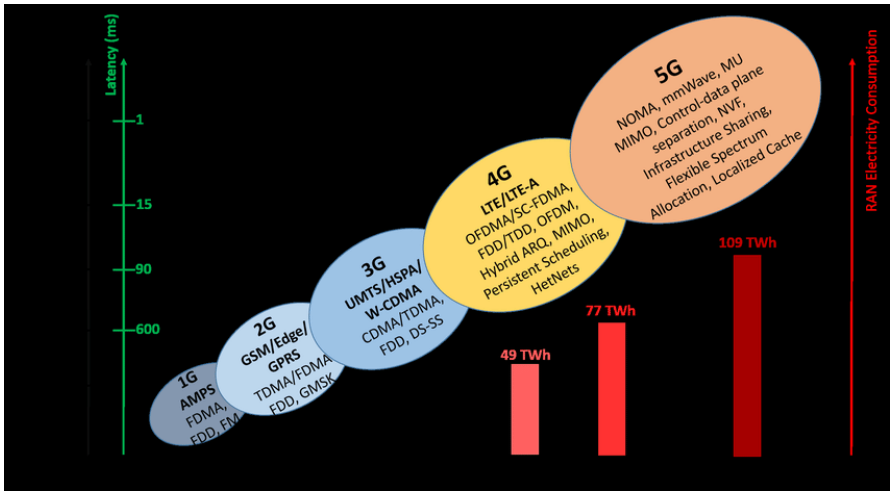
2. KRATKA POVIJEST RAZVOJA MOBILNIH KOMUNIKACIJSKIH SUSTAVA

Ukupna povijest razvoja mobilnih komunikacijskih sustava može se podijeliti u pet razvojnih generacija.

- 1G Analogni sustavi za mobilni telefon koji podržavaju kanalsko spajanje.
- 2G digitalni sustavi mobilnih telefona koji također podržavaju kanalsko spajanje čak i onda kada se radi o prijenosu paketske informacije (GSM 9,6 Kbit/s).
- 3G digitalni sustavi mobilnih telefona uvode uz kanalsko i paketsko prospajanje.
- 4G Digitalni sustavi mobilnih telefona osnivaju se isključivo na paketskom prospajanju. [2]
- 5G Digitalni sustavi mobilnih telefona nastoje; znatno povećati ukupni kapacitet prijenosa informacija, objediniti sve komunikacijske potrebe, te široko podržati IoT.

Slika 1 Kratka povijest razvoja mobilnih komunikacijskih sustava

Kroz generacije sustava mobilnog komuniciranja dolazi do ubrzanog rasta raspoloživog informacijskog volumena, smanjenja kašnjenja i optimizacije u korištenju energije (baterija).



Slika 2. Kratka povijest razvoja mobilnih komunikacijskih sustava

3. FREKVENCIJSKI OPSEZI NAMIJENJENI RADU 5G MREŽA U EU

Za Europu se očekuje da će 5G sustav raditi u frekvencijskim opsezima:

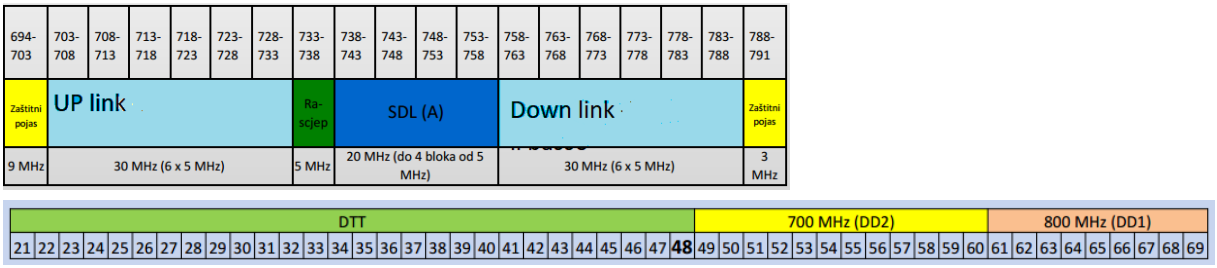
- Frekvencijski opseg oko 700 MHz
- Frekvencijski raspon oko 3,5 GHz
- Frekvencijski raspon iznad 26 GHz

Tabela 1: Frekventni opsezi predviđeni za rad 5G sustava

Frekvencijsko područje namijenjeno za 5G mobilne sustave	700 MHz 694 – 791 MHZ	3,4 -3,6 MHz i 3,6 -3,8 MHz	26 GHz
Što danas radi u tim frekvencijskim opsezima?	DVBT1	WiMAX RR HEP	RR Policijski radari od 24,050 MHz do 24,250 MHz za K-opseg i od 33.400 MHz do 36,000 MHz za Ka opseg.

3.1. Karakteristike RF spektra u pojasu od 700 MHz

Rad mobilnih komunikacijskih sustava 5G predviđen je među ostalim i u 700 MHz području. Međutim, u ovom **području u diljem Europe radi emitiranje signala digitalne televizije DVBT1**. Oslobađanje ovog područja i njegova prenamjena za mobilne komunikacijske sustave uvjetovana je prelaskom na emitiranje **digitalnih televizijskih programa po standardu DVBT2**. Ovaj prijelaz u Republici Hrvatskoj bio je predviđen za početak ljeta 2020., međutim zbog COVID 19 prelazak je odgođen za kraj listopada ove godine.



Slika 3. Oslobađanje dijela spektra koji se je koristio za emitiranje televizijskih programa i njegova prenamjena za mobilne komunikacijske sustave.

Područje frekvencija oko 700 MHZ povoljno je za mobilne komunikacijske sustave jer je dobar ogib elektromagnetskih valova. Navedeno omogućava dostatnu razinu signala u područjima „sjene“. Što je frekvencija EMV niža to je bolji ogib EMV, preko 11 GHz predajna antena mora imati izravnu optičku vidljivost sa prijemnom antenom [*LOS – line of sight*].

Područje frekvencija oko 700 MHz danas se koristi za emitiranje digitalnih televizijskih signala, a prosječna snaga DVBT1 odašiljača varira od 250W do 2KW. Nakon što dođe do prenamjene RF područja izlazna snaga 5G baznih stanica bit će maksimalno oko 23 dBm, što odgovara 0,2 W. Razina emitirane snage smanjit će se za 1250 puta u odnosu na DVBT odašiljače.

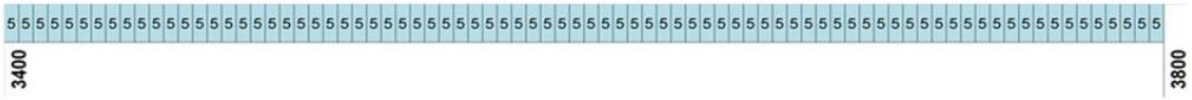
Prijemna snaga u mobilnoj telefoniji kreće se od -50 dBm (10 nW) do -68 dBm (0,15 nW). (1 nW milijarditi dio 1 W.)

Kad se DVBT-1 napusti i prebaci na 5G, na 700 MHz zagađenje elektromagnetskim valovima bit će manje nego što je trenutno.

3.2. Karakteristike RF spektra u pojasu od 3,4 – 3,8 MHz

Trenutačno se većina testnih emitiranja za 5G sustave u Republici Hrvatskoj odvija na frekvencijama oko 3,5 GHz. U ovom frekvencijskom području danas u Republici Hrvatskoj u dvije Županje rade WiMAX sustavi (sustavi za fiksni bežični pristup širokopojasnoj mreži, standardi IEEE 802.16). Međutim u skoro vrijeme te koncesije istječu. Također ovo područje frekvencija danas se koristi i za usmjerene radijske veze (poglavito kada se radi o većim udaljenostima). Međutim, kada govorimo o prenamjeni ovog područja za 5G sustave, onda je važno napomenuti da se u ovom RF području predviđa rad u vremenskom dupleksu TDD (Time Duplex), i to rad s radijskim kanalima širine 5 MHz.

Predviđena radijska kanalizacija prikazana je na slici 4.



Slika 4. Predviđena radijska kanalizacija za rad 5G sustava u području od 3,4 do 3,8 GHz

TDD način rada omogućava manipuliranje sa količinom informacijskog volumena za silazni i uzlazni smjer. Međutim, ovaj način rada zbog kašnjenja prostiranja ne dozvoljava područje pokrivanja koje je veliko (najviše do nekoliko 100 m promjera ćelije - sustav mikroćelija).

Malo područje pokrivanja zahtijeva maksimalnu izlaznu snagu od 12 do 8 dBm (16 - 6,3 mW).

Na frekvencijama od 3,4 do 3,8 GHz postoji signal u sjeni ,ali nije tako snažan kao na nižim frekvencijama što za mobilni pristup može predstavljati značajniji problem ako je ćelija relativno velika.

Dakle u našem okruženju već postoje EMV iz ovog dijela RF spektra, a prenamjenom neće doći do povećanja izlazne snage kod 5G baznih postaja u odnosu na danas postojeće sustave, već suprotno na blago smanjenje izlaznih snaga.

3.3. Karakteristike RF spektra u pojasu iznad 26 GHz

Područje od 26 pa na više GHz danas se u radio komunikacijama koristi isključivo za sustave usmjerenih radijskih veta na kraćim udaljenostima od svega nekoliko kilometara. Ovo frekvencijsko područje osim slabljenja uslijed prostiranja elektromagnetskih valova izloženo je i slabljenu koje je uvjetovano interakcijom čestica u atmosferi sa EMV (valna duljina je manja od 12 mm). Na frekvencijama od 24 GHz predajna antena i prijemna antena moraju ostvariti izravnu optičku vidljivost jer nema signala u sjeni (i najmanja prepreka sprječava širenje EMV). Zbog svih ovih razloga ovo područje nije podesno za komunikaciju na većim udaljenostima.

Međutim, danas policijski radari za kontrolu prometa među ostalim koriste i frekvencije oko 26 GHz.

Učestalosti signala koji emitiraju policijski radari unutar je opsega:

- od 10.500 MHz do 10.550 MHz za X opseg,
- **od 24.050 MHz do 24.250 MHz za K-opseg i**
- od 33.400 MHz do 36.000 MHz za Ka opseg.

Tolerancija frekvencije za X opseg je ± 25 MHz, a za K i Ka opsege ± 100 MHz.

Policijski radari koriste se za mjerenje brzine vozila na udaljenostima do 300 m, radi toga nemaju visoke izlazne snage kao radari namijenjeni za dublje promatranje prostora. Njihova izlazna snaga iznosi nekoliko Wata što u dBm iznosi od 30 do maksimalno 38 dBm-a.

Ako bi se frekvencije oko 26 GHz koristile za mobilno komuniciranje, tada bi ćelija bila ograničena na vrlo mali prostor od svega nekoliko desetaka metara, u okviru jedne prostorije ili otvorenog prostora bez prepreka. U takovom slučaju izlazne snage bi se kretale u redu od nekoliko mW (od 0 do 8 dBm).

U svim područjima radijskih frekvencija koje su predviđene za rad 5G sustava ili u području bliskih frekvencija, već danas postoje elektronički uređaji koji u svom radu koriste EMV sa većim ili bliskim razinama snaga koje će se koristiti u radio komunikaciji 5G sustava. Dakle ne možemo govoriti o povećanju zagađenja EMV u našem okolišu koje će uslijediti zbog korištenja 5G tehnologija mobilnih komunikacijskih uređaja.

4. UTJECAJ EMV NA LJUDSKO ZDRAVLJE

Kada govorimo o utjecaju EMV na ljudsko zdravlje onda efekti koje ovi valovi izazivaju ovise o:

- frekvenciji koja se koristi,
- snazi izvora EMV,
- dužini trajanja izloženosti našeg tijela izboru EMV.

4.1. Frekvencije EMV koje se koriste

Energija kvanta EMV može se izračunati iz produkta frekvencije i Planck-ove konstante (h= 6,62607015 10^{-34} Js ≈ 4,13566743 10^{-15} eVs). Dakle linearno kako raste frekvencija raste i energija kvanta EMV. Energija kvanta koja može promijeniti našu DNK molekulu iznosi oko 1 eV i nalazi se na području frekvencija koje poznajemo pod nazivom UV zračenje. Sve frekvencije koje spadaju u opseg radijskog zračenja spadaju u ne ionizirajuća zračenja. Na primjer za frekvencije oko 1GHz energija kvanta iznosi cca 3,7 μeV, a za frekvencije neposredno iznad 24 GHz 90 μeV.

4.2. Snaga zračenja EMV

Jedan od efekata koji mogu prouzrokovati EMV je zagrijavanje materijala koji su im izloženi pa tako i zagrijavanje ljudskog tijela. Svako zagrijavanje ljudskog tijela koje prelazi 1 ^{0}C izaziva štetne efekte po ljudsko zdravlje. Koliko je štetno EMZ iz mobilnih telefona ili baznih postaja najbolje je da procijenimo iz jednog primjera.

Mikrovalna pećnica koja se nalazi u veliko broju naših domaćinstava radi na frekvencijama od 2,430 GHz. Mobilni telefoni 3G (UMTS) ili 4 G (UMTS) jedno od područja svojih rada imaju UL 2,5 -2,57 GHz i DL 2,62 2,69 GHz. Dakle radi se o vrlo bliskom frekvencijskom području. Zašto sa mobilnim telefonom ne možemo podgrijavati hranu a sa mikrovalnom pećnicom možemo ? Zato jer je znatna razlika u snazi emisije.

Mikrovalna pećnica konzumira od 300 do 800 W (ovisno o tipu i proizvođaču).

Mobilni telefon emitira snagu od 1 do 16 mW (od 0 do 12 dBm)

Dakle, mobilni telefon emitira EMV koji su slabiji od valova iz mikrovalne pećnice za od 50000 do 300000 puta. Najinteresantnije je da se mobilnog telefona bojimo a mikrovalne pećnice ne!

Ljudsko tijelo je obnavljajući sustav dinamičke ravnoteže, biološke ili kemijske nečistoće možemo podnijeti ako su u malim količinama koje tijelo može izbaciti i ne možemo podnijeti kada se radi o pretjeranim količinama. Doktori će vam vrlo često kazati kako je količina do jedne čaše crnog vina zdrava, jer ima antioksidante u sebi i ta mala količina alkohola djeluje pozitivno. Naravno ako popijete dvije litre čistog alkohola umrijet će te u roku od nekoliko desetaka sekundi. Omjer između korisne i štetne razine je daleko manji nego što je omjer emitirane snage između mobilnog uređaja i mikrovalne pećnice.

4.3. Dužina trajanja izloženosti EMV

Snažnim izvorima EMV ne bi smjeli biti izloženi dugo vremena, kod slabijih izvora izloženost može biti duža.

Uzmimo jedan jednostavni primjer. Kada se vozimo brodom koji ima uključen radar. Važno je znati da brodski radar ima usku horizontalnu laticu zračenja (negdje oko 1^0) i široku vertikalnu laticu zračenja (oko $30\ ^0$) dakle, redovno pere svojim snopom palubu na kojoj se nalazimo. Njegova izlazna impulsna snaga je minimalno oko 10 KW. (impuls traje 1μs a pauza među impulsima 1 ms što bi u prosjeku bilo kao da smo izloženi konstantnom izvoru zračenja od 10 W , mada je snažno impulsno zračenje zasigurno puno štetnije nego što bi bila izloženost prosječnoj snazi). Dakle tog izvora se ne bojimo a u ruci držimo mobitel kojeg se bojimo a njegova izlazna snaga je reda mW, dakle, 10 000 puta manja.

Znamo da je povremeno i umjereno izlaganje sunčevoj svjetlosti zdravo za naše tijelo, dolazi do proizvodne vitamina D, ali i drugih pozitivnih učinaka. Pretjerano izlaganje sunčevoj svjetlosti može izazvati karcinom kože, a i druge negativne efekte. Omjer između umjerenog i pretjeranog izlaganja suncu daleko je manji nego omjer emitirane snage između mobilnog uređaja i mikrovalne pećnice.

5. ZNANSTVENA ISTRAŽIVANJA KOJA SE PROVODE U SVRHU PROCJENE ŠTETNOSTI EMZ NA LJUDSKO ZDRAVLJE

U proteklih 30 godina objavljeno je približno 25 000 znanstvenih članaka na području bioloških učinaka i medicinske primjene ne ionizirajućih zračenja.

Vrste istraživanja koje se provode kako bi se utvrdili štetni učinci ne ionizirajućeg zračenja na ljudsko zdravlje (EMV iz domena radijskih frekvencija).

- in vitro,
- in vivo,
- epidemiološke studije.

In Vitro su istraživanja na organskim materijalima koji su izuzeti iz živih organizama. Dakle, neživi dijelovi tkiva izlažu se često vrlo snažnim EMV kako bi se utvrdio njihovo djelovanje na tkiva. Ova istraživanja samo djelom mogu pokazati negativne utjecaje.

In Vivo su istraživanja koja se sprovode na živim organizmima, laboratorijskim životinjama, najčešće miševima. Međutim zbog različitosti u organizmima samom dijelom ova istraživanja mogu pokazati mogući utjecaj EMV na ljudsko zdravlje.

Epidemiološke studije pravde se na ljudima koji su u svom profesionalnom radu ili običnom životu izloženi većoj razini snage EMV ili elektromagnetskih polja. Ove studije se svode na statističko povezivanje, traženje korelacije između određenih tipova oboljenja i profesionalnoj izloženosti.

Epidemiološke studije su pokazale da ljudi koji žive ispod dalekovoda ili blizu rasklopnih postrojenja (ovdje se ne radi o EMV nego o promjenjivim elektromagnetskim poljima visoke snage) imaju veću vjerojatnost da obole od leukemije od opeče populacije.

Svi se negativni učinci EMV na ljudsko zdravlje mogu podijeliti u dvije grupe;

- Ne termalni učinci (O ne termalnim negativnim učincima ne postoje egzaktni i mjerljivi pokazatelji, još uvijek se o ovim učincima govori sa hipotetskog stanovišta. Ne termalni negativni učinci djelom se svode na ometanje bioloških struja koje naš organizam koristi za svoje funkcioniranje, te negativnom efektu koje mogu inducirane struje prouzročiti u našem tijelu a koji nisu termalnog karaktera.)
- Toplinski učinci (Elektromagnetski valovi radio frekvencija uzrokuju porast tjelesne temperature. Ovi učinci su mjerljivi i svi propisi i **norme za zaštitu temelje se na termalnim efektima koje EMV može proizvesti na ljudsko tijelo**. Povećanje tjelesne temperature od 1 Celzijev stupanj može stvoriti prve štetne učinke.)

4.1. Termalni negativni efekti na ljudsko zdravlje i norme zaštite.

Toplinska energija svake komponente biološkog tkiva (ioni, molekule, stanice) ima prosječnu vrijednost (1).

$$E = k \cdot T \qquad (1)$$

gdje je:

k - Boltzman je konstanta (k = 1,38x10-23 J/K), a

T - apsolutna temperatura u stupnjevima Kelvina (K = °C + 273,15).

Na sobnoj temperaturi T = 300 K produkt kT iznosi 26 MeV. Ako su toplinski učinci elektromagnetskih valova takovi da su promjene oko vrijednosti 26 MeV neznatne, tada je ukupni utjecaj na ljudsko zdravlje je minoran. Međutim ako je

promjena tjelesne temperature dijela ili cijelog tijela za više od 1^0 C mogu nastupiti štetni efekti. Manji porast tjelesne temperature uslijed utjecaja elektromagnetskog zračenja može dovesti do;

- srčanog udara,
- ozljede mozga,
- neplodnosti,
- zamagljenog vid (ozljede oka),
- glavobolje,
- općeg osjećaja slabost,
- gubitak apetita,
- poremećaji spavanja ili
- pojave opeklina na koži.

Sve norme koje se propisuju za zaštitu od elektromagnetskog ne ionizirajućeg zračenja temelje se na termalnim efektima. Kada smo izloženi djelom ili cijelim tijelom utjecaju EMV prema važećim normama za zaštitu tjelesna temperatura ne bi smjela priječi povišenje za $0,1^0$ C .

Međunarodni standardi za zaštitu od ne ionizirajućeg elektromagnetskog zračenja definiraju maksimalnu količinu snage EM zračenja po kilogramu mase tijela koju čovjek može primiti, to je jedinica koja se naziva SAR (*Specific Absorption Rate*) (W / kg) i dopuštene vrijednosti za profesionalno osoblje i opću populaciju prikazane su u tabeli 2.

Tabela 2: Dozvoljene vrijednosti SAR za profesionalnu izloženost EM zračenju i za izloženost opće populacije.

Frekvencija	SAR kod izloženosti cijelog tijela (W/kg)		SAR kod izloženosti glave i trupa (W/kg)		SAR kod izloženosti ruku i nogu (W/kg)	
10 MHz – 10 GHz	Profesionalna izloženost	Izloženost opće populacije	Profesionalna izloženost	Izloženost opće populacije	Profesionalna izloženost	Izloženost opće populacije
	0,4	0,08	10	2	20	4

Druga veličina koju norme definiraju je dopuštena gustoća struje koja smije poticati ljudskim tijelom J (A/m^2). Dozvoljene gustoće struja reda su mA. Struja koju ćemo osjetiti da potiče našim tijelom je između 20 i 40 mA. Mišićne kontrakcije započinju kada struje prelaze 50 mA, a jaki trzaji muskulature i štetni efekti po ljudsko zdravlje započinju na vrijednostima od 160 do 320 mA.

Da bi došlo do porasta tjelesne temperature za 1^0C na radio frekvencijama od 10 MHz do 10 GHz apsorbirana snaga morala bi biti 4 W/kg. Radi se o **velikoj apsorbiranoj snazi elektromagnetske energije** i o graničnim vrijednostima normi koje su deset, odnosno četrdeset puta ispod navedene vrijednosti. Glavni problem ovog temeljnog ograničenja apsorbirane snage i gustoće struje je **nemogućnost izravnog mjerenja.**

Zbog nemogućnosti izravnog mjerenja navedenih veličina **empirijskim modeliranjem** dolazi se do izračuna za veličine snage i gustoće struje u ljudskom tijelu preko jakosti elektromagnetskih polja (u svakom frekvencijskom području).

Jakost elektromagnetskih polja u odgovarajućem prostoru je relativno lako mjerljiva veličina.

Na temelju toga međunarodna komisija za zaštitu od ne ionizirajućeg zračenja (ICNIRP) je preporučila **referentne razine za ograničavanja izloženosti elektromagnetskim poljima za profesionalno osoblje i za opću populaciju.**

Tabela 4: Maksimalno dozvoljena jakost i gustoća Električnih i magnetskih polja.

Frekvencije od 400 MHz do 2 GHz	Maksimalno dozvoljena jakost električnog polja E(V/m)	Maksimalno dozvoljena jakost magnetskog polja H(A/m)	Gustoća magnetskog toka B(μT)	Ekvivalentna gustoća snage ravnog vala S(W/m^2)
Profesionalna izloženost	$3\ f^{1/2}$	$0,008\ f^{1/2}$	$0,01\ f^{1/2}$	f/40
Izloženost opće populacije.	$1,375\ f^{1/2}$	$0,0037\ f^{1/2}$	$0,0046\ f^{1/2}$	f/200
Napomena	Frekvencija se uvrštava u MHz			

Na primjer za frekvencije oko 1 GHz (GSM 890 do 960 MHz) dozvoljena snaga električnog pola za profesionalnu populaciju je 93 V/m a za opću populaciju 42 V/m. Dakle, relativno visoke vrijednosti snage električnog polja koje teško da će moći ostvariti bilo koja bazna ili mobilna postaja.

U niže navedenoj tabeli 5. Možemo vidjeti granične vrijednosti koje propisuju pojedine zemlje i koje su uglavnom strože od normi ICNIRP.

Tabela 5: Granične vrijednosti električnog polja za frekvenciju 935 MHz

Država	Granične vrijednosti električnog polja za frekvenciju 935 MHz što odgovara baznim postajama GSM-a	
	Profesionalna izloženost	Opća populacija
ICNIRP	**91,7 V/m**	**42,0 V/m**
Austrija	109,09 V/m	49,0 V/m
Velika Britanija, Francuska, Irska, Španjolska	91,7 V/m	42,0 V/m
Švedska	60,0 V/m	42,0 V/m
Švicarska	42,0 V/m	42,0 V/m
Slovenija	41,9 V/m	13,1 V/m
Hrvatska	**42,0 V/m**	**16,8 V/m**

Norme koje su postavljene u Hrvatskoj daleko su strožije od graničnih vrijednosti kojepropisuje ICNIRP.

ICNIRP (International Committee on Non-Ionizing Radiation) datira od trećeg međunarodnog kongresa IRPA-e 1973. godine, kada je prvi put organizirano zasjedanje o zaštiti od neionizirajućih zračenja. 1974. godine osnovana je radna skupina za neionizirajuće zračenje, a 1975. studijska skupina. 1977., za vrijeme četvrtog međunarodnog kongresa IRPA, rođen je INIRC (kratica za Međunarodni odbor za neionizirajuće zračenje), preteča ICNIRP-a koji je stvoren kao neovisna komisija 1992. tijekom osmog međunarodnog kongresa IRPA. Od tada je cilj ICNIRP-a pružiti najmjerodavnije neovisno znanstveno mišljenje o svim pitanjima koja se tiču interakcije neionizirajućih zračenja i zdravlja ljudi. [4]

Razine električnih polja kojima smo izloženi od kućanskih aparata kada smo na udaljenosti od 30 cm znaju biti također visoke, ali se najčešće radi o znatno nižim frekvencijama pa iz toga proizlazi pretpostavka da bi i štetni utjecaji mogli viti znatno manji. [5], [6]

6. ZAKLJUČAK

Europska komisija u digitalizaciji europskog društva sagledava daljnji gospodarski rast i prosperitet Europe. Danas se u dokumentima Europske komisije i Europskog parlamenta sve više pojavljuje termin „Gigabitno društvo“. Veli značaj daje se na daljinu implementaciju 5G tehnologije. Zadatak 5G tehnologije je da objedini sve komunikacijske potrebe među kojima je i IoT. Predviđa se izuzetno veliki porast terminala koji će se priključivati na 5G mrežu zbog čaga će ona imati puno gušći raspored baznih postaja. Mobilni sustav sa gušćim rasporedom baznih postaja u prostoru ujedno znači i rad sa nižim snagama emisije i djeluje ne ukupno smanjenje snage EMV u prostoru. U EU predviđa se rad 5G sustava u frekvencijskim područjima 700 MHz, 3,5 GHz i preko 26 GHz. Danas u svim ovim ili njima bliskim područjima postoje uređaji i sustavi koji rade s nešto većim izlaznim snagama nego što će raditi 5G. Korištenje pametnih antena u 5G sustavu omogućit će još veću uštedu ili smanjenje snaga jer se za svakog korisnika emitira u ciljanom snopu.

Pojavom 5G tehnologija u našem okolišu neće doći do znatnog „zagađenja“ EMV, stanje će biti vrlo slično današnjem stanju, te neće doći do znatnijeg uvećanja rizika za ljudsko zdravlje, sigurno ne većeg od onog koje je danas prisutno.

Literatura:

1 Što je 5G? – Sve o mobilnim mrežama pete generacije https://mob.hr/sto-je-5g-sve-o-mobilnim-mrezama-pete-generacije/

2 Cell Phone Generations 1G, 2G, 3G and now 4G http://forums.techeblog.com/others-cell-phone/1205-cell-phone-generations-1g-2g-3g-now-4g.html

3 Effects of 5G wireless communication on human health. Author: Miroslava Karaboytcheva Members' Research Service
PE 646.172 – March 2020. European Parliament

4 International Commission on Non-Ionizing Radiation Protection (ICNIRP); Gudelines for Limiting Expouse to Time-Varying, Electric, Magnetic, and Electromagnetic Fields (up to 300 GHz), Health Physics, Vol. 74, No. 4, (1998) 494-522

5 Pravilnik o zaštiti od elektromagnetskih polja NN 204/2003

6 Zakon o zaštiti od neionizirajućeg zračenja NN 105/99

7 Emanuele Piuzzi, Paolo Bernardi, Marta Cavagnaro, Stefano Pisa and James C. Lin: Analysis of Adult and Child Exposure to Uniforme Plane Waves at Mobile Communication Systems Frequencies (900 MHz-3GHz), 2010 IEEE

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 zvršnog 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 (C2) jezikom.

Sveučilište u Splitu
Sveučilišni odjel za stručne studije
Kopilica 5, 21000 Split Croatia

UNAPRJEĐENJE SIGURNOSTI SERVISA ELEKTRONIČKE POŠTE KROZ PRIMJENU DMARC PROTOKOLA

IMPROVING THE SECURITY OF EMAIL SERVICE THROUGH THE APPLICATION OF THE DMARC PROTOCOL

Dražen Pranić

SAŽETAK:

Mnoga sigurnosna izvješća i analize kontinuirano ukazuju na servis elektroničke pošte kao uzrok odnosno polazišnu točku brojnih sigurnosnih incidenata. Nesigurnost servisa elektroničke pošte ima za posljedicu stalni porast financijskih prijevara uzrokovanih kompromitacijom korisničkih računa. Korištenje naprednije autentifikacije značajno umanjuje rizik od navedenih prijetnji. U radu će biti opisan razvoj mehanizama autentifikacije servisa elektroničke pošte. Detaljno će se pojasniti DMARC protokol te njegova uloga u unaprjeđenju autentifikacije servisa elektroničke pošte. Istaknuti će se brojne prednosti DMARC protokola, pojasniti koraci i izazovi u implementaciji. Budući trendovi u autentifikaciji servisa elektroničke pošte biti će opisani u završnom dijelu rada.

ABSTRACT:

Many security reports and analyses continuously emphasize email service as the cause or starting point of numerous security incidents. The insecurity of email service results in a steady increase in financial fraud caused by compromising user accounts. The use of more advanced authentication significantly reduces the risk of these threats. The paper will describe the development of email service authentication mechanisms. The DMARC protocol and its role in improving email service authentication will be explained in detail. Numerous advantages of the DMARC protocol will be highlighted, steps and challenges in implementation will be clarified. Future trends in email service authentication will be described in the final part of the paper.

1. UVOD

Nesigurnost servisa elektroničke pošte je činjenica na koju redovito ukazuju različita sigurnosna izvješća i analize. Analize brojnih sigurnosnih incidenata ukazuju na to da je jedan od najčešćih inicijalnih vektora napada upravo zlonamjerna phishing email poruka. Tako prema jednom od najcjenjenijih sigurnosnih izvještaja „Verizon Data Breach Reportu“ phishing email napadi odgovorni su za preko 20% analiziranih incidenata. Ova poražavajuća statistika je nažalost godinama nepromjenjiva.

Jedna od najvećih osiguravajućih tvrtki u svijetu, AIG, u svom godišnjem izvješću navodi slijedeće: “Kompromitacija poslovnih računa servisa elektroničke pošte (engl. BEC) pretekla je ransomware kao glavni razlog zahtjeva za isplatom štete.“ Dalje se u izvješću pojašnjava kako je kompromitacija poslovnog računa servisa elektroničke pošte uzrokovana phishing napadima.

Koliko je navedeni problem velik odnosno koji su njegovi financijski razmjeri detaljnije je pojašnjeno u Izvještaju koji na godišnjoj razini radi američki FBI. Naime u periodu od srpnja 2016. do srpnja 2019. ukupna prijavljena šteta od kompromitacije servisa elektroničke pošte iznosila je 26,2 mlrd. $. Navedena šteta je nastala temeljem ukupno 166.349 prijavljenih incidenata.

Analitička tvrtka Gartner predviđa da će se do 2023. godine napadi kompromitacijom servisa elektroničke pošte udvostručavati svake godine i iznositi će preko 5 mlrd. $ godišnje te će mnogim tvrtkama biti izvor ogromnih financijskih gubitaka. Upravo u tom dokumentu Gartner ističe slabosti u autentifikaciji servisa elektroničke pošte kao jedan od važnijih razloga zašto su ovi jednostavni napadi i dalje aktualni. Preporučuju implementaciju napredne autentifikaciju kao jednu od kontrola koja značajno umanjuje rizik od ovakvih napada. Navedeno ćemo detaljnije pojasniti u slijedećim poglavljima ovog rada.

2. VAŽNOST EMAIL AUTENTIFIKACIJE

Napadi koji imaju za cilj kompromitaciju poslovnog korisničkog računa servisa elektroničke pošte su u pravilu vrlo jednostavni. Najčešći oblik takvih napada je kada napadač promijeni, lažira „pošiljatelj“ polje poruke elektroničke pošte (engl. from) kako bi prevario primatelja poruke.

Pogotovo je učinkovita metoda napada „direktorska prijevara“ u kojoj se napadači pretvaraju da su dio managementa organizacije te zaposlenike navode da uplate novčani iznos na lažni račun ili da neovlašteno doznače novce s

poslovnog računa. Prema Ministarstvu unutarnjih poslova Republike Hrvatske to je jedna od sedam najčešće korištenih financijskih online prijevara.

Email autentifikacija ima za cilj umanjiti rizik od ovakvih i sličnih prijevara koje koriste nesigurnost servisa elektroničke pošte. Osnovni zadatak email autentifikacije je proces slanja i primanja poruka elektroničke pošte učiniti što sigurnijim odnosno identificirati pošiljatelja same poruke te preusmjeriti je primatelju.

Svaka poruka elektroničke pošte pošiljatelja detaljno se analizira od strane dolaznog poslužitelja primatelja. Osnovni kriterij analize same poruke temelji se na metodi provjere autentičnosti koju definira pošiljatelj poruke. Temeljem tih podataka poslužitelj primatelja elektroničke pošte definira hoće li s zaprimljena poruka isporučiti, staviti u karantenu ili odbaciti.

avedeni proces se primjenjuje neovisno o tome koja vrsta autentifikacije se koristi. Poslužitelj primatelja elektroničke pošte prilikom zaprimanja poruke analizira određene podatke u poruci te DNS zapisima domene pošiljatelja. Poslužitelj primatelja poruka temeljem analize korištenih metoda autentifikacije donosi odluku o autentičnost poslane poruke.

U slijedećim poglavljima ovog rada pojasniti ćemo najvažnije metode autentifikacije:

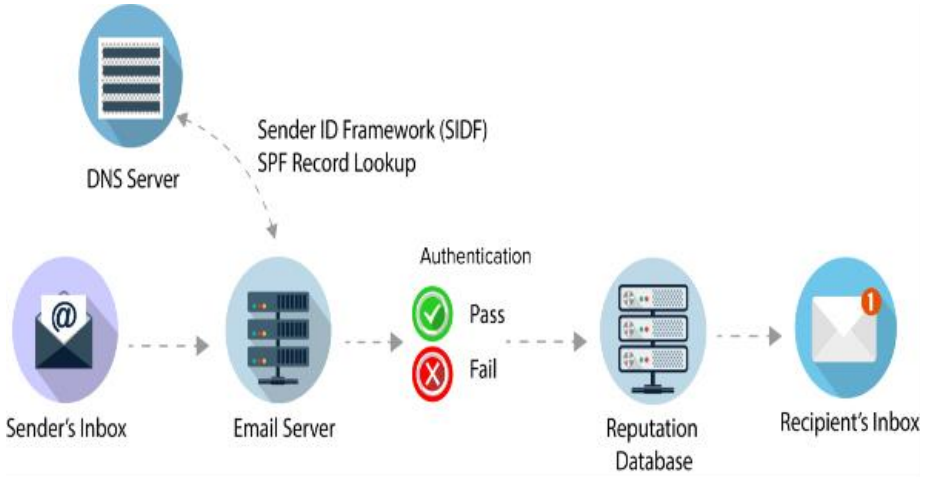
- Sender Policy Framework (SPF),
- DomainKeys Identified Mail (DKIM),
- Domain Message Authentication Reporting and Conformance (DMARC),
- Brand Indicators for Message Identification (BIMI).

3. SENDER POLICY FRAMEWORK

Sender Policy Framework ili SPF je metoda autentifikacije elektroničke pošte čija je svrha detekcija krivotvorenja adrese pošiljatelja tijekom isporuke poruke elektroničke pošte. SPF je u DNS zapis koji određuje koje IP adrese i/ili poslužitelji smiju slati poruke elektroničke pošte s određene domene. Pri tom SPF koristi vrijednost povratnog puta (engl. Return-Path) za provjeru izvornog poslužitelja elektroničke pošte. Povratni put je adresa elektroničke pošte koju poslužitelji primatelja koriste za obavještavanje pošiljatelja elektroničke pošte o problemima s isporukom poruke.

Provjera SPF zapisa poruke elektroničke pošte provodi se na način prikazan na Slici 1.:

1. Poslužitelj primatelja elektroničke pošte zaprima SPF zapis iz DNS poslužitelja za domenu pošiljatelja „primjer.hr“.
2. Poslužitelj primatelja provjerava SPF zapis te analizira popis IP adresa koje su u zapisu ovlaštene slati elektroničku poštu za domenu „primjer.hr“.
3. Ukoliko IP adresa pošiljatelja zadovolji SPF provjeru tada će poslužitelj primatelja nastaviti procesirati poruku elektroničke pošte.
4. Ukoliko SPF provjera nije uspješna tada će poruka elektroničke pošte biti odbačena sukladno postavkama poslužitelja primatelja



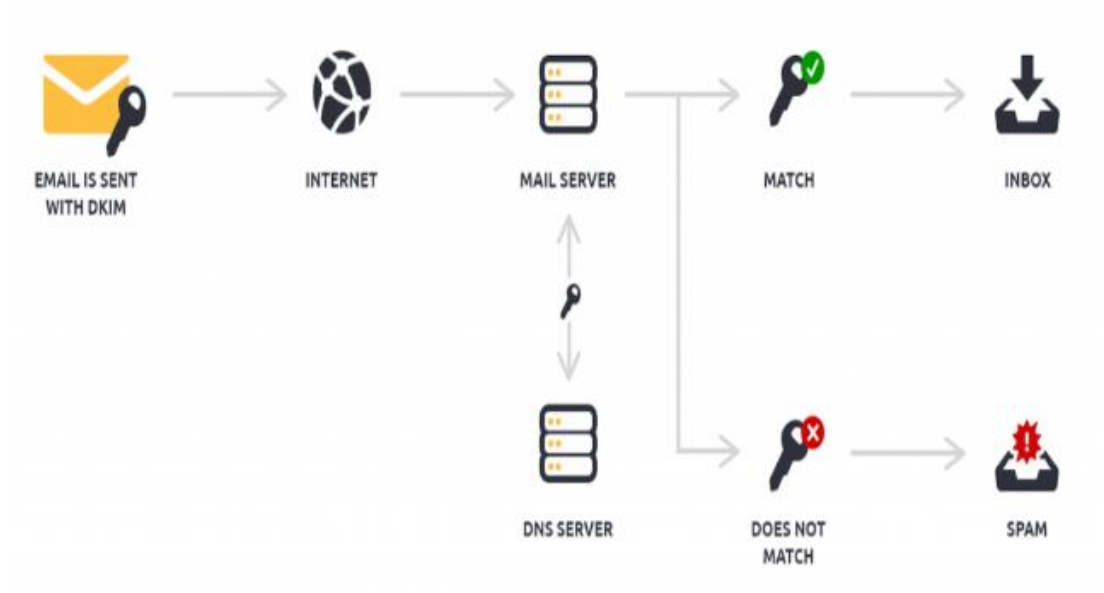
Slika 1. Provjera SPF zapisa

SPF metoda autentifikacije ime nekoliko ozbiljnih nedostataka. Metoda nema odgovor na problem prosljeđivanja email poruka. Ukoliko primjerice pošaljemo poruku elektroničke pošte na adresu netko@primjer.hr a ta adresa ima podešeno prosljeđivanje na netko@primjer.com SPF provjera neće proći. U ovom primjer poruka elektroničke pošte izgleda kao da dolazi s poslužitelja koji nemaju veze s domenom pošiljatelja.

Međutim, najveći nedostatak SPF metode autentifikacije je što se ne provjerava polje pošiljatelja koje je jedino vidljivo primatelju elektroničke pošte. Napadač može u vrijednost povratnog puta navesti svoju domenu, zadovoljiti SPF provjeru, te promijeniti polje pošiljatelja i tako prevariti primatelja poruke elektroničke pošte.

4. DOMAINKEYS IDENTIFIED MAIL (DKIM)

DKIM koristi kriptografiju javnog ključa za autentifikaciju pojedinačnih poruka elektroničke pošte. Poslužitelj pošiljatelja elektroničke pošte dodaje digitalni potpis poruke u samu poruku korištenjem svog privatnog ključa. Poslužitelj primatelja elektroničke pošte provjerava digitalni potpis poruke javnog ključa pošiljatelja. Javni ključ pošiljatelja dostupan je pomoću odgovarajućeg DKIM zapisa u DNS domeni pošiljatelja. Način provjera DKIM zapisa prikazan je na Slici 2.



Slika 2. Provjera DKIM zapisa

Primjena DKIIM autentifikacije sprječava napadača da presretne poruku elektroničke pošte, promijeni je te tako promijenjenu pošalje primatelju.

Međutim kao i kod SPF metode autentifikacije DKIM ne provjerava polje pošiljatelja koje je jedino vidljivo primatelju elektroničke pošte. Napadač može poslati sa svoje domene DKIM potpisanu poruku ali može promijeniti polje pošiljatelja poruke i tako prevariti primatelja poruke elektroničke pošte.

5. DOMAIN MESSAGE AUTHENTICATION REPORTING AND CONFORMANCE (DMARC)

DMARC je metoda autentifikacije odnosno protokol koji omogućava detaljnu provjeru autentičnosti email i izvještavanje. Temelji se na široko primijenjenim SPF i DKIM protokolima koji su pojašnjeni u prethodnim poglavljima. Rješava najvažniji problem autentifikacije email poruka a to je neovlašteno mijenjanje polja pošiljatelja poruke.

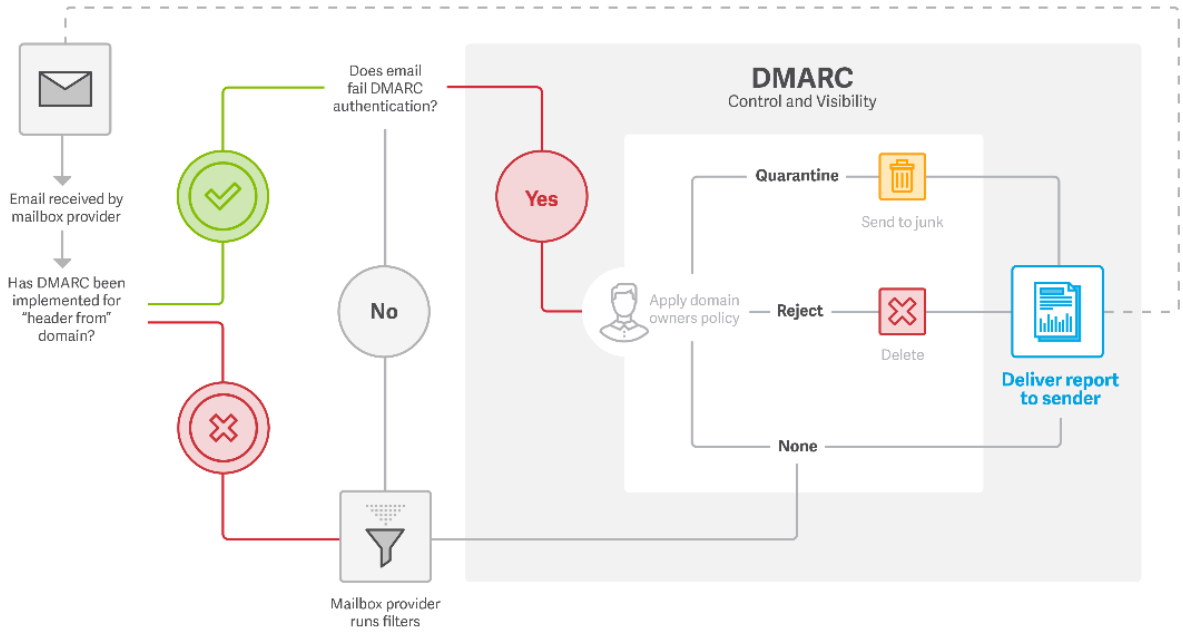
DMARC se temelji na rezultatima SPF-a i/ili DKIM-a, te je za implementaciju neophodna barem jedna od navedenih metoda autentifikacije. Preporuka je koristiti obje metode autentifikacije kao potporu DMARC mehanizmu, zbog dodatne razine provjere. U slučaju korištenja samo jedne metode autentifikacije, preporuča se korištenje DKIM-a, zbog činjenice da isti pruža veću razinu sigurnosti i eliminira pojedine sigurnosne probleme prisutne kod SPF mehanizma.

Važno je napomenut da DMARC koristi DNS za objavu zapisa slično kao i SPF/DKIM.

DMARC zapisi definiraju na koji način upravljate s porukama koje nisu usklađene s ovom metodom autentifikacije. Razlikujemo tri DMARC zapisa/politike:

1. p=none
   Email promet se samo nadzire. Ne poduzimaju su nikakve dodatne aktivnosti
2. p=quarantine
   Neovlaštene poruke elektroničke pošte se šalju u karantenu ili u spam.
3. p=reject
   Neovlaštene poruke se ne isporučuju.

Prilikom provjere usklađenja poruke elektroničke pošte s DMARC protokolom najprije se provjerava jesu li ispunjeni SPF i/ili DKIM zahtjevi. Ukoliko jesu tada se provjerava usklađenje s DMARC zahtjevima. Ovisno o rezultatima DMARC usklađenja i definiranim DMARC politikama poruke elektroničke pošte se isporučuju primatelju. Navedeni proces je detaljno prikazan na Slici 3.

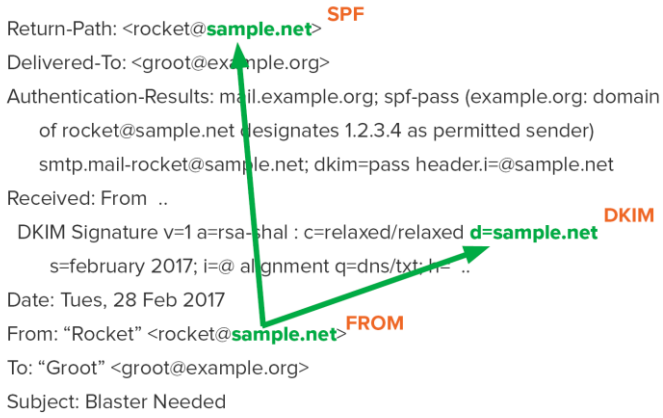


Slika 3. DMARC provjera poruke elektroničke pošte

DMARC usklađenje zapravo rješava najveći nedostatak SPF i DKIM metoda autentifikacije poruka elektroničke pošte. Naime DMARC povezuje rezultate SPF i DKIM metoda sa sadržajem poruke elektroničke pošte odnosno sa sadržajem polja pošiljatelja poruka. Domena pronađena u polju pošiljatelja poruke elektroničke pošte je dio poruke koji povezuje cjelokupno DMARC procesiranje.

Danas bilo tko može kupiti domenu i podesiti odgovarajuće SPF i DKIM zapise. Stoga rezultat analize SPF i DKIM zapisa mora biti povezan s domenom koja se nalazi u polju pošiljatelja poruke elektroničke pošte. Navedeni koncept je poznat kao usklađivanje identifikatora (engl. identifier alignment). Usklađivanje identifikatora omogućuje da postojeće metode autentifikacije servisa elektroničke pošte postanu relevantne za sadržaj same poruke. Usklađivanje svih identifikatora je najveći izazova prilikom implementacije DMARC protokola.

Koncept usklađivanje identifikatora ćemo pojasniti na jednostavnom primjeru u kojem je neka poruka elektroničke pošte poslana s domene *sample.net.* DMARC protokol će verificirati SPF identifikator odnosno adresu pošiljatelja u omotnici poruke (engl. ReturnPath) i DKIM identifikator odnosno d parametar. Navedeno je prikazano na Slici 4.



Slika 4. DMARC usklađivanje identifikatora

Važan dio usklađivanja identifikatora odnosi se na vanjske pružatelje usluga koji u ime pošiljatelja komuniciraju s ostalim sudionicima.

5.1 DMARC izvještavanje

DMARC protokol donosi veliki doprinos u domeni izvještavanja. Naime bez DMARC protokola nije moguće dati odgovor na jednostavno pitanje: „Koji sve pošiljatelji poruka elektroničke pošte šalju poruke u ime moje organizacije"? Bez DMARC protokola odgovor na navedeno pitanje nije moguć.

DMARC protokol omogućuje skupna ili detaljna forenzička izvješća. Skupna izvješća šalju se automatski prema definiranom intervalu na adrese specificirane u *rua* parametru, od strane svih poslužitelja elektroničke pošte na strani primatelja koji su postavljeni na odgovarajući način. U skupnim izvješćima nalaze se slijedeće najvažnije informacije:

- osnovni podaci o organizaciji koja šalje izvješće i vremenski interval izvješća,
- sažetak rezultata provjere (IP adrese s koje su poruke poslana, broj poruka poslanih s tih adrese i rezultati SPF i/ili DKIM provjera te provjera usklađenosti).

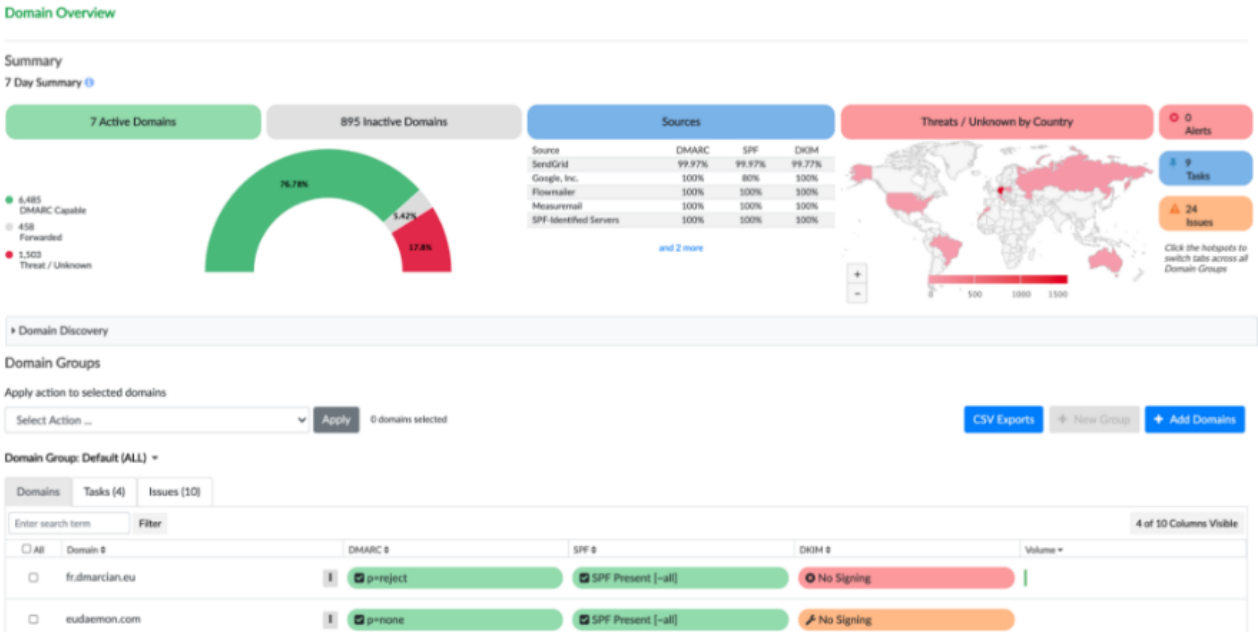
Forenzička izvješća, uz gornje informacije, sadržavaju i dodatne podatke poput naslova poruke, zaglavlja poruke i URL poveznica koje se nalaze u tijelu poruke. Forenzička izvješća se u pravilu šalju odmah nakon primitka poruke koja ne zadovoljava DMARC metodu autentifikacije.

Upravljanje DMARC izvješćima može predstavljati izazov za mnoge organizacije. Naime organizacije s velikom brojem korisnika servisa elektroničke pošte mogu očekivati ogroman broj zaprimljenih DMARC izvještaja koje je potrebno analizirati. Dodatni izazov predstavlja što DMARC koristi XML kao format izvještaja. Navedeno ukazuje na potrebu korištenja gotovih programskih rješenja ili platformi za analizu DMARC izvješća.

Kako je DMARC protokol rastao na popularnosti tako na tržištu postoji sve više takvih platformi koje nude brojne funkcionalnosti od kojih su najvažnije:

- arhiviranje DMARC izvješća,
- obogaćivanje DMARC statusa,
- provjera status SPF/DKIM/DMARC zapisa,
- podešavanje različitih notifikacija,
- integracija s različitim sigurnosnim rješenjima.

Na slici 5. je prikazan izgled jedne od najraširenijih DMARC platformi: Dmarcian.com.



Slika 5. Prikaz Dmarcian.com DMARC platforme

6. IZAZOVI IMPLEMENTACIJE DMARC PROTOKOLA

Implementacija DMARC protokola/metode autentifikacije je dugotrajan proces. Navedeno se pogotovo odnosi za velike organizacije. Takve organizacije često surađuju s vanjskim pružateljima različitih usluga koji imaju za potrebu slati poruke elektroničke pošte u ime organizacije. Prije pune implementacije DMARC protokola mora se osigurati DMARC uskladivost email infrastrukture organizacije i vanjskih pružatelja usluga koje organizacija koristi. Ukoliko se DMARC implementira brzopleto riskira se odbacivanje velikog broja legitimnih poruka elektroničke pošte.

Primjerice na Dmarcian.com platformi se svi takvi vanjski pružatelji usluga automatski i evaluiraju što organizacijama omogućuje jednostavni uvid u uskladivost SPF/DKIM/DMARC metodama autentifikacije.

Sources			
Source	DMARC	SPF	DKIM
SendGrid	99.97%	99.97%	99.77%
Google, Inc.	100%	80%	100%
Flowmailer	100%	100%	100%
Measuremail	100%	100%	100%
SPF-Identified Servers	100%	100%	100%

Slika 6. SPF/DKIM/DMARC uskladivost vanjskih pružatelja usluga

Najbolja praksa za implementaciju DMARC protokola naglašava važnost i potrebu za faznim pristupom. U prvoj fazi je potrebno na sve domene organizacije koje su sposobne poslati poruke elektroničke pošte, odnosno imaju odgovarajući MX zapis u DNS poslužitelju, postaviti tzv. DMARC auditing. Odnosno potrebno je podesiti odgovarajući DMARC zapis primjerice: v=DMARC1;p=none;rua=mailto:dmarc@domena.hr. Navedenim zapisom, DMARC politika postavlja se u nadzorni način rada, što znači da

poruke elektroničke neće biti odbijene u slučaju neuspješne DMARC provjere. Time ne riskiramo gubitak bilo koje poruke elektroničke pošte. Rua parametar definira adresu elektroničke pošte na koju će se slati skupna DMARC izvješća.

Nakon što je DMARC implementiran u nadzorni način rada počinje najzahtjevniji dio implementacije: analiza svih poslužitelja elektroničke pošte koji šalju poruke u ime organizacije.

Nakon što se DMARC politika postavi u nadzorni način rada mora proći minimalno barem mjesec dana da se prikupi dovoljan broj skupnih DMARC izvješća te tako identificiraju svi interni i vanjski pošiljatelji poruka elektroničke pošte.

Pri tome nisu svi email poslužitelji (koji pripadaju samoj organizaciji ili vanjskim pružateljima usluge) na istoj razini DMARC uskladivosti. Neki su na samom početku te imaju primjerice samo implementiran SPF zapis a drugi mogu biti skoro pa usklađeni s DMARC protokolom. Ovisno o broju pošiljatelja te njihovoj razini uskladivosti s DMARC protokolom ovisi koliko vremena je potrebno za punu implementaciju DMARC protokola. Navedeno podrazumijeva postavljanje DMARC zapisa/politike *p=reject*. U našem primjeru DMARC zapis bi izgledao ovako: *v=DMARC1;p=reject;rua=mailto:dmarc@domena.hr*.

Često navedena aktivnosti može trajati puno duže od vremena potrebnog za prikupljanje inicijalnih podataka. Mnoge organizacije nemaju dovoljno znanja, ljudskih resursa za navedenu implementaciju. Zbog toga mnoge organizacije rizik odbacivanja email poruka vide mnogo većim od rizika primanja neovlaštenih poruka. Stoga ne čudi da se na tržištu osim usluge eksternalizacije DMARC platformi nudi i zagarantirana usluga pune implementacije DMARC protokola. Pružatelji takve usluge garantiraju primjenu *p=reject* za dogovorenu domenu.

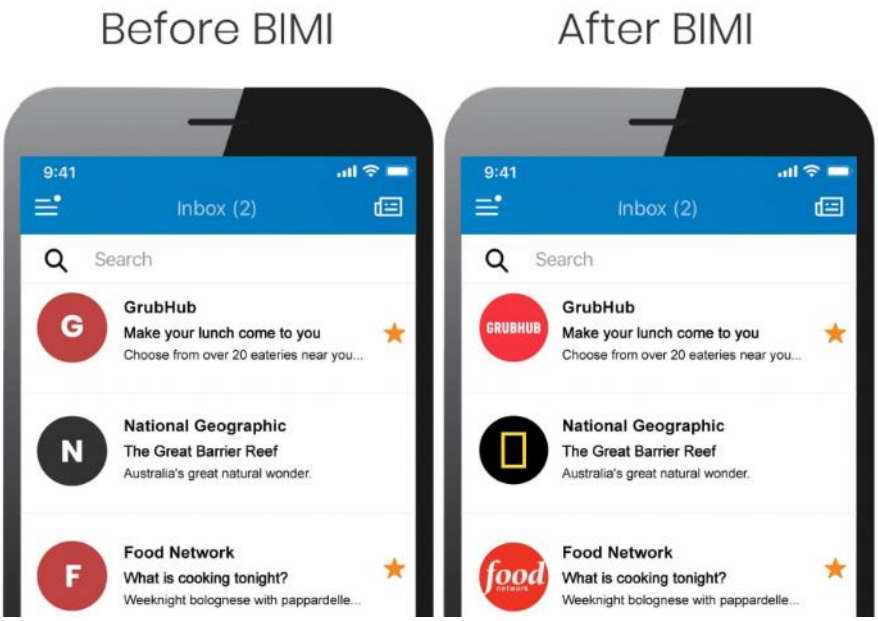
Možda upravo sve veći broj pružatelja DMARC usluga unaprijedi dosta lošu statistiku oko korištenja DMARC protokola. Naime prema sigurnosnoj tvrtki Agari svega 15 % najvećih svjetskih kompanija je ostvarilo punu uskladivost s DMARC protokolom. U iznimno detaljnoj analizi Agari je obuhvatio 366 milijuna Internet domena. Ustanovljeno je veliko povećanje korištenja DMARC protokola od 83%. Međutim u konačnici svega 11.6 milijuna domena ga stvarno koristi.

7. BUDUĆI TRENDOVI

Iako kao relativno nova metoda autentifikacije DMARC nije zadnje što je razvijeno po pitanju email autentifikacije. Puno se nade po pitanju email autentifikacije polaže u Brand Indicators for Message Identification (BIMI) metodu email autentifikacije.

BIMI je otvoreni standard koji tvrtkama omogućuje potvrdu identiteta i njihovo lako prepoznavanje u pristiglim porukama elektroničke pošte. Slično kao SPF, DKIM, DMARC i BIMI je DNS zapis. Radi u suradnji zajedno sa sve ostale tri metode autentifikacije. Neki ovu metodu autentifikacije elektroničke pošte nazivaju i DMARC 2.0. Zapravo BIMI tvrtkama omogućuje da stave svoj logotip na svaku poslanu poruku elektroničke pošte.

Na slici 7. prikazani su primjeri poruka elektroničke pošte prije i poslije BIMI implementacije.



Slika 7. Primjeri poruka elektroničke pošte prije i poslije BIMI implementacije

Osnovni preduvjet za BIMI implementaciju je puna implementacija DMARC protokola odnos DMARC zapis/politika *p=reject.* Osim toga potrebno je dodati BIMI dns zapis koji sadrži logotip organizacije kao u ovom primjeru: *v=BIMI1; l=https://www.example.com/images/logo.sv*gS obzirom je BIMI standard nastao 2019. godine mali broj kompanija ga aktivno koristi. Međutim radi očite marketinške koristi za očekivati je da će se u budućnosti sve više koristiti.

8. ZAKLJUČAK

Nesigurnost servisa elektroničke pošte je nažalost uvijek aktualna tema. Poslovno i privatno vrlo često dobijemo sumnjive poruke u kojoj se od nas očekuje neka brza akcija ili nepromišljena aktivnost. Pri tome napadači koriste sve što će privući pozornost primatelja poruke elektroničke pošte. Pojavom COVID-19 virusa značajno je porastao broj zlonamjernih poruka koje upravo temu COVID-19 zloupotrebljavaju. Naravno pri tome će činiti sve da takve poruke budu što autentičnije. Upravo metoda lažiranja polja pošiljatelja poruke im u tome uveliko pomaže.

Puna implementacija DMARC protokola značajno umanjuje rizik od takvih jednostavnih metoda napada. Iz rada je jasno vidljivo da navedeno nije nimalo jednostavan zadatak. Međutim prema mišljenju mnogi koristi od DMARC implementacije uvelike premašuje rizike koja ta implementacija donosi. Za organizacije koje nemaju dostatne resurse za samostalnu implementaciju najbolja strategija je ključ u ruke. Navedeno uključuje izbor vanjskog partnera koji osim specijalizirane DMARC platforme nudi i uslugu zagarantirane DMARC implementacije. Dobra vijest je da na tržištu postoji sve više takvih specijaliziranih tvrtki.

Važno je također napomenuti da DMARC nije rješenje za sve pokušaje prijevare korištenjem poruka elektroničke pošte.

U ovom radu su predstavljene najvažnije metode autentifikacije servisa elektroničke pošte: SPF, DKIM, Nesigurnost servisa elektroničke pošte je činjenica na koju redovito ukazuju različita sigurnosna izvješća i analize. Analize brojnih sigurnosnih incidenata ukazuju na to da je jedan od najčešćih inicijalnih vektora napada upravo zlonamjerna phishing email poruka.

Važno je napomenuti da DMARC ne rješava sve probleme. Naime korištenje vanjskih javnih servisa elektroničke pošte predstavlja veliki izvor prijetnji kada se radi o kompromitaciji poslovnih računa servisa elektroničke pošte (engl. BEC). Prema analizi tvrtke Agari oko 60% ovakvih napada odvija se putem adresa elektroničke pošte registriranih na Gmail.com, Outlook.com, Yahoo.com i sličnim javnim besplatnim servisima elektroničke pošte.

Iako DMARC ne rješava sve metode napada na servis elektroničke pošte svakako predstavlja važan aspekt u podizanju razine sigurnost tog iznimno važnog servisa za svaku organizaciju.

Literatura:

1 2020 Data Breach Investigations Report, https://enterprise.verizon.com/resources/reports/dbir/

2 Cyber Claims: GDPR and business email compromise drive greater frequencies, https://www.aig.ae/content/dam/aig/emea/uae/documents/aig-cyber-claims-report-2020.pdf

3 Business Email Compromise The $26 Billion Scam, https://www.ic3.gov/Media/Y2019/PSA190910, 10.09.2019.

4 Protecting Against Business Email Compromise Phishing, https://www.gartner.com/doc/reprints?id=1-1Z0GPFC0&ct=200512&st=sg, 19.03.2020.

5 Internet prijevare, https://policija.gov.hr/prevencija/racunalna-sigurnost/internet-prijevare/456

6 Tehničke preporuke za povećanje sigurnosti komunikacije elektroničkom poštom, https://www.zsis.hr/UserDocsImages/Tehni%C4%8Dke_preporuke_za_pove%C4%87anje_sigurnosti_komunikacije_elektroni%C4%8Dkom_po%C5%A1tom.pdf

7 How To: Email Authentication: SPF, DKIM, DMARC and BIMI, https://www.emailout.com/email-authentication-methods/, 14.09.2020

8 Dmarcian SaaS platforma, https://dmarcian.com/dmarc-saas-platform/

9 Q1 2020 Email Fraud & Identity Deception Trends, https://www.agari.com/email-security-blog/dmarc-q1-2020-email-fraud-report/

Podaci o autoru:

Dražen Pranić
Atlantic Grupa d.d.
Miramarska 23
10000 Zagreb
e-mail: Drazen.Pranic@atlanticgrupa.com

Diplomirao je na Fakultetu elektrotehnike i računarstva, smjer Telekomunikacije te magistrirao na Ekonomskom Fakultetu u Zagrebu, smjer Informatički management. Ima položen certifikat iz revizije informacijskog sustava Certified Information System Auditor (CISA).

Stekao je bogato iskustvo u području informacijske sigurnosti i revizije informacijskih sustava radeći u Agrokoru, Hrvatskoj narodnoj banci, Plivi, Privrednoj banci Zagreb i Tele2. Trenutno je zaposlen kao Voditelj IT sigurnosti u Atlantic Grupi d.d gdje je odgovoran za sve aspekte sigurnosti informacijskih sustava.

Predavač je na Visokom učilištu Algebra u Zagrebu i to na kolegijima Sigurnost informacijskih sustava, Sigurnost elektroničkog poslovanja i Upravljanje identitetima.

NASTAVA NA DALJINU PROBLEMI I RJEŠENJA

DISTANCE LEARNING PROBLEMS AND SOLUTIONS

dr.sc. Winton Afrić

SAŽETAK:

U prvom dijelu rada govori se o odabiru tehnologije za obavljanje rada na daljinu i problemima koji su vezani uz taj odabir. U programskom smislu nedostatak jedinstvene programske podrške koja može podržati sve tražene aktivnosti. Nastava na daljinu mora omogućiti, držanje predavanja, auditornih vježbi, laboratorijskih vježbi i provjere znanja putem testova. Značaj izrade videozapisa predavanja i auditornih vježbi kojim se studentima omogućava ponavljanje sadržaja. U nastavku rada se govori o praktičnim iskustvima i problema kod obavljanja nastavnog procesa na taj način. Značajni problemi uočeni su u nedostatku kvalitetne pristupne širokopojasne infrastrukture koja bi mogla podržati rad na daljinu. Na dalje se analiziraju DESI parametara koji opisuju stanje širokopojasne pristupne infrastrukture u RH, te se vrši usporedba sa drugim zemljama EU. U zaključku se daju prijedlozi za unapređenje procesa nastave na daljinu s aspekta tehnologija.

ABSTRACT:

The first part of the paper discusses the choice of technology for remote work and the problems associated with that choice. In terms of distance learning, the lack of unique software that can support all required activities for that process. Distance learning must enable lectures, auditory exercises, laboratory exercises, and knowledge tests. The importance of making videos of lectures and auditory exercises that allow students to repeat the content. In the continuation of the paper, we talk about practical experiences and problems in performing the teaching process. Significant problems observed in the lack of quality access broadband infrastructure that could support teleworking. Furthermore, analyzed DESI parameters that describe the state of broadband access infrastructure in the Republic of Croatia and a comparison with other EU countries is made.

1. UVOD

U zadnjih godinu dana došlo je do „eksplozije“ rada na daljinu. Brzo širenje rada na daljinu uvjetovano je pojavom globalne pandemije izazvane COVID 19 virusom. Rad na daljinu, „online“, doprinosi mjerama zaštite od širenja virusa jer je značajno smanjio fizičke kontakte među ljudima. Međutim, rad na daljinu pokazao je i druge prednosti koje se poglavito očituju u smanjenju troškova. Na primjer kada su učilišta za vrijeme lockdown-a prešle raditi na daljinu znatno su se smanjili troškovi vezani uz potrošnju energije, grijanje prostora, električna rasvjeta, potrošnja vode i tome slično.

Mnogi poslovi omogućavaju nesmetani rad na daljinu, a za kompaniju to često znači i manje troškove poslovanja. Rad na daljinu pruža i stanoviti komoditet ljudima, jer rade od svog doma iz svog obiteljskog okruženja te lakše rješavaju probleme koji su vezani uz čuvanje djece i održavanje domaćinstva. Naravno da rad na daljinu ima svojih negativnih strana, ništa ne može u cijelosti zamijeniti osobni kontakt. A i ljudi zbog nedostatka socijalnog kontakta mogu osjetit i određene psihološke probleme. Vjerojatno u budućnosti kombinirani model rada na daljinu i neposrednog kontakta će predstavljati idealnu kombinaciju. Dakle, dobar dio poslova obavljat će se na daljinu ali će se timovi s vremena na vrijeme neposredno sastajati.

Kao što je već rečeno zbog stanja izazvanog globalnom pandemijom, ne samo naša zemlja nego i mnoge druge zemlje relativno naglo su ušle u domenu obavljana poslova na daljinu. Pred postavljalo se da će se takovi oblici rada postupno uvoditi i prevladati u daljoj ili bližoj budućnosti a odjednom preko noći takav oblik rada nametnu se je kao imperativ.

I prije nego što je došlo do novo nastale situacije izazvane globalnom pandemijom određeni broj ljudi radio je na daljinu, obavljajući posao za domaće ili strane tvrtke iz svog doma. Međutim, nagli zahtjev prelaska na „online“ za većinu koja to do sada nije iskusila predstavljao je stanoviti šok. Također, novonastala situacija ne samo da je da je od pojedinaca zahtijevala brzo usvajanja novih znanja i vještina koje zahtijeva ovaj rad, već je i od kompanija koje podržavaju svojom opremom i programskom podrškom zahtijevala brzo prilagođavanje uvjetima naglog porasta prometnog opterećenja. Novonastale potrebe za zadovoljenjem prometnih zahtjeva narasle su gotovo trenutno za preko 90%.

Za rad na daljinu javili su se i drugi problemi koji su vezani uz stanje širokopojasne pristupne infrastrukture, poglavito u Republici Hrvatskoj.

Kada se kućni širokopojasni priključak koristi za gledanje TV programa (otprilike 4 Mbit/s po jednom video toku) i pretraživanje interneta (otprilike 1 Mbit/s po video sadržaju na primjer sa YouTube), onda su pristupne brzine od

svega par Mbit/s bile zadovoljavajuće. Međutim, rad na daljinu zahtijeva znatno veći informacijski volumen, naravno koliki ovisi o tipu i vrsti rada na daljinu. Situacija u kojoj je naše kućno računalo samo terminal koji radi na s aplikacijom u oblaku (*cloud*) što u fizičkom smislu znači s aplikacijom koja se nalazi u informatičkom centru, potreban nam je informacijski volumen onog reda veličine kojeg imamo na današnjim LAN mrežama kada je naše računalo također krajnji terminal, a aplikacija je na serveru, dakle brzina od 100 Mbit/s. Međutim, broj takovih kućnih priključaka u Republici Hrvatskoj je izuzetno mali i to loše stanje širokopojasne pristupne infrastrukture u velikoj mjeri izazvalo je nemogućnost rada na daljinu ili velike probleme u ostvarivanju takova rada.

U ovom radu težište će bit na analizi upravo tih otegotnih okolnosti za rad na daljinu poglavito kada se radi o učenju na daljinu koje i nije toliko zahtjevno kao obavljanje nekih drugih poslova „online“.

2. ODABIR PROGRAMSKE PODRŠKE ZA USPJEŠNO ODRŽAVANJE NASTAVE NA DALJINU

Situacija u kojoj se zahtijevalo od nastavnog osoblja na visokoškolskim ustanovama da pređu na održavanje nastave „online“ došla je toliko naglo da te ustanove i ti djelatnici bili potpuno nepripremljeni a vremena za prilagodbu na novo nastalu situaciju nije bilo. Međutim, vrlo brzo u svega nekoliko dana nastava se je počela odvijati na daljinu. Dakle, djelatnici bez obzira na svoj stručni profil izuzetno brzo su se prilagodili novo nastaloj situaciji.

Većina visokoškolskih ustanova koristila je za komunikaciju sa studentima određene programske podrške kao što je na primjer Moodl-e, ali te programske podrške nisu namijenjene za držanje nastave na daljinu. One su virtualni prostor na kojem se postavljaju nastavni materijali za studente, u okviru kojih se mogu generirati testovi za studente, preko kojih se šalju obavijesti studentima i tome slično, ali nisu prostor koji je namijenjen održavanju online predavanja, laboratorijskih vježbi ili sličnih stvari.

Prva stvar koju je trebalo razriješiti bio je odabir videokonferencijske platforme za odabir učenja na daljinu. Videokonferencijskih platformi preko kojih bi mogli obavljati nastavu na daljinu ima jako mnogo. Međutim, ne može svaki nastavnik koji učestvuje u realizaciji određenog studenskog programa izabrati platformu po svom nahođenju, jer bi tada studenti morali imati veći broj različitih platformi što nije nemoguće, ali nije ni racionalno. Uglavnom su se pojedine sastavnice naših sveučilišta odlučile za jednu od mogućih programskih platformi koja je onda bila obavezna za sve nastavnike i za sve studente. Platformi ima jako mnogo, a ovdje ćemo napomenuti samo neke; Zoom, MS Teams, Skype i tako dalje. Sve one u većoj ili manjoj mjeri omogućavaju priključivanje većeg broja disperziranih korisnika, audiovizualni kontakt među njima i dijeljenje nastavnih materija koji su vezani uz nastavni proces držanja predavanja ili auditornih vježbi. Neke od ovih aplikacija omogućavaju snimanje procesa izvođenja nastave te se kasnije snimka može postaviti na programske podrške kao što je Moodle kako bi je naknadno pregledali studenti koji nisu bili priključeni ili koji su imali u tom trenutku tako loš priključak (vezu) da im je učestvovanje bilo otežano ili nemoguće.

Mnogi studiji, pogotovo tehnički zahtijevaju i laboratorijske vježbe. Jedan veliki broj tih laboratorijskih vježbi zahtijeva neposredno učestvovanje studenata u praktičnom radu i nije ih uvijek moguće prebaciti u virtualno okruženje. Dakle, oni su kod nas (Sveučilište u Splitu, Odjel za stručne studije) bile organizirane neposredno ali uz poštivanje svih epidemioloških mjera, razmaka, među osobama, broja učesnika, nošenja maski i druge zaštitne opreme. Organiziranje laboratorijske nastave zbog svega navedenoga zahtijevalo je veću satnicu od ranije predviđene koja je uglavnom odrađena gratis preko nominalnog opterećenja. Na Sveučilištu u Splitu odjelu stručnih studija od programske podrške korišten je program Zoom za neposredno održavanje nastave u kombinaciji sa prethodno korištenom programskom podrškom Moodle za komunikaciju sa studentima izvan predviđenih termina nastave. Program Zoom također, je korišten za održavanje konsultacija sa studentima i diplomantima. Neke druge sastavnice Sveučilišta u Splitu koristile su MS Teams koji pruža nešto malo drugačiji spektar mogućnosti.

Nastavno osoblje se je vrlo brzo prilagodilo novo nastaloj situaciji, ovladalo sa korištenjem nove programske podrške za online nastavu u našem slučaju Zoom, te izvršilo pripremu nastavnih materijala za ovakvo izvođenje nastave.

Ono što seje pokazalo kao izuzetno veliki problem je kvaliteta širokopojasne pristupne mreže, odnosno raspoloživa brzina koja za sve učesnike nije bila dostatna te oni nisu mogli normalno pratiti ovakav nastavni proces. Zaleđivanje slike, ton niskog kvaliteta, veće vremensko kašnjenje u interaktivnoj komunikaciji, prekidanje veze i tako dalje. Glavni problem za ovaka način rada na daljinu kod nastavnog procesa lociran je u kvaliteti mreže.

Do sada se je proces nastave na daljinu drugačije koncipirao nego u ovom slučaju. U prethodnim godina nastava na daljinu uglavnom se je odvijala na način kada ste imali predavača koji je bio na udaljenoj lokaciji u specijaliziranoj učionici i kada se je on povezivo sa studentima koji su bili također u specijaliziranoj učionici za takovu svrhu. Međutim, to je sasvim druga dimenzija obavljanja nastavnog procesa u odnosu na novonastalu situaciju u kojoj nitko nije u specijaliziranoj učionici, kada su svi učesnici dislocirani i raspršeni u prostoru najčešće kod svojih kuća za svojim kućnim računalima, tabletima ili pametnim mobitelima. Poneki studenti koji kod nas studiraju, a iz drugih su zemalja okruženja (najčešće BiH) također su bili kod svojih kuća.

Mnogi studenti ne samo iz prigradski i ruralnih područja Republike Hrvatske nego i iz centra visoko urbanih gradova praktički nisu mogli pratiti ovakav nastavni proces zbog lošeg stanja širokopojasne pristupne infrastrukture.

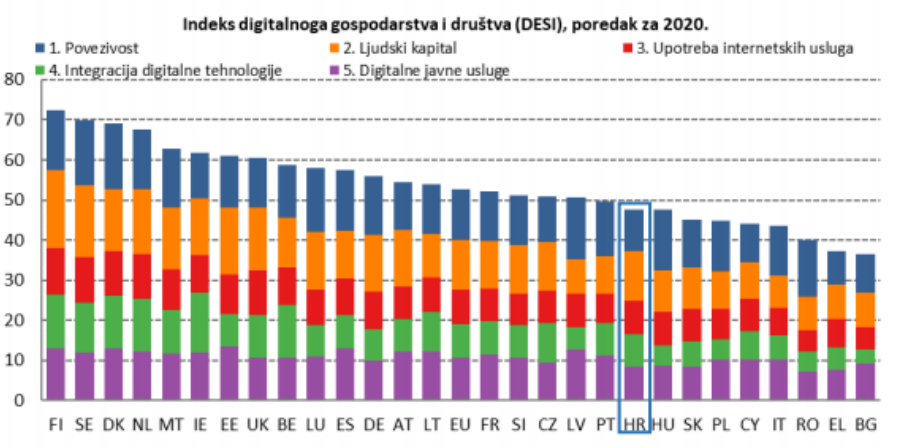
3. STANJE ŠIROKOPOJASNE PRISTUPNE INFRASTRUKTURE U REPUBLICI HRVATSKOJ

Na loše stanje širokopojasne pristupne infrastrukture na mnogim savjetovanjima u RH pa i na KOM-ovim savjetovanjima upozorava se već dugi niz godina. Međutim, ta se upozorenja nisu shvaćala ozbiljno, te smo došli u

situaciju u kojoj znamo da je ugrožen ili nemoguć rad zbog lošeg ili nepostojećeg stanja širokopojasne pristupne infrastrukture. Na naše gospodarstvo takovo stanje će se i dalje odraziti sa vrlo negativnim posljedicama.

Najbolje je da pogledamo položaj Republike Hrvatske unutar EU po DESI pokazateljima koje prati Europska komisija. Prema ukupnim DESI pokazateljima 2020 godine Republika Hrvatska se nalazi na dvadesetom mjestu od dvadesetosma zemalja unije. Taj položaji po ukupnim DESI pokazateljima nije tako loš, a prikazan je na slici 1.

	Hrvatska		EU
	rang	rezultat	rezultat
DESI 2020.	**20**	**47,6**	**52,6**
DESI 2019.	20	44,3	49,4
DESI 2018.	21	40,8	46,5

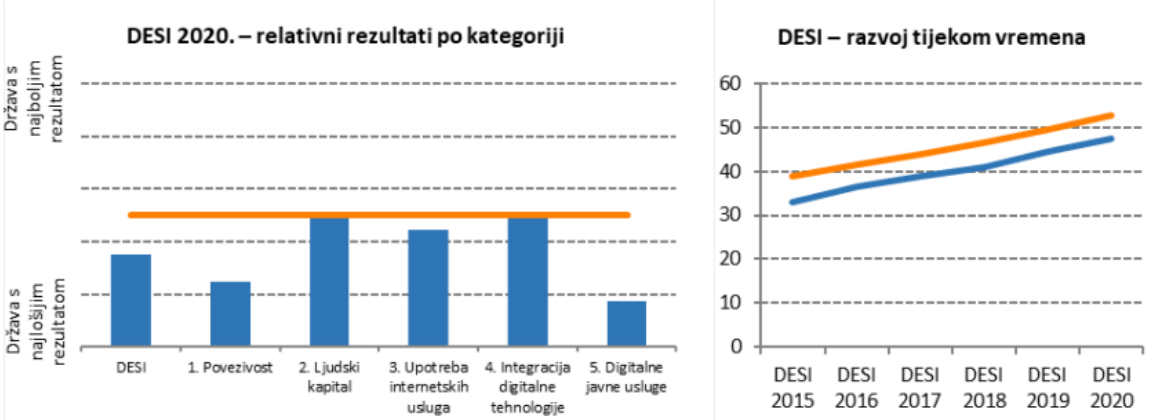


Slika 1: Pozicija Republike Hrvatske po ukupnim DESI parametrima među zemljama EU

Međutim, tom položaju u većoj ili manjoj mjeri pridonose pojedini pod pokazatelji DESI parametara, dok neki drugi DESI pokazatelji vuku nas prema dnu poretka. DESI pokazatelj sastoji se od;

- Povezivosti (po ovom parametru RH je u 2020 godini na 25-tom mjestu)
- Ljudski kapital (po ovom parametru RH je u 2020 godini na 13-tom mjestu)
- Upotreba internetskih usluga (po ovom parametru RH je u 2020 godini na 15-tom mjestu)
- Integracija digitalne tehnologije (po ovom parametru RH je u 2020 godini na 12-tom mjestu)
- Digitalne javne usluge (po ovom parametru RH je u 2020 godini na 25-tom mjestu)

RH po prvom (Povezivost) i petom (Digitalne javne usluge) pod pokazatelju pozicionirana je vrlo nisko. Ako želimo napredovati po ukupnom DESI pokazatelju trebamo poraditi na podizanju kvalitete telekomunikacijske pristupne širokopojasne infrastrukture i razvoju digitalnih javnih usluga to jest online komunikacija sa državnom i lokalnom upravom. Slika 2.Prikazuje nam odnos pojedinih pod pokazatelja prema EU prosjeku i ukupnu promjenu DESI pokazatelja u posljednjih nekoliko goda u usporedbi s ukupnom promjenom EU prosjeka.



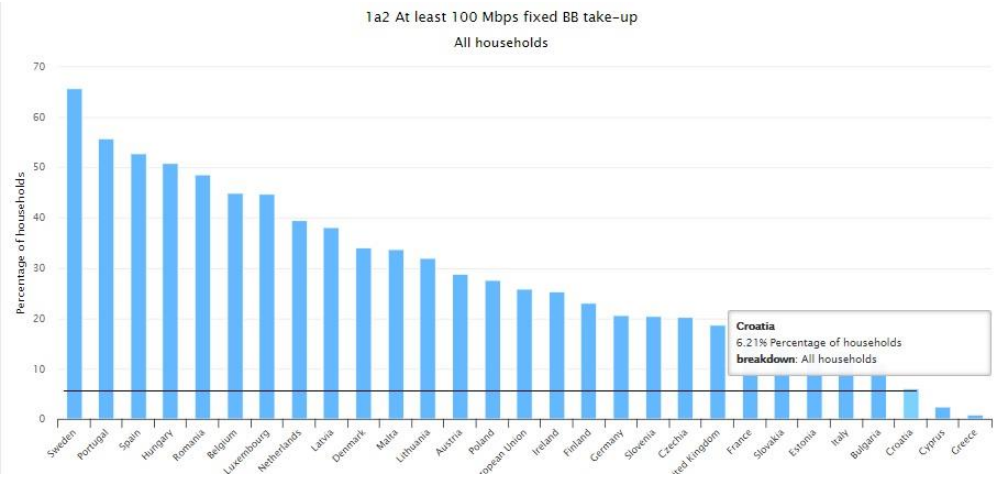
Slika 2. Odnos pojedinih pod pokazatelja DESI prema EU prosjeku i ukupnu promjenu DESI pokazatelja u posljednjih nekoliko goda u usporedbi s ukupnom promjenom EU prosjeka.

Po ljudskom kapitalu i po Integraciji digitalnih tehnologija nalazimo se u EU prosjeku, a da je kvaliteta mreže viša i da su razvijenije javne digitalne usluge ne samo da bi po ukupnim DESI parametrima bili bolje pozicionirani nego bi i pod-pokazatelji Integracije digitalnih tehnologija i Upotrebe internetskih usluga bili znatno bolji jer ti pokazatelji u velikoj mjeri ovise o stanju razvijenosti širokopojasne pristupne infrastrukture.

Kada pogledamo pod pokazatelj ljudski kapital onda je sasvim jasno da postoji znanje, a i svijest o važnosti digitalizacije kao procesa kod opće populacije, ali ta je spoznaja daleko od državne i lokalne uprave jer taj je parametar najlošiji.

Kad govorimo o povezivosti (Connectivity) od nas su jedini lošije pozicionirane, Bugarska, Cipar (koji je dijelom okupiran podijeljen na Turski i Grčki dio) i Grčka (koja je praktički bankrotirala). Međutim, ako gledamo širokopojasne priključke od 100 Mbit/s, što znači postavljanje optike do krajnjeg korisnika, od nas je samo lošije pozicioniran Cipar i Grčka. Dakle, mi smo na dnu razvoja (ulaganja) u brzi širokopojasni pristup.

Operateri nemaju interesa za daljnja ulaganja u razvoj svjetlovodne pristupne infrastrukture. Cijena koju sada već ostvaruju na XDSL modemima je takova da teško mogu od krajnjeg korisnika naplatiti više. Dakle, ulagati u promjenu infrastrukture, a da pri tome znaju da ne dolazi do uvećanja dobiti nema poslovne logike. Operateri neće ulagati nigdje, a ne samo na područjima niske naseljenosti. Jedino gdje operateri imaju interes ulaganja u svjetlovodnu infrastrukturu su veliki poslovni korisnici, jer se tu radi o mogućnosti da mogu izgubiti veliku dobit ako ne osiguraju kvalitetu.



Slika 3. Položaj Republike Hrvatske po broju priključaka koji dosežu 100 Mbit/s i više.

Iako su statistički pokazatelji iskazani kao broj priključaka od 100 Mbit/s i više po broju domaćinstava u RH radi se većinom o svjetlovodnim priključcima koji spajaju velike korisnike (poduzeća). Brpk svjetlovodnih priključaka koji uistinu ide do domaćinstava najmanji je u EU. Dakle, kod nas je penetracija optičkih priključaka nešto preko 6%, ali to u najvećoj mjeri nisu domaćinstva nego priključci prema velikim poslovnim korisnicima.

Gledajući pokazatelj indeksa cijene Republika Hrvatska se od 20 zemalja nalazi na solidnom 18 mjestu. Dakle, ne samo da imamo jednu od naj ne kvalitetnijih mreža nego za nekvalitetnu uslugu plaćamo relativno visoku cijenu u odnosu na prosječni bruto prihod domaćinstva.

Državna uprava još od 2015. pa sve do 2020. godine nastoje oformiti Nacionalno vijeće za digitalnu ekonomiju. Ali nekog strateškog razmišljanja o digitalizaciji društva nema pa se onda ne treba ni čuditi da smo po pod-pokazateljima povezivosti i digitalne javne usluge na samom repu EU. Nedostatak sredstava nikome nije opravdanje jer EU već čitav niz godina kroz čitav niz fondova spremna je u dobroj mjeri pratiti financiranje ovakvih aktivnosti. Dakle, ne trebamo se čuditi da smo u postupku digitalizacije društva (koja je neophodna za daljnji privredni prosperitet) loše pozicionirani. Kada govorimo o digitalizaciji Hrvatskog društva mi uopće ne znamo i ne želimo znati čemu nam to služi (Nema strategije razvoja digitalnog gospodarstva, nema strategije razvoja naših računanih centara, a Strategije razvoja širokopojasnih priključaka više su popis želja nego strateški plan).

Još prije šest godina vlada je imenovala poduzeće OiV infrastrukturnim operaterom. U RH postoji na magistralnoj razini snažan optička infrastruktura koja je u vlasništvu poduzeća koja su u većinskom Hrvatskom vlasništvu, međutim ona i dalje ostaje neiskorištena, a njeno postojanje nije transparentno pokazano. U ovom procesu nad općim društvenim interesom prevladali su partikularni interesi pojedinih poduzeća i to je dovelo do postupnog odumiranja ove inicijative.

Kada se govori o ulaganjima u razvoj širokopojasne pristupne infrastrukture u Republici Hrvatskoj onda moramo znati da ukupna sredstva za ove aktivnosti nisu ni približno tako velika kao što su kod drugih kapitalnih investicija u RH, mostova, cesta, autoputova i sl., a u dobroj mjeri su praćena i EU fondovima.

Cijena koju će Republika Hrvatska platiti zbog loše i na pojedinim prostorima nepostojeće pristupne Infrastrukture bit će u daljnjem gospodarskom zaostajanju za ostalim članicama EU, te u daljnjem iseljavanju domicilnog stanovništva. Naseljena mjesta u kojima nema pristupne širokopojasne infrastrukture u skoroj budućnosti posta će nenaseljena mjesta.

4. ZAKLJUČAK

Pojava COVID-19 virusa i borba protiv njegova širenja uzrokovala je vrlo snažnu potrebu brzog prelaska na rad od kuće. Svatko tko je mogao obavljati rad od kuće u trenutku „lockdown-a“ prešao je na takav oblik rada. Problemi koji su se najčešće javljali kod prelaska na ovakav način rada nisu bili vezani za posjedovanje odgovarajućih znanja ili terminalne opreme, kao ni nedostatak programske podrške za povezivanje, već su bili vezani za kvalitetu ili nedostatak širokopojasne pristupne infrastrukture.

Kada govorimo o nastavnom procesu na daljinu (što je nešto širi pojam od učenja na daljinu) problemi koji se javljaju kod interaktivnog održavanja i praćenje nastave za disperzirane slušače vezani su u najvećoj mjeri za kvaliteta njihova terminalnih priključka širokopojasna priključka. Određeni broj studenata nije u stanju pratiti nastavni proces preko svojih kućnih širokopojasnih priključaka te zbog toga u praćenju koriste mobilni pristup preko mobitela i tableta opet uz biranje lokacije ali i smanjenja mogućnosti interaktivnog učešća u nastavi zbog ograničenja svojih krajnjih terminala.

Analiza stanja širokopojasne pristupne infrastrukture pokazala je da se po kvaliteti fiksne širokopojasne pristupne infrastrukture nalazimo na samom začelju EU. To je stanje nedopustivo jer smo po većini drugih pod pokazatelja u ili vrlo blizu EU prosjeku.

Da bi se ovo stanje promijenilo potreban je jači angažman centralne uprave, te jača koordinacija između centralne i lokale uprave. Potrebno je definirati strategiju digitalnog gospodarstva, a posebno u njoj sagledati ulogu i značaj naših nacionalnih informacijskih centara. Isključivo korištenje usluga inozemnih informacijskih centara znači odljev kapitala iz Republike Hrvatske, ali i odljev stručnih ljudi te dovodi RH u položaj ovisnosti.

Literatura:

1 Što je 5G? – Sve o mobilnim mrežama pete generacije https://mob.hr/sto-je-5g-sve-o-mobilnim-mrezama-pete-generacije/

2 Measuring the economic impact of cloud computing in Europe, European Commission.

3 Uptake of Cloud in Europa, European Commission.

4 STRATEGIJA RAZVOJA ŠIROKOPOJASNOG PRISTUPA U REPUBLICI HRVATSKOJ U RAZDOBLJU OD 2016. DO 2020. GODINE, VLADA REPUBLIKE HRVATSKE, Zagreb, srpanj 2016.

5 Strategija e-Hrvatska, Ministarstvo uprave - svibanj 2017

6 Akcijski plan za provedbu Strategije e-Hrvatska 2020, https://uprava.gov.hr/strategija-e-hrvatska-2020/14630

7 DESI pokazatelji, https://ec.europa.eu/digital-single-market/en/news/digital-economy-and-society-index-desi-2017

8 Strategija e-Hrvatska, Ministarstvo uprave - svibanj 2017

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 zvršnog 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 (C2) jezikom.

Sveučilište u Splitu
Sveučilišni odjel za stručne studije
Kopilica 5, 21000 Split Croatia

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 2021

Zlatni pokrovitelj

citus grupa
Internet Development Company

Srebrni pokrovitelj

foi

Brončani pokrovitelji

konto d.o.o.
Projektiranje informacijskih sustava

VELEUČILIŠTE U RIJECI
COLLEGIUM FLUMINENSE

ODJEL ZA INFORMATIKU
SVEUČILIŠTE U RIJECI

SVEUČILIŠTE JURJA DOBRILE U PULI

FIPU Fakultet informatike u Puli

22.02.-23.02.2021.

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, 2021

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

SUSTAV MJERENJA ORGANIZACIJSKIH PERFORMANSI BALANCED SCORECARD I DIGITALNA TRANSFORMACIJA

prof. dr. sc. Robert Fabac

SAŽETAK:

Balanced Scorecard dobro je prihvaćeni koncept koji omogućuje mjerenje performansi organizacije na strateškoj razini. Pokazalo se kroz vrijeme da Balanced Scorecard može poslužiti kao snažni alat potpore provedbi strategije i ostvarenju strateških ciljeva. Pored financijske perspektive koja se tradicionalno uvažava kao temeljna glede odabira pokazatelja uspjeha, Balanced Scorecard uključuje i perspektive kupaca, unutrašnjih procesa, te učenja i rasta. Pokazatelji odnosno mjere iz navedenih perspektiva pažljivo se odabiru i povezuju unutar strateških mapa čineći tako dinamični sustav pokazatelja s uzročno posljedičnim vezama. Novije inačice Balanced Scorecarda uvažavaju vrijednosti održivog razvoja te se unutar standardnog okvira ugrađuju veličine održivog razvoja i korporativne održivosti, kroz različite arhitekture modela „održivog Balanced Scorecarda". Na drugoj strani, rastući fenomen digitalna transformacija kao ekosustav tehnologija i aplikacija koje imaju sve intenzivniju i brojniju primjenu od strane pojedinaca, kompanija i državnog sektora uključuje komponente poput sljedećih: Internet stvari, Big Data analitika, tehnologije oblaka, umjetna inteligencija, blockchain tehnologija, 3D tisak i druge. Tako primjerice, u praćenju digitalne transformacije, različite države se mogu međusobno komparirati preko specifičnih indeksa integracije digitalne tehnologija i faktora koji omogućuju digitalnu transformaciju. Primjena pojedine tehnologije iz spektra digitalne transformacije u poduzeću ima utjecaj na pojedine funkcije poduzeća te ima svoj karakterističan ishod u smislu otvaranja novih tržišta, poboljšanja proizvoda i slično. Iz razloga uspješnog ostvarenja strateških razvojnih ciljeva i promjena, a koje uključuju inicijative digitalne transformacije, potrebno je razmotriti integraciju pokazatelja i mjera u mapama Balanced Scorecarda. Dakle, analiza utjecaja i veza između inicijativa digitalne transformacije te ključnih ciljeva i mjera tradicionalnih perspektiva u lancu vrijednosti, predstavlja doprinos prema boljem razumijevanju performansi i ostvarenja strateških ciljeva organizacije, financijskih i drugih.

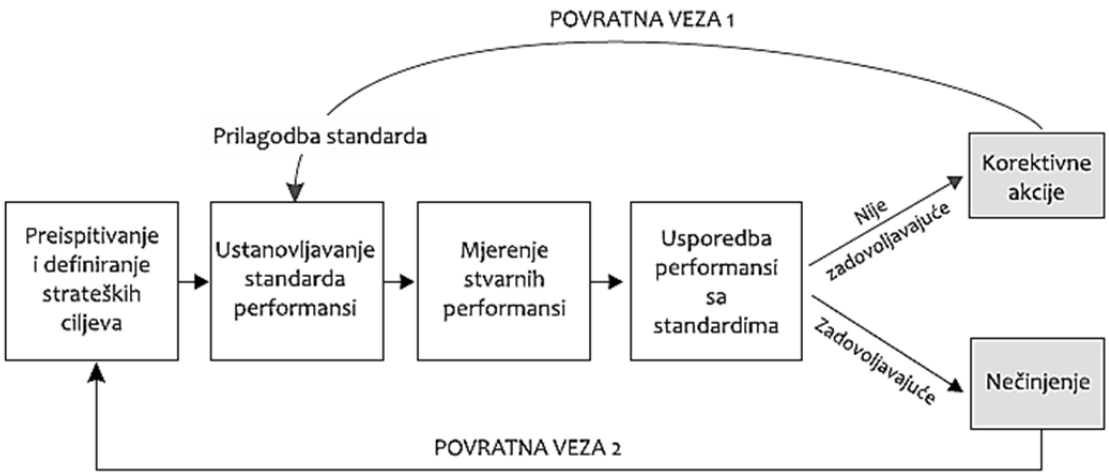
1. UVOD – MJERENJE PERFORMANSI

Balanced Scorecard (BSC) kreirali su devedesetih godina prošlog stoljeća R.S. Kaplan i D. P. Norton kao alat za mjerenje organizacijskih performansi. Nasuprot tradicionalnom gledištu koje preferira financijske pokazatelje pri evaluaciji performansi, model BSC uključuje i pokazatelje iz drugih važnih organizacijskih perspektiva te ukupni okvir obuhvaća: učenje i rast, poslovne procese, kupce, te financije.

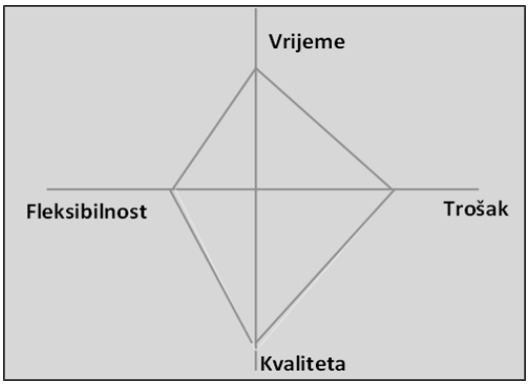
Iz same definicije performansi, kao primjerice u rječniku Merriam-Webster (dostupno na https://www.merriam-webster.com/dictionary/performance), pojam performanse odnosi se na izvršenje neke akcije, na sposobnost izvršenja, na ostvarenje. Primjenjujući koncept performansi u kontekstu organizacije razmatra se organizacijska učinkovitost, odnosno cilj je u sagledavanju rezultata organizacijskih aktivnosti. Samo mjerenje rezultata odnosno performansi pretpostavlja postojanje odgovarajućeg modela, alata, definiranog postupka za procjenu i mjerenja učinkovitosti organizacije.

Mjerenje organizacijskih performansi tijekom vremena je evoluiralo od prakse tzv. kibernetičkog gledišta gdje su fokusirane financijske mjere, prema praksi sukladno holističkom gledištu pri čemu su tada uključene i druge mjere osim financijskih. Ideja kontrole sadržana je u sustavskom, kibernetičkom pristupu i ona podrazumijeva aktivnosti korekcije. Treba naglasiti da mjerenje performansi ne podrazumijeva samo stratešku organizacijsku razinu već nerijetko organizacije prate različite i brojne indikatore koji nemaju presudan utjecaj na njihovu ukupnu uspješnost. Pojedini alati kao što je Balanced Scorecard, kroz mjerenje performansi ostvaruju ključnu ulogu u razvoju strateških planova i vrjednovanju ostvarenja organizacijskih ciljeva. (Henri, Jean-Francois, 2004), (Ittner, C. D. & D. F. Larcker, 1998), (Spitzer, D. R., 2007).

Rummler i Brache (2013) analiziraju općeniti pristup performansama organizacije te navode tri glavna aspekta odnosno razine: organizacijsku, procesnu te onu radnog mjesta (djelatnika-izvršitelja). Za konzultantske i IT tvrtke, razmjeno su najčešći projekti poboljšanja poslovnih procesa, koji nemaju osobite performanse a mogu se unaprijediti. Suglasnost većeg broja teoretičara postoji glede nekoliko **glavnih dimenzija performansi procesa.** To su *vrijeme, trošak i kvaliteta; a zbog* opcije promjena pridodaje se ovdje i četvrta varijabla, a to je *fleksibilnost*. Svaka od navedenih dimenzija performanse može se raščlaniti dalje u mjere poznate kao *key performance indicators (KPI)*. Poboljšanje procesa kroz skraćivanje vremena često donosi izazov održanja kvalitete, dok ostvarenje fleksibilnosti treba balansirati s veličinom troška, itd.



Slika 1. Mjerenje performansi i korekcije (prema Daft, R.L., 2009)



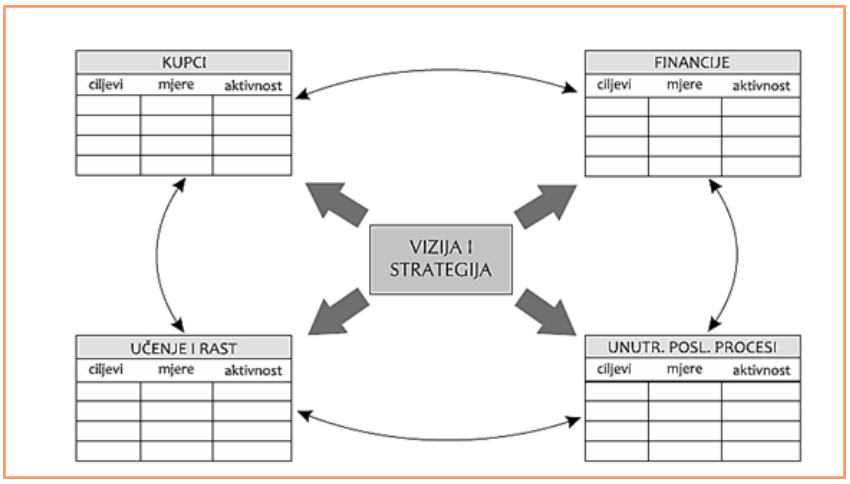
Slika 2. Mjerenje performansi procesa – „đavolji četverokut“ (prema Dumas, et al , 2013)

Balanced Scorecard (BSC) uključuje poslovne procese kao jednu od svojih perspektiva unutar koje treba sagledavati pokazatelje. Ovaj alat je strateškog karaktera jer kroz mjerenje performansi podupire provedbu strategije kako je zamišljena. BSC očvršćuje i podupire dugoročne ciljeve na račun kratkoročnih uspjeha, sugeriranjem formuliranja potrebnih aktivnosti i projekata. Aktiviranje modela BSC-a izrazito je korisno kod primjene razvojnih strategija, kod organizacijskih transformacija tj. većih organizacijskih promjena. Po ovoj analogiji BSC može osigurati i potporu provedbi **digitalne transformacije**. Usklađivanje strategije digitalne transformacije s drugim organizacijskim strategijama i programima izazovan je poduhvat, o čemu su pisali Matt et al (2015), između ostalih. BSC model ima određene važne atribute koji mogu značajno doprinijeti uspješnoj implementaciji digitalne transformacije, što je objašnjeno u nastavku.

2. BALANCED SCORECARD

BSC predstavlja „…sustav mjerenja i upravljanja izveden iz strategija i sposobnosti…. te on prevodi organizacijsku viziju i strategiju u koherentan skup mjera.” BSC je kreiran kao više - dimenzionalni izbalansirani skup mjera koji ujedinjuje indikatore iz četiri perspektive:

- *financijska, kupci, unutrašnja perspektiva te učenje i rast* (Norton& Kaplan, 1992)



Slika 3. Perspektive BSC-a (prema Norton& Kaplan, 1992)

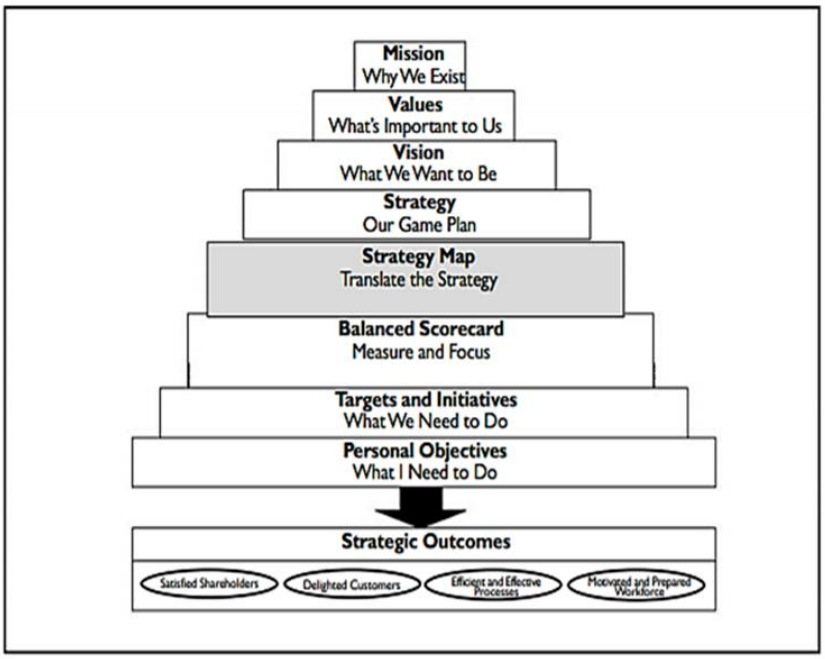
Prije pojave BSC-a u domeni potpore provedbi strategije, općenito je **preveliki naglasak bio na kratkoročnim financijskim ciljevima** i ciljevima budžeta. Menadžeri u brojnim kompanijama nerijetko odbijaju investicije u razvoj i inovacijski potencijal čime ostvaruju financijske uštede i mogu prezentirati dobre kratkoročne rezultate, kroz kvartalna i druga izvješća. Tako se stvara privid da je stanje performansi organizacije dobro ali ovakvo ponašanje vodi prema gašenju pokretača razvoja, zaostajanju u prihvaćanju novih tehnologija i općenito prema slabljenju konkurentnosti.

Osim problema prenaglašavanja izoliranih financijskih ciljeva i pokazatelja, još jedan izazov uspješno je moguće adresirati primjenom odgovarajućeg BSC modela mjerenja performansi. Riječ je o fenomenu kojeg navodi Mintzberg (1987), o zabrinjavajućem **jazu između strategije izražene kroz planove i strategije koja se pokazuje u praksi,** temeljem stvarnih aktivnosti. Proizvoljne i slobodne interpretacije strateških formulacija vode prema različitim mogućim izvedbama strategije na nižim razinama – kroz okvir sadržanih organizacijskih projekata, programa, inicijativa. Smanjenje ovog jaza i potpora provedbi strategija može se poduprijeti uporabom primjerenih alata, među koje pripada BSC (poznat je tako i sustav tzv. PPBS koji se primjenjuje sa sličnom svrhom u ministarstvu obrane SAD; primjerice Haynes (1992))

Balanced Scorecard integrira mjere financijskog i nefinancijskog karaktera pri čemu:

- Financijska perspektiva identificira kako kompanija želi biti viđena od svojih dioničara i interesnih grupa
- Perspektiva kupaca određuje kako kompanija želi izgledati gledana „očima kupaca“
- Unutrašnja perspektiva (procesna) prepoznaje procese u kojima kompanija želi biti osobito dobra, da bi zadovoljila kupce i dioničare
- Perspektiva učenja i rasta uključuje potrebne promjene i poboljšanja radi ostvarenja vizije, uključuje usavršavanje zaposlenih

Ključne zadaće i aktivnosti u vezi implementacije modela Balanced Scorecard u organizaciji poželjno je da slijede proceduru: pojašnjenje misije i vizije, identificiranje strateških ciljeva, formuliranje implementacijskih strategija (SWOT) za strateške ciljeve, definiranje prioritetnih organizacijskih inicijativa, te pripadnih ciljeva i mjera kroz perspektive (slika 4.)



Slika 4. Piramida BSC-a: od misije do strateških ishoda (R.S. Kaplan & D.P. Norton, 2001)

Izradu modela BSC-a za javni sektor, pojašnjavamo na primjeru potpore ostvarenju ciljeva iz dokumenta „Dugoročni plan razvoja oružanih snaga Republike Hrvatske 2006. – 2015.“ (DPR) (slučaj BSC-a koji je obrađen u materijalima Ž. Dobrovića (nastava na kolegiju Mjerenje organizacijskih performansi, FOI Varaždin, Zagrebačko sveučilište). Plan razvoja predstavlja zapis željenog budućeg organizacijskog razvoja, dakle uključujući i organizacijske promjene koje će se dogoditi ostvarenjem strateških ciljeva. Za promjene koje bi bile iz kategorije digitalne transformacije možemo reći kako je temeljna logika analogna.

Slika 5. sadrži izvadak iz navedenog DPR-a s fokusom na razvojne ciljeve organizacije. Među svim navedenim ciljevima izdvajamo M1: „Uspostava ciljane organizacijske i personalne strukture“ (Hrvatski sabor, NN 81/2006).

Model BSC za organizaciju razvija se po proceduri sa slike 4, pri čemu su ključni prvi koraci. U svrhu razrade strateških ciljeva moguće je primijeniti metodu koja se bazira na SWOT analizi, i ta je metoda prikazana u značajnom broju radova.

Dakle, za odeđeni strateški cilj identificiraju se SWOT elementi (snage, slabosti, vanjske prilike i vanjske prijetnje). Nakon toga formuliraaju se moguće (SWOT) strategije uparivanjem karakterističnih elemenata: S-O (iskorištavanje vanjskih prilika angažiranjem unutrašnjih snaga), S-T (angažiranje unutrašnjih snaga za neutraliziranje vanjskih prijetnji), i druge. Pojedini elementi povezani s resursima koji stvaraju pozitivan učinak moraju biti dominantni ili relativno komparabilni naspram elemenata negativnih učinaka.

Razvojni ciljevi OS RH razrađeni su za područja ljudskih resursa, vojne izobrazbe, materijalnih resursa, logistike, doktrine i obuke, međunarodne vojne suradnje te istraživanja i razvoja.

Glavni razvojni ciljevi OS RH u razdoblju od 2006. do 2015. godine su:

Ciljevi usmjereni na dosezanje vojnih sposobnosti

1. Uspostaviti ciljanu organizacijsku i personalnu strukturu.
2. Opremiti OS RH potrebnim naoružanjem i vojnom opremom.
3. Pojačati sudjelovanje u međunarodnim vojnim operacijama.

Ciljevi usmjereni na uređenje doktrinarno-normativnog okvira

4. Uspostaviti ciljani sustav obuke i izobrazbe.
5. Prilagoditi zakonsku i drugu regulativu novom obrambenom konceptu, usvojiti potrebne doktrinarne dokumente.

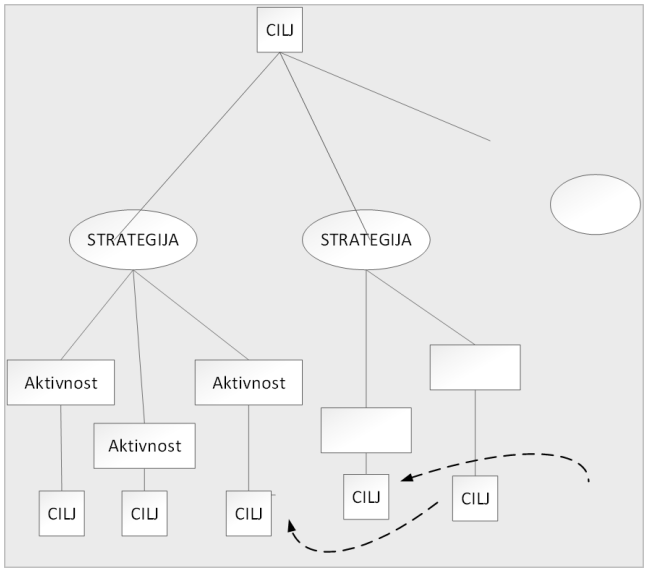
Ciljevi usmjereni na povećanje učinkovitosti OS RH

6. Popuniti OS RH dragovoljcima (profesionalnim vojnim osobljem, vojnicima-dragovoljnim ročnicima i pripadnicima ugovorne pričuve).

Slika 5. Izvadak iz dokumenta „DPR OS Republike Hrvatske 2006. – 2015.

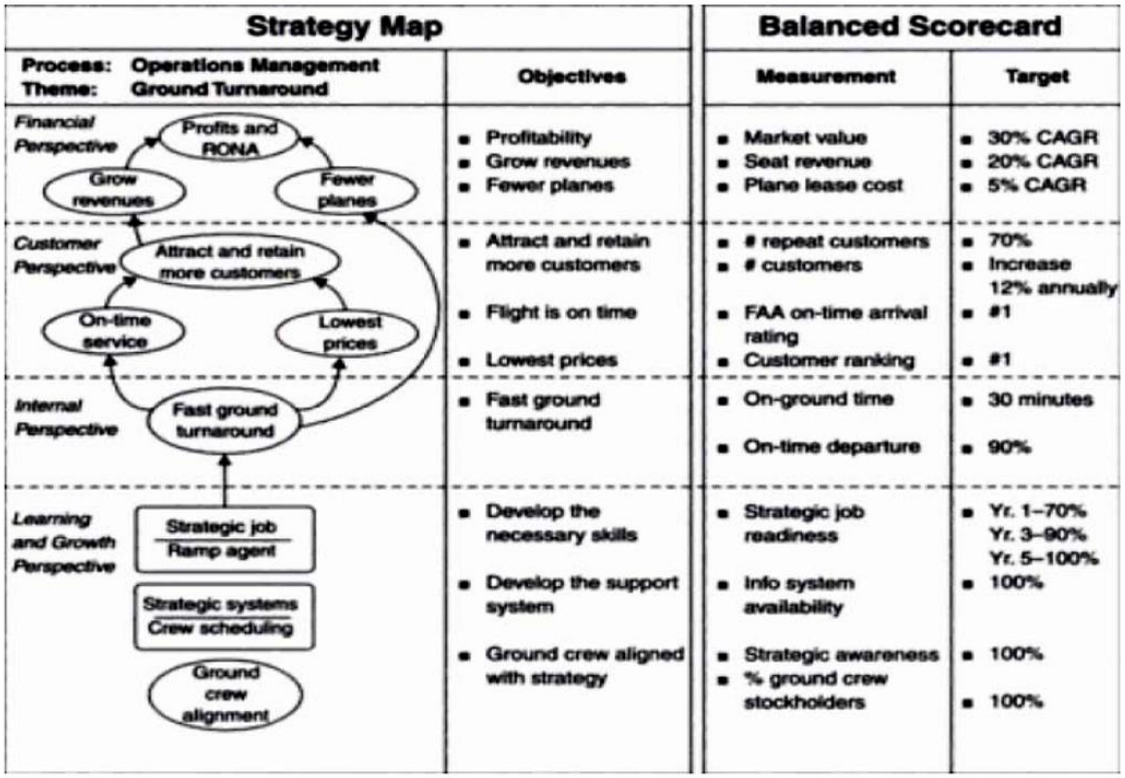
Za strateški cilj M1 a tako i za druge ciljeve, razrada kroz BSC model provodi se najčešće uz pomoć tablice u koju upisujemo elemente: Strategije – Aktivnosti – Ciljevi – Lanac uzroka i posljedica. Tako primjerice strategija S2-W2 podrazumijeva poduzimanje aktivnosti uslijed nezadovoljavajuće trome organizacijske strukture. U okviru ove strategije poduzimaju se aktivnosti edukacije djelatnika o potrebi promjena te implementacija odgovarajućeg sustava nagrađivanja. Za prvu aktivnost definira se cilj glede obima ciljane edukacije voditelja (oznaka M1.U1) a za drugu aktivnost - cilj pridobivanja glavnine djelatnika (M1.I1). (analogno razradi danoj u Dobrović, Ž.). Ovdje moramo primijetiti da kod kreiranja BSC modela prije određivanja samih mjera i imamo opredjeljenje za prioritetne inicijative (aktivnosti) koje doprinose ostvarenju strateških ciljeva. Na taj način BSC izravno **podupire provedbu strategije**, jer njegova dosljedne primjena nalaže gdje će se angažirati organizacijski resursi u budućnosti.

Razrada strateškog cilja ima hijerarhijsko i proceduralno obilježje, te kroz niz koraka ovdje formuliramo (kao na slici 6.): prvo strategije (SWOT), zatim pripadajuće aktivnosti za provedbu tih strategija, pa onda ciljeve koji idu uz aktivnosti (koji su općenito raspodijeljeni u svim perspektivama BSC-a). Ovi ciljevi trebaju biti uzročno-posljedično povezani. Ciljevi nižih razina (učenje i rast; procesi) pretpostavljamo da ostvaruju učinak na ciljeve viših razina. U nastavku formuliranja BSC-a, svakom pojedinom cilju pridružuje se mjera – potrebno je opisati za svaku mjeru – način mjerenja, granične vrijednosti, utjecaj na druge mjere (u skladu s utjecajima ciljeva).



Slika 6. Ilustracija formuliranja aktivnosti i ciljeva niže razine u BSC modelu.

Na slici 7, prikazan je cjeloviti okvir jednog BSC modela (Kaplan & Norton, 2004). Ono što je izostalo, to su opisi (prioritetnih) inicijativa, aktivnosti. Inicijative moraju podupirati ostvarenje ciljeva (inače ih organizacija može zanemariti). U slučaju digitalne transformacije na koju bi se primijenio BSC, riječ je o aktivnostima glede digitalizacije, iz spektra poznatih tehnologija koje se navode kasnije.



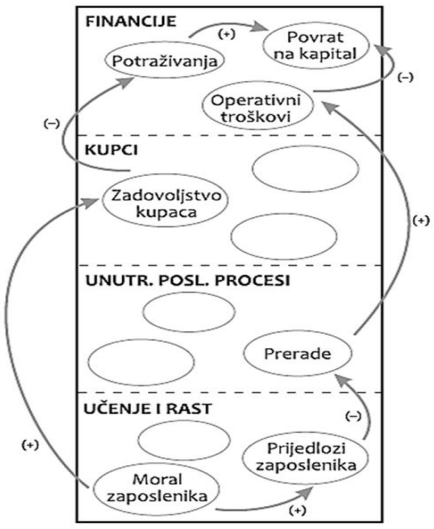
Slika 7. Model BSC (Kaplan & Norton, 2004).

Na slici 7, s lijeve strane vidimo dio koji predstavlja karakterističnu stratešku mapu. Strateška mapa ilustrira međudjelovanje ciljeva (i mjera) iz perspektiva, sve do ostvarenja glavnih – financijskih ciljeva. Za organizacije iz sektora gospodarstva, dakle za profitni sektor, krajnji ciljevi pozicionirani su u domeni financijske perspektive. Ako primijenimo BSC na organizaciju iz javnog sektora, tada su ciljevi na vrhu – oni ciljevi iz domene misije organizacije.

Većina autora izvješćuje o BSC modelima koji su strukturirani hijerarhijski, dakle jedna perspektiva te njezini ciljevi i mjere striktno ispod neke druge perspektive. Međutim, imamo i koncepte nehijerarhijskih strateških mapa, dakle različite arhitekture, o čemu se može pročitati u radu Hansen & Schaltegger (2014). Mjere iz nižih perspektiva pretežito su pokretači (lead-indikatori) dok su mjere iz gornjih perspektiva uglavnom posljedične (lag-indikatori).

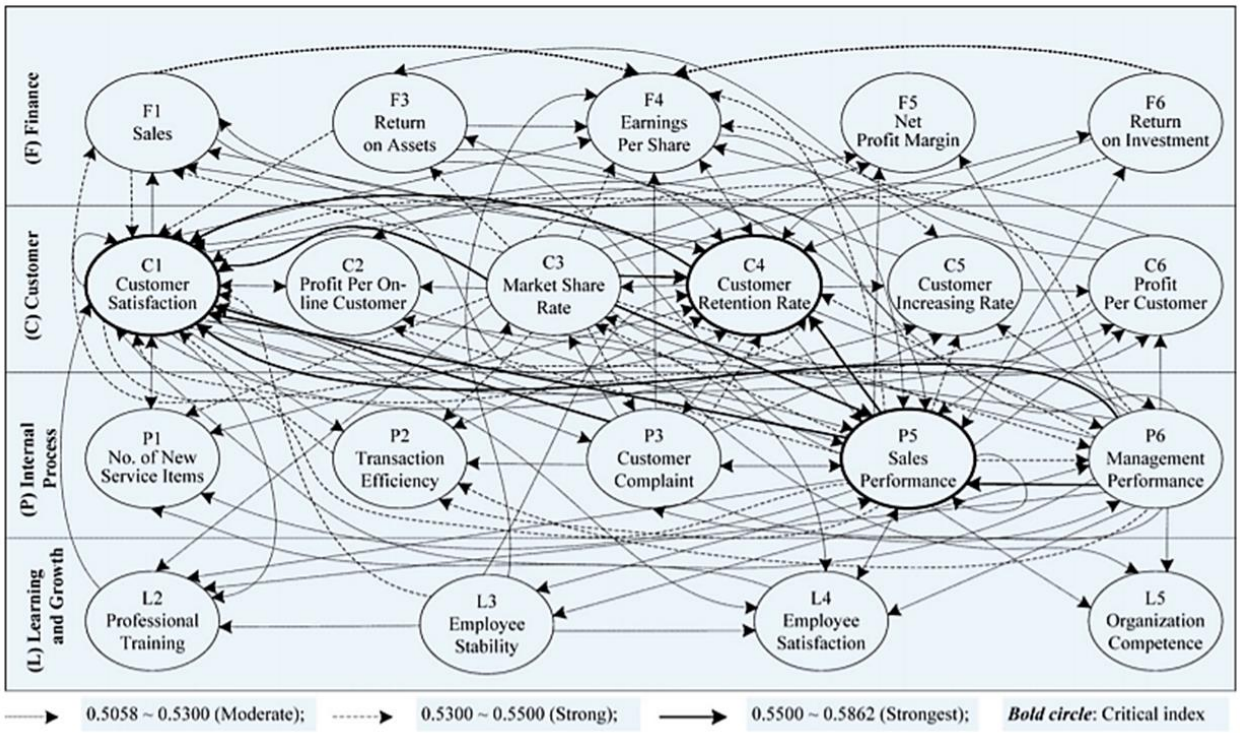
3. POVEZANOST CILJEVA I MJERA KROZ STRATEŠKE MAPE BSC-A

Strateške mape ciljeva i mjera prikazuju sve ciljeve (mjere) u nekom modelu Balanced Scorecarda te njihov međusobni utjecaj odnosno pozicioniranje. Da bi neki BSC model podupirao provedbu strategije tada raspored mjera i ciljeva u mapama mora biti dosljedan te ne može postojati neki indikator ili cilj koji nije povezan s drugim ciljevima. Na ovim uzročno-posljedičnim vezama zasniva se struktura strateških mapa (slika 8). Zbog navedenih uvjeta postoji izazov modeliranja digitalne transformacije uz primjenu BSC okvira, jer bilo koji smisleni indikator glede digitalne transformacije postaviti bilo u mapu – to nije dovoljno. U primjeru mape hipotetske organizacije vidimo da „moral zaposlenika“ može djelovati na „zadovoljstvo kupaca“, i tako dalje.



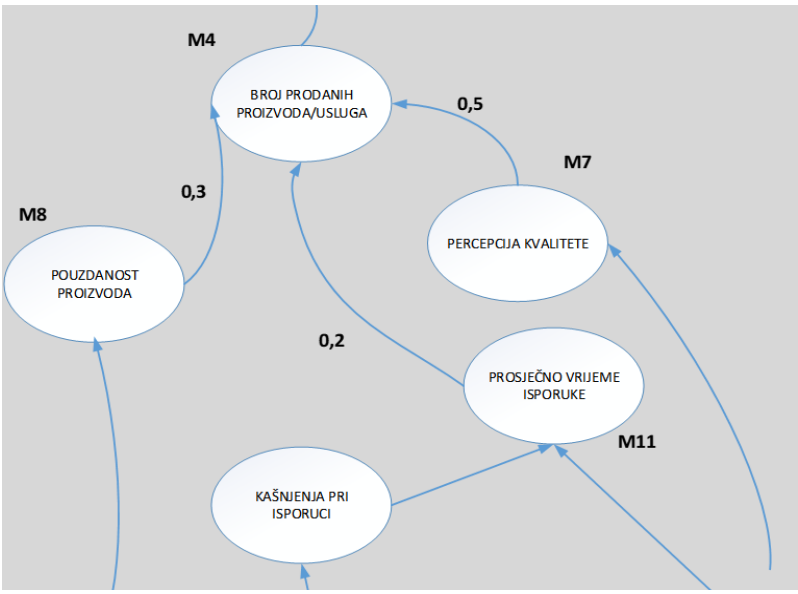
Slika 8. Strateška mapa za model BSC – ilustracija (Fabac, 2020)

Strateške mape mogu biti izuzetno složene, kao što se vidi na primjeru iz sektora bankarstva (slika 9, Hung-Yi Wu, 2011). Treba primijetiti da svaki model BSC-a zahtjeva validaciju, te je za ovako složeni model koji je prikazan na slici 9, takav zahtjev izazovan. Napor se treba uložiti i pri ažuriranju BSC-a, temeljem spoznaja o poslovanju koje se kreiraju kroz vrijeme.



Slika 9. Strateška mapa – primjer iz sektora bankarstva (Hung-Yi Wu, 2011).

Strateški ciljevi razrađuju se kroz perspektive i svega nekoliko pokazatelja koji su okosnica sustava BSC unutar različitih perspektiva - pokazatelji su povezani s ključnim projektima, inicijativama, aktivnostima koje se trebaju ostvariti u organizaciji.

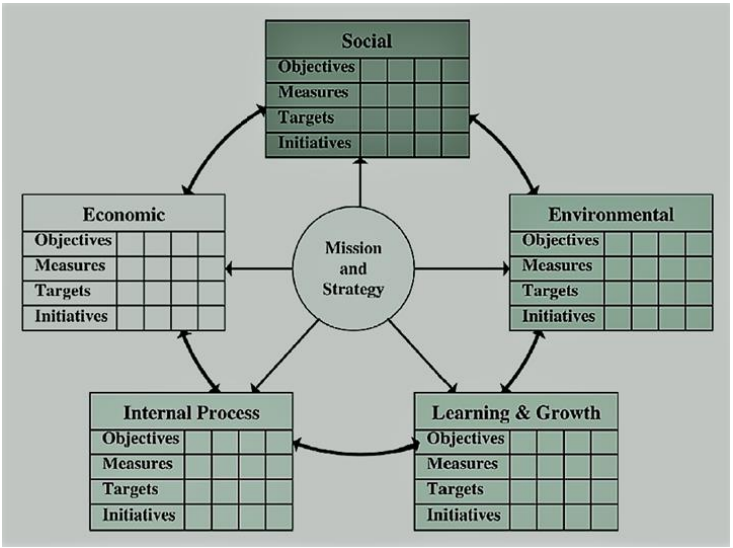


Slika 10. Izvadak iz hipotetske BSC strateške mape.

Uzročno posljedične veze u mapi na slici 10, prikazuju da mjera M4 (broj prodanih proizvoda i usluga) ima funkcijsku ovisnost *f* (M7, M8, M11). U pravilu se pretpostavlja linearna ovisnost tako da M4 zavisna mjera trpi utjecaj tri mjere uz neke težinske faktore (primjerice 0.5; 0.3; 0.2). Određenje ovih težinskih faktora zasebna je priča i nije posve trivijalno kako ih odrediti, ali je ta „preciznost“ od iznimne važnosti za ispravnost modela. Kod skupine analitičara može se primijeniti neka od metoda više-kriterijalnog vrednovanja (primjerice AHP).

4. PRIMJENE SUSTAVA S-BSC

Odgovarajuće inačice BSC-a uspješno su tijekom godina primjenjivane u raznim industrijama privatnog te u organizacijama javnog sektora. U novije vrijeme razmjerno veliku popularnost ostvario je koncept **BSC-a održivog razvoja.** Veći broj istraživanja ponudio je različita rješenja piramide (Bieker T., 2003) i ugradnje **komponenata održivosti** (eng. sustainability) u izvornu strukturu BSC-a (primjerice kao na slici 11).

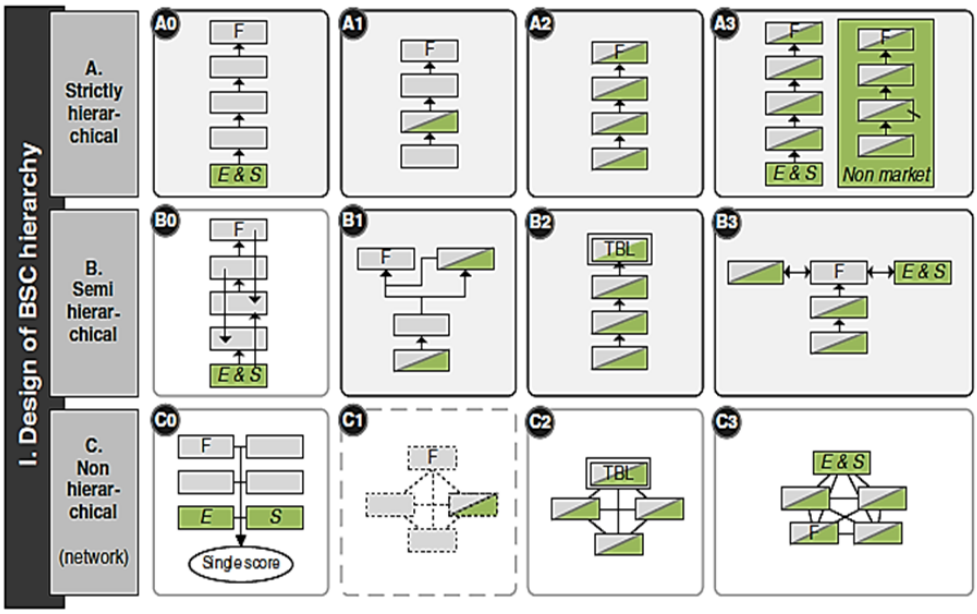


Slika 11. S- BSC tradicionalne 4 perspektive i perspektiva „environmental“ (Rabbani et al, 2014)

Među najistaknutijim globalno poznatim kompanijama koje primjenjuju BS, prema izvoru https://www.clearpointstrategy.com) nalaze se, u različitim industrijama:

- Automotive: Volkswagen, Ford Motor Company,
- Banking: Wells Fargo, Citibank, TD Canada Trust
- Energy: Mobil North America Marketing and Refining (NAM&R)
- Healthcare: Sunnybrook Health Sciences Centre at the University of Toronto Hospital
- Manufacturing: Borealis, FMC Corporation
- Technology: Apple, Microsoft Latin America
- Telecommunications: Verizon, AT&T;
- Te istaknimo da uz neke kompanije pouzdano ide i model **S-BSC,** a tu su: BMW, Daimler i dr.

Glede strateških mapa BSC-općenito, ali i posebno u vezi S-BSC modela, treba istaknuti kako nije nužno da se formiraju strogo hijerarhijske inačice. O mogućim arhitekturama u svojem radu daju razmatranja Hansen & Schaltegger (2014). Održivost općenito ima komponentu brige o okolišu („zelena industrija“), ali uključuje i komponentu društvene odgovornosti. Ovdje postoji dodirna točka s konceptom koji ima svoju popularnost unutar strateškog menadžmenta – „corporate social responsibility“.



Slika 12. Moguće izvedbe modela S-BSC (prema Hansen & Schaltegger, 2014).

5. DIGITALIZACIJA I DIGITALNA TRANSFORMACIJA – INDIKATORI

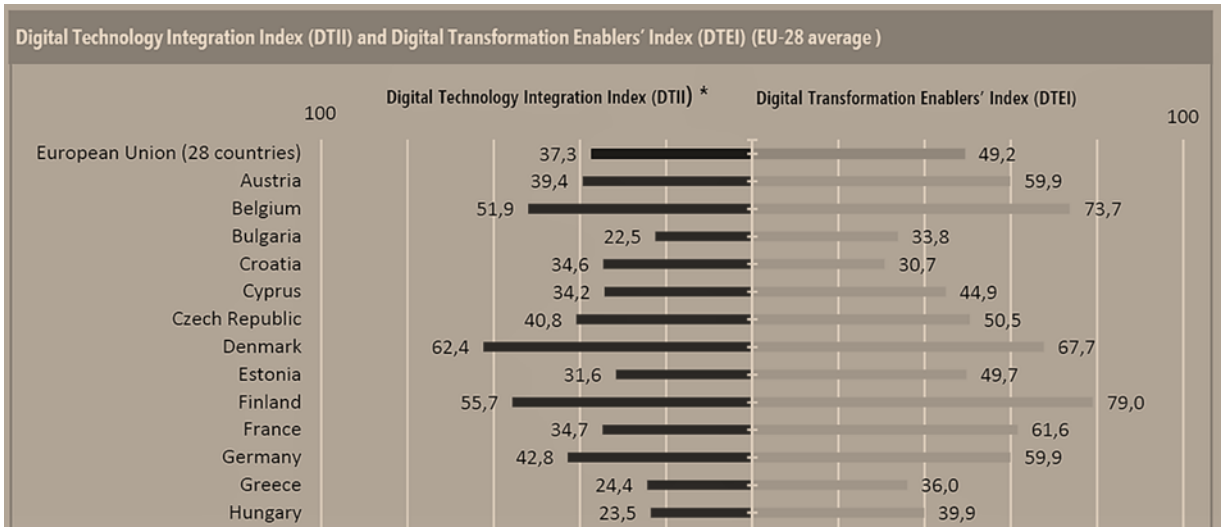
Digitalna transformacija pojavljuje se kao globalni fenomen uslijed kombiniranog funkcioniranja stalnih mobilnih veza i pojave širokog spektra novih aplikacija, proizvoda i usluga, čime se ostvaruje rastući ekosustav tehnologija i aplikacija koje primjenjuju pojedinci, kompanije i državni sektor. **Digitalizacija i digitalna transformacija** u modernim organizacijama pojavljuju se kao svojevrsna **nova paradigma** koja vodi prema poboljšanju organizacije, te je preduvjet osiguranja konkurentnosti i konkurentske prednosti.

U spektar identificiranih novih tehnologija, pripadaju Big data & data analytics, rješenja kibernetičke sigurnosti, robotika i automatizirani strojevi, 3D tisak, mobilne usluge, društveniji mediji, tehnologije oblaka (Cloud technologies), Internet stvari, umjetna inteligencija. Ove tehnologije primjenjuju se u lancu stvaranja vrijednosti kod naprednih organizacija te se govori o digitalizaciji usluga, digitalizaciji procesa, digitalizaciji proizvoda, i tako dalje.

Za cijele države ili nacije poželjno je ostvarivanje digitalizacije pa tako Europska Unija prati prihvaćanje digitalnih tehnologija u društvu i u poslovnim aktivnostima za zemlje članice. Podaci i odgovarajući indikatori mogu se pronaći u dokumentu Transformation Scoreboard (slika 12, DTS verzija iz 2018.). Ovaj dokument daje uvid u relevantnu statistiku i potporu digitalnoj transformaciji, osigurava izvješća glede tehnoloških i industrijskih izazova te pregled političkih (nacionalnih) inicijativa iz domene **DT.**

U navedenom dokumentu EK, države EU međusobno se kompariraju preko **specifičnih indeksa** integracije digitalne tehnologija i faktora koji omogućuju digitalnu transformaciju - Digital Technology Integration Index te DTII te Digital Transformation Enablers' Index – DTEI.

Digital Transformation Scoreboard (DTS) uključuje nadgledanje digitalne transformacije postojećih industrija i poduzeća, uvažava nacionalne pokazatelje za praćenje digitalne transformacije u Europi s geografskim fokusom i iz makro-perspektive.



Slika 13. Komparacija zemalja EU (prema Digital Transformation Scoreboard 2018)

Sastavni elementi podupiratelja (enablers) DT, sadrže podsustave: digitalna infrastruktura, investicije i dostupnost financiranja, raširenost digitalnih vještina, e- vodstvo (edukacija i usavršavanje glede digitalnih vještina), poduzetnička kultura. Za navedena podsustave postoje pripadni skupovi indikatora.

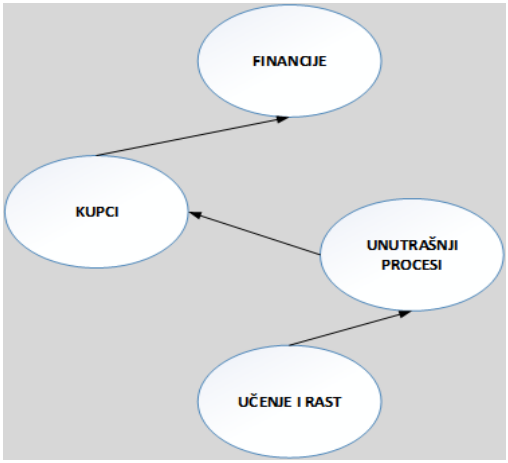
Po pitanju indikatora DT općenito treba uočiti razliku između područja DT koja obuhvaćaju „skupine organizacija" (nacionalno gospodarstvo, javni sektor, pojedine industrije i sl.) te područja poduzeća, „pojedinačne organizacije", razine odjela, timova, pojedinaca. Veliki je značaj digitalizacije javnog sektora (Mergel et al, 2019). Tako se karakteristični pokazatelji za e-Upravu u vezi realiziranih primjena (broj online servisa, postotak ureda vlade s web-stranicama, postotak građana koji koriste vladine web-stranice s elektroničkim uslugama, i dr.; primjerice u Janssen et al, 2004).

5.1 Indikatori DT u perspektivama BSC-a

Jedno produktivno sagledavanje digitalne transformacije je glede tzv. komponenata DT: Big Data, Cloud Computing, Digital Security, Mobility Management, pri čemu se analizira primjena novih tehnologija s utjecajem na tri organizacijske dimenzije: vanjska dimenzija (s naglaskom na digitalno snaženje iskustva korisnika/kupaca), unutarnja dimenzija (u djelovanju na poslovne operacije (procese), odlučivanje i organizacijsku strukturu) te holistička dimenzija (gdje su zahvaćeni svi poslovni segmenti i uvodi se novi poslovni model) (Kaufman & Horton, 2015; Hess et al, 2016; M. H. Ismail, et al, 2017).

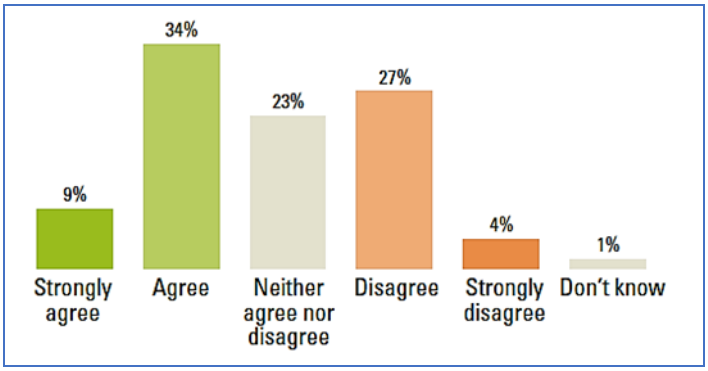
5.2 Perspektiva učenje i rast

Upravo ova navedena podjela fokusira **perspektive BSC-a,** dakle kod implementacije DT može se koncipirati paralelno sa BSC hijerarhijom strukturom strat. mapa ili kao dio BSC modela (slika 14). Ako razmotrimo DT u modelu/sustavu BSC-a unutar **perspektive rast i učenje**zaposlenih, tada se opravdanost povezivanja i mogućnosti formuliranja indikatora naziru u većem broju radova poput Schuchmann & Seufert (2015) ili (Verhoef et al (2021) gdje se razmatraju resursi digitalne transformacije pa tako i ljudski resursi. U studiji MIT & Deloitte (2016) analiziraju se barijere provedbi digitalne transformacije.



Slika 14. Komparacija Perspektive BSC-a i naznaka uzročno-posljedičnih veza.

Tako u vezi pitanja: imaju li zaposlenici dovoljno znanja i sposobnosti za izvršenje digitalne strategije, distribucija odgovora je prikazana na slici 15. Vidi se da je veća razina slaganja s tvrdnjom u ne tako velikom broju slučajeva (43%). Slična distribucija odgovora (zeleno ukupno 47%) dobivena je glede tvrdnje da organizacija zaposlenika i njegove kolege dovoljno opskrbljuje resursima i mogućnostima stjecanja potrebnih vještina da bi se iskoristili rastući digitalni trendovi.



Slika 15. Znanja i sposobnosti zaposlenika za izvršenje strategije DT (MIT & Deloitte, 2016)

Isto istraživanje upućuje da za organizacije veće zrelosti glavne barijere kod digitalne transformacije su: previše prioriteta, brige glede sigurnosti, **nedovoljna tehnička znanja, vještine**.

5.3 Perspektiva kupaca te organizacijski proizvodi i procesi

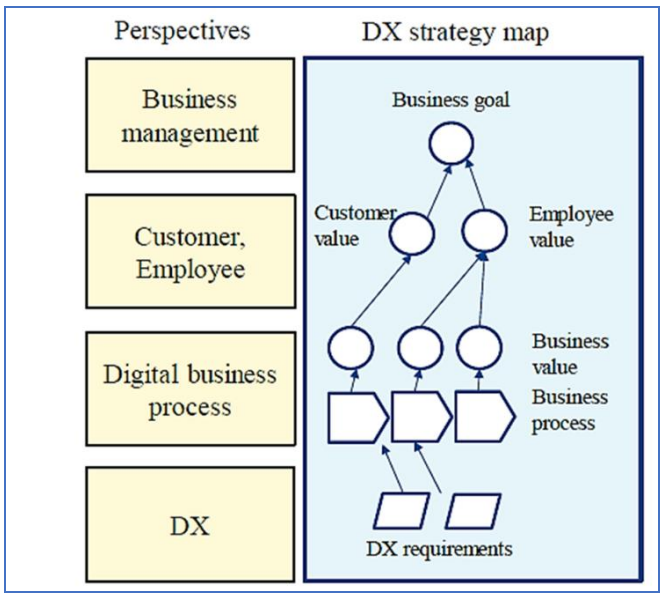
Razmještaj indikatora digitalne transformacije treba sagledavati i naspram položaja indikatora (mjera, ciljeva) tradicionalnog BSC-a, u perspektivama tradicionalnim ili novim (primjerice F: profit, ROE, i sl.; K: veći broj kupaca, novi kupci i sl.; P: skraćivanje trajanja, postotak digitaliziranih procesa, i dr.).

Ako se usmjerimo na digitalnu transformaciju i pojave u **perspektivi kupaca** u BSC modelu, tada gotovo neizbježno zahvaćamo gotovo cijelu mapu uzročno-posljedičnih veza (slika 14.). Primjer njemačke tvrtke Ravensburger analiziran je u radu Hess et al (2016). Prijašnji *core-business* ovog poduzeća bile su analogne igre i knjige za djecu. Organizacijska promjena uslijed digitalne transformacije učinila je da Ravensburger ulazi na novo tržište - „e-book & online gaming". Kako bi se uspješno transformirala, tvrtka je ustanovila potrebu snaženje (digitalnih) kompetencija i sposobnosti – dakle nužnost pokretanja inicijativa uvježbavanja/educiranja osoblja te zapošljavanja novih ljudi. Ovaj primjer i neki drugi (primjerice u Chanias et al, 2019), gotovo tipičnog prelaska na digitalizaciju u domeni kupaca (BSC, slika) zahtjeva i promjene u tzv. nižim perspektivama poslovnih procesa te učenja i rasta. Glede druge perspektive BSC-a (procesi) Mergel et al (2019), primjerice, naglašavaju važnost poslovnih procesa kod teme digitalne transformacije, ističući da su procesi najčešći objekt kod provedbe DT.

6. BSC U POTPORI DIGITALNOJ TRANSFORMACIJI

Izravna primjena te mogućnosti primjene BSC modela za digitalnu transformaciju izložena je u nekoliko radova novijeg datuma. Strateške mape perspektiva za digitalnu transformaciju globalnih linija za prijevoz kontejnera razradili su Maydanova et al (2019). Mogući modeli ugradnje elemenata DT u BSC dobro se naznačuju u primjeru kojeg je ilustrirao autor Yamamoto (2020).

Slijedom formulacije indeksa DXPI (Digital Transformation Promotion Indeks) od strane japanskog ministarstva (Ministry of Economy, Trade and Industry) koji uključuje kvalitativne i kvantitativne pokazatelje, Yamamoto identificira te ilustrira digitalni Balanced Scorecard kao jednu od metoda potpore digitalnoj transformaciji (slika 16).



Slika 16. Model BSC-a u potpori digitalnoj transformaciji (Yamamoto, 2020)

Treba primijetiti da se koncept digitalnog BSC kojeg Yamamoto predlaže razlikuje od klasične inačice BSC-a posebno u prvoj perspektivi (na dnu), koja je ovdje glede zahtjeva digitalne transformacije.

Kroz izvedbu hijerahijskog utjecaja prema višim perspektivama, trebali bi se ostvariti i potrebni ciljevi koji autor naziva elementima, te razlikuje po mapama: (BM- veći profit, smanjenje troška; CE- vrijednost, kupca i zaposlenika; DBP- digitalni eko sustav; DX – digitalizirani podaci).

Nadalje, u radu se razmatra ideja slojeva koji predstavljaju različite digitalne sposobnosti, a one se povezuju kroz strateške mape sukladno prioritetima poslovnih ciljeva. Istaknimo primjer klasifikacije digitalnih sposobnosti Zhou and Wu (2010):

- Pribavljanje ključnih digitalnih tehnologija
- Identificiranje novih digitalnih mogućnosti
- Odgovor na digitalnu transformaciju
- Upravljanje *state-of-the-art* digitalnim tehnologijama
- Razvoj inovativnih proizvoda i usluga primjenom digitalne tehnologije

7. ZAKLJUČAK

Balanced Scorecard je strateški alat mjerenja performansi i upravljanja performansama, kroz podupiranje ostvarenja strateških i razvojnih ciljeva organizacije. Na drugoj strani, digitalna transformacija (DT) u pravilu je danas visoki prioritet za menadžere brojnih organizacija. Da bi se elementi ili komponente digitalizacije sustavno uvodili u organizaciju, trebaju biti dio strategije DT. Strategija DT (SDT) može se provoditi učinkovito integriranjem u razvojne i strateške dokumente organizacije. Međutim, ta implementacija predstavlja izazov s obzirom na prioritete, ograničene resurse, ranija opredjeljenja i drugo.

Osobina modela BSC je da se njegovom primjenom glavni ciljevi iz organizacijskih razvojnih dokumenata/strategija – dosljedno raščlanjuju u operativne aktivnosti (inicijative, projekte), s pripadnim međusobno povezanim ciljevima u perspektivama strateških mapa.

Ako je dijelom ukupne strategije, ili skupa glavnih strategija organizacije tada SDT postaje sastavnica koja ulazi u operacionalizaciju te se definiraju i provode odgovarajuće inicijative, projekti digitalnih promjena, digitalne transformacije. Ove inicijative imaju svoje ciljeve i mjere koje su povezane s tradicionalnim ciljevima i mjerama (financijskim, procesnim, glede kupaca, glede učenja i rasta).

Kroz takav model razrade primjenom BSC okvira, DT nije sama sebi cilj, već je jasan učinak ostvarenja ciljeva DT na ciljeve, sve do onih glavnih, u domeni financija.

DT kroz ovakav model integriranja u BSC okvir može zaživjeti u punoj mjeri te zahvaljujući cikličnosti sustava, moguće je pratiti učinak transformacije na organizacijski uspjeh kroz vrijeme. Moguće je i poželjno izvoditi upravo i potrebne korekcije, poboljšanja, u odnosu na prvotno razrađene koncepte digitalne transformacije.

Na kraju, istaknimo i spoznaju iznesenu u tekstu (Businesswire, 2021) gdje se tvrdi da prema izvješću organizacije Accenture, one Europske tvrtke koje ubrzaju vlastite digitalne tranzicije kao i održive tranzicije vjerojatno će se brže oporaviti i izaći će snažnije iz krize COVID-19. BSC model pruža mogućnost integriranja ciljeva održivosti kao i ciljeva digitalne transformacije, te osigurava mjerenje performansi kao i dosljednu operacionalizaciju srednjoročnih i dugoročnih strategija i ciljeva.

Literatura:

1 Armitage H. M. & Scholey C. (2007) Using strategy map to drive performance, ABI/INFORM Global

2 Balanced scorecard, https://www.clearpointstrategy.com/

3 Bieker, T. (2003) Sustainability Management with the Balanced Scorecard. 2003. - 5. International Summer Academy on Technology Studies. – Deutschlandsberg

4 Chanias, S., Myers, M., Hess, T. (2019) Digital transformation strategy making in pre-digital organizations: the case of a financial services provider. J. Strateg. Inf. Syst. 28(1), 17–33

5 Daft, R.L. (2009) Management, 9th ed, Cengage Learning, 2009

6 Deloitte University Press and MIT Sloan Management (2016) Aligning the Organization for Its Digital Future: 2016 Digital Business Global Executive Study and Research Project; Gerald C. Kane, Doug Palmer, Anh Nguyen Phillips, David Kiron, and Natasha Buckley, http://sloanreview.mit.edu/projects/aligning-for-digital-future

7 Dictionary, http://dictionary.cambridge.org/dictionary/english/performance

8 Dictionary, https://www.merriam-webster.com/dictionary/performance

9 Dobrović, Ž (2017) Mjerenje organizacijskih performansi; predavanja, FOI Varaždin

10 Dumas, M., La Rosa, M., Mendling, J., & Reijers, H. A. (2013) Fundamentals of business process management. Springer

11 EU (2018) Digital Transformation Scoreboard 2018 - EU businesses go digital: Opportunities, outcomes and uptake; Luxembourg: Publications Office of the European Union, 2018

12 Fabac, R. (2020) Organizacijska teorija - s naglaskom na teoriju igara, Jastrebarsko: Naklada Slap; Varaždin: Fakultet organizacije i informatike, 2020.

13 Figge, F.; Hahn, T.; Schaltegger, S. and Wagner, M. (2001) Sustainability Balanced Scorecard, Wertorientiertes Nachhaltigkeitsmanagement mit der Balanced Scorecard, Lüneburg, Center for Sustainability Management, 2001

14 Hansen, E.G. & S. Schaltegger (2016) The sustainability balanced scorecard: A systematic review of architectures, J. Business Ethichs,133 (2016),193-221

15 Haynes, Howard H. (1992) Planning, Programming and Budgeting System (PPBS); NPS, Monterey

16 Henri, Jean-Francois. (2004). Performance measurement and organizational effectiveness: Bridging the gap. Managerial Finance. 30. 93-123.

17 Hess, T. & Matt, C. & Benlian, A. & Wiesböck, F. (2016) Options for Formulating a Digital Transformation Strategy. MIS Quarterly Executive. 15. 123-139.

18 Hess, Thomas & Matt, Christian & Benlian, Alexander & Wiesböck, Florian (2016) Options for Formulating a Digital Transformation Strategy. MIS Quarterly Executive. 15. 123-139.

19 Hrvatski sabor, NN 81/2006 (2006) Dugoročni plan razvoja oružanih snaga Republike Hrvatske 2006. – 2015., Zagreb

20 Ittner, C., & Larcker, D. (1998). Are Nonfinancial Measures Leading Indicators of Financial Performance? An Analysis of Customer Satisfaction. Journal of Accounting Research, 36, 1-35.

21 Janssen, D., Rotthier, S., Snijkers, K. (2004) If you measure it, they will score: an assessment of international eGovernment benchmarking. Information Polity 9, 121–130 (2004)

22 Jun Xin & Yuhui Wei (2009) How to develop a Balanced Scorecard into a Strategy Map - A case study of Ericsson, Master thesis, Lund University

23 Kaplan, R. S. and D.P. Norton (1992) The Balanced Scorecard: Measures that Drive Performance, Harvard Business Review, (January-February): 71-79.

24 Kaplan, R. S. and D.P. Norton (1996) The Balanced Scorecard: Translating Strategy into Action, Boston: HBS Press.

25 Kaplan, R. S.: Conceptual foundations of the balanced scorecard. In: Chapman, C. S./Hopwood, A. G./Shields, M. D. (Hrsg.): Handbook of Management Accounting Research, Band 3, Amsterdam 2009, S. 1253-1269.

26 Kaplan, Robert S., and David P. Norton (2004) Strategy Maps: Converting Intangible Assets into Tangible Outcomes. Boston: Harvard Business School Press, 2004

27 Kaufman, I. & Horton, C. (2015) Digital Transformation: Leveraging Digital Technology with Core Values to Achieve Sustainable Business Goals. The European Financial Review (December–January), 63–67.

28 M. H. Ismail, M. Khater, M. Zaki (2017) Digital Business Transformation and Strategy: What Do We Know So Far? Cambridge Service Alliance 2017

29 Matt , C.; T. Hess and A. Benlian (2015) Digital Transformation Strategies, Business & Information Systems Engineering: The International Journal of WIRTSCHAFTSINFORMATIK, Springer; Gesellschaft für Informatik e.V. (GI), vol. 57(5), 339-343

30 Maydanova, S., Ilin, I., Jahn, C., Lange, A.K., Korablev, V. (2019) Balanced scorecard for the digital transformation of global container shipping lines. In: International Conference on Digital Transformation in Logistics and Infrastructure (ICDTLI 2019), Russia (2019)

31 Mergel, I.; Edelmann, N.; Haug, N. (2019) Defining digital transformation: Results from expert interviews, Gov. Inf. Q. 2019

32 Mintzberg, H. (1987) The Strategy Concept I: Five Ps for Strategy, California Management Review 30 (1), 11-24

33 MIT and Deloitte (2016) Aligning the Organization for Its Digital Future. Gerald C. Kane, Doug Palmer, Anh Nguyen Phillips, David Kiron, Natasha Buckley. MIT Sloan Management Review Research Paper (July 26, 2016)

34 Rabbani, A., Zamani, M., Yazdani-Chamzini, A. & Zavadskas, E.K. (2014) Proposing a new integrated model based on sustainability balanced scorecard (SBSC) and MCDM approaches by using linguistic variables for the performance evaluation of oil producing companies, Expert Systems with Applications 41, 7316–7327

35 Robert S. Kaplan and David P. Norton (2001) The Strategy-Focused Organization: Harvard Business School Press, Boston, 2001

36 Rummler, G. A., & Brache, A. P. (2013) Improving performance: How to manage the white space on the organization chart (3rd ed.). US: Jossey-Bass

37 Schuchmann, D. & Seufert, S. (2015) Corporate Learning in Times of Digital Transformation: A Conceptual Framework and Service Portfolio for the Learning Function in Banking Organisations. International Journal of Advanced Corporate Learning

38 Spitzer, D. R. (2007) Transforming performance measurement: Rethinking the Way We Measure and Drive Organizational Success. New York, USA: American Management Association

39 Wu, H.Y. (2012) Constructing a strategy map for banking institutions with key performance indicators of the balanced scorecard, Evaluation and Program Planning, Vol. 35, No. 3, 303–320

40 Yamamoto, S. (2020) A Strategic Map for Digital Transformation, KES

41 Zhou, K.Z. & Wu, F. (2010) Technology capability, strategic flexibility and product innovation, Strategic Management Journal, Vol. 31 No. 5-6, 547-561.

42 Review of the new report from Accenture (2021), Businesswire; dostupno na: https://www.businesswire.com/news/home/20210125005034/en/European-Companies-That-Accelerate-Both-Digital-and-Sustainability-Transitions-Will-Recover-Faster-from-the-COVID-19-Crisis-Finds-Research-from-Accenture

Podaci o autoru:

Prof. dr. sc. Robert Fabac

Fakultet organizacije i informatike

Pavlinska 2, 42000 Varaždin

e-mail: rfabac@foi.hr

Robert Fabac diplomirao je na Prirodoslovno-matematičkom fakultetu u Zagrebu, a nakon pohađanja Sveučilišnog poslijediplomskog studija „Vođenje i upravljanje pokretnim objektima“ stječe titulu magistra znanosti. Od 1999. do 2007. radi u Institutu za obrambene studije, istraživanje i razvoj Ministarstva obrane RH, te 2004. stječe zvanje doktora informacijskih znanosti. Od 2007. radi na Fakultetu organizacije i informatike u Varaždinu, a na mjestu redovitog profesora od 2018. Nositelj je predmeta „Organizacijsko projektiranje“, „Organizacijska teorija“, „Mjerenje organizacijskih performansi“, „Komuniciranje u organizaciji“. Na doktorskom studiju FOI nositelj je kolegija „Odabrana poglavlja upravljanja organizacijom“ a na doktorskom studiju „Pomorstvo“ pri PFRI, Sveučilište u Rijeci, nositelj je kolegija „Strategijsko planiranje i vođenje“. Na vojnim studijima Sveučilišta u Zagrebu nositelj je predmeta „Planiranje i upravljanje resursima obrane“. Sudjelovao je više godina u izvođenje nastave na RŠ „Ban Josip Jelačić“ pri Hrvatskom vojnom učilištu Ministarstva obrane. Sudjelovao je u radu na većem broju znanstvenih i stručnih projekata. Član je uređivačkog odbora u više međunarodnih znanstvenih časopisa. Član je upravnog odbora Hrvatskog interdisciplinarnog društva. U dva navrata biran je za pročelnika Katedre za organizaciju, na FOI. Područje njegovog znanstvenog interesa je organizacija u širem smislu, te pored toga discipline teorija igara, teorija odlučivanja i strategijski menadžment.

NEO4J - GRAFOVSKA BAZA PODATAKA: OD IDEJE, KREIRANJA, PUNJENJA DO WEB SERVISA

prof. dr. sc. Dragutin Kermek, prof. dr. sc. Kornelije Rabuzin, dr. sc. Matija Novak

SAŽETAK:

Opisano je zašto su Grafovske baze podataka izvrstan odabir kada se radi o podacima nad kojima treba provoditi modeliranje u kojem se odnosi između podataka jednako tretiraju kao i sami podaci. Slijedi opis zašto ne postoji unaprijed definirani model podataka tako da se podaci pohranjuju u svojem prirodnom sadržaju uz mogućnost kasnijeg dodavanja ili brisanja svojstava na razini pojedinačnog zapisa tj. čvora. Opisano je zašto se kod grafovskih baza podataka koristi struktura grafa za semantičke upite. Neo4j je sustav za upravljanje grafovskim bazama podataka koji ima vlastiti grafovski upitni jezik (eng. Graph Query Language) pod nazivom Cypher i preglednik pod nazivom Neo4j Desktop. Prikazan je način na koji se kreira grafovska baza podataka u Neo4j i kako se kreiraju pojedini čvorovi te relacije između čvorova. U nastavku je opisano kako se postavljaju upiti putem jezika Cypher unutar Neo4j Desktop, koji prikazuje rezultate upita u obliku grafa. Posebno se obrađuje prijelaz sa SQL baze podataka na Neo4j u slučaju kada se radi u velikoj količini podataka. Slijedi opis kako je realiziran web servis na bazi Neo4j.Na kraju je kratak zaključak.

Ključne riječi: Neo4j, grafofska baza podataka, QGL, Cypher, ETL, web servis,

ABSTRACT*:*

The paper describes why graph databases are an excellent choice when it comes to data that needs to be modeled in which the relationships between the data are treated in the same way as the data itself. The following is a description of why there is no predefined data model so that the data is stored in its natural content with the possibility of later adding or deleting properties at the level of an individual record, ie a node. It is described why graph structures use a graph structure for semantic queries. Neo4j is a graph database management system that has its own Graph Query Language called Cypher and a browser called Neo4j Desktop. The way in which the graphical database in Neo4j is created and how individual nodes and relations between nodes are created are shown. The following describes how to query through the Cypher language within Neo4j Desktop, which displays the query results in graph form. The transition from SQL database to Neo4j is especially processed in the case when working in a large amount of data. The following is a description of how the Neo4j-based web service was implemented. Finally, there is a brief conclusion.

1. UVOD

Relacijske baze podataka dominiraju svijetom baza podataka već posljednjih pedesetak godina. Dobra svojstva samog modela kao i utemeljenost na konceptima od kojih su neki stari preko 2000 godina (npr. teorija skupova), garancija su da one neće nestati, iako pojedini autor s vremena na vrijeme spominju da bi se upravo to moglo dogoditi. Unatoč tome, promjene koje su se zbile u posljednjih desetak godina ipak imaju utjecaj i važne su za daljnji smjer razvoja baza podataka.

Prva velika promjena koja je zasigurno bitna, počelo se generirati iznimno velike količine podataka, posebno sa zrelom primjenom koncepta Interneta stvari (en. Internet of Things, IoT). Količina generiranih podataka udvostručuje se svakih dvije do tri godine. I dok u kontekstu klasičnih poslovnih aplikacija (primjerice ERP ili CRM sustavi) nema toliko potrebe za promjenama, pohrana velikih količina podataka koji su generirani na Internetu ipak zahtijeva nova rješenja prilagođena specifičnim zahtjevima. Stoga se spominje i pojam „Big Data“ koji označava velike količine heterogenih podataka koji brzo nastaju i moraju biti obrađeni, a s kojima postojeće tehnologije imaju problema. Činjenica je da relacijske baze podataka imaju problema s iznimno velikim količinama podataka, u smislu pohrane, ali i u smislu obrade, pogotovo obrade u realnom vremenu.

Može se pogledati primjer društvenih mreža; broj novih korisnika svakodnevno raste i mjeri se u tisućama, a broj poruka generiranih poruka mjeri se u stotinama tisuća odnosno u milijunima. U jednoj minuti generira se oko 4 milijuna pretraga za što se koristi Google, a generira se i preko 40 milijuna poruka putem aplikacija Facebook ili WhatsApp. Sljedeći primjer iz [7] gdje autori prenose podatke o provedenom eksperimentu autora „Partner i Vukotic“ u potrazi prijatelja na društvenoj mreži (tzv. eng. Friend of a Friend, foaf). Testiranjem je pokazano da traženje prijatelja u relacijskoj bazi podataka funkcionira, ali samo na prve dvije odnosno tri razine, dok grafovske baze podataka s takvim upitima nemaju nikakvih problema. Konkretno, uz milijun korisnika, od kojih svaki ima 50 prijatelja, pretraga prijateljevih prijatelja se kod relacijskih baza podataka mjeri u sekundama, dok grafovske odgovor daju u milisekundama. Na petoj razini relacijski sustav ne daje odgovor u realnom vremenu, dok je kod grafovskih baza podataka vrijeme odgovora tek nešto veće (dvije do tri sekunde). U ovom slučaju nije toliko problem u pohrani podataka, što je u načelu izvedivo, no obradu odnosno dohvaćanje traženih podataka nije moguće provesti u realnom vremenu. Ponekad je uvjet da se podaci obrade u realnom vremenu, drugim riječima, nema vremena na pretek i podaci su potrebni odmah.

Kod obrade velikih količina podataka vrijeme procesiranja može se lako mjeriti u minutama, desecima minuta, pa čak i satima. To je posebno vidljivo kod skladišta podataka i sustava poslovne inteligencije, primjerice, kod kreiranja kocki kao više dimenzijskih struktura podataka za čiju pripremu i procesiranje također može biti potrebno po nekoliko sati, iako količine podataka nisu prevelike (nekoliko desetaka ili stotina milijuna slogova).

Što se strukture i heterogenosti generiranih podataka tiče, podaci više nisu uniformni u smislu da imaju istu strukturu. Stoga i pokušaj pohrane istih u predefinirane strukture, odnosno tablice kod relacijskih baza podataka, nije uvijek najbolje rješenje.

Temeljem svega navedenog u posljednjih desetak godina (intenzivno) tražila su se rješenja za spomenute probleme. Temeljeno na rješavanju tih problema razvila se i nova kategorija sustava za upravljanje bazama podataka, koji su poznati kao NoSQL sustavi. Iako je jedna od prvih interpretacija skraćenice bila „No to SQL“, kasnije je prevladalo značenje „Not Only SQL“. Kad se govori o NoSQL sustavima, razlikuje se četiri podvrste:

1. dokumentu orijentirane baze podataka (eng. document – oriented databases)
2. stupcu orijentirane baze podataka (eng. column-oriented databases)
3. grafovske baze podataka (eng graph databases)
4. ključ-vrijednost baze podataka (eng. key value databases).

Kod prve spomenute vrste podaci su pohranjeni u dokumente, no ne .doc ili .pdf, već je bitno razumijevanje JSON (eng. Java Script Object Notation) formata kao takvog. Drugi sustavi najsličniji su relacijskim sustavima jer se podaci pohranjuju u tablice, no razlika je u samoj shemi koja ne mora biti fiksna (ista) za svaki zapis, te u samoj pohrani podataka. Kod ključ-vrijednost sustava podaci se pohranjuju korištenjem ključa za koji se specificira određena vrijednost; za dohvat vrijednosti moramo znati ključ. Kako se ovaj rad bavi grafovskim bazama podataka, neće biti dubljeg prikaza preostalih podvrsta, nego se fokusirati na grafovske baze podataka. Za detaljnije informacije o preostalim spomenutim sustavima postoji puno dostupnih izvora i knjiga na tu temu.

Nadalje, valja imati na umu da NoSQL sustavi imaju određenih prednosti, ali i nedostatke. Postoji skraćenica YAQL, koja dolazi od eng. Yet Another Query Language, a što upravo odgovara stanju stvari kad se govori o NoSQL sustavima. Svaki sustav ima svoj upitni jezik i ne standardiziranog jezika, kao što je kod relacijskih sustava SQL. S druge strane, koncepti koji su poznati i razvijeni kod relacijskih sustava još uvijek nisu podržani kod NoSQL sustava. Tako je jedan od autora ovog rada objavio niz radova na temu proširenja grafovskih sustava s konceptima koji se već dugi niz godina koriste kod relacijskih sustava:

- implementacija integritetnih ograničenja [4], [5] i [6];
- implementacija okidača (eng. trigger) [2];
- nasljeđivanja u grafovskim bazama podataka [3].

2. GRAFOVSKE BAZE PODATAKA

Grafovske baze podataka počivaju na teoriji grafova. Iako se može činiti da je ideja nova, to nije tako. Naime, relacijskim bazama podataka prethodile su tzv. mrežne baze podataka, za koje se može reći da su imale sličnu ideju pohrane kao i grafovske baze podataka. Kako su relacijske baze podataka ubrzo prevladale, mrežne nikad nisu zaživjele u punom smislu kao takve, no može se reći da se temeljna ideja mrežnih baza podataka danas koristi kod grafovskih baza podataka. Što se samog razvoja i trendova u razvoju grafovskih sustava tiče, iste su opisane u [1].

Svaki graf sastoji se od čvorova i bridova koji povezuju dane čvorove. Društvena mreža se također sastoji od korisnika (čvorova) koji su međusobno povezani. Mreža aerodroma se također sastoji od aerodroma (čvorovi) koji su međusobno povezani (ako postoji let od prvog prema drugom, odnosno obrnuto). Dakle, kako što je i Euler zaključio, grafovima se mogu modelirati različite domene, uključujući društvene mreže, promet, itd. Postoje drugi koncepti poznati u teoriji grafova, no njima se ovaj rad ne bavi jer nisu presudni za rad s grafovskim bazama podataka.

Za implementaciju grafovskih baza podataka mogu se koristiti klasični relacijski sustavi koji su prošireni na način da podržavaju koncepte grafovskih baza (primjer Oracle, MS SQL Server …), ili „čisto grafovske” (eng. native) sustave. Slika 1 prikazuje grafovske baze podataka i koji model podataka podržava pojedina od njih.

☐ include secondary database models — 32 systems in ranking, February 2021

Rank			DBMS	Database Model	Score		
Feb 2021	Jan 2021	Feb 2020			Feb 2021	Jan 2021	Feb 2020
1.	1.	1.	Neo4j	Graph	52.16	-1.62	+0.96
2.	2.	2.	Microsoft Azure Cosmos DB	Multi-model	31.66	-1.31	-0.29
3.	3.	3.	OrientDB	Multi-model	5.13	-0.20	+0.19
4.	4.	4.	ArangoDB	Multi-model	5.07	-0.22	+0.22
5.	5.	↑ 7.	JanusGraph	Graph	2.53	-0.05	+0.65
6.	↑ 7.	↓ 5.	Virtuoso	Multi-model	2.37	+0.22	-0.40
7.	↑ 8.	↑ 9.	GraphDB	Multi-model	2.14	+0.03	+1.00
8.	↓ 6.	↓ 6.	Amazon Neptune	Multi-model	2.07	-0.24	+0.11
9.	9.	↑ 10.	FaunaDB	Multi-model	1.91	0.00	+0.92
10.	↑ 11.	↑ 12.	Stardog	Multi-model	1.46	0.00	+0.56

Slika 1. Pregled grafovskih sustava (izvor: https://db-engines.com)

Kad se govori o kreiranju pitanje je kako kreirati čvorove i veza između čvorova. Čvorovi i veze mogu imati svojstva, s time da struktura svakog čvora odnosno veze ne mora biti identična. Drugim riječima, drugi čvor „istog tipa“ ne mora nužno imati identična svojstva odnosno atribute, a što vrijedi i za veze. Nadalje, čvorovi mogu biti označeni i na taj način grupirani, no to nije nužno. Ako čvorove označimo (eng. labeled), moguće je razlikovati čvorove prema oznakama, a što je poželjno kod izvršavanja upita jer se ne mora pretraživati cijela baza s velikim brojem čvorova, već samo čvorovi koji imaju traženu oznaku.

Kako se grafovska baza podataka sastoji od velikog broja čvorova i veza među njima, javlja se pitanje kako se relacijska baza podataka može migrirati u grafovsku bazu podataka. Temeljno pitanje je dakle što odgovara čemu. Generalno možemo reći da redak iz tablice u relacijskoj bazi postaje čvor u grafovskoj bazi. Drugim riječima, čvor predstavlja instancu entiteta. Ako su između redaka različitih tablica postojale veze, konkretno jedna tablica t1 ima vanjski ključ na drugu tablicu t2, kreiraju se čvorovi i veza između čvorova.

Koje su prednosti korištenja grafovskih baza podataka? Kao prvo, svakako valja izdvojiti fleksibilnu strukturu. Čvorovi dakle ne moraju imati identična svojstva, što također vrijedi i za veze. Nadalje, kad govorimo o skaliranju, moguće je dodati nove čvorove i nove veze s time da ne postoji potreba za promjenom postojećih, već kreiranih čvorova i/ili veza. Isto tako, upiti koji se izvršavaju nad grafovskim bazama podataka, a „traže“ povezane podatke, generalno se izvršavaju brže nego kod relacijskih sustava.

Kada se govori o čisto grafovskim sustavima, svakako valja istaknuti sustav Neo4j. To je sustav koji je prisutan već dosta godina, te ima niz dobrih i poželjnih svojstava kad govorimo o grafovskim bazama podataka. Upravo je to sustav koji se koristi u nastavku ovog rada.

3. NEO4J

Neo4j je „čista” (en. native) grafovska baza podataka (sustav za upravljanje bazama podataka). Kao i svaka druga baza podataka tako i Neo4j prvenstveno se koristi za pohranjivanja podataka. Značajno je da je Neo4j od svog temelja građena da se mogu koristiti odnosi između podataka za različite namjene (pohranjivanje podataka, upiti, uvjeti i sl.).

Ideja za razvoj Neo4J nastala je 2000. godine kada su autori budućeg sustava Neo4J radili na određenom projektu te uočili velika ograničenja relacijskih baza podataka. Razvoj Neo4J započeo je 2007. godine, a prva verzija Neo4J 1.0 izašla je 2010. godine. Trenutno je važeća verzija 4.2. Akronim Neo4J dolazi od Network Exploration and Optimization 4 Java. Iz punog naziva jasno se otkriva da je Neo4J implementiran u programskom jeziku Java.

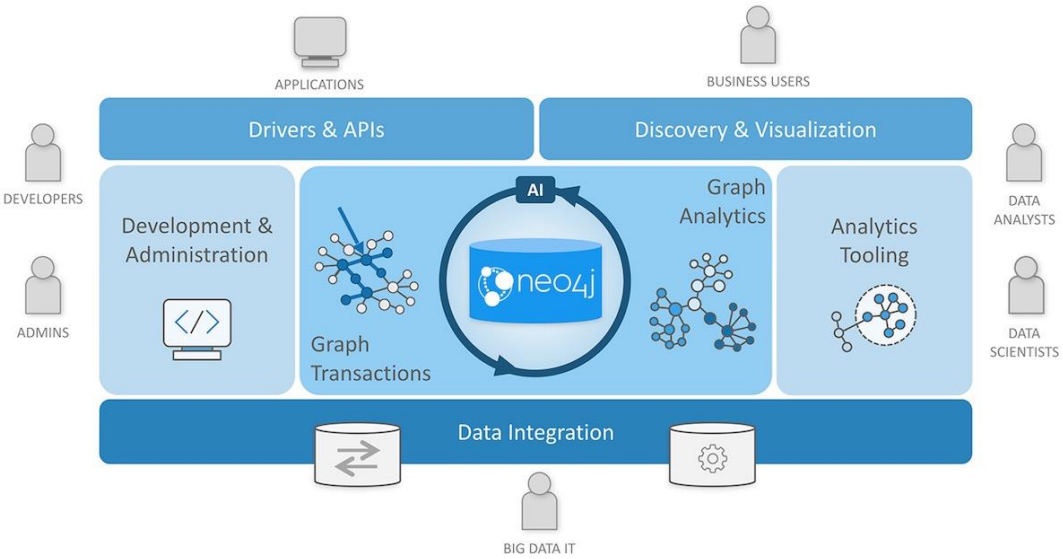
Jedan od opisa [8] kaže da je Neo4j „stvoren za učinkovito pohranjivanje, rukovanje i izvršavanje upita na visoko povezanim podacima u podatkovnom modelu. Pomoću moćnog i fleksibilnog podatkovnog modela mogu se predstaviti stvarne informacije promjenjive strukture bez gubitka svog bogatstva.“ Glavne osobine Neo4j su:

- skalabilnost
- sigurnost
- fleksibilnost
- performanse
- visoka raspoloživost
- usmjerenost razvojnim inženjerima.

Neo4J ima izdanja: Community i Enterprise tako da ima dualnu licencu: GPL v3 i komercijalnu licencu. Community Edition je besplatna ali ima ograničenja kao što su kreiranje samo jedne baze podataka, rad na samo jednom računalnom čvoru (poslužitelju) jer nije dopušteno izvršavanje na klasteru računala, sigurnosna zaštita podataka tijekom rad (eng. hot backup) i sl. Enterprise Edition otvara ograničenja koja postoje kod Community Edition.

Za realizaciju složenih projekata koji se temelje na velikim količinama pohranjenih podataka slobodne strukture, izvršavanju različitih upita na tim podacima koji mogu uključivati vrlo složene algoritme, analizi dobivenih podataka izvršavanja upita i njihovoj vizualizaciji, potrebno je više od same grafovske baze podataka. Zbog toga Neo4J ima grafovsku bazu podataka u sredini oko koje su izgrađeni ostali elementi, što sve zajedno čini Neo4J platformu (Slika 2). Važne komponente Neo4j platforme su [8]:

- Neo4j Graph Database – grafovska baza podataka
- Neo4j Desktop – aplikacija za upravljanje lokalnim instancama Neo4J
- Neo4j Browser – online pregledničko sučelje za postavljanje upita i pregled podataka u bazi podataka. Cypher se koristi za osnovne mogućnosti vizualizacije podataka.
- Neo4j Bloom – vizualizacijski alat za poslovne korisnike koji ne zahtjeva programski kod ili vještine programiranja za pogled i analizu podataka
- Neo4j ETL Tool – alat za migraciju podataka iz relacijske baze podataka u Neo4J.
- Neo4j APOC Library (Awesome Procedures On Cypher) – standardna biblioteka dodatnih procedura za NeoJ
- Neo4j Graph Data Science– biblioteka algoritama na grafovima za Neo4J.
- Neo4j GraphQL – specifikacija za upit dijela aplikacijskog grafa kojim je pretražuju podaci koji točno odgovaraju korisničkom pogledu, bez obzira odakle se povlače podaci.



Slika 2. Komponente Neo4J platforme (izvor: [8])

3.1. Cypher

Neosporna je činjenica da je osnovna uloga Neo4J platforme pohranjivanje podataka. To se jasno može zaključiti iz prethodno prikazanim glavnim osobinama Neo4J kao i prve dvije komponente Neo4J platforme. Osnovu rada s podacima čini Cypher tj. Neo4j grafovski upitni jezik (JSON se koristi). „Cyper je snažan upitni jezik koji je optimiran za grafove i koji razumije i koristi prednosti spremljenih veza. Cypher je nadahnut SQL-om, uz dodatak podudaranja uzoraka posuđenih iz SPARQL-a.“ [9].

Primjeri Cypher koda koji se izvršava u komponenti Neo4J Browser za kreiranje 4 čvora aerodroma i četiri leta aviona između aerodroma:

```
CREATE
(a1:airportAll {ident: 'LDZA', name: 'Zagreb Airport', coordinates: '16.068, 45.742',
continent: 'EU', iso_country: 'HR'}),
(a2: airportAll {ident: 'LDVA', name: 'Varaždin Airport', coordinates: '16.382, 46.294',
continent: 'EU', iso_country: 'HR'}),
(a3: airportAll {ident: 'LOWW', name: 'Vienna International Airport', coordinates:
'16.569, 48.110', continent: 'EU', iso_country: 'AT'}),
(a4: airportAll {ident: 'LEBL', name: 'Barcelona International Airport', coordinates:
'16.569, 48.110', continent: 'EU', iso_country: 'ES'})
```

```
CREATE
(l1:airplanes {icao24: '440824', estDepartureAirport: 'LOWW', estArrivalAirport:
'LDZA'}),
(l2:airplanes {icao24: '501d1e', estDepartureAirport: 'LOWW', estArrivalAirport:
'LDZA'}),
(l3:airplanes {icao24: '346186', estDepartureAirport: 'LOWW', estArrivalAirport:
'LEBL'}),
(l4:airplanes {icao24: '345644', estDepartureAirport: 'LDZA', estArrivalAirport: 'LEBL'})
```

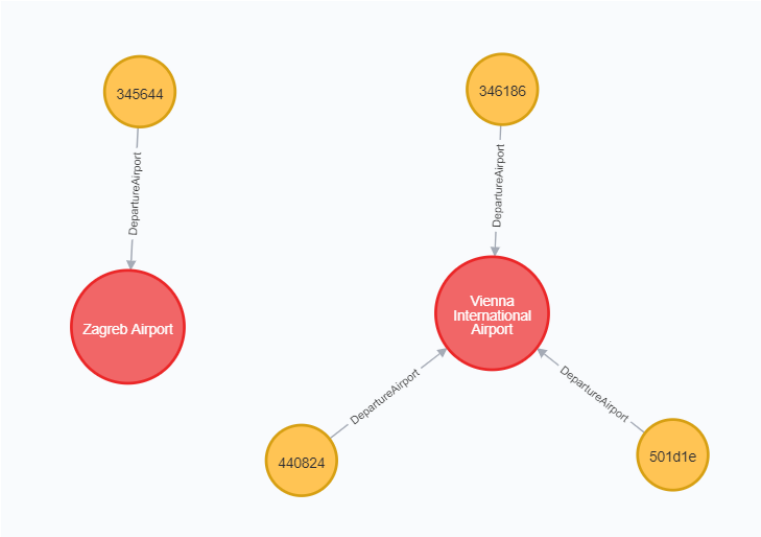
Slijede primjeri Cypher koda za kreiranje veza između letova aviona i polaznih aerodroma, a nakon toga i za kreiranje veza između letova aviona i odredišnih aerodroma.

```
MATCH (a:airplanes)
MATCH (aa:airportsAll {ident: a.estDepartureAirport})
CREATE (a)-[r:DepartureAirport]->(aa)
RETURN count(r)
```

```
MATCH (a:airplanes)
MATCH (aa:airportsAll {ident: a.estArrivalAirport})
CREATE (a)-[r:ArrivalAirport]->(aa)
RETURN count(r)
```

Nakon što se izvrše prethodna dva bloka Cypher instrukcija u grafovskoj bazi podataka postoje čvorovi (airportAll i airplanes) te relacije (DepartureAirport i ArrivalAirport). To znači da se mogu postaviti upiti i dobiti vizualizacija odabranih bridova tj. veza između čvorova. Slijedi primjer Cypher koda za prikaz veza između letova aviona i polaznih aerodroma, a nakon toga vizualizacija u Neo4J Browser (Slika 3).

```
MATCH (a:airplanes)-[r:DepartureAirport]->(aa)
RETURN a, r, aa
```



Slika 3. Vizualizacija veza između letova aviona i polazišnih aerodroma

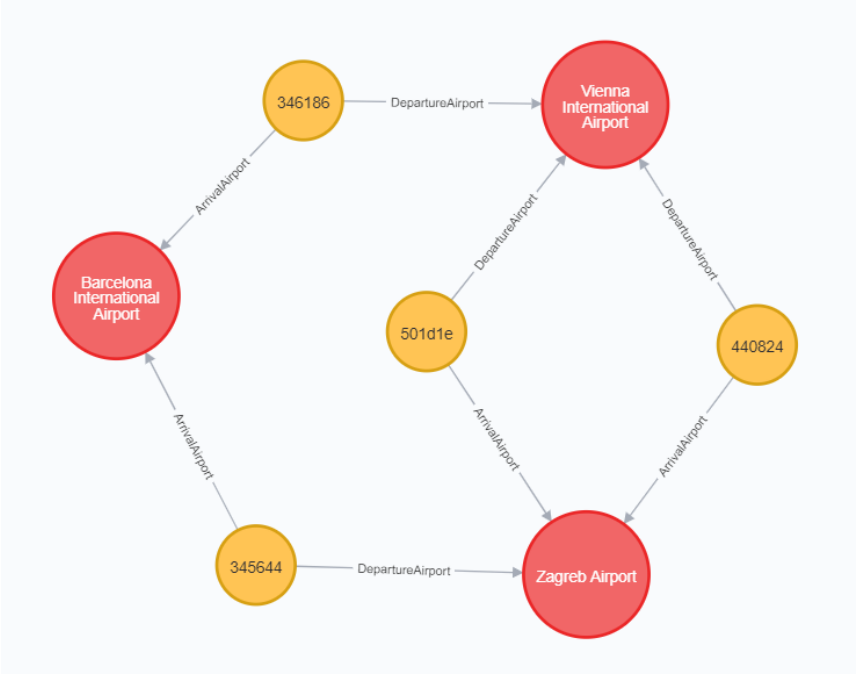
Vrlo često je potrebno analizirati složenije odnose između čvorova tako da se koriste dva ili više brida odnosno veza. Slijedi 2 primjera Cypher koda za prikaz veza letova aviona i njihovih polazišnih i odredišnih aerodroma kao i vizualizacija u Neo4J Browser (Slika 4). U prvom primjeru koriste se dvije veze leta aviona (l1) i to između polazišnog aerodroma (a1) i odredišnog aerodroma (a2). U drugom primjeru koristi se operator ili (|) kako bi se uzela bilo koja od tih dviju veza (DepartureAirport|, ArrivalAirport) između veza aviona i nekog drugog nedefiniranog tipa čvora (aa).

```
MATCH (a1:airportsAll) <-[da1:DepartureAirport]-(l1:airplanes)-[aa1:ArrivalAirport]->(a2:airportsAll)
RETURN a1, da1, l1, aa1, a2
```

```
MATCH (a:airplanes)-[r:DepartureAirport| ArrivalAirport]->(aa)
RETURN a, r, aa
```

Spomenuto je ranije da se određena svojstva mogu pridružiti ne samo čvorovima (u relacijskom modelu redcima u tablici baze podataka) nego u bridovima tj. vezama/relacijama. Slijedi primjer Cypher koda za kreiranje veza između aerodroma koja sadrži atribut njihove udaljenosti. Veza se kreira na temelju letova aviona između dva aerodroma.

```
MATCH (a1:airportsAll) <-[da1:DepartureAirport]-(l1:airplanes)-[aa1:ArrivalAirport]->(a2:airportsAll)
WITH a1, l1, a2, distance(
point({latitude: a1.latitude, longitude: a1.longitude}),
point({latitude: a2.latitude, longitude: a2.longitude})) as udaljenost
MERGE (a1)-[ac:AIRPORT_CONNECTION {distance: udaljenost}]->(a2)
RETURN count(ac)
```



Slika 4. Vizualizacija veza između letova aviona i polazišnih i odredišnih aerodroma

4. PUNJENJE GRAFOVSKE BAZE PODATAKA

U Laboratoriju za web tehnologije, servise i sučelja (WATTS) na Fakultetu organizacije i informatike provodi se interni projekt na kojem se istražuje područje strujanja podataka (eng. Data Streaming). Taj projekt za svoje potrebe preuzima podatke iz različitih otvorenih izvora podataka kao su OpenWeatherMap[*], Open Sky Network[†] i dr. Podaci o letovima aerodroma rezultat su projekta OpenSky kroz koji se gradi velika mreža ADS-B[‡] senzora, koja služi za istraživanje [10]. Projekt laboratorija WATSS od početka za pohranjivanje podataka koristi Java DB tj. Apache Derby, Apache DB[§]. Radi se o relacijskoj bazi podataka otvorenog programskog koda koja je u potpunosti implementirana u programskom jeziku Java te ima prednosti kao što su:

- mala veličina (3.5 MB)
- temeljen na programskom jeziku Java, JDBC i SQL standardu
- jednostavno se instalira, isporučuje i koristi itd.

Postoje različiti moduli u projektu koji se bave preuzimanjem podataka iz određenih izvora. Neki od njih rade s podacima u stvarnom vremenu te ih preuzimaju u određenim vremenskim ciklusima za definirani skup objekata. Npr. svakim 15 min preuzimaju se meteorološki podaci za 200 aerodroma u svijetu, a u pauzi između kraja preuzimanja jednog ciklusa i početka novog ciklusa preuzimaju se podaci za preostali skup aerodroma u svijetu između kojim postoje letovi aviona odabranih 200 aerodroma (više od 13.000). Ciklus tog preuzimanje je 6 sati. Računica pokazuje da se dnevno upisuje oko 70.000 meteoroloških zapisa u bazu podataka, što daje na razini godine dana oko 19.000.000 zapisa.

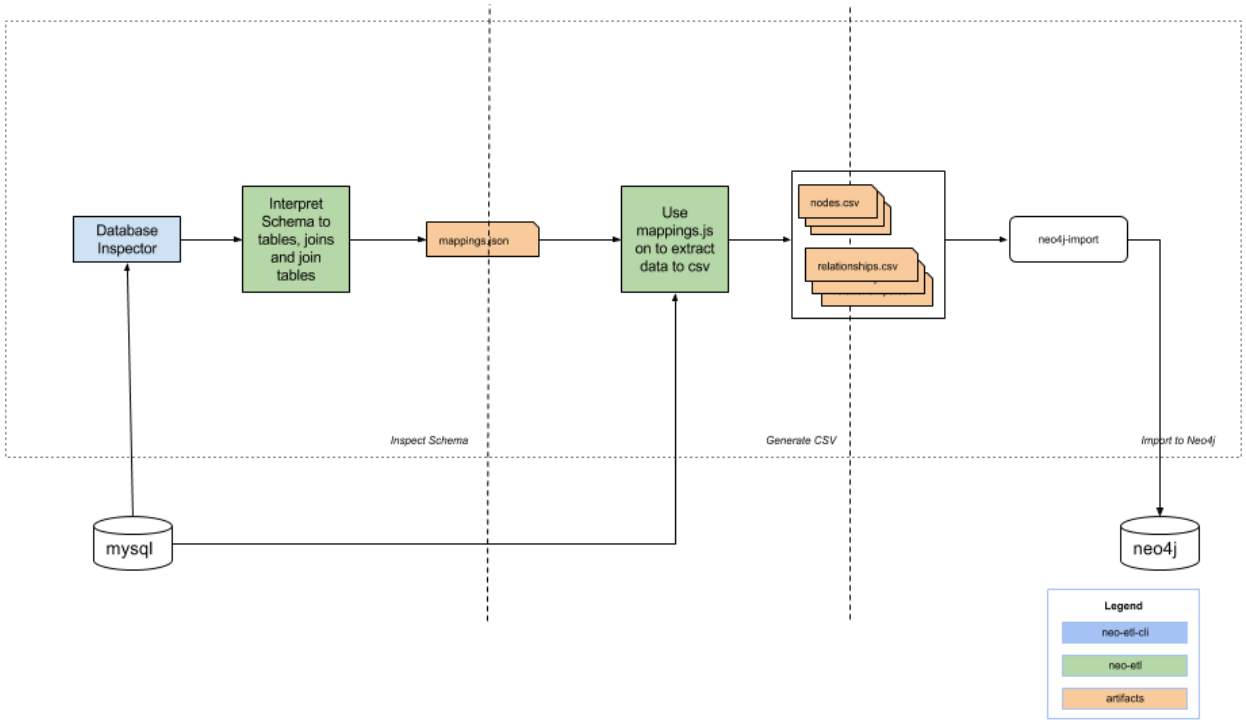
Druga vrsta podataka koja se preuzima odnosi se na letove aviona i ne radi se o podacima u stvarnom vremenu. Određeni podaci o letovima aviona mogu se preuzeti i do godina dana nakon leta, a neki drugi podaci o letovima aviona imaju znatno kraće vrijeme u kojem se mogu preuzeti i to je do 30 dana. S odabranih 200 aerodroma dnevno poleti između 0 i 600 letova što ovisi o danu u tjednu, meteorološkim uvjetima, a u zadnje vrijeme i o epidemiološkim uvjetima. Ako se pretpostavi da je 200 prosječan dnevni broj letova aviona na svakom od odabranih 200 aerodromima tada je ukupan dnevni broj letova aviona 40.000, a godišnji oko 15.000.000. Za većinu letova aviona mogu se preuzeti podaci o samom letu a za određene letove aviona mogu se preuzeti podaci i o njihovom putu od polazišnog do odredišnog aerodroma. Frekvencija podataka ovisi o promjeni podataka leta aviona (brzina, visina, smjer, GPS lokacija i sl). prosječan let aviona ima nekoliko stotina zapisa (npr. 200). Kada se pomnoži dnevni broj letova aviona s promatranim aerodroma i prosječan broj zapisa putanje leta aviona dođe se do broja 8.000.000 zapisa dnevno, odnosno 3.000.000.000 zapisa godišnje. Ako se podaci spremaju i čuvaju više godina tada je jasno da se radi o velikoj količini podataka. A posebno je pitanje veza između pojedinih entiteta u relacijskom modelu, broju indeksa i sl.

Analiza prikupljenih podataka i njihovih veza došla je određenih granica zbog količine podataka. Tada je obrana Neo4J baza podataka zbog svojih grafovskih osobina i podrške algoritmima na grafovima u svojoj biblioteci Graph Data Science. Svi podaci i dalje se prikupljaju na postojeći način jer su potrebni web servisi i ostali dio podrške koji su bazirani na platformi Java DB. Prvi dio prijelaza na Neo4J obavljen je za modul koji preuzima meteorološke podatke u stvarnom vremenu jer ima najmanje podataka, a i ciklusi preuzimanja dopuštaju da se u pauzi između njih dostigne ažurnost podataka Java DB i Neo4J. A nakon toga može se obaviti zamjena da se podaci upisuju u obje baze podataka ili samo u Neo4J. Odnosno može se izgraditi viša razina ETL arhitekture koja je temeljena na slanju poruka [11] kako bi se postigla paralelnost u radu i veći broj spremišta podataka. Znatno veći problem bio je kod podataka o letovima aviona i putanjama letova aviona.

Neo4j sadrži sljedeće mogućnosti za preuzimanje podataka iz relacijske baze podataka:

- Neo4j ETL[**] – alat s grafičkim sučeljem i podržava većinu baza podataka (MySQL, PostgreSQL, Oracle, MS SQL i ostale primjenom JDBC). U situacijama manje količine podataka i kada se ne radi o neprestanom dodavanju novih zapisa to je idealan model rada. Slika 5 prikazuje dijagram njegove arhitekture. Očigledno da ETL koristi csv datoteke za prijenos podataka između relacijske baze podataka i učitavanje u Neo4J.
- Cypher punjenje iz csv datoteke
- Cypher/APOC punjenje iz JDBC veze
- Cypher bulk import – programsko korištenje.

[*] https://openweathermap.org/api
[†] https://opensky-network.org/apidoc/rest.html
[‡] Automatic Dependent Surveillance–Broadcast (ADS–B). Radi se o tehnologiji nadzora putem koje avion utvrđuje svoju poziciju putem satelitske navigacije ili drugih senzora i šalje ju periodično kako bi se mogla pratiti njegova pozicija.
[§] https://db.apache.org/derby/
[**] Extract, transform, load – (ETL) – izdvoji, transformiraj, napuni (učitaj)



Slika 5. Dijagram arhitekture Neo4J ETL (izvor: https://neo4j.com/labs/etl-tool/1.5.0/)

Za određene tablice postojali su podaci u csv datotekama tako da se njihovo punjenje moglo obaviti bez većih problema putem Cypher-a, npr.

```
LOAD CSV WITH HEADERS FROM 'aerports.csv' as row
CREATE (aa:airportsAll {ident: row.IDENT, name: row.NAME, continent: row.CONTINENT,
iso_country: row.ISO_COUNTRY,
coordinates: row.COORDINATES})
```

Kao alternativa moglo se koristiti Cypher uz pomoć APOC biblioteke kojom bi se punjenje odradilo uz pomoć JDBC veze na bazu podataka Java DB, npr.

```
CALL apoc.load.jdbc('jdbc:derby://localhost:1527/watts1,
'SELECT * FROM airportsAll', [],
{credentials:{user:'watss1',password:'123456'}})
YIELD row
CREATE (aa:airportsAll {ident: row.IDENT, name: row.NAME, continent: row.CONTINENT,
iso_country: row.ISO_COUNTRY,
coordinates: row.COORDINATES})
```

Kod većih količina podataka prethodni oblici (osim ETL) nisu pouzdani jer potrebni memorijski resursi na poslužitelju vrlo često nisu dovoljni tako da dolazi do pada/prekida prilikom preuzimanja i punjenja. Rješenje se nalazi u skupnom obliku u kojem se preuzimaju podaci u fiksnim blokovima (npr. 100.000 zapisa). Opet su u suradnji Cypher i APOC biblioteka, npr.

```
CALL apoc.periodic.iterate(
"CALL apoc.load.jdbc('jdbc:derby://localhost:1527/watss1,
'SELECT * FROM airportsAll', [],
{credentials:{user:'watss1',password:'123456'}})
YIELD row
CREATE (aa:airportsAll {ident: row.IDENT, name: row.NAME, continent: row.CONTINENT,
iso_country: row.ISO_COUNTRY,
coordinates: row.COORDINATES})", {batchSize: 100000})
```

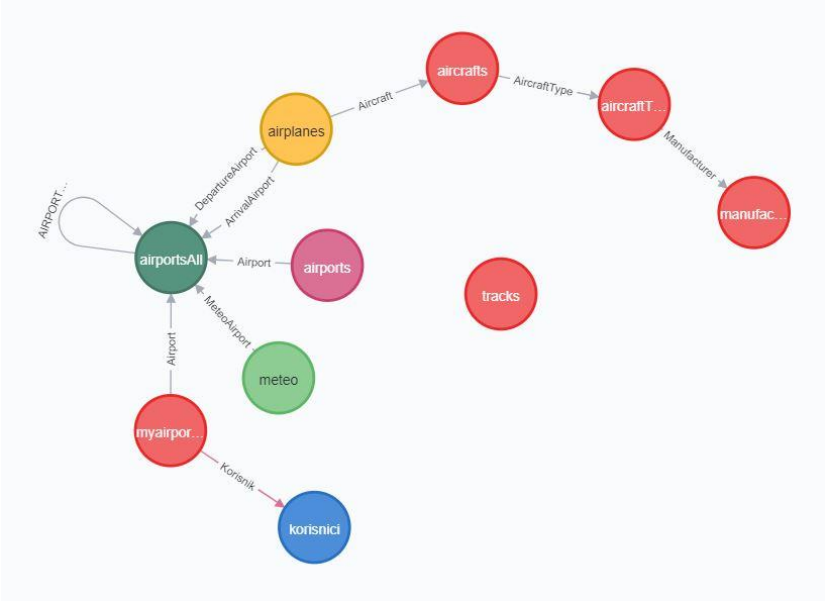
Zadnji problem koji je trebalo riješiti odnosi se na podatke koji imaju stalni priljev i nije moguće privremeno zaustaviti taj proces. S jedne strane postoji dnevni priljev od 8.000.000 zapisa, s druge strane važno je da ne dođe preskakanja pojedinih podataka u Java DB čime neće biti preneseni u Neo4J ili do duplog prijenosa podataka iz Java DB u Neo4J, s treće strane važno je da treba biti automatizirano na dnevnoj razini i četvrto da cijeli proces ne smije alocirati previše memorije i trošiti previše CPU. Rješenje se temelji na pokretanju pozadinske dretve s cikličkim radom koji u skupnom obliku preuzima podatke u fiksnim blokovima (npr. 100.000 zapisa). Za ovo rješenje korišten je programski jezik Java u suradnji Cypher-om i APOC biblioteka. U programskom jeziku ugrađena je logika rada s dretvom i ona u svakom ciklusu rada pomiče svoju početnu poziciju preuzimanja na bazi indeksiranog stupca id koji se automatski povećava kod upisa. Podaci se preuzimaju u skupnom obliku po 10.000 zapisa, a količina podataka koja se preuzima obično je manja od dnevnog priljeva podataka. Postavljanje početne pozicije za sljedeće preuzimanje ovisi o ukupnom broju preuzetih podataka u važećem ciklusu. Dio rješenja s Java, Cypher i APOC izgleda npr.

```
session.run("CALL apoc.load.jdbc('jdbc:derby://localhost:1527/watss1',"
 + "'SELECT * FROM tracks WHERE id > ? fetch first ? rows only', "
 + "[$broj, $kolicina],{batchSize: 10000,
```

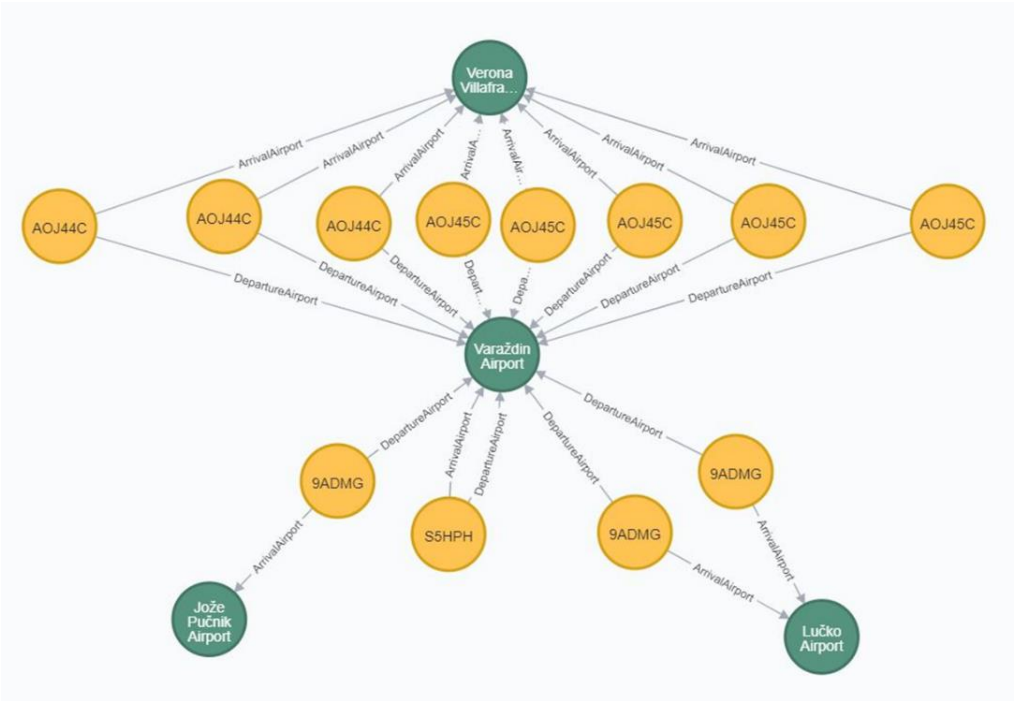
```
    credentials:{user:'watss1',password:'123456'}})"
+ "YIELD row "
+ "CREATE (aa:tracks {icao24: row.ICAO24, …});",
parameters("broj", priviZapis + (broj * kolicina), "kolicina", kolicina));
```

5. NEO4J GRAFOVSKA BAZA PODATAKA U PRIMJENI

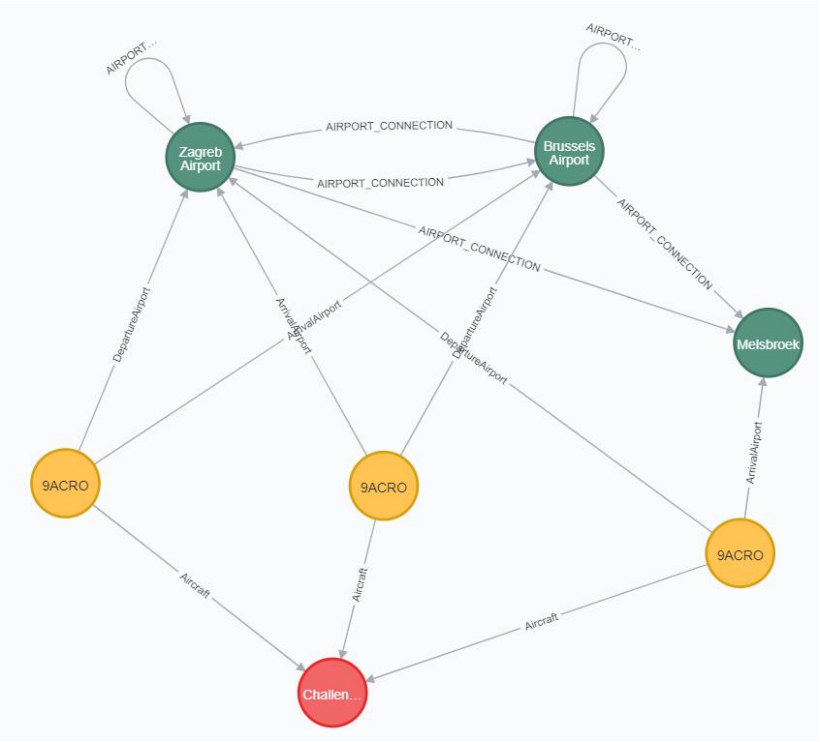
Preuzimanjem većeg dijela podataka te uspostavom automatiziranog preuzimanja preostalih podataka ostvareni su preduvjeti za rad s grafovskom bazom podataka. Slika 6 prikazuje dijagram dijela strukture rješenja. Slika 7 prikazuje letove aviona s aerodroma Varaždin u određenom razdoblju. Slika 8 prikazuje podatke izabranog aviona, njegove letove u 10. mjesecu 2020. godine i veze između aerodroma s kojih je avion poletio ili na koje je sletio.



Slika 6. Dijagram dijela strukture rješenja



Slika 7. Letovi aviona s aerodroma Varaždin u određenom razdoblju



Slika 8. Izabrani avion i njegovi letovi u 10. mjesecu 2020. godine

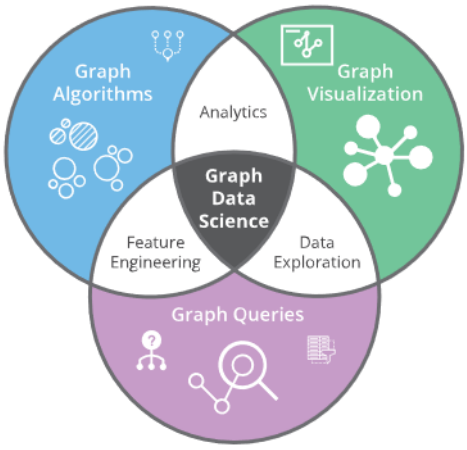
Vizualizacija podataka pomoću grafova predstavlja značajnu prednost kod analize podataka u odnosu na tablični oblik. Tu se Neo4J platforma prikazala vrlo korisna. Sljedeće iskušenje koje je postavljeno pred Neo4J bili su algoritmi na grafovima. Za primjer kreiran je memorijski graf u kojem su čvorovi aerodromi iz 3 države: Hrvatske, Austrije i Njemačke. Primijenjen je algoritam na grafu pod nazivom Yen's Shortest Path[††]. Traži se najkraći putevi između dva aerodroma: LDZA (Zagreb) i EDDP (Leipzig) pri čemu se pod najkraćim putem misli na broj kilometara koje treba preletjeti. Slika 9 prikazuje rezultat algoritma u vidu 5 najkraćih puteva s čvorovima (aerodromima) koji se nalaze na putu. Očito je najkraći put pod brojem 0 koji ima 692 km, a to je direktni let između dva aerodroma. U ostalim putevima sudjeluje barem još jedan ili dva aerodroma.

index	sourceNodeName	targetNodeName	round(totalCost/1000)	nodeNames	costs
0	"Zagreb Airport"	"Leipzig/Halle Airport"	692.0	["Zagreb Airport", "Leipzig/Halle Airport"]	[0.0, 692299.0335620967]
1	"Zagreb Airport"	"Leipzig/Halle Airport"	749.0	["Zagreb Airport", "Vienna International Airport", "Leipzig/Halle Airport"]	[0.0, 266230.225163568, 748967.869423795]
2	"Zagreb Airport"	"Leipzig/Halle Airport"	770.0	["Zagreb Airport", "Vienna International Airport", "Dresden Airport", "Leipzig/Halle Airport"]	[0.0, 266230.225163568, 658726.3456587654, 770146.917426127]
3	"Zagreb Airport"	"Leipzig/Halle Airport"	779.0	["Zagreb Airport", "Munich Airport", "Leipzig/Halle Airport"]	[0.0, 435789.57296136563, 779059.2136247279]
4	"Zagreb Airport"	"Leipzig/Halle Airport"	888.0	["Zagreb Airport", "Munich Airport", "Dresden Airport", "Leipzig/Halle Airport"]	[0.0, 435789.57296136563, 776359.0958633162, 887779.6676306778]

Slika 9. Rezultat 5 najkraćih puteva između aerodroma LDZA i EDDP

Kroz dva zadnja poglavlja prikazane su mogućnost Neo4J kao grafovske baze podataka. Te mogućnosti mogu se zvati područjima. Započelo se upitima za punjenje podataka. Slijedeći dio bavio se vizualizacijom podataka u obliku grafova. Na kraju je prikazana jedna od mogućnosti izvršavanje algoritma na grafovima. Slika 10 prikazuje spomenuta tri područja s njihovim međusobnim vezama. Središnje mjesto presjeka triju područja (grafovski upiti, vizualizacija grafa i algorotama na grafovima)) čini

[††] https://neo4j.com/docs/graph-data-science/current/algorithms/yens/



Slika 10. Područja Neo4J (izvor: https://neo4j.com/product/)

6. WEB SERVIS S PODACIMA U NEO4J

Podaci u Neo4j grafovskoj bazi podataka mogu se izložiti/dohvatiti putem različitih vrsta web servisa: REST i GraphQL. Neo4j ima službenu podršku za programske jezike:

- GO
- Java
- Javascript
- .NET
- Python

Neo4j zajednice daje podršku za programske jezika: PHP, Ruby, R, Erlang, Clojure, C/C++. Za potrebe projekta odlučeno je da će implementacija biti u programskom jeziku Java uz REST web servisa rađene za novu Jakarta EE 9 platformu. Alternativno rješenje koristi programski jazik Java i GraphQL na Spring platformi. Druga implementacija GraphQL usmjerena je na druge tehnologije koje su objedinjene pod nazivom GRANDstack. Radi se o sljedećim tehnologijama:

- GraphQL
- React
- Apollo
- Neo4J baza podataka.

7. ZAKLJUČAK

Na iskustvima rada na ranije spomenutom projektu može se zaključiti da Neo4j kao grafovska baza podataka i šire kao platforma svojim osobina daje velike mogućnosti, od pohranjivanja velike količine podataka, modeliranja do znanosti o podacima. Potrebno je vrijeme za „mentalni" prijelaz s relacijskog modela podataka na „paradigmu" grafovske baze podataka kako bi se stvarno mogle iskoristiti njezina prednosti. Prijelaz na razini podataka nije uvijek jednostavan kada se radi o velikoj količini podataka na koju su postavljeni uvjeti da ne smije biti gubitaka podataka ili njihovog dupliciranja. Drugu dimenziju problema predstavlja aktivna produkcijska baza podataka.

Literatura:

1 Rabuzin Kornelije, Martina Šestak: Graph Database Management Systems: The Past, the Present, and the Future, Encyclopedia of Information Science and Technology, Fifth Edition, poglavlje 53, Information Resources Management Association, IGI, USA, 2020.

2 Rabuzin, Kornelije; Šestak, Martina: Creating Triggers with Trigger-By-Example in Graph Databases, Proceedings of the 8th International Conference on Data Science, Technology and Applications - Volume 1, Prag, Češka: SCITEPRESS - Science and Technology Publications, 137-144, 2019.

3 Rabuzin, Kornelije; Šestak, Martina: Towards Inheritance in Graph Databases, Proceedings of 2018 International Conference on Information Management and Processing, London, Ujedinjeno Kraljevstvo: IEEE, 115-119, 2018.

4 Rabuzin, Kornelije; Konecki, Mario; Šestak, Martina: Implementing CHECK Integrity Constraint in Graph Databases, Proceedings of the 82nd The IIER International Conference, Suresh, P. (ur.). Odisha: IRAJ, Berlin, Njemačka, 19-22, 2016.

5 Rabuzin, Kornelije; Šestak, Martina; Konecki, Mladen. Implementing UNIQUE Integrity Constraint in Graph Databases, Proceedings of The Eleventh International Multi-Conference on Computing in the Global Information Technology, Westphall, C. M.; Nygard, K. ; Ravve, E. (ur.). Barcelona, 48-53, 2016.

6 Šestak, Martina; Rabuzin, Kornelije; Novak, Matija: Integrity constraints in graph databases - implementation challenges, Proceedings of Central European Conference on Information and Intelligent Systems, Hunjak, Tihomir;

Kirinić, Valentina; Konecki, Mario (ur.). Varazdin: Faculty of Organization and Informatics, University of Zagreb, 23-30, 2016.

7 Robinson, Ian; Webber, Jim; Eifrem, Emil: Introduction to graph databases, O'Reilly, USA, 2013.

8 Neo4J platforma, https://neo4j.com/developer/graph-platform/ (19.01.2021.)

9 Neo4J, https://neo4j.com/product/ (19.01.2021.)

10 Matthias Schäfer, Martin Strohmeier, Vincent Lenders, Ivan Martinovic, Matthias Wilhelm. Bringing up OpenSky: A large-scale ADS-B sensor network for research, ACM/IEEE International Conference on Information Processing in Sensor Networks, April 2014.

11 Novak, Matija; Kermek, Dragutin; Magdalenić, Ivan. Proposed architecture for ETL workflow generator // Proceedings of Central European Conference on Information and Intelligent Systems / Strahonja, Vjeran ; Hertweck, Dieter ; Kirinić, Valentina (ur.). Varaždin: Faculty of Organization and Informatics, University of Zagreb, 2019. str. 297-304

12 GRANDstack, https://grandstack.io/ (19.01.2021.)

Podaci o autorima:

prof. dr. sc. 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.

prof. dr. sc. Kornelije Rabuzin

e-mail: konelije.rabuzin @foi.hr

dr. sc. 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

RANJIVOST I SIGURNOST WEB SERVISA NA BAZI GRAPHQL

dr. sc. Matija Novak, prof. dr. sc. Dragutin Kermek, mag. inf. Matija Kaniški

SAŽETAK:

Korištenost web servisa u razvoju aplikacija sve je veća. Osim popularnih web servisa baziranih na SOAP i REST tehnologijama danas se sve više koristi GraphQL. GraphQL razvio je Facebook 2011 godine, a od 2015. godine je otvorenog koda i dostupan javnosti. GraphQL kao i svaka druga tehnologija ima svoje prednosti i nedostatke. U ovom radu uspoređen je GraphQL i REST te su dani razlozi zašto koristiti GraphQL. Opisana je implementacija web servisa baziranih na GraphQL tehnologiji, a nakon toga rad se fokusira na sigurnost web servisa baziranih na GraphQL tehnologiji. Opisane su ranjivosti koje GraphQL ima te načini kako učiti GraphQL web servise sigurnijima.

ABSTRACT:

The use of web services in application development is increasing. In addition to popular web services based on SOAP and REST technologies, GraphQL is more and more frequently used today. GraphQL was developed by Facebook in 2011 and has been open source and available to the public since 2015. GraphQL like any other technology has its advantages and disadvantages. In this paper, GraphQL and REST are compared and the reasons why to use GraphQL are given. Then the implementation of web services based on GraphQL technology is described. After that the paper focuses on the security of web services based on GraphQL technology. In specific vulnerabilities of GraphQL are described, as well as ways to make GraphQL web services more secure.

1. UVOD

U razvoju Web aplikacija sve više se koriste razne vrste web servisa (eng. Web Services). Web servisi su, pojednostavljeno rečeno, softverske komponente (aplikacije) koje putem mrežnih protokola (primarno preko HTTP ili HTTPS) pružaju neku uslugu (npr. pristup vremenskim podacima) drugim aplikacijama koristeći pritom *standardizirane* tehnologije. Glavni problem kojeg rješavaju web servisi jest *način komunikacije* između aplikacija. Web servisi, omogućuju da različite aplikacije međusobno razmjenjuju podatke putem mreže neovisno o programskom jeziku, platformi, operativnom sustavu i sl. Da bi neka komponenta (aplikacija) bila web servis mora biti *samoopisujuća* kako bi se znalo što radi, mora je se moći *locirati* i *pozvati* [1].

Prema tome pojam web servisi dolazi od činjenice da je to servis kojem se pristupa putem web protokola. S duge strane, prema [2] naziv web servisi dolazi od pojma „Web of Services", što znači da distribuirane aplikacije se grade kao mreža povezanih softverskih servisa, slično kao što se web aplikacije grade kao mreža povezanih HTML stranica. Postoje i druge definicije koje se mogu pronaći na webu [3]. Razmatranje što je točno značenje i od kuda dolazi pojam Web Servis je izvan fokusa ovog članka. Ono što je važno za ovaj članak jest da web servisi se bave *načinom komunikacije* između aplikacija koristeći *standardizirane tehnologije*.

U prvoj rečenici spomenuto je da postoje različite vrste web servisa. Pod vrste misli se na različite tehnologije/standarde koji se koriste da bi se implementirao web servis. Najpoznatije vrste su: SOAP (eng. Siple Object Access Protocol) i REST (eng. Representational State Transfer). SOAP specifikacija 1.0 objavljena je 1999. godine od strane Microsofta, a prva javno prihvaćena verzija 1.1 objavljena je 2000. godine kada se IBM zainteresirao za SOAP [1, 4].

REST je definirano Roy T. Fielding 2000. godine u svojoj doktorskoj disertaciji [5]. Jedno vrijeme ove dvije tehnologije bile su dva standarda koja su se većinom koristila za izgradnju web servisa. Danas sve više postaju popularniji web servisi koji su bazirani na GraphQL tehnologiji. GraphQL razvio je Facebook 2011 godine, a od 2015. godine je otvorenog koda.

U ovom radu fokus je na web servisima baziranih na GraphQL tehnologiji. Kao i svaka druga tehnologija GraphQL nije savršen i ima određenih ranjivosti. Stoga u ovom radu dati će se pregled poznatih ranjivosti i opisati načini kako takve web servise učiniti sigurnijima. U poglavlju 2 ukratko su opisane karakteristike i dana je usporedba web servisa baziranih na SOAP, REST i GraphQL tehnologijama. U poglavlju 3 opisan je način kreiranja web servisa baziranih na GraphQL tehnologiji. Poglavlje 4 daje opis ranjivosti i sigurnosne savjete za izgradnju web servisa općenito i poseban je osvrt na GraphQL tehnologiju. Poglavlje 5 daje zaključak.

2. VRSTE WEB SERVISA

U uvodnom poglavlju navedene su tri vrste web servisa baziranih na: SOAP, REST i GraphQL tehnologiji. U ovom poglavlju dan je kratak opis sve tri tehnologije i njihova usporedba.

2.1. SOAP

SOAP je nastao iz XML-RPC protkola kojeg je inicijalno definirao Dave Winer. SOAP se bazira na razmjeni XML dokumenata primarno preko HTTP protokola, koji se nazivaju SOAP poruke. SOAP prouke mogu se slati i putem drugih protokola poput SMTP [6] no u specifikaciji se navodi HTTP [7]. SOAP poruka sastoji se od omotnice (eng. envelope) koja u sebi sadrži zaglavlje (eng. head) i tijelo (eng. body). Omotnica označava gdje počinje i gdje završava SOAP poruka. Zaglavlje je opcionalno i može sadržavati informacije o pošiljatelju, primatelju i sl. Tijelo poruke sadrži podatke zahtjeva, odgovora i poruke o grešci. Kada aplikacija koristi SOAP klijent kreira XML dokument koji sadrži ispravno formatiranu SOAP poruku prema SAOP specifikaciji. Server će pročitati tu poruku, obraditi zahtjev i vratiti ispravno formatiranu SOAP poruku s odgovorom.

Dodatno na svaku poruku veže se i određena XML shema koja govori primatelju SOAP poruke kako „čitati" SOAP poruku. Za specifikaciju sheme koristi se WSDL (eng. Web Services Description Language). Za više detalja oko SOAP protokola pogledajte [4]

Glavni nedostatak SOAP-a jest da XML dokumenti za komunikaciju mogu brzo postati jako kompleksni. Ukoliko je u nekom trenutku potrebno ručno kreirati SOAP poruku to može biti izuzetno problematično jer poruka mora biti ispravno formatirana. Na svu sreću većina razvojnih okruženja danas omogućuje automatsko generiranje SOAP poruka što je zahvaljujući WSDL specifikaciji. Prednost SOAP poruke jest da imaju ugrađeno upravljanje pogreškama što olakšava ispravljanje grešaka kada se one pojave.

2.2. REST

Kada se REST pojavio on je postao jednostavnija alternativa za SOAP. Glavna prednost REST-a je da slanje samih zahtjeva je puno jednostavnije jer često je samo potrebno definirati URL do servisa, a sam zahtjev je već unutar tog URL-a. Na primjer, http://domena.hr/servis/korisnici/{id} gdje se id samo zamjeni s id-om korisnika. U SOAP varijanti treba kreirati ispravno formatiranu SOAP poruku da bi se dohvatila ista stvar.

Za razliku od SOAP servisa, REST servisi vraćaju odgovor u primarno u JSON formatu, ali se mogu koristiti i drugi formati. Ukoliko je potrebno REST servisi mogu imati parametre. Za više detalja oko REST servisa može se pronaći u [8].

Kao što SOAP ima WSDL tako REST ima WADL (eng. Web Application Description Language). WADL je XML jezik za opis REST web servisa. Kako je REST jednostavniji WADL nije obavezna komponenta što je ujedno i nedostatak. Često se mogu pronaći servisi koji nemaju WADL te je nemoguće automatsko generiranje klijenta od strane razvojnog okruženja za komuniciranje sa servisom [8]. Međutim potrebno je napomenuti da REST ima određena ograničenja kojih se treba pridržavati prilikom dizajna servisa.

Ako REST servis se pridržava svih tih ograničenja tada ga možemo nazvati RESTful servis. Dobar opis kako doći do ispravnog RESTful servisa opisao je Martin Fowler na svom blogu [9] gdje opisuje RESTful model zrelosti kojeg je osmislio Leonard Richardson [10].

Jedna specifičnost REST servisa je također da su u potpunosti bazirani na HTTP protokolu te koristi HTTP metode POST, PUT, DELETE i sl. za različite vrste akcija nad servisom. Kako SOAP servis kreira XML poruku koja nije nužno ovisna o HTTP-u može se koristiti i neki drugi protokol za komunikaciju.

2.4. REST ili SOAP

Iako izgleda da je REST nastao da zamjeni SOAP to nije točno. REST i SOAP imaju različitu namjenu. Svaki ima svoje prednosti i nedostatke. Prema tome SOAP je na primjer bolji u situacijama kada je: potrebno imati jasnu specifikaciju komunikacije između aplikacija (obavezan WSDL), kada je potrebna već gotova podrška za upravljanje greškama, kada je XML jedini format komunikacije, kada se ne želi koristiti HTTP i sl.

REST s druge strane je na primjer bolji kada: postoji više formata komunikacije (JSON, XML, CSV, ...), kada se koristi JavaScript programski jezik (JSON format), kada nam je bitna jednostavnost korištenja, kada se želi smanjiti količina podataka prilikom prijenosa (nema zaglavlja) i sl. Jednostavna tablica usporedbe SOAP i REST može se pronaći ovdje [11].

2.5. GraphQL

GraphQL je najmlađa od sve tri tehnologije i nastala je kao rješenje za probleme koje imaju REST servisi. Usporedba REST i GraphQL dana je na kraju sljedećeg poglavlja [12].

Prema [13] je jezik upita za aplikativna sučelja (eng. Application Program Interface - API) i softver na poslužitelju za izvršavanje istih. GraphQL nije vezan uz nijedan programski jezik ili bazu podataka. Na službenoj stranici [14] mogu se pronaći brojne biblioteke za razne programske jezike koje olakšavaju izgradnju GraphQL servisa.

GraphQL servis kreira se definiranjem tipova i atributa te funkcija za dohvaćanje atributa. Kada je gotov pristupa mu se putem njegove URL adrese. Prilikom poziva GraphQL servisa šalje mu se upit koji se validira i izvršava na serveru [13].

3. KREIRANJE GRAPHQL SERVISA

Kako bi se kreirao GraphQL servis potrebno je odabrati programski jezik. Za potrebe ovog članka odabran je programski jezik PHP. Postoje brojne biblioteke koje se mogu koristiti [14] kako bi se ubrzao razvoj. Neke biblioteke su specifično rađene za neki PHP okvir (npr. Laravel, Symfony i sl.) dok su druge rađene za čisti PHP. U ovom radu koristi se biblioteka *graphql-php* [15]. Ova biblioteka jest implementacija GraphQL specifikacije [16] za programski jezik PHP.

Kako bi se koristila graphql-php biblioteka najjednostavnije je instalirati ju korištenjem *composer* upravitelja [17].

```
composer require webonyx/graphql-php
```

Ako se ne želi koristiti composer tada je moguće preuzeti izvorni kod dodatka sa [18] gdje je kod već obrađen i može se ga koristiti u vlastitom projektu bez instalacije. U ovom radu koristila se varijanta bez instalacije. Nakon raspakiranja kopira se *vendor* direktorij u projekt, te je potrebno uključiti *autoload.php* skriptu u vlastiti projekt. Ta skripta dalje uključuje sve ostale skripte iz biblioteke. Nakon uključivanja skripte potrebno je uključiti osnovne klase (BuildSchema, GraphQL) za rad. Sam kod nakon toga izgleda ovako:

```
require_once __DIR__ . '/../vendor/autoload.php';
use GraphQL\Utils\BuildSchema;
use GraphQL\GraphQL;
```

3.1. GraphQL SDL

Sljedeći korak je definirati shemu. Shema je opis strukture je specifikacija našeg servisa i opisuje način korištenja servisa, na neki način shema je kao WADL ili WSDL kod REST ili SOAP servisa. Postoji više načina kako se shema može definirati s ovom bibliotekom [19], a u ovom radu koristit će se GraphQL SDL (eng. Schema Definition Language) [20, 21].

U ovom jednostavnom primjeru napravit će se servis koji radi s korisnicima te je potrebno definirati odgovarajuću shemu. Prvo je potrebno kreirati početni element *schema* koji ima standardni atribut *query*. U ovom primjeru atribut *query* vraća podatak tipa *Upit* koji predstavlja korijenski tip u ovom primjeru. Nakon toga definiran je sam tip *Upit* koji će vraćati sve korisnike koji postoje u sustavu te on ima atribut *korisnici* koji je tipa *Korisnik*. No kako su stavljene uglate zagrade oko tipa *Korisnik* to znači da se vraća lista korisnika.

```
schema { query: Upit }
type Upit {  korisnici: [Korisnik] }
```

Za svakog korisnika postoje tri podatka (ime, prezime, telefon). Potrebno je definirati tip *Korisnik* i pripadajuće atribute s tipom podatka. Kada atributi imaju jednostavne tipove podataka koje u GraphQL-u nazivaju skalarni tipovi (eng. scalar type) tada smo došli do takozvanih listova. Listovi predstavljaju konkretne podatke koje će servis vraćati. Uskličnik na kraju tipa nekog atributa (npr. ime) označava da će servis svaki puta vratiti taj podatak ako nema uskličnika tada se taj podatak može, ali ne mora vratiti (npr. telefon).

```
type Korisnik {
  ime: String!
  prezime: String!
  telefon: Int
}
```

Kreiranu shemu spremamo u datoteku pod nazivom shema.graphql. Glavni program nakon toga mora pročitati shemu i generirati objekt za rad.

```
$shemaSpecifikacija = file_get_contents('shema.graphql');
$shema = BuildSchema::build($shemaSpecifikacija);
```

3.2. Obrada GraphQL upita

Kada je shema spremna potrebno je napraviti logiku koja čita ulazni GraphQL upit u JSON formatu i obrađuje ga te vraća odgovor u JSON formatu. Pretpostavka je da je korijenski element u zahtjevu pod nazivom *query* koji dalje sadrži GraphQL upit. Program tada čita zahtjev, vadi upit te ga izvršava sa executeQuery funkcijom kojoj mora dati shemu, upit i podatke. Kada se upit obradi podaci se prebacuju u JSON format i vraćaju kao odgovor. Ovaj dio stavljen je u *try-catch* blok te ako dođe do greške vraća se poruka o grešci.

```
try{
   $zahtjev = file_get_contents('php://input');
   $zahtjevJSON = json_decode($zahtjevText, true);
   $upit = $zahtjevJSON['query'];
   $result = GraphQL::executeQuery($shema, $upit, $podaci);
   $rezultat = GraphQL::executeQuery($shema, $upit, $podaci);
   $odgovor = $rezultat->toArray();
} catch (\Exception $e) {
    $odgovor = [ 'greska' => [ 'poruka' => $e->getMessage()] ];  }
header('Content-Type: application/json; charset=UTF-8');
echo json_encode($odgovor);
```

Osim sheme i upita potrebni su i podaci, u ovom primjeru podaci se čitaju iz CSV datoteke koja sadrži podatke odvojene zarezom. Te se kreira asocijativni niz za svakog korisnika i dodaje u indeksirani niz korisnici. Anonimna funkcija koja se ovdje koristi poziva se prilikom izvršavanja upita.

```
$podaci = [
  'korisnici' => function($rootValue, $args, $context) {
    $korisnici=array();
    $d=fopen("podaci.csv","r");
    while($red = fgetcsv($d)){
       $korisnik["ime"] = $red[0];
```

```
        $korisnik["prezime"] = $red[1];
        $korisnik["telefon"] = $red[2];
        $korisnici[] = $korisnik;
    }
    fclose($d);
    return $korisnici;
}];
```

3.3. Korištenje GraphQL servisa

Kako bi lakše pristupali i koristili GraphQL servis može se instalirati ChromeiQL dodatak u preglednik [22]. Ovaj dodatak je omotač oko GraphiQL razvojnog okruženja te omogućuje pristup bilo kojem GraphQL servisu. Za pristup servisu potrebno je samo definirati URL do servisa i naravno željeni upit.

Ako želimo dohvatiti sve korisnike iz prethodnog primjera stavimo:

```
{ Upit : korisnici {ime, prezime} }
```

Odgovor koji se dobiva jest:

```
{ "data": { "Upit": [
      { "ime": "Matija", "prezime": "Novak" },
      { "ime": "Matija", "prezime": "Kaniški"},
      { "ime": "Dragutin", "prezime": "Kermek" }
    ]
}}
```

Važno je napomenuti kada se koristi dodatak query dio se automatski dodaje. Ako bi se isti upit izvršavao ručno npr. curl komandom tada je treba staviti i query ključnu riječ.

```
curl http://localhost:8080 -d '{"query": "{Upit : korisnici{ime,prezime}}" }'
```

3.4. Filtiranje i dodavanje podataka

Ako se želi dodati funkcionalnost dohvata podataka po imenu tada treba ažurirati tip korisnici u shemi da može primati argumente.

```
type Upit { korisnici(ime: String): [Korisnik] }
```

Te u *while* petlju koja vraća podatke doda se logika za filtriranje. Informacija o unesenom imenu dobiva se iz *args* varijable koju prima anonimna funkcija.

```
if(isset($args["ime"])){
   if(strpos($korisnik["ime"],$args["ime"])!==false)     $korisnici[]     =     $korisnik;
} else { $korisnici[] = $korisnik; }
```

Na kraju upit tada izgleda ovako:

```
{Upit : korisnici(ime : "M") {ime, prezime}}
```

Detaljnije objašnjavanje funkcioniranja GraphQL-a je van ovog rada i ovaj primjer samo služi kao brzi uvod u GraphQL kako bi se lakše razumjele ranjivosti koje su opisane u sljedećem poglavlju. Više informacija o GraphQL-u može se pronaći u [23] ili na web stranicama [13] ili [21] gdje postoje besplatni tečajevi. Kod ovog primjera dostupan je na https://github.com/matnovak-foi/phpGraphQLExample .

3.5 REST ili GraphQL

Iz prethodnog primjera mogu se vidjeti sličnosti REST i GraphQL servisa. Oboja dvije vrste ne pamte stanje, koriste JSON, imaju strukturirani pristup podacima, oba koriste HTTP i sl. Međutim ključna razlika jest u tome da kod GraphQL-a postoji samo jedna krajnja točka (jedan url) preko koje se dohvaćaju podaci. Ako bi napravili prethodni primjer kao REST servis tada bi imali barem dvije putanje /korisnici za dohvat svih korisnika i /korisnici/{id} za dohvat jednog korisnika. Kako bi složenost aplikacije rasla sve više putanja bi imali, a kako bi trebali sve više podataka potrebno je raditi više poziva na servis. Kod GraphQL-a to se ne dešava jer se upit uvijek šalje na istu krajnju točku i unutar njega se definiraju željeni podaci. Kako bi složenost aplikacije rasla upiti bi isto bili složeniji no i dalje bi se slao samo jedan zahtjev na servis neovisno koliko podataka dohvaćali. Razumljivo je da kod aplikacija koje imaju brdo korisnika u isto vrijeme to može biti vrlo korisno.

Druga razlika je u tome da REST ima fiksnu strukturu koju vraća kod svakog poziva kod GraphQL-a struktura nije fiksna i varira ovisno o podacima. Najjednostavniji primjer jest koje podatke o korisniku želimo u GraphQL upitu može se točno specificirati dok kod REST servisa to nije moguće. Zbog toga GraphQL nema problem da dohvati previše ili premalo podataka jer korisnik prilikom poziva specificira što točno želi. Postoje i druge razlike poput brzine razvoja servisa, performanse i sl. koje sada se neće navoditi no mogu se pronaći u [24, 25].

No bez obzira na prednosti GraphQL nije savršen i ima određene probleme. Jedna domena jest sigurnost te kao i svaki drugi servis i GraphQL servisi imaju svoje ranjivosti.

4. RANJIVOSTI GRAPHQL-A

Svaka nova tehnologija ima prednosti, ali također može izložiti aplikaciju novim napadima. GraphQL tu nije iznimka. Svrha ovog poglavlja je ukazati na uobičajene pogreške u konfiguraciji i moguće napade na aplikacije koje koriste GraphQL. Neki napadi su specifični za GraphQL (npr. introspekcija, višestruki napadi), a neki su generički za sve vrste API-je (npr. SQL/NoSQL umetanje, XSS napadi) [26].

4.1. Introspekcija

Introspekcija (eng. Introspection) je metoda koja putem upita nad GraphQL shemom vraća detaljnije informacije o istoj. Upitom introspekcije moguće je saznati sve upite koja shema podržava, dostupne podataka i koristan je za lakše testiranje i razvoj GraphQL-a [27].

4.1.1. GraphQL sustav introspekcije

Postoji nekoliko načina za dohvaćanje ovih podataka i vizualizaciju rezultata. Najjednostavniji način je slanje HTTP zahtjeva sa sljedećim korisnim teretom (eng. Payload) kao u primjeru [28]:

```
query IntrospectionQuery {
  __schema {
    queryType { name  }
    mutationType {
      name
    }
    subscriptionType { name  }
    types {
      ...FullType
    }
    directives {
      name
      description
      locations
      args {
        ...InputValue
      }   }  }  }
fragment FullType on __Type {
  kind
  name
  description
  fields(includeDeprecated: true) {
    name
    description
        args {
      ...InputValue
    }
    type {
      ...TypeRef
    }
    isDeprecated
    deprecationReason
  }
inputFields {
    ...InputValue
  }
interfaces {
    ...TypeRef
  }
  enumValues(includeDeprecated: true) {
    name
    description
    isDeprecated
    deprecationReason
  }
  possibleTypes {    ...TypeRef   }
}
fragment InputValue on __InputValue {
  name
  description
  type {
    ...TypeRef
  }
  defaultValue
}
fragment TypeRef on __Type {
  kind
  name
  ofType {
    kind
    name
    ofType {
      kind
      name
      ofType {
        kind
        name
        ofType {
          kind
          name
          ofType {
            kind
            name
            ofType {
              kind
              name
              ofType {
                kind
                name
                ofType {
                  kind
                  name
                }
}}}}}}
```

Rezultat upita introspekcije je obično vrlo dugačak i sadržava cijelu shemu implementacije GraphQL-a. Primjer naše sheme (ovdje je skraćen):

```
defaultValue {
  "data": {
    "__schema": {
      "queryType": {
        "name": "Upit"
      },
      "mutationType": {
        "name": "Upravljanje"
      },
      "subscriptionType": null,
      "types": [
        {
          "kind": "OBJECT",
          "name": "Upit",
          "description": null,
          "fields": [
            {
              "name": "korisnici",
              "description": null,
              "args": [
                {
                  "name": "ime",
                  "description": null,
                  "type": {
                    "kind": "SCALAR",
                    "name": "String",
                    "ofType": null
                  },
                  "defaultValue": null
                }
              ],
              "type": {
                "kind": "LIST",
                "name": null,
                "ofType": {
                  "kind": "OBJECT",
                  "name": "Korisnik",
                  "ofType": null
                }
              },
              "isDeprecated": false,
              "deprecationReason": null
            }
          ],
          "inputFields": null,
          "interfaces": [],
          "enumValues": null,
          "possibleTypes": null
        },
        {...},
        {
          "kind": "OBJECT",
          "name": "Korisnik",
          "description": null,
          "fields": [
            {
              "name": "ime",
              "description": null,
              "args": [],
              "type": {
                "kind": "NON_NULL",
                "name": null,
                "ofType": {
                  "kind": "SCALAR",
                  "name": "String",
                  "ofType": null
                }
              },
              "isDeprecated": false,
              "deprecationReason": null
            },
            {
              "name": "prezime",
              "description": null,
              "args": [],
              "type": {
                "kind": "NON_NULL",
                "name": null,
                "ofType": {
                  "kind": "SCALAR",
                  "name": "String",
                  "ofType": null
                }
              },
              "isDeprecated": false,
              "deprecationReason": null
            },
            {
              "name": "telefon",
              "description": null,
              "args": [],
              "type": {
                "kind": "SCALAR",
                "name": "Int",
                "ofType": null
              },
              "isDeprecated": false,
              "deprecationReason": null
            }
          ],
          "inputFields": null,
          "interfaces": [],
          "enumValues": null,
          "possibleTypes": null
        },
        {...},
        {...},
        {...},
        {...},
        {...},
        {...},
        {...},
        {...},
        {...},
        {...},
        {...},
        {...},
        {...},
        {...},
      ],
      "directives":{...},
    }
  }
```


4.1.2. GraphiQL Voyager

Alat kao što je GraphQL Voyager [29] može se koristiti za bolje razumijevanje krajnje točke GraphQL-a. U alatu se odabere izrada sheme (eng. Create Schema) i pod Introspection karticom se zalijepi rezultat upita introspekcije. Iz sheme

GraphQL-a kreira se dijagrama entiteta i veza (ERA). ERA modela olakšava izradu upita. Slike 1. prikazuje ERA model sheme našeg primjera iz kojeg se vidi koje sve atribute i tipove podataka ima tablica korisnik.

Slika 1. ERA model GraphQL sheme

Nedostatak korištenja GraphQL Voyager-a je što ne prikazuje baš sve za unesenu shemu. Ne vide se primjerice mutacije (eng. Mutation) na ERA dijagramu. Zbog toga se GraphQL Voyager-a koristi najčešće zajedno s GraphiQL-a [30] ili GraphQL Playground-a [31].

4.1.3. GraphiQL

GraphiQL je web bazirano integrirano razvojno okruženje (IDE) za GraphQL. Dio je projekta GraphQL-a i uglavnom se koristi za uklanjanje pogrešaka ili u svrhu razvoja. GraphiQL ima karticu za dokumentaciju koji koristi podatke iz sheme kako bi stvorio dokument instance GraphQL koja se koristi. Kreirani dokument sadrži vrste podataka, mutacije i svaki podatak koji se može dobiti putem introspekcije.

4.1.4. GraphQL Playground

GraphQL Playground je klijent GraphQL-a. Može se koristiti za testiranje različitih upita, kao i za dijeljenje GraphQL IDE-a na različite projekte grupiranje po temi ili nazivu. GraphQL Playground za razliku od GraphiQL automatski može stvoriti dokument instance GraphQL-a. Potrebno je samo definirati URL do krajnje točke GraphQL-a ili odabrati datoteku sheme. Koristi se za testiranje ranjivosti.

4.2 Primjeri napada

Rezultat upita za introspekciju u ranijem primjeru sadrži upit nazvan korisnici. Ovo se čini kao dobro mjesto za otkrivanje osjetljivih podataka kao što su ime, prezime, telefon, itd.

Upit:

```
query {
  korisnici(ime: "M") {
    ime,
    prezime,
    telefon
  }
}
```

Rezultat tog upita je:

```
{ "data": {
    "korisnici": [
      {
        "ime": "Matija",
        "prezime": "Novak",
        "telefon": 9999999
      },
      {
        "ime": "Matija",
        "prezime": "Kaniški",
        "telefon": 8888888
      }
    ]
  }
}
```

Postupak autorizacije razlikuje se od implementacije do implementacije jer će svaka shema imati različite osjetljive podatke. U ranijem primjeru svaki korisnik (čak i neautentificiran) može dobiti pristup podacima svakog korisnika. Informacije dobivene iz sheme mogu se koristiti i za izvršavanje radnji koje su definirane shemom. Sljedeći primjer koristi mutaciju Upravljanje za kreiranje novog korisnika:

```
mutation {
  dodajKorisnika(ime: "Pero", prezime: "Perić", telefon: 888888) {
    ime
    prezime
  }
}
```

Ovu vrstu radnje moguće je izvršiti jer ne postoji provjera autorizacije korisnika.

4.2.1. SQL/NoSQL umetanje

GraphQL sam po sebi ne sprječava bilo kakvu vrstu napada. Aplikacija bez parametriziranih upita može biti ranjiva na napade SQL umetanja. Tako prema [32] dodavanje jednog navodnika (**'**) argumentu dovoljno je za generiranje sintaksne pogreške MySQL-a.

```
query {
    korisnici(ime: "M'") {
        ime,
        prezime,
        telefon
    }
}
```

Aplikacija možda neće izbaciti pogrešku ali i dalje može biti ranjiva na napade SQL umetanja. Skalarni tipovi isto tako mogu biti ranjivi. U većini slučajeva dovoljno umetnuti zlonamjerni kod između dvostrukih navodnika (").

Autor u [33] autor govori da je GraphQL otporan na NoSQL Injection no to nije točno kao što i isti autor kasnije sam prepoznaje [34]. U svojem primjeru prihvaća kriterije pretraživanja kao JSON objekt pomoću vlastitog skalarnog tipa.

```
type Query {users(search: JSON!, options: JSON!): [User]}
```

Polja search i options izloženi su napadu. Napad za search polje bi mogao izgledati ovako:

```
{
    users(search: "{\"email\": {\"$gte\": \"\"}}",
          options: "{\"skip\": 0, \"limit\": 10}") {
        _id
        username
        fullname
        email
    }
}
```

$gte odabire dokumente u kojima je vrijednost polja veća ili jednaka određenoj vrijednosti. Čitav JSON search objekt prosljeđuje se izravno u upit i ovaj upit vraća sve korisnike u kolekciji.

4.2.2. XSS umetanje

XXS napad je umetanje zlonamjernog kôda u aplikaciju koji potom pokreće preglednik [26]. U sljedećem primjeru pogrešna obrada ulaznih podataka uzrokuje izvršavanje XSS-a.

```
query xss  {
  korisnici(ime:"<script>alert('1')</script>") {
    ime,
    prezime
  }}
```

4.2.3. DoS napadi

GraphQL nudi jednostavno sučelje koje programerima omogućuje upotrebu ugniježđenih upita i ugniježđenih objekata. Ta se osobina može koristiti na zlonamjerni način pozivanjem duboko ugniježđenog upita. Upit sličan rekurzivnoj funkciji uzrokuje uskraćivanja usluge korištenjem viška CPU-a, memorije ili drugih računarskih resursa. Primjer jednog takvog upita je [35]:

```
query evil {            # dubina: 0
  album(id: 42) {       # dubina: 1
    songs {             # dubina: 2
      album {           # dubina: 3
        ...             # dubina: ...
        album {id: N}   # dubina: N
}}}}
```

4.2.4. Višestruki napadi

GraphQL podržava grupiranje više upita u jednom zahtjevu. To omogućuje korisnicima da zatraže više objekata ili više primjeraka objekata. Međutim, napadač može koristiti ovu metodu kako bi izvršio višestruki napad. Slanje više upita u jednom zahtjevu izgledalo bi ovako [35]:

```
[  {
    upit: < upit 0 >,
    varijabla: < varijable za upit 0 >,
  },  {
    upit: < upit 1 >,
    varijabla: < varijable za upit 1 >,
  },  {
    upit: < upit n >
    varijabla: < varijable za upit n >,
  }]
```

U sljedećem primjeru definirano je više upita u jednom zahtjevu za dohvaćanje svih imena korisnika koji zadovoljavaju kriterije.

```
query {
  korisnici(ime: "M") {    ime   }
  rezultat1: korisnici(ime: "D") {    ime  }
  rezultat2: korisnici (ime: "P") {   ime  }
}
```

Takvi će se zahtjevi teže detektirati od strane poslužitelja. Odnosno omogućuje se učinkovito izvršavanja napada grubom silom obrade bez da se otkrije pokušaj napada. Višestruki napadi mogu se koristiti za zaobilaženje mnogih sigurnosnih mjera implementiranim na web mjestima. Također primjenjuju se za enumeraciju objekata i pokušaj probijanja višefaktorske autentifikacije grube sile ili drugih osjetljivih podataka.

4.3. OWASP API sigurnost

Sve veće korištenje web API-ja rezultiralo je učestalim napadima na iste. Tako je u 2019. godini do 75% ukupnih napada bilo usmjereno na API-je. U svrhu zaštite API-a iste godine objavljene su OWASP smjernice od 10 najčešćih napada. Glavna razlika API-ja u odnosnu na klasične web aplikacije jest da je poslovna logika izravno dostupna putem njih. U nastavku opisani su mehanizmi zaštite REST, SOAP i GraphQL web servisa prema smjernicama OWASP-a za najčešćih 10 vrsta API napada 2019. godine [36].

4.3.1. Napadi na objekt autorizacije

Napad na objekt autorizacije (eng. Broken Object Level Authorization) iskorištava identifikator resursa za pristup resursima. Karakteristika ovog napada je nepostojanje odgovarajuće kontrole pristupa i predvidljivost ID resursa. Prvi korak za smanjenje takve vrste napada bila bi provjera identiteta korisnika koji je poslao zahtjev. U REST i GraphQL web servisima to se može postići korištenjem JSON web tokena (JWS). Kod SOAP-a isto tako će prvi korak biti identificirati korisnika koji šalje zahtjev. Alternativa je korištenje HTTP-a provjere autentičnosti. Najkorišteniji mehanizmi za provjeru autentičnosti su: UsernameToken Profile i X.508 Certificates Token Profile. Drugi korak nakon potvrde identiteta korisnika koji šalje zahtjev je autorizacija. Autorizacija osobito je važna kada se koristi GraphQL jer postoji više načina za dohvaćanje određenog polja ovisno o shemi. Dodatna mjera zaštite mogla bi biti sekvencijalno kreiranje ID-a ili generiranje univerzalno jedinstvenih identifikatora.

4.3.2. Napadi na autentikaciju

Web servisima obično nije namijenjen izravni pristup. Presudno je stoga implementirati mehanizam provjere autentičnosti. Web servis mora autenticirati korisnika koji ga zahtijeva. Uobičajen način autentikacije korisnika je kombinacijom korisničkog imena i lozinke. Nakon uspješne autentifikacije generira se JWS token za pristup REST i GraphQL web servisima. Jednom kada je token poslan klijentu važno je da se taj token ne može mijenjati i da nema neograničeno trajanje. Bez ograničavanja broja pristupa web servisu isti postaje ranjiv protiv automatiziranih napada. Napad na autentikaciju (eng. Broken Authentication) se može smanjiti postepenim povećavanjem vremena ponovnog pristupa nakon svakog neuspjelog pokušaja. Korištenje vremena kao mjere za usporavanje napada povezanih s autentifikacijom su primjenjive i na SOAP. SOAP za to može koristi UsernameToken Profil-a mehanizam za autentifikaciju korisnika. Automatizirani napadi mogu se dodatno smanjiti provjerom autentičnosti putem unaprijed generiranog i podijeljenog korisničkog certifikata.

4.3.3. Pretjerana izloženost podacima

Web servisi nerijetko ovise o korisničkoj strani kako bi filtrirali sadržaj koji se želi dohvatiti u zahtjevu. Odgovor po zahtjevu za SOAP i REST web servise se može razlikovati u smislu fleksibilnosti. Zahtjev koji preuzima podatke obično dijeli rezultat kako bi se izbjeglo mijenjanje svojstava resursa koji se je tražio. Jedna od ključnih prednosti GraphQL-a je njegova fleksibilnost kada je u pitanju traženje informacija. Jedan se upit može definirati tako da vraća različita polja iz istog objekta bez promjene implementacije upita. Pretjerana izloženost podacima (eng. Excessive Data Exposure) u kontekstu GraphQL uvelike ovisi o poljima koja se traže u upitu. Preporučuje se poslati više zahtjeva koji će isključivo dohvatiti tražene podatke umjesto da se šalje manji broj zahtjeva i time riskira vraćanje podataka koji nisu traženi od korisnika.

4.3.4. Nedostatak resursa i ograničenje složenosti

Mehanizam provjere autentičnosti može se ugroziti kada je korisniku dopušteno kontinuirano slanje zahtjeva web servisu bez ikakvih ograničenja. Radi se o napadu poznatom kao Nedostatak resursa i ograničenje složenosti (eng. Lack of Resources & Rate Limiting). Cilj ovog napada ponekad nije ostvariti pristup već da iscrpiti dostupne resursi API-a. Web servis koji prihvaća HTTP zahtjeve za autentifikaciju, ali uvijek odgovara HTTPS-om može dovesti do uskraćivanja usluge (DoS) na poslužitelju. Ovisno o shemi GraphQL-a dijelovi grafa mogu biti ciklični što znači da se mogu zloupotrijebiti kako bi se generirao ugniježđeni upit. Bez postavljanja maksimalne dubine upita ovakvi ili slični upiti mogli bi obuhvatiti čitavu strukturu. Na taj način se potencijalno uzrokuje uskraćivanje usluge slanjem nekoliko dubinskih upita. Potrebno je voditi računa i koliko složeno može biti dohvaćanje određenih polja upitom. Sukladno tome kod GraphQL-a potrebno je postaviti maksimalnu dubinu upita i maksimalnu složenosti upita. Druga mogućnost u ublažavanju ove ranjivosti je postavljanje ograničenja na broj zahtjeva dopuštenih u definiranom vremenskom rasponu. Još jedna mogućnost je definirati dodatna ograničenja resursima dodijeljeni web servisima.

4.3.5. Napadi na autorizaciju funkcije

Kao što je navedeno u ranije presudno je ograničiti pristup do određenih resursa. Jednako tako važno je osigurati da sve funkcije nisu dostupne svim korisnicima. Pronalaženje dodatnih funkcionalnosti često je rezultat nagađanja temeljeno na strukturi drugih krajnjih točaka ili dostupnih operacija. Ovaj napad moguće je izvršiti nad SOAP-om i GraphQL-om kada su javno dostupni WSDL i shema. Dok bi se za REST web servise trebala uključiti Hypermedia komponenta. Cilj napada na autorizaciju funkcije (eng. Broken Function Level Authorization) je u odgovore na zahtjev uključiti krajnje točke s kojima bi korisnik mogao komunicirati. Napadaču bi to omogućilo da utvrdi koje su dostupne operacije s potencijalno.

Zaštita ove vrste napada je za provedba kontrole pristupa zasnovane na ulogama. Definirana uloga korisnika ne smije biti vrijednost koju je moguće uređivati na poslužitelju putem zahtjeva, već vrijednost koju je moguće dobiti s poslužitelja. Nakon što se odredi uloga korisnika slijedi dopuštanje zahtjeva temeljem dodijeljene uloge korisnika.

4.3.6. Masovno dodjeljivanje

Masovno dodjeljivanje (eng. Mass Assignment) podrazumijeva utvrđivanje svojstva objekta putem zahtjeva koji nije za to namijenjeni. Korisno opterećenje kod REST, SOAP i GraphQL zahtjeva šalje u XML-u ili JSON-u formatu. Parametri u zahtjevu se mogu pogoditi sljedeći logiku konvencije imenovanja web API-ja ili pomoću različitim alata. To bi bio slučaj za SOAP-om i GraphQL-om. WSDL i GraphQL shema izričito navodi koja operacija što očekuje. Druga mogućnost bila bi pronalaženje dodatnih parametara u drugoj vrsti zahtjeva koja je namijenjena istom resursu. Zaštita od takve vrste napada je izričita zabrana obrade bilo kakvog neočekivanog parametara dodan zahtjevu. Pristup osjetljivim svojstvima izvan predviđenog protoka ne smije se dopustiti. Javno objavljivanje WSDL-a za SOAP web servise i GraphQL sheme treba zabraniti. Onemogućavanje tzv. introspekcije sprječava se dobivanje informacija o shemu i slanja upita prema poljima sheme.

4.3.7. Pogrešna konfiguracija sigurnosti

Naknadno se mogu pronaći različiti nedostaci web servisa koji su bili prisutni tijekom razvoja i ugrožavaju sigurnost web API-ja. Komponenata API koja nije ispravljena može izložiti API napadima koji inače nisu izvedivi. Radi se o pogrešnoj konfiguraciji sustava (eng. Security Misconfiguration). Najčešće, greške u konfiguraciji su pristup do web servisa bez obavezne upotrebe HTTPS-a što predstavlja veliki rizik za web servis jer napadaču olakšava prisluškivanje komunikacije.

Nadalje slijedi pristup do web servisa bez provjere autentičnosti. I posljednje, ali ne najmanje važno, su ispisi poruka o pogreškama. Neispravnom konfiguracijom poruka o pogreškama mogu se otkriti podaci komponente koje koristi web servis. GraphQL je vrlo sklon slanja poruke o pogreškama koja otkriva gdje je pogreška pronađena i što ju je pokrenulo. Kada se dogodi pogreška, klijentu treba vratiti zasebnu poruku koja objašnjava uzrok pogreške na generičan način i bez otkrivanja nepotrebnih detalja. Potpuni detalji poruka o pogreškama trebali bi se dokumentirati samo interno. Komunikacija između klijenta i krajnje točke trebala bi biti omogućena što je više moguće putem HTTPS-a. Isto tako treba utvrditi nesigurne tokove podataka kako bi se izbjeglo otkrivanje informacija putem njih. Komponente treba redovito provjeravati zbog mogućih objavljenih ranjivosti. Zaštitu od takvih napada primjenjive su na sve navedene tehnologije web servisa.

4.3.8. Napadi umetanjem

Napadi umetanjem ili ubrizgavanjem (eng. Injection) su usmjereni na izvršavanje proizvoljnih naredbi koristeći korisnikov unos. Karakteristično za web servise popularna vrsta umetanja, kao što je ranije bilo rečeno, su SQL/NoSQL umetanje. No, postoje i druge vrste umetanja. Korištenje razvojnih okvira i ORM-a smanjenje vjerojatnost da se takvi napadi mogu izvršiti. Preporuka je koristiti ugrađene ili parametrizirane upute kod pisanja upita. SOAP API-ji verzije starije od SOAP 1.2 dopuštaju vanjske entitete u SOAP poruci. To znači da bi web servis koji koristi SOAP 1.0 ili 1.1 mogao biti izložen napadima poput proširenja XML entiteta ili XML obrade vanjskog entiteta. Isto tako ti napadi mogu se primijeniti i na REST API-je koji komuniciraju s XML korisnim teretom (eng. Payload). Provjeru ispravnosti korisničkog unosa uvijek je korisno napraviti pa čak i kada se koriste pripremljeni upiti. Posebno se to odnosi na GraphQL ali i na REST i SOAP kako se ne bi u potpunosti oslanjali na ugrađene mehanizme.

4.3.9. Neispravno upravljanje resursima

Neispravno upravljanje resursima (eng. Improper Assets Management) podrazumijeva zastarjele verzije web API-ja koje mogu ostati u produkciji i dalje ostati dostupne. Dostupnost zastarjelih web API-a može otkriti ranjivosti što utječe na iste resurse koji se upotrebljavaju u kasnijim verzijama. REST web servisi skloniji su takvoj ranjivosti iz razloga što obično imaju više od jedne krajnje točke. Kod GraphQL-a to bi bili stari upiti ili mutacije u shemi. Dok za SOAP to su operacije u WSDL-u. Općenito to se može izbjeći pravilnim dokumentiranjem web API-ja. Ispravna dokumentacija također može pomoći u identificiranju slomljenog objekta i autorizacije jer dostupni ista pomaže u utvrđivanju ispravnih slučajeva korištenja.

4.3.10. Nedovoljno bilježenje i nadzor

Nedovoljno bilježenje i nadzor (eng. Insufficient Logging & Monitoring) za razliku od ostalih napada, nije ranjivost koja će izravno utjecati na web servis. Preporuča se bilježiti svaku radnju web servis u dnevnike. Ako se ne vode dnevnici, potencijalne napade na web servis nije moguće otkriti. Iako nema ugrađenih funkcionalnosti za bilježenje u REST, SOAP i GraphQL web servisima, obično se to rješava razvojnim okvirima u kojim se implementira API.

4.4. Usporedba sigurnosti/ranjivosti REST-a i GraphQL-a

REST implementira nekoliko mehanizama za povećanje sigurnosti API-ja. Sigurnost REST API-ja osigurava se primjenom različitih metoda provjere autentičnosti. U to se ubraja HTTP provjera autentičnosti (osjetljivi podaci šalju se u HTTP zaglavljima), JSON Web Token-i (osjetljivi podaci šalju se kao JSON podatkovne strukture) ili standardni OAUth 2.0 mehanizmi. GraphQL-u nedostaju mehanizmi autentifikacije i autorizacije što često dovodi do sigurnosnih propusta i pristup do podacima. Teže realizira mehanizme protiv napada uskraćivanja usluge (DoS). Nema provjere korektnosti unosa vlastitih skalarnih tipova podataka [37].

Neke od ključnih razlike su još [38]:

- GraphQL koristi arhitekturu temeljenu na klijentu, a REST arhitekturu temeljenu na poslužitelju.
- GraphQL je tehnologija koja služi za izvršavanje upita s postojećim podacima, dok je REST arhitektura koji definira skup ograničenja za kreiranje web servisa.
- GraphQL ima shemu, a REST-a više krajnjih točaka.
- Razvoj GraphQL aplikacija je brži u odnosnu na REST.
- Format podataka za GraphQL je isključivo JSON, dok kod REST-a nema ograničenja oko formata podataka.

5. ZAKLJUČAK

GraphQL je nova tehnologija koja je nastala kao rješenje za probleme koje imaju REST servisi. Jezik je upita za API i softver na poslužitelju za izvršavanje tih upita te nije vezan uz programski jezik ili bazu podataka. GraphQL ima svoju shemu koja je javna. Shema je opis strukture našeg servisa u kojoj se definiraju tipovi i atributu te funkcije za dohvaćanje atributa. GraphQL shema je kao WADL ili WSDL kod REST ili SOAP servisa. Kao i svaka druga tehnologija GraphQL nije savršen i ima određenih ranjivosti. Neki napadi su specifični za GraphQL kao introspekcija ili višestruki napadi a neki napadi kao što su napadi umetanjem (SQL/NoSQL) su generički za sve API-je. Najčešća ranjivost GraphQL-a je napad putem introspekcije. Introspekcija je metoda koja pomoću upitima nad GraphQL shemom vraća detaljnije informacije o istoj. Introspekcija je vrlo korisna prilikom razvoja ali se može upotrijebiti u zlonamjerne svrhe. Obrana od ove vrste napada je ograničavanje pristupa upitima za introspekciju. GraphQL nema način provjere unosa podataka i time je ranjiv na napade umetanjem. Razvojni okviri implementiraju parametrizirane upite koji smanjuju ali ne i eliminiraju takve napade. Preporuka je svakako provjeriti korektnost unosa za svaki unos korisnika. Upiti mogu se koristiti i u svrhu uskraćivanja usluge. Stoga je potrebno provesti sigurnosne mjere protiv zloupotreba korištenja GraphQL resursa. To se radi temeljem vremenskog ograničenja i ograničenja složenosti upita. Vremensko ograničenje podrazumijeva vrijeme u kojem je dopušteno izvršavanje upita i vrijeme kako dugo korisnik smije koristiti servis. Ograničenje složenosti odnosi se na maksimalnu dubinu upita i ukupne složenosti upita koje korisnik smije izvršiti. Time se može spriječiti da preduboki upiti zlouporabe resurse. GraphQL omogućuje slanje više upita u jednom zahtjevu. Navedenu osobinu iskorištavaju tzv. višestruki napadi. Obrana od ovog napada je ograničavanje broja upita koji se mogu istodobno pokrenuti i dodavanje ograničenje vremena zahtjeva za objekt. Za kraj neuspjeli GraphQL upit vraćaju poruku o pogrešci. Preporučuje se da poruke budu što generičke kako ne bi otkrivali detalje implementacije upita.

Literatura:

1 Muschamp, P. (2004). An introduction to web services.

2 Informit. Glass, G. (2002). The Evolution of Web Services. Pristupljeno 15.02.2021. Dostupno na: https://www.informit.com/articles/article.aspx?p=27295

3 Tutorialspoint. What are Web Services? Pristupljeno 15.02.2021. Dostupno na: https://www.tutorialspoint.com/webservices/what_are_web_services.htm

4 Newcomer, E. (2002). Understanding Web Services: XML, Wsdl, Soap, and UDDI. Addison-Wesley Professional.

5 Fielding, R. T. (2000). REST: architectural styles and the design of network-based software architectures. Doctoral dissertation, University of California.

6 PocketSOAP. SMTP Transport Binding for SOAP 1.1. Pristupljeno 15.02.2021. Dostupno na: https://www.pocketsoap.com/specs/smtpbinding/

7 W3C. Simple Object Access Protocol (SOAP) 1.1. Pristupljeno 15.02.2021. Dostupno na: https://www.w3.org/TR/2000/NOTE-SOAP-20000508/

8 Richardson, L., & Ruby, S. (2008). RESTful web services. " O'Reilly Media, Inc.". Chapter 1. The Programmable Web and Its Inhabitants. Pristupljeno 15.02.2021. Dostupno na: https://www.oreilly.com/library/view/restful-web-services/9780596529260/ch01.html

9 Martinfowler Blog. Fowler M. (2010). Richardson Maturity Model: steps toward the glory of REST. Pristupljeno 15.02.2021. Dostupno na: https://martinfowler.com/articles/richardsonMaturityModel.html

10 Richardson, L. (2008, November). Justice will take us millions of intricate moves. In International Software Development Conference (QCon). Pristupljeno 15.02.2021. Dostupno na: https://www.crummy.com/writing/speaking/2008-QCon/act3.html

11 Octoperf. Loisel J. (2021). Soap vs Rest (Why comparing them is a nonsense). Pristupljeno 15.02.2021. Dostupno na: https://octoperf.com/blog/2018/03/26/soap-vs-rest/

12 Kajdič, D., Jurič, M.B. (2010). Web services with GraphQL. Elektrotehniski Vestnik/Electrotechnical Review. Pristupljeno 15.02.2021. Dostupno na: https://www.researchgate.net/publication/333563199_Web_services_with_GraphQL

13 GraphQL. Introduction to GraphQL. Pristupljeno 15.02.2021. Dostupno na: https://graphql.org/learn/

14 GraphQL. Code using GraphQL. Pristupljeno 15.02.2021. Dostupno na: https://graphql.org/code/

15 Github. Franke, B. (spawnia). Repository: webonyx/graphql-php. Pristupljeno 15.02.2021. Dostupno na: https://github.com/webonyx/graphql-php

16 Github. Warner B. (brianwarner). Repository: graphql/graphql-spec. Pristupljeno 15.02.2021. Dostupno na: https://github.com/graphql/graphql-spec

17 Getcomposer. Download Composer. Pristupljeno 15.02.2021. Dostupno na: https://getcomposer.org/download/

18 PHP download. Download the PHP package webonyx/graphql-php without Composer. Pristupljeno 15.02.2021. Dostupno na: https://php-download.com/package/webonyx/graphql-php

19 Webonyx. Type Definitions: Introduction. Pristupljeno 15.02.2021. Dostupno na: https://webonyx.github.io/graphql-php/type-system/

20 GraphQL. Schemas and Types. Pristupljeno 15.02.2021. Dostupno na: https://graphql.org/learn/schema/

21 How to GraphQL. Core Concepts. Pristupljeno 15.02.2021. Dostupno na: https://www.howtographql.com/basics/2-core-concepts/

22 Chrome web store. ChromeiQL. Pristupljeno 15.02.2021. Dostupno na: https://chrome.google.com/webstore/detail/chromeiql/fkkiamalmpiidkljmicmjfbieiclmeij

23 Porcello, E., & Banks, A. (2018). Learning GraphQL: declarative data fetching for modern web apps. " O'Reilly Media, Inc.".

24 Apollo Blog. Stubailo S. (2017). GraphQL vs. REST. Pristupljeno 15.02.2021. Dostupno na: https://www.apollographql.com/blog/graphql-vs-rest-5d425123e34b/

25 How to GraphQL. GraphQL is the better REST. Pristupljeno 15.02.2021. Dostupno na: https://www.howtographql.com/basics/1-graphql-is-the-better-rest

26 OWASP. Testing GraphQL. Pristupljeno 15.02.2021. Dostupno na: https://owasp.org/www-project-web-security-testing-guide/latest/4-Web_Application_Security_Testing/12-API_Testing/01-Testing_GraphQL

27 GraphQL. Introspection. Pristupljeno 15.02.2021. Dostupno na: https://graphql.org/learn/introspection

28 Medium. Rizwan B. (2020). GraphQL - Common vulnerabilities & how to exploit them. Pristupljeno 15.02.2021. Dostupno na: https://the-bilal-rizwan.medium.com/graphql-common-vulnerabilities-how-to-exploit-them-464f9fdce696

29 GraphQL Voyager. Pristupljeno 15.02.2021. Dostupno na: https://apis.guru/graphql-voyager/

30 Github. Repository: graphql/graphiql. Pristupljeno 15.02.2021. Dostupno na: https://github.com/graphql/graphiql

31 Github. Warner B. (brianwarner). Repository: graphql/graphql-playground. Pristupljeno 15.02.2021. Dostupno na: https://github.com/graphql/graphql-playground

32 Medium. Choren M. (2018). Discovering GraphQL endpoints and SQLi vulnerabilities. Pristupljeno 15.02.2021. Dostupno na: https://medium.com/@localh0t/discovering-graphql-endpoints-and-sqli-vulnerabilities-5d39f26cea2e

33 Pete Corey Blog. Corey, P. (2016). NoSQL Injection and GraphQL. Pristupljeno 15.02.2021. Dostupno na: http://www.petecorey.com/blog/2016/06/13/nosql-injection-and-graphql/?from=east5th.co]

34 Medium. Corey, P. (2017). GraphQL NoSQL Injection Through JSON Types. Pristupljeno 15.02.2021. Dostupno na: https://medium.com/@petecorey/graphql-nosql-injection-through-json-types-a1a0a310c759

35 OWASP Cheat Sheet Series. GraphQL Cheat Sheet. Pristupljeno 15.02.2021. Dostupno na: https://cheatsheetseries.owasp.org/cheatsheets/GraphQL_Cheat_Sheet.html

36 SANS Institute Information Security Reading Room. Cabezas, E. (2021). Leveraging the OWASP API Security top 10 to build secure web services. Pristupljeno 15.02.2021. Dostupno: https://www.sans.org/reading-room/whitepapers/application/leveraging-owasp-api-security-top-10-build-secure-web-services-39935

37 Rakuten RapidAPI. GraphQL vs. REST: A Comprehensive Comparison. Pristupljeno 15.02.2021. Dostupno: https://blog.api.rakuten.net/graphql-vs-rest/#REST_and_GraphQL_comparison

38 Guru99. GraphQL vs REST: What's the Difference? Pristupljeno 15.02.2021. Dostupno: https://www.guru99.com/graphql-vs-rest-apis.html

Podaci o autorima:

dr. sc. 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.

prof. dr. sc. 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.

mag. inf. Matija Kaniški

e-mail: makaniski@foi.hr

Matija Kaniški, mag. inf. je asistent na Katedri za teorijske i primijenjene osnove informacijskih znanosti i doktorand na Fakultetu organizacije i informatike. Njegova područja interesa su: Web programiranje, informacijsku sigurnost, arhitekturu softvera, strojno učenje i e-učenje. Vještine koje posjeduje su redom. Analiziranje i procjena sigurnosti danih informacijskih sustava s obzirom na norme sigurnosti i računalnu forenziku. Izrada online tečajeva uključujući psihološke elemente. Korištenje tehnika meta-modeliranja, generativnog programiranja i projektiranja arhitekture temeljene na komponentama. Izgradnja algoritama temeljem zahtijevane vremenske i/ili prostorne složenosti.

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

GENERIRANJE SADRŽAJA POMOĆU DUBOKIH NEURONSKIH MREŽA

CONTENT GENERATION WITH DEEP NEURAL NETWORKS

mag.oec. Ingrid Hrga

SAŽETAK:

Vještina stvaranja nečeg novog i jedinstvenog oduvijek se smatrala osobinom prvenstveno svojstvenom ljudima. Mentalne sposobnosti koje su pritom potrebne uglavnom nisu pripisivane drugim živim bićima, a pogotovo ne strojevima, čak niti onima pogonjenima umjetnom inteligencijom. Međutim, posljednjih nekoliko godina svjedočimo procvatu rješenja baziranih na dubokim neuronskim mrežama. Neuronske mreže pomažu u izvršavanju zadataka bez ljudske intervencije, a njihova sve impresivnija sposobnost u stvaranju najraznovrsnijih sadržaja predstavlja očiti dokaz ubrzanom smanjenju granica između ljudi i strojeva. Lažne video snimke (deep fakes), digitalno komponiranje ili automatsko opisivanje slika samo su neki od primjera. U radu se daje prikaz dubokih neuronskih mreža za stvaranje digitalnih sadržaja, s posebnim osvrtom na generative suparničke mreže koje u tom području trenutno drže primat. Nakon opisa njihove osnovne arhitekture i glavnih karakteristika te načina na koji one uče, detaljnije se razlažu uspješni primjeri njihove primjene. Primjeri variraju od jednostavnije augmentacije podataka pa do kompleksnijeg stvaranja vizualnih, tekstualnih i ostalih sadržaja koji ponekad graniče i s umjetnošću.

ABSTRACT:

The skill of creating something new and unique has always been considered a trait primarily inherent in humans. The mental abilities required for creation are generally not attributed to other living beings, and certainly not to machines, not even those driven by artificial intelligence. However, in the last few years we have witnessed the flourishing of solutions based on deep neural networks. Neural networks help to perform tasks without human intervention, and their increasingly impressive ability to create a wide variety of content provides special evidence of the rapidly diminishing difference between humans and machines. Fake videos (deep fakes), digital composing or automatic image captioning are just some of the examples. This paper surveys deep neural networks for digital content creation, with special emphasis on the generative adversarial networks, currently considered state-of-the-art. After describing their basic architecture, their main characteristics and the way they learn, some successful examples of application are explained in more detail. Examples range from the simpler data augmentation to the more complex creation of visual, textual and other content that sometimes borders on art.

1. UVOD

Iako začeci umjetnih neuronskih mreža sežu još u prvu polovicu prošloga stoljeća [1], tek je s ispunjenjem određenih pretpostavki unazad desetak godina stvoreno okruženje koje je potaklo njihov ubrzani napredak i opću prihvaćenost. Razvoj naprednih algoritama koji omogućavaju mrežama učenje kompleksnih zadataka, dostupnost velikih skupova podataka [2] s milijunima označenih primjera iz kojih mreže uče te smanjenje cijena modernog hardvera, pogotovo grafičkih procesora (GPU-a), dovelo je do toga da u kratkom roku umjetne neuronske mreže prerastu akademske okvire te se nađu u pozadini mnogih inteligentnih komercijalnih proizvoda. Samovozeći automobili, roboti, pa i neke manje, svakodnevne aplikacije poput onih za prevođenje [3] ili za zabavu [4] temelje se na primjeni dubokih neuronskih mreža. Ono što posebno zapanjuje jest lakoća s kojom mreže sve češće izvršavaju zadatke čije izvršavanje zahtijeva sposobnosti donedavno smatrane svojstvenima samo ljudima. To su zadaci koji uključuju percepciju i izražavanje osjetilnih informacija [5, 6] uz stvaranje nečeg novog i jedinstvenog.

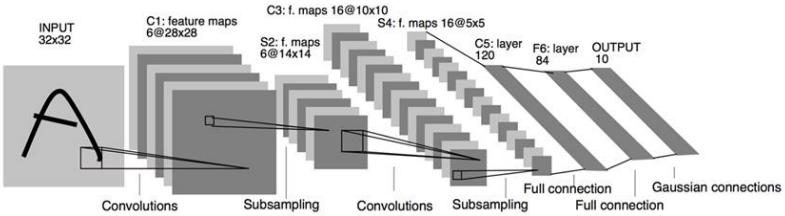
Generativne suparničke mreže (eng. Generative Adversarial Networks – GANs) [7] obuhvaćaju posebnu skupinu dubokih neuronskih mreža koje postižu prilično impresivne rezultate u generiranju raznovrsnih digitalnih sadržaja, prvenstveno onih vizualnih te su okosnica ovoga rada. Nakon kratkog uvoda o dubokim neuronskim mrežama u drugom poglavlju, u trećem se poglavlju predstavlja osnovni model generativnih suparničkih mreža. Objašnjena je njihova arhitektura te način kako uče generirati nove podatke, a zatim su prikazana i neka od značajnijih unaprjeđenja osnovnog modela. U četvrtom poglavlju predstavljaju se primjeri uspješne primjene, među kojima prevladavaju oni vizualno atraktivni. Povezanost umjetnosti i generativnih suparničkih mreža izdvojena je u posebno, peto poglavlje, a prije samog zaključka navedena su i neka od postojećih ograničenja.

2. DUBOKE NEURONSKE MREŽE

Umjetne neuronske mreže inspirirane su ljudskim živčanim sustavom, naročito mozgom, te se sastoje od procesnih jedinica, popularno nazvanih neuronima. Neuroni vrše transformaciju ulaznih podataka u određeni odgovor na izlazu iz mreže. Na primjer, ako želimo izgraditi neuronsku mrežu za rješavanje zadatka klasifikacije objekata na slikama, tada su ulazni podaci slike, tj. vrijednosti piksela, a na izlazu mreža daje odgovor u obliku vjerojatnosti da se na slici nalazi određeni objekt. Skupina unaprijednih mreža (eng. *feed-forward*) obuhvaća mreže kod kojih su neuroni organizirani u slojeve i to na način da izlaz jednog sloja predstavlja ulaz u drugi sloj. Između ulaznog i izlaznog sloja nalaze se tzv.

skriveni slojevi u kojima se vrše transformacije podataka. U tome važnu ulogu imaju nelinearne aktivacijske funkcije koje daju mrežama sposobnost učenja nelinearnih veza u podacima.

Neuronske mreže koje imaju više od dva skrivena sloja nazivaju se dubokim neuronskim mrežama (eng. *Deep Neural Networks ili kraće DNN*). Današnje mreže često imaju znatno više od dva sloja [8] s obzirom da je veći broj slojeva povezan sa većom sposobnošću rješavanja kompleksnijih zadataka.



Slika 1. Primjer konvolucijske mreže za klasifikaciju slika [9]

U području računalnog vida najzastupljenije su tzv. konvolucijske neuronske mreže (eng. *Convolutional Neural Networks ili kraće CNN*) [9] koje su, zahvaljujući svojoj specifičnoj arhitekturi, posebno prilagođene procesiranju vizualnih sadržaja, poput slika i videa. Međutim, njihova je primjena znatno šira te se uspješno koriste i za druge vrste nestrukturiranih podataka, poput teksta ili zvuka.

Konvolucijske neuronske mreže (slika 1) posebna su vrsta unaprijednih neuronskih mreža. Inspirirane su funkcioniranjem vidnog sustava sisavaca [10] na način da se prolazom podataka kroz mrežu vrši njihova transformacija u reprezentaciju s više razina apstrakcije [11], odnosno stvara se hijerarhija značajki. Ako se zadržimo na prethodnom primjeru klasifikacije, mreža u svojim donjim slojevima uči prepoznavati jednostavne oblike poput krugova, linija i sl., a zatim se kroz daljnje slojeve takve značajke povezuju, tvoreći sve kompleksnije oblike a da bi se na kraju mreže izdvojio glavni objekt na slici. Može se reći da mreža kroz svoje slojeve potiskuje nebitno, a ističe samo ono najbitnije za rješavanje određenog zadatka.

Konvolucijske mreže svoj su naziv dobile prema konvolucijskom sloju u kojem se vrši linearna transformacija odnosno operacija konvolucije. Konkrenta arhitektura svake mreže ovisi o zadatku kojem je namijenjena, a nastavljajući s primjerom klasifikacije slika, mogu se izdvojiti njezina dva osnovna dijela: konvolucijska baza, putem koje se izvlače značajke te dio za klasifikaciju. Konvolucijska baza sastoji se jednog ili više konvolucijskih slojeva koji se uobičajeno izmjenjuju sa slojevima za sažimanje (eng. *pooling*). Dio za klasifikaciju uobičajeno sadrži potpuno povezane slojeve, a završava aktivacijskom funkcijom koja daje vjerojatnost da se na slici nalazi određeni objekt.

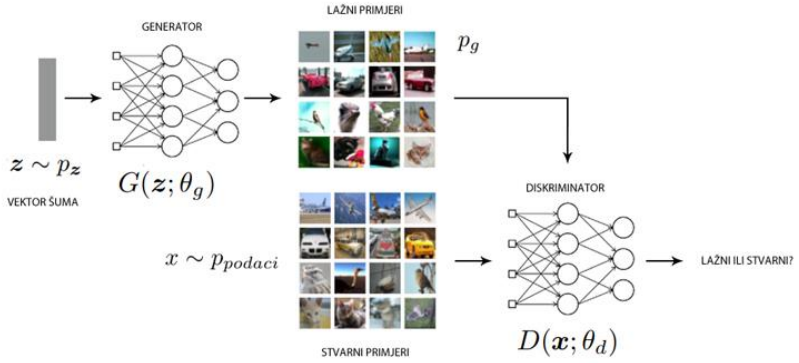
3. GENERATIVNE SUPARNIČKE MREŽE

U ovom poglavlju najprije se daje prikaz osnovnog modela generativnih suparničkih mreža, a zatim se predstavljaju neka od značajnijih poboljšanja. Kako je fokus ovoga rada na generiranju sadržaja, između velikog broja objavljenih radova odabrano je samo nekoliko reprezentativnih, usmjerenih na stvaranje vizualno atraktivnih rezultata.

3.1. Osnovni model generativnih suparničkih mreža

Generativne suparničke mreže (u daljnjem tekstu GAN) pripadaju skupini generativnih modela koji uče distribuciju nekog skupa podataka, a naknadnim uzorkovanjem iz naučene distribucije moguće je generirati nove sintetičke primjere koji su jako sliči izvornom skupu podataka.

GAN predstavlja inovativni način treniranja generativnih modela jer se sastoji se od dviju neuronskih mreža u stalnom međusobnom natjecanju (slika 2). Jedna se mreža zove generator, a druga diskriminator. Zadatak generatora je naučiti generirati nove, realistične primjere, npr. slike, ali tako da tijekom učenja uopće ne vidi primjere stvarnih podataka. Kako bi generator uopće znao jesu li generirani primjeri dovoljno dobri, u tome mu pomaže signal koji dobiva od diskriminatora. Zadatak diskriminatora je naučiti razlikovati generirane, odnosno lažne primjere od stvarnih, a svojim odgovorom ukazuje na uvjerljivost određenog primjera te time pomaže generatoru. Natjecanje mreža sastoji se u tome da, s jedne strane, generator želi „nadmudriti" diskriminatora stvaranjem primjera koje on neće moći razlikovati od stvarnih, a s druge strane, diskriminator želi biti što bolji u razlikovanju pravih od lažnih primjera. U takvom međusobom nadmetanju obje mreže postaju sve bolje.



Slika 2. Osnovni model generativnih suparničkih mreža

Algoritam za učenje [7] sastoji se od dva dijela: učenja diskriminatora i učenja generatora. Na slici 2 obje su mreže prikazane kao potpuno povezane neuronske mreže. Obje mreže imaju svoj skup parametara (θ_g i θ_d, pri čemu je g oznaka generatora a d diskriminatora) koji se podešavaju tijekom učenja. Učenje započinje uzorkovanjem vektora šuma z iz distribucije p_z poput uniformne ili Gaussove. Generator G na temelju vektora šuma generira nove primjere $x = G(z)$. U prikazu na slici 2 to su nove slike. Diskriminator D je zapravo binarni klasifikator koji na ulazu prima kako stvarne $x \sim p_{podaci}$ tako i generirane primjere $x \sim p_g$ te za svaki primjer daje vjerojatnost da je on stvaran, odnosno da dolazi iz distribucije stvarnih podataka p_{podaci}. Funkcija cilja J daje informaciju o tome koliko dobro pojedina mreža uči, a u ovom slučaju, takva funkcija svake od mreža ovisi o parametrima one druge mreže na koje ona sama nema utjecaj. Zato kažemo da se ovdje radi o igri. A s obzirom da mreže imaju oprečne ciljeve, generator i diskriminator igraju igru sa sumom nula (eng. *zero-sum game*). Cilj diskriminatora je minimizirati unakrsnu entropiju stvarnih oznaka i predikcija, odnosno odgovora mreže i pravog odgovora je li slika stvarna. To postiže mijenjanjem svojih parametara θ_d:

$$J_d(\theta_d, \theta_g) = -\frac{1}{2}\mathbb{E}_{x\sim p_{podaci}} \log D(x) - \frac{1}{2}\mathbb{E}_z \log(1 - D(G(z))) \quad (1)$$

Niska unakrsna entropija znači da mreža može dobro raspoznavati stvarne od lažnih primjera. Za generatora vrijedi obratno. Generator želi maksimizirati unakrsnu entropiju diskriminatora:

$$J_g = -J_d \quad (2)$$

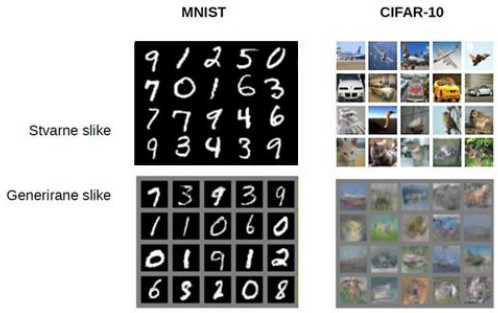
Povezivanjem fukcija cilja generatora i diskriminatora dobije se minimax formulacija funkcije vrijednosti V gdje je rješenje:

$$\theta_g^* = \arg\min_{\theta_g} \max_{\theta_d} V(\theta_d, \theta_g) \quad (3)$$

Tijekom treniranja, odnosno postupka tijekom kojeg mreže uče, može se opaziti kako slike postupno od šuma postaju sve sličnije stvarnim podacima, iako se tijekom treniranja mogu pojaviti određeni problemi koji često ponište postignuti napredak. Nakon učenja, ako je cilj bio samo generirati nove primjere, diskriminator više nije potreban. Tad je dovoljno generatoru proslijediti novi vektor šuma i dobit ćemo sliku koja izgleda kao da je izvučena iz pravog skupa slika. Naravno, to vrijedi samo ako je učenje bilo uspješno.

Na slici 3 prikazani su primjeri iz izvornog rada [7] u kojem su po prvi puta predstavljene generativne suparničke mreže. U gornjem redu nalaze se stvarni primjeri, a u donjem su prikazani generirani primjeri. Može se vidjeti da je mreža trenirana na skupu MNIST [12] uspjela generirati realistične nove primjere. Međutim, radi se o skupu koji se sastoji od jednostavnih crno-bijelih slika rukom pisanih znamenki te je zadatak generatora lakši nego u slučaju drugog skupa, CIFAR10 [13], koji se sastoji od slika u boji s raznovrsnijim objektima i s puno više detalja. Na tom drugom skupu mreža je ostvarila znatno lošiji rezultat. Iako je mreža u drugom slučaju uspjela „uhvatiti" boje i naznake glavnih objekata, nije uspjela generirati uvjerljive detalje.

Međutim, taj je rad predstavio prekretnicu te je potaknuo iznimni interes znanstvenika. U svega nekoliko godina ostvaren je eksponencijalni rast broja objavljenih znanstvenih radova koji predlažu neko poboljšanje ili novu primjenu GAN-ova [14].

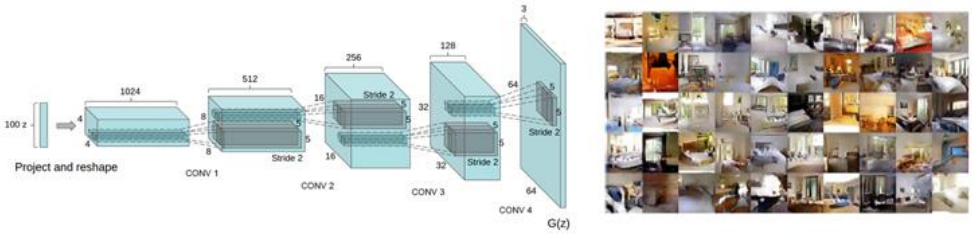


Slika 3. Usporedba stvarnih i generiranih primjera na skupovima slika MNIST i CIFAR-10. Prilagođeno prema [7]

3.2. Deep Convolutional Generative Adversarial Network (DCGAN)

Među prvim značajnijim poboljšanjima jest mreža pod nazivom duboka konvolucijska generativna supranička mreža (eng. *Deep Convolutional Generative Adversarial Network ili kraće DCGAN*) [15] koja donosi nekoliko važnih promjena. Prva skupina promjena tiče se arhitekture mreže, a druga se skupina promjena odnosi se na hiperparametre, odnosno postavke treniranja.

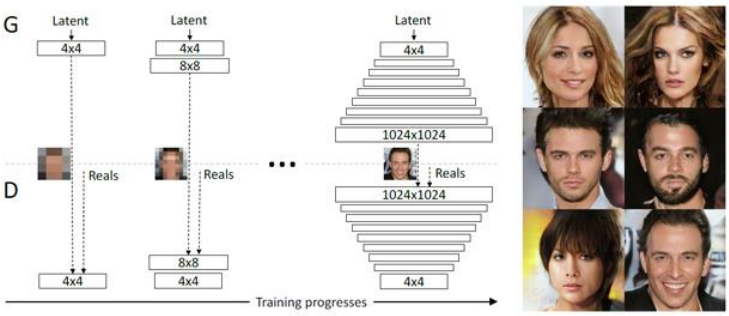
S obzirom da konvolucijske mreže pokazuju nadmoć u radu sa slikama [8, 16], potpuno povezani slojevi sad su zamijenjeni konvolucijskim slojevima u diskriminatoru te slojevima unatražne konvolucije u generatoru. Na slici 4 prikazana je predložena arhitektura generatora. Slojevima unatražne konvolucije vrši se potrebno uvećavanje kako bi se od vektora šuma dobile slike određene rezolucije (npr. u prikazu je to 64x64 piksela). Dodatne izmjene odnose se na uvođenje posebnih slojeva za normalizaciju po grupama (eng. *batch normalization*) [17], korištenje konvolucijskog koraka umjesto slojeva za sažimanje (eng. *pooling layers*) te primjenu određenih aktivacijskih funkcija, poput ReLU [19] i LeakyReLU [20]. Sve to doprinosi stabilizaciji treniranja koje se zbog kompetitivne igre generatora i diskriminatora pokazalo iznimno problematičnim. Na slici 4 može se vidjeti da generirane slike sad imaju puno više detalja.



Slika 4. Arhitektura generatora DCGAN-a (lijevo) i primjeri generiranih slika (desno) [15]

3.3. Progressive GAN (ProGAN)

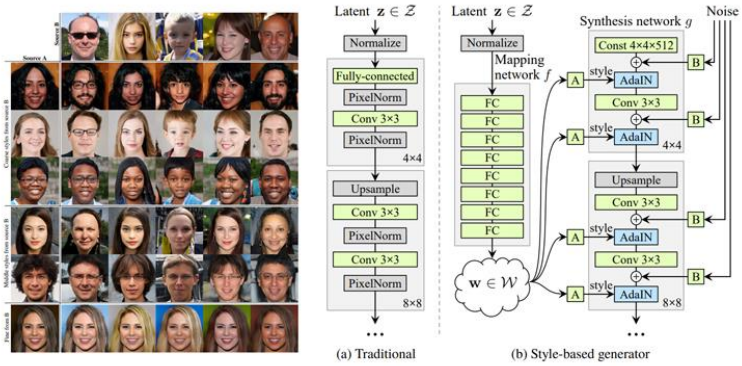
Prethodno navedene mreže imale su sposobnst generiranja slika uglavnom niske rezolucije, npr. 32x32, 64x64 piksela zbog znatnih hardverskih zahtjeva i problema prilikom treniranja. Mreža pod nazivom Progressive GAN (ProGAN) [21] omogućava generiranje iznimno realističnih slika rezolucije od 1024x1024 piksela. Postupak treniranja je ponešto izmijenjen te se provodi se kroz nekoliko faza. Najprije se stvarne slike smanjuju na nižu rezoluciju npr. 4x4 ili 8x8 piksela te mreža uči generirati slike takve niže rezolucije. Zatim se mreži postupno dodaju slojevi kako bi učila stvarati slike veće rezolucije. To je tzv. višeskalna arhitektura (eng. *multiscale architecture*) prikazana na slici 5. Druga inovacija odnosi se na način dodavanja novih slojeva. Pri prijelazu iz sloja niže u sloj više rezolucije vrši se njihova interpolacija te se vrijednosti slike množe faktorom između 0 i 1. Taj se faktor postupno tijekom treniranja mijenja čime se stvaraju glatki prijelazi iz manje u veću rezoluciju pridonoseći većoj kvaliteti krajnjeg rezultata.



Slika 5. Shematski prikaz postupka progresivnog treniranja ProGAN-a (lijevo) i primjeri generiranih slika visoke rezolucije (desno) [21].

3.4. StyleGAN

Autori mreže pod nazivom StyleGAN [22] koriste unaprjeđenja ProGAN-a u vidu postupnog povećavanja rezolucije tijekom treniranja, ali i predlažu dodatna unaprjeđenja koja omogućavaju bolju kontrolu nad svim aspektima slike. Predložena arhitektura dobiva na kompleksnosti zbog uvođenja dodatne mreže (na slici 6 označena s f) koja na izlazu daje dodatni vektor w. Taj vektor pomaže u razdvajanju i boljoj kontroli faktora varijacije. Omogućeno je jako fino podešavanje svih detalje slike, poput boje kose, ali i pojedinih vlasi kose!



Slika 6. Modifikacija generatora (desno) koja omogućava bolju kontrolu finih detalja na generiranim slikama (lijevo) [22]

Vezano uz ovaj rad objavljene su i popularne web stranice „This person/cat/... does not exist" [23]. Sa svakim novim učitavanjem stranice prikaže se nova fotografija osobe/mačke/... koja ne postoji. Fotografija je zapravo generiran slika. Iako su slike iznimno realistične, ipak se među primjerima mogu pronaći i oni s pokojom pogreškom (slika 7).

Slika 7. Primjeri generiranih na web stranicama [23]. koje prate rad [22]. Slika mačke (krajnje desno) ima pogrešku u vidu loše definiranih usta

4. PRIMJENA GENERATIVNIH SUPARNIČKIH MREŽA

U ovom se poglavlju daje dodatni prikaz primjera uspješne primjene generativnih suparničkih mreža u rješavanju raznovrsnih zadataka, pri čemu su neki od njih već postali sastavni dio brojnih komercijalnih proizvoda, pogotovo onih za obradu slika ili videa.

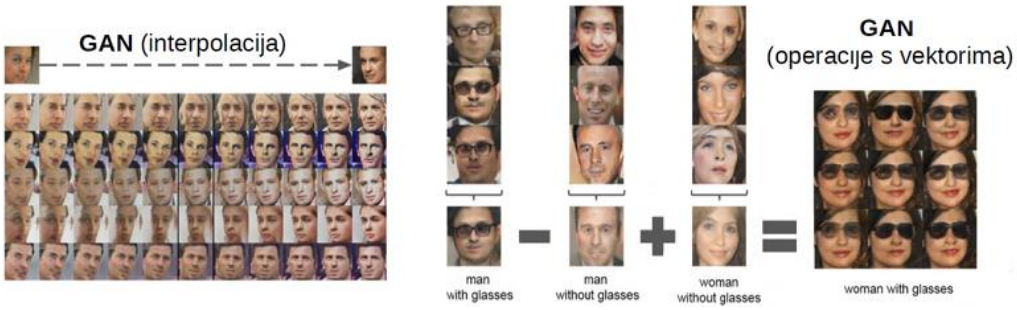
4.1. Augmentacija podataka

Pri primjeni dubokih neuronskih mreža, koje ponekad imaju i milijarde parametara [6], čest je problem nedostatka odgovarajućih podataka za učenje. Da bi neuronska mreža mogla naučiti rješavati kompleksne zadatke, treba imati dovoljan kapacitet, što u praksi znači dovoljan broj slojeva. Međutim, to povlači za sobom i potrebu da se osigura veća količina odgovarajućih podataka kako bi se izbjegla pojava overfittinga, odnosno situacije kad mreža memorira podatke za učenje umjesto da iz njih uči. Kasnije se takav problem očituje lošom sposobnošću generalizacije, odnosno sposobnosti mreže da primijeni naučeno na novim podacima. Npr. ako mreža uči raspoznavati objekte na slikama koje sadrže kompleksne scene, trebat će više podataka nego kad se radi o jednostavnim scenama. Međutim, često nije moguće nabaviti dodatne podatke, npr. zbog visokih troškova ili zbog zaštite identiteta kao što je to slučaj u području medicine. Tada je potrebno na odgovarajući način iskoristiti postojeće podatke.

Slika 8. Slika prikazuje postupak augmentacije podataka jednostavnim transformacijama postojećeg skupa

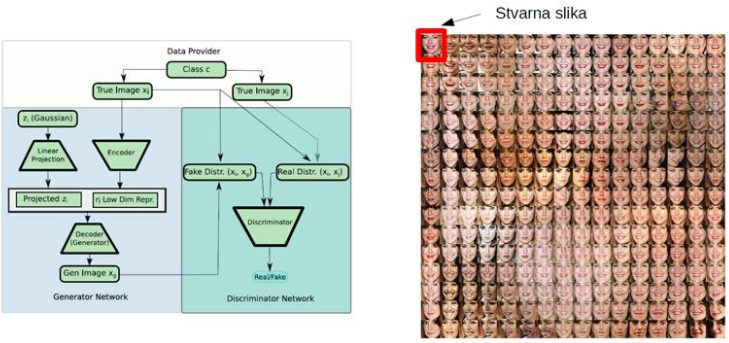
Uobičajena praksa je transformacija postojećih podataka tako da se mijenjaju samo neki njihovi aspekti bez promjene klase kojoj podaci pripadaju (slika 8). Ako radimo sa slikama, to može biti rotacija slike, promjena intenziteta boja, dodavanje šuma i sl. To su sve standardni postupci koji daju dobre rezultate, ali njima ne nastaje nova scena.

S GAN-ovima je moguće stvoriti potpuno nove slike kakve se ne nalaze u izvornom skupu. Na slici 9 prikazano je kako se interpolacijom dviju točaka mogu generirati prijelazi između dviju slika. A s obzirom da su slike predstavljene vektorima, operacijama s vektorima moguće je kreirati potpuno nove, drugačije primjere koji kombiniraju svojstava više slika (slika 9 desno). Ako od vektora koji predstavlja sliku muškarca s naočalama oduzmemo vektor muškarca te dodamo vektor žene, dobit ćemo sliku žene s naočalama.



Slika 9. Stvaranje novih slika interpolacijom dviju točaka (lijevo) i operacijama s vektorima (desno). Prilagođeno prema [15]

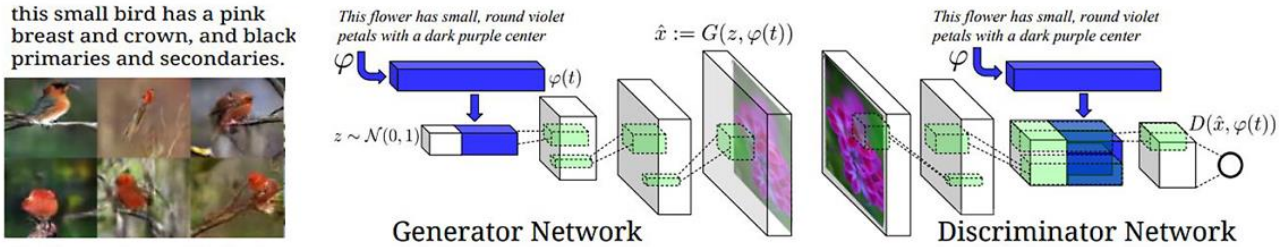
Autori u [24] predložili su posebnu mrežu, nazvanu Data Augmentation GAN (DAGAN), namijenjenu upravo augmentaciji podataka. Na slici 10 desno samo je jedna slika stvarna, dok su svi prikazani prijelazi generirani. Mreža pripada posebnoj skupini uvjetnih generativnih suparničkih mreža kod kojih su generator i diskriminator uvjetovani dodatnim inputom. Generator DAGAN mreže baziran je na arhitekturi kodera i dekodera. Na ulazu uzima slike samo jedne klase te ih mreža kodera pretvara u reprezentaciju smanjene dimenzionalnosti. To se dodaje uobičajenom vektoru šuma kako bi mreža dekodera na izlazu dala augmentiranu sliku. Diskriminator i dalje ima zadatak razlikovanja stvarnih i generiranih primjera.



Slika 10. Arhitektura DAGAN mreže za augmentaciju podataka (lijevo) s primjerima interpolacije (desno). Prilagođeno prema [24]

4.2. Generiranje slika na temelju teksta

Povezivanje slika i teksta poprilično je izazovan zadatak jer obuhvaća problem odgovarajuće reprezentacije i usklađivanja dvaju modaliteta. Jedna strana takvog povezivanja obuhvaća generiranje opisa slika (više o tome u [25]) pri čemu se GAN-ovi mogu koristiti za povećanje raznovrsnosti generiranih opisa. Drugu stranu predstavlja mogućnost generiranja slika na temelju opisa. Iako se u oba slučaja za jedan input može generirati veliki broj outputa, potonji je zadadtak kudikamo teži zbog većeg prostora mogućih rezultata (slika 11 lijevo). Rad u kojem je predstavljena mreža za generiranje slika na temelju opisa [26] objavljen je 2016. godine pa koristi nešto stariju DCGAN arhitekturu za uvjetni GAN. Generator na ulazu dobiva tekst koji kodira i povezuje s vektorom šuma, dok diskriminator prima tri kombinacije inputa: stvarne slika i ispravan tekst, stvarne slike i lažni tekst, te lažne slike i ispravni tekst. Proces učenja tijekom kojeg diskriminator uči ne samo razlikovati stvarne od lažnih slika već i ispravne kombinacije slika i teksta, daje generatoru dodatni signal koji mu pomaže tijekom generiranja outputa, odnosno slika.



Slika 11. Arhitektura GAN-a za generiranje slika na temelju teksta (desno) te nekolliko primjera slika generiranih na temelju iste rečenice (lijevo)[26]

4.3. Super rezolucija slika

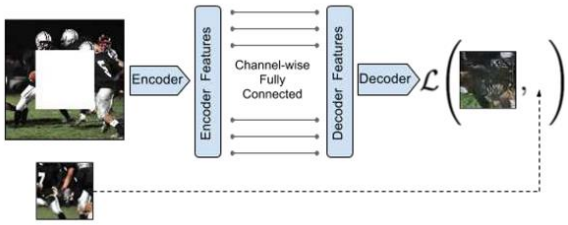
Super rezolucije predstavlja zadatak dobivanja odgovarajuće slike visoke rezolucije na temelju slike niske rezolucije. Nedostaci klasičnih metoda interpolacije, koji se očituju u gubitku detalja nakon uvećanja, uspješno se izbjegavaju primjenom GAN-ova. Jedan od značajnijih radova koji u tu svrhu koristi GAN jest [27]. Autori predlažu mrežu nazvanu Superresolution GAN (SRGAN) kod koje se generator temelji na posebnoj arhitekturi CNN mreže s rezidualnim blokovima [8] te koristi novu formulaciju ciljne funkcije od dva dijela. Uz standardni dio, uobičajen u zadacima super rezolucije, dodan je i dio koji potiče mrežu na stvaranje uvećanja koja djeluju prirodno. Na slici 12 prikazana je usporedba rezultata povećanja rezolucije bikubičnom interpolacijom te pomoću SRGAN-a. Može se primijetiti očita superiornost SRGAN-a, unatoč tome što nedostaju određeni detalji na kapi ili ovratniku.

Slika 12. Usporedba rezultata bikubične interpolacije (lijevo) i SRGAN-a (u sredini) u povećanju rezolucije slike. Izvorna slika visoke rezolucije nalazi se desno [27]

4.4. Rekonstrukcija nedostajućih dijelova slika

Rekonstrukcija nedostajućih dijelova (eng. *Image inpainting*) predstavlja zadatak nadopunjavanja slike objektima ili dijelovima scena koji nedostaju. Autori u [28] predlažu mrežu nazvanu Context Encoder (slika 13) koja se bazira na

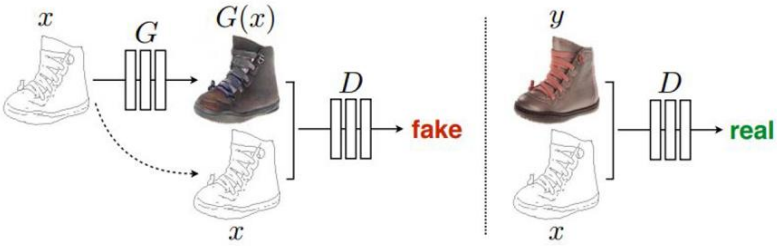
arhitekturi kodera i dekodera. Koder sliku s nedostajućim dijelom pretvara u reprezentaciju na temelju koje dekoder zatim generira nedostajući dio. Pri tome je važno uzeti u obzir kontekst u kojem se nalazi nedostajući dio, ali i općenito značenje slike kako bi generiranje dalo uvjerljiv rezultat. Slično kao i kod problema opisanog u 4.2., velik je prostor mogućih rezultata. Kako bi se, s jedne strane sačuvala općenita struktura slike, a s druge strane ona nadopunila prikladnim dijelom, autori predlažu ciljnu fukciju koja se sastoji od gubitka rekonstrukcije i suparničkog gubitka, po uzoru na GAN. U radu autori prikazuju i jednu usporedbu rezultata umjetnika i mreže na zadatku rekonstukcije slike u kojem se mreža pokazala uspješnijom.



Slika 13. Shematski prikaz Context Encodera za rekonstrukciju slika [28]

4.5. Prevođenje slike u sliku

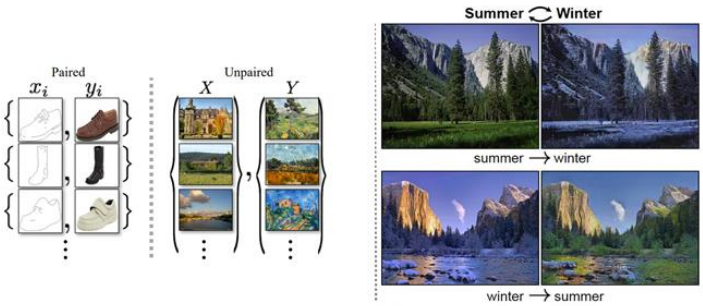
U prethodnim su primjerima kako izvorne tako i generirane slike bile iz iste domene (npr. fotografije lica). Međutim, autori u [29] predložili su mrežu nazvanu Pix2Pix koja može primjere jedne domene pretvoriti u primjere druge domene, npr. fotografiju u skicu ili obratno. Takvo pretvaranje slika (eng. *image-to-image translation*) omogućeno je zahvaljujući promjenama na strani generatora, ali i diskriminatora. Autori koriste uvjetnu generativnu mrežu (slika 14) kod koje su generator i diskriminator uvjetovani dodatnim inputom, u ovom slučaju drugom slikom. Ako se npr. iz skice želi dobiti fotografiju, tada generator, osim vektora šuma, na ulazu dobije i skicu čime se kontrolira u što treba pretvoriti šum. Određene promjene učinjene su i u arhitekturi mreža pa tako sad generator poprima arhitekturu U-Net mreže [30] dok diskriminator preuzima arhitekturu koju autori nazivaju PatchGAN jer, umjesto cijele slike, klasificira njezine dijelove (eng. *patch*).



Slika 14. Pojednostavljeni prikaz arhitekture Pix2Pix mreže [29]

Jedno od ograničenja Pix2Pix mreže jest potreba za uparenim primjerima za učenje. Ako želimo pretvarati skice u fotografije, za učenje trebamo imati na raspolaganju primjere i skica i fotografija istog objekta ili scene. Mreža pod nazivom CycleGAN [31] zaobilazi takvo ograničenje. Za pretvaranje slika jedne domene u slike druge domene dovoljno je imati dva skupa pri čemu slike ne moraju biti povezane ni na koji način. Jedan skup predstavlja slike čiji sadržaj želimo zadržati dok drugi skup slika predstavlja stil ili ugođaj koji želimo prenijeti na prvi skup. Kako bi se takvo pretvaranje uspješno realiziralo, autori su koristili dva generatora i dva diskriminatora, svaki za po jedan skup slika.

Očuvanje strukture slike pri prijenosu stila s neke druge slike autori su postigli pomoću nove funkcije gubitka rekonstrukcije nazvane kružno dosljednom funkcijom gubitka (eng. *cycle consistency loss*) pri čemu rekonstrukcijski gubitak penalizira kvadratnu pogrešku u rekonstrukciji originalne slike. Time je omogućeno pretvaranje slike A u sliku B, te ponovno vraćanje slike B u sliku A. Popularni primjer primjene mreže ovakvog tipa predstavlja FaceApp aplikacija [4]

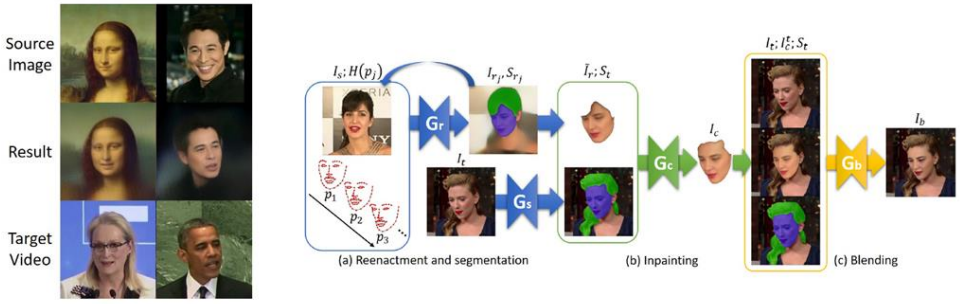


Slika 15. Za pretvaranje slika iz jedne domene u drugu Pix2Pix mreža (lijevo) treba uparene primjere za učenje dok CycleGAN (u sredini i desno) to ne treba [31]

4.6. Zamjena lica u video snimkama

Aplikacije za zamjenu lica potaknule su brojne kontroverze krajem 2017. godine [32] Društvene mreže bile su odjednom preplavljene tzv. deep fakes snimkama, odnosno video snimkama u kojima su osobama zamijenjena lica kako bi bile iskorištene u neetične svrhe. To je potaknulo brojne rasprave o opasnostima tehnologije koja ipak ima i pozitivnu primjenu npr. u industriji zabave kako navode u [33].

Pri zamjeni lica možemo razlikovati dva zadatka. Prvi se odnosi na prijenos fizionomije jedne osobe na drugu uz zadržavanje mimike lica druge osbe (eng. *face swapping*) dok drugi zadatak podrazumijeva prijenos mimike lica jedne osobe na drugu uz zadržavanje fizionomije lica druge osobe (eng. *face reenactment*). Oba zadatka prilično su izazovna jer zahtijevaju posebnu pažnju pri prilagodbi osvjetljenja, poza, ekspresija i općenitog izgleda osoba.



Slika 16. Shematski prikaz generatora FSGAN-a (desno) te nekoliko primjera zamjene fizionomije lica (lijevo) [33]

Autori u [33] predlažu tzv. Face Swapping GAN (FSGAN) koji objedinjuje oba zadatka te može vršiti zamjenu lica određenih osoba bez potrebe da se raspolaže slikama tih osoba za treniranje mreže. Ovakva mreža ima i znantno kompleksniju arhitekturu te se za generiranje koriste čak četiri generatora (slika 16): generator za zamjenu ekspresija lica, generator za segmentaciju lica, generator za rekonstrukciju neodstajućih dijelova te generator za zaglađivanje. Na slici 16 lijevo mogu se vidjeti primjeri zamjene fizionomije lica, iako je za potpuni doživljaj uspješnosti mreže potrebno pogledati video snimku.

4.7. Generiranje teksta

Za razliku od generiranja vizualnih sadržaja, GAN-ovi se u znatno manjoj mjeri koriste za generiranje tekstualnih sadržaja. Poteškoće rada s tekstom proizlaze iz njegove diskretne prirode dok GAN-ovi preferiraju raditi s kontinuiranim podacima. Uz to, dodatne poteškoće nastaju u radu s dužim tekstovima. Metode koje su se pokazale uspješnima u radu s kratkim tekstovima (do otprilike 20 riječi) baziraju se na kombinaciji GAN-ova i učenja s podrškom (eng. *Reinforcement Learning ili kraće RL*). Autori u [34] predložili su SeqGAN za rad sa sekvencijalnim, diskretnim podacima kao što je tekst pri čemu je generiranje teksa modelirano kao sekvencijalni proces odlučivanja. Stanje je predstavljeno prethodno generiranim tekstom, akcija je riječ koju sljedeću treba generirati a generator je stohastična strategija koja preslikava trenutno stanje u distribuciju prostora akcija [35]. Nakon što proces generiranja teksta završi, generirani se tekst prosljeđuje diskriminatoru koji ga klasificira te se nagrada određuje na temelju prosudbe o uvjerljivosti generiranog teksta. Kako navode u [35] jedan od problema s ovakvim modelom jest taj što signal koji dobiva generator od diskriminatora ne zadržava dovoljno informacija o strukturi i značenju koje bi generatoru pomogle tijekom učenja i omogućile generiranje dužih tekstova.

Autori u [35] zato predlažu LeakGAN, model koji zaobilazi navedene poteškoće. I u ovom slučaju autori kombiniraju GAN s podržanim učenjem, ali koriste varijantu hijerarhijskog učenja s podrškom (eng. *hierarchical reinforcement learning*). Hijerarhijski generator sadrži dodatne module (menadžera i radnika) koji olakšavaju prijenos signala tijekom treniranja i to prije nego se dovrši generiranje čitave rečenice, umjesto da se to vrši tek po završetku generiranja kao što je to bio slučaj u ranijim radovima [34]. Na taj je način omogućeno generiranje znatno dužih tekstova.

4.8. Generiranje zvuka

Generiranje audio sadržaja neuronskim mrežama također predstavlja znatno teži zadatak od generiranja slika zbog izrazite osjetljivosti ljudskog sluha na karakteristike i nepravilnosti zvuka. Autori u [36] istražuju prikladnu arhitekturu GAN mreža i odgovarajuću reprezentaciju za sintetiziranje zvuka te predlažu GANSynth. Postupni proces treniranja predstavljen u [21] preuzimaju uz prilagodbe koje se prvo odnose na zamjenu slika zvukom, a dalje na uvjetovanje generatora dodatnim izvorom informacija o visini tona. Taj dodatni input dodaje se zatim latentnom vektoru s ciljem ostvrarivanja pojedinačne kontrole visine tona i timbra. Također, arhitekturu proširuju dodatnim klasifikatorom po uzoru na [37] koji predviđa oznaku visine tona. Generirani primjeri mogu se poslušati na [38].

5. GENERATIVNE SUPARNIČKE MREŽE I UMJETNOST

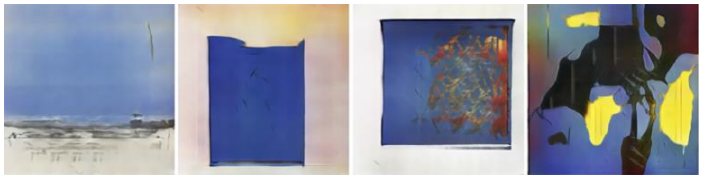
Ako se pokuša definirati što je to umjetnost, naići će se na brojne definicije. Jedna od poznatijih je ona Lava Nikolajevića Tolstoja [39] kojom ističe da je to ljudski, svjesni čin prijenosa emocija na druge. Zbog čvrstog povezivanja umjetnosti s kreativnošću, specifičnom estetikom i jedinstvenošću, ne čudi da su generativne mreže postale sredstvo izražavanja brojnih umjetnika ili onih koji se takvima osjećaju. Pri tome neki od njih stvaralaštvo u potpunosti prepuštaju mrežama, a drugi ih koriste samo za pomicanje granica svojih stvaralačkih mogućnosti.

Povezanost umjetnosti i GAN-ova izdvojena je u posebno poglavlje zbog drukčijeg pogleda na sam rezultat i očekivanja od mreža. U odnosu na primjere opisane u prethodnim poglavljima, u kojima je cilj bio postići što veću vjernost

generiranih podatak pravome skupu, zbog navedenih karakteristika umjetnosti puka emulacija postojećeg nije dostatna da bi se nešto proglasilo umjetničkim činom ili djelom [40].

Općenito se autori koji koriste GAN-ove u kontekstu umjetnosti mogu podijeliti na one koji nisu umjetnici, ali mrežama pokušavaju udahnuti karakteristike koje čovjeka čine umjetnikom, dok drugi koriste mreže u vlastitom stvaralaštvu.

Jedna grupa autora koja pripada prvoj skupini predlaže mrežu pod nazivom Creative Adversarial Networks (CAN) [40]. Mreža uči generirati umjetnička djela na temelju velikog broja primjera različitih umjetničkih stilova. Autori inspiraciju pronalaze u istraživanjima koja pokazuju da ona umjetnička djela koja izazivaju najviše pažnje dobro balansiraju količinu ekscitacije, izazvanu devijacijom od uobičajenog i predvidljivog, te količinu takve devijacije. Devijacija ne smije biti pretjerana kako ne bi proizvela negativni učinak. Autori modificiraju ciljnu funkciju kako bi implementirali ove spoznaje, a funkcija treba osigurati da mreža generira umjetnička djela koja ne pripadaju niti jednom poznatom stilu. Diskriminator ima dvostruki zadatak: uz standardni zadatak klasifikacije pravih i lažnih primjera, treba naučiti klasificirati i kojem stilu određena slika pripada. Takav signal predstavlja dodatnu pomoć generatoru. Koliko je mreža uspješna može se vidjeti na nekoliko primjera (slika 17).



Slika 17. Primjeri umjetničkih djela generiranih mrežom CAN [40]

U skupinu autora koji GAN-ove koriste s primarnim ciljem umjetničkog stvralaštva pripadaju Helena Sarin [41] ili Scott Eaton [42], umjetnici koji za treniranje mreža koriste vlastita umjetnička djela (slika 18).

Slika 18. Radovi umjetnika koji koriste vlastita umjetnička djela za treniranje GAN-ova: Helena Sarin (lijevo) [41] i Scott Eaton (desno) [42]

Međutim, nije rijetkost da se kao primarni kriterij prilikom prosudbe vrijednosti umjetničkog djela koristi novčani iznos koji je netko za njega voljan izdvojiti, pa je tako i portret prikazan na slici 19 prodan 2018. godine na aukciji za 432,500 USD [43]. Portret je generirala GAN mreža nakon učenja na 15,000 drugih portreta, a potpis “autora” u obliku ciljne fukcije može se vidjeti u donjem desnom kutu slike. Iako neki osporavaju umjetničku vrijednost takvog djela, iza njega ipak stoji skupina umjetnika pod nazivom Obvious [44].

Slika 19. Portret generiran pomoću GAN mreže prodan je na aukciji za 432,500 USD [43]

6. OGRANIČENJA GENERATIVNIH SUPARNIČKIH MREŽA

Iako je u kratkom roku, od svega nekoliko godina, postignut značajan napredak u generiranju sadržaja pomoću GAN-ova, i to s naglaskom na vizulanim sadržajima, još uvijek postoje određena ograničenja koja ujedno predstavljaju i aktivna područja istraživanja.

Za postizanje uvjerljivih rezultata često su potrebni značajni resursi, i to ne samo u vidu velikih količina podataka za treniranje već i jakog hardvera, što priječi njihovu širu primjenu. Npr. za treniranje mreže StyleGAN i generiranje primjera rezolucije 1024x1024 piksela potrebno je više od 6 dana ako se koristi osam Tesla V100 GPU-a, odnosno više od 41 dana ako se koristi samo jedan GPU.

Treniranje je i dalje teško i nestabilno iako su veliki pomaci učinjeni i u tom smjeru. Dodatnu poteškoću predstavlja nedostatak odgovarajućih metrika za evaluaciju rezultata generiranja. Iako su neke od predloženih metrika ušle u redovnu primjenu, kao što su to Inception Score (IS) [45], ili Multi-Scale Structural Similarity (MS-SSIM) [37] ne postoji opći konsenzus oko njihove dostatnosti.

7. ZAKLJUČAK

Duboke neuronske mreže posljednjih nekoliko godina ostvaruju značajne rezultate na brojnim područjima i zadacima koji su donedavno bili teško rješivi te su postale prvi izbor u radu s nestrukturiranim podacima, poput slika, videa, zvuka ili teksta na inače percepcijskim zadacima. Jedno od područja koje je doživjelo veliki zamah jest generiranje različitih digitalnih sadržaja, posebno onih vizualnih, u čemu prednjače generativne suparničke mreže.

U radu su prikazani neki od modela generativnih suparničkih mreža koji imaju sposobnost stvaranja atraktivnih sadržaja. Objašnjena je njihova temeljna arhitektura, način učenja te su dani prikazi nekih od uspješnih primjena. Od generiranih sličica veličine svega 28x28 piksela iz 2014. godine pa do generiranih slika visoke rezolucije prošlo je samo nekoliko godina. Iako određene poteškoće i dalje postoje, sve je veća primjena GAN-ova u komercijalnim proizvodima. Međutim, sve to ne ostaje bez kontroverzi i etičkih dvojbi. Ako se nastavi takav ubrzani napredak te mreže postanu još uvjerljivije u reproduciranju stvarnosti, ali i stvaranju neke nove, ostaje pitanje hoćemo li uskoro više moći razlikovati stvarnost od iluzije.

Literatura:

1 McCulloch, W. S., & Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. The bulletin of mathematical biophysics, 5(4), 115-133.

2 Deng, J., Dong, W., Socher, R., Li, L. J., Li, K., & Fei-Fei, L. (2009, June). Imagenet: A large-scale hierarchical image database. In 2009 IEEE conference on computer vision and pattern recognition (pp. 248-255). IEEE.

3 Google Translate [Online] Dostupno na: https://translate.google.com [Pristupljeno: 16.2.2021.]

4 FaceApp [Online] Dostupno na: https://www.faceapp.com/ [Pristupljeno: 16.2.2021.]

5 Xu, K., Ba, J., Kiros, R., Cho, K., Courville, A., Salakhudinov, R., ... & Bengio, Y. (2015, June). Show, attend and tell: Neural image caption generation with visual attention. In International conference on machine learning (pp. 2048-2057). PMLR.

6 Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., ... & Amodei, D. (2020). Language models are few-shot learners. arXiv preprint arXiv:2005.14165.

7 Goodfellow, I. J., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., ...& Bengio, Y. (2014). Generative adversarial networks. arXiv preprint arXiv:1406.2661.

8 He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 770-778).

9 LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. Proceedings of the IEEE, 86(11), 2278-2324.

10 Hubel, D. H., & Wiesel, T. N. (1962). Receptive fields, binocular interaction and functional architecture in the cat's visual cortex. The Journal of physiology, 160(1), 106-154.

11 LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. Nature, 521(7553), 436-444.

12 Deng, L. (2012). The mnist database of handwritten digit images for machine learning research [best of the web]. IEEE Signal Processing Magazine, 29(6), 141-142.

13 Krizhevsky, A., & Hinton, G. (2009). Learning multiple layers of features from tiny images.

14 The GAN Zoo [Online] Dostupno na: https://github.com/hindupuravinash/the-gan-zoo [Pristupljeno: 15.2.2021.]

15 Radford, A., Metz, L., & Chintala, S. (2015). Unsupervised representation learning with deep convolutional generative adversarial networks. arXiv preprint arXiv:1511.06434.

16 Simonyan, K., & Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556.

17 Ioffe, S., & Szegedy, C. (2015, June). Batch normalization: Accelerating deep network training by reducing internal covariate shift. In International conference on machine learning (pp. 448-456). PMLR.

18

19 Glorot, X., Bordes, A., & Bengio, Y. (2011, June). Deep sparse rectifier neural networks. In Proceedings of the fourteenth international conference on artificial intelligence and statistics (pp. 315-323). JMLR Workshop and Conference Proceedings.

20 Maas, A. L., Hannun, A. Y., & Ng, A. Y. (2013, June). Rectifier nonlinearities improve neural network acoustic models. In *Proc. icml* (Vol. 30, No. 1, p. 3).

21 Karras, T., Aila, T., Laine, S., & Lehtinen, J. (2017). Progressive growing of gans for improved quality, stability, and variation. ArXiv preprint arXiv:1710.10196.

22 Karras, T., Laine, S., & Aila, T. (2019). A style-based generator architecture for generative adversarial networks. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (pp. 4401-4410).

23 This person/chemical/artwork/cat does not exist [Online] Dostupno na: https://www.this + person/chemical/artwork/cat + doesnotexist.com [Pristupljeno: 15.2.2021.]

24 Antoniou, A., Storkey, A., & Edwards, H. (2017). Data augmentation generative adversarial networks. arXiv preprint arXiv:1711.04340.

25 Hrga, I., & Ivašić-Kos, M. (2019, May). Deep image captioning: An overview. In 2019 42nd International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO) (pp. 995-1000). IEEE.

26 Reed, S., Akata, Z., Yan, X., Logeswaran, L., Schiele, B., & Lee, H. (2016, June). Generative adversarial text to image synthesis. In International Conference on Machine Learning (pp. 1060-1069). PMLR.

27 Ledig, C., Theis, L., Huszár, F., Caballero, J., Cunningham, A., Acosta, A., ... & Shi, W. (2017). Photo-realistic single image superresolution using a generative adversarial network. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 4681-4690).

28 Pathak, D., Krahenbuhl, P., Donahue, J., Darrell, T., & Efros, A. A. (2016). Context encoders: Feature learning by inpainting. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 2536-2544).

29 Isola, P., Zhu, J. Y., Zhou, T., & Efros, A. A. (2017). Image-to-image translation with conditional adversarial networks. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 1125-1134).

30 Li, X., Chen, H., Qi, X., Dou, Q., Fu, C. W., & Heng, P. A. (2018). H-DenseUNet: hybrid densely connected UNet for liver and tumor segmentation from CT volumes. IEEE transactions on medical imaging, 37(12), 2663-2674.

31 Zhu, J. Y., Park, T., Isola, P., & Efros, A. A. (2017). Unpaired image-to-image translation using cycle-consistent adversarial networks. In Proceedings of the IEEE international conference on computer vision (pp. 2223-2232).

32 FaceSwap [Online] Dostupno na: https://faceswap.dev [Pristupljeno: 18.2.2021.]

33 Nirkin, Y., Keller, Y., & Hassner, T. (2019). Fsgan: Subject agnostic face swapping and reenactment. In Proceedings of the IEEE/CVF International Conference on Computer Vision (pp. 7184-7193).

34 Yu, L., Zhang, W., Wang, J., & Yu, Y. (2017, February). Seqgan: Sequence generative adversarial nets with policy gradient. In Proceedings of the AAAI conference on artificial intelligence (Vol. 31, No. 1).

35 Guo, J., Lu, S., Cai, H., Zhang, W., Yu, Y., & Wang, J. (2018, April). Long text generation via adversarial training with leaked information. In Proceedings of the AAAI Conference on Artificial Intelligence (Vol. 32, No. 1).

36 Engel, J., Agrawal, K. K., Chen, S., Gulrajani, I., Donahue, C., & Roberts, A. (2019). Gansynth: Adversarial neural audio synthesis. arXiv preprint arXiv:1902.08710.

37 Odena, A., Olah, C., & Shlens, J. (2017, July). Conditional image synthesis with auxiliary classifier gans. In International conference on machine learning (pp. 2642-2651). PMLR.

38 GANSynth: Making music with GANs [Online] Dostupno na: https://magenta.tensorflow.org/gansynth [Pristupljeno: 16.2.2021.]

39 Tolstoj, L. N. (1960). What is Art? Translated from the Russian Original by Aylmer Maude. With an Introduction by Vincent Tomas.Liberal Arts Press.

40 Elgammal, A., Liu, B., Elhoseiny, M., & Mazzone, M. (2017). Can: Creative adversarial networks, generating" art" by learning about styles and deviating from style norms. arXiv preprint arXiv:1706.07068.

41 Helena Sarin [Online] Dostupno na: https://aiartists.org/helena-sarin [Pristupljeno: 18.2.2021.]

42 Scott Eaton [Online] Dostupno na: http://www.scott-eaton.com [Pristupljeno: 18.2.2021.]

43 Christie's: Is artificial intelligence set to become art's next medium? [Online] Dostupno na: https://www.christies.com/features/A-collaboration-between-two-artists-one-human-one-a-machine-9332-1.aspx [Pristupljeno: 18.2.2021.]

44 Obious [Online] Dostupno na: https://obvious-art.com [Pristupljeno: 18.2.2021.]

45 Salimans, T., Goodfellow, I., Zaremba, W., Cheung, V., Radford, A., & Chen, X. (2016). Improved techniques for training gans. arXiv preprint arXiv:1606.03498.

Podaci o autoru:

mag.oec. Ingrid Hrga
E-mail: ingrid.hrga@gmail.hr

Ingrid Hrga je doktorandica Odjela za informatiku, Sveučilišta u Rijeci, modul Inteligentni računalni sustavi te asistentica na Fakultetu informatike, Sveučilišta Jurja Dobrile u Puli gdje je angažirana na kolegijima Umjetna inteligencija, Primijenjena statistika, Operacijska istraživanja te Multimedijalni sustavi. Područje njenog primarnog istraživačkog interesa su rješenja bazirana na dubokim neuronskim mrežama. Usavršavala se u zemlji i inozemstvu te je izlagala na znanstvenim i stručnim konferencijama.

Fakultet informatike
Sveučilište Jurja Dobrile u Puli
Rovinjska 14
52100 Pula

MIKROSERVISNA ARHITEKTURA - PREGLED RAZVOJNIH PLATFORMI

Vedran Zdešić, mag. ing. el.

SAŽETAK:

Od trenutka kada je razvoj složenih sustava intenzivno krenuo u smjeru mikro servisne arhitekture kakvu danas znamo, početni entuzijazam je oscilirao. Sada nam je puno jasnije kada treba koristiti takvu arhitekturu, a kada su možda druga rješenja bolja. Bolje razumijemo cijenu i potreban trud razvoja takve arhitekture. Znamo gdje se troši najviše vremena i ljudskih resursa tijekom razvoja i održavanja. Nova generacija alata i platformi usmjerena je na rješavanje uočenih problema apstrahirajući mehanizme komunikacije među servisima, te procese pokretanja i distribucije. Ovo predavanje daje pregled evolucije tih alata i platformi te ih uspoređuje sa starijim alatima i platformama koje polako gube trku na tržištu.

KEYWORDS: ASF, ASF Mesh, K8s, AKS, Service Mesh, DAPR, TYE, OAM, Serverless, Cloud, Docker.

ABSTRACT:

From the very first moment, when development of complex systems begins to follow so called microservices architecture, we learned a lot along the way. We now understand when to use microservices, and when some other solutions can be more useful. We now better understand the price and effort needed to develop product according to the microservices architecture. We know where we spend the most time and resources during development and maintenance of such a software product. New generation of the related tools and platforms tries to solve all these problems making complex parts of the system easy to use by developers. It helps us to put our focus to the business logic, rather than spending significant effort to implement communication between services, creating and using service bus or saving the data to underlying storage. This lecture gives the overview of the evolution of these tools and platforms. We will compare new and old generation, and we will try to explain why old generation is losing the race on the market.

KEYWORDS: ASF, ASF Mesh, K8s, AKS, Service Mesh, DAPR, TYE, OAM, Serverless, Cloud, Docker.

1. UVOD

Cilj ovog dokumenta je omogućiti lakši odabir odgovarajuće platforme za razvoj mikro servisa ovisno o slijedećim parametrima:

- Tip aplikacije/sistema/proizvoda
- Veličina tima
- Dostupno znanje u timu
- Proračun
- Vremenska ograničenja

Zaključci doneseni ovdje su bazirani na praktičnom iskustvu, ne na knjiškom znanju. U pojedinim dijelovima rada osvrnuti ću se na momente kada je teorija i praksa u sukobu prema mom vlastitom iskustvu.

2. ZA I PROTIV MIKRO SERVISA

Za početak, budimo svjesni da su uspješne svjetske kompanije (Netflix, Uber, Airbnb) kretale sa monolitnim arhitekturama i tek kada su bile spremne za tranziciju, migrirali su postepeno na mikro servise. Iz praktičnog iskustva možemo reći da se korištenje mikro servisa isplati tek od onog trenutka kada aplikacija postane prevelika za održavanje, uključujući razvoj, testiranje i isporuku. Sa točke gledišta pouzdanosti, postoji puno veća mogućnost greške tijekom komunikacije između različitih mikro servisa nego što je to u monolitnoj okolini. Kritična točka je također i cijena pokretanja, orkestracije i nadgledanja mikro servisa. Njihov razvoj i održavanje ima svoju cijenu i dobro je kretati u razvoj imajući to na umu. Navedimo prvo neke od situacija kada se kretanje u razvoj mikro servisa čini neopravdanim:

- Gradim prototip (namijenjen investitorima ili managementu)
- Proizvod još uvijek nema dobro definiranu poslovnu logiku, te se ne želimo istovremeno suočavati sa tehničkom i poslovnom kompleksnošću sustava.
- Nismo se u stanju suočiti sa kompleksnošću dizajna i razvoja mikro servisa:
  - Moj tim nije dovoljno jak i brojčano velik.
  - Znanje mog tima nije u skladu sa najnovijim trendovima.
  - Imam stroge vremenske rokove.
  - Moj tim nema dovoljno DevOps znanja.

Postoje i suprotne situacije. Nekada je već od početka jasno da će najbolji odabir arhitekture biti mikro servisna arhitektura. Primjerice, najvjerojatnije želimo razvijati mikro servise već od samog početka u slijedećim slučajevima:

- Postoji zahtjev za brzom i neovisnom isporukom servisa. U tom slučaju servisi moraju biti mali, kompaktni i autonomni.
- Dijelovi sistema moraju biti ekstremno efikasni. Tada će njihovo skaliranje i sveukupno održavanje, ali i nadgledanje biti puno jednostavnije ako su to mikro servisi.
- Imam plan kako ću vremenom ojačati svoj tim. Projekt može započeti jaka jezgra tima sastavljena od arhitekata koji će ispravno postaviti platformu i smjernice razvoja, ali moramo biti svjesni da vremenom moramo osnažiti tim kako bi na vrijeme i temeljito pokrili sve aktivnosti potrebne za ostvarenje cijelog projekta.

Svaka specifična situacija, svaka tvrtka, ima rješenje koje najbolje odgovara njenim potrebama, ali da bi se do tog rješenja došlo, mora se analizirati konkretna situacija. Konačno rješenje može uključivati neko od slijedećih kombinacija arhitektura:

- Monolitna
- Mikro servisna
- Cloud nativna
- Miješana/hibridna.

Moram ovdje pojasniti razliku između mikro servisne i Cloud nativne arhitekture. Opišimo to na primjeru izreke koja se može susresti u literaturi, a koja kaže: „Što je kompanija manja, to je važnije da u svom razvoju razmišlja na način mikro servisa.“. Što ova izreka znači? Prema gornjoj klasifikaciji, čini se da neki autori miješaju mikro servisnu i Cloud nativnu arhitekturu. Da li oni žele reći da svaka mala kompanija svoje softverske proizvode treba razvijati u skladu sa mikro servisnom arhitekturom? Naravno, ne. To bi moglo biti potpuno kontra produktivno. Poanta ove izreke je da čak i kada razvijamo monolitne aplikacije trebali bi koristiti sve prednosti Cloud okoline koja će nam omogućiti jednostavniji razvoj, testiranje, konfiguraciju i isporuku. Dakle, to ne uključuje nužno mikro servisnu arhitekturu pojedine aplikacije kao zahtijevanu. Nadalje, svaka aplikacija građena prema mikro servisnoj arhitekturi se može pokretati u potpunosti izvan Clouda, na lokalnim serverima, On-Premises. Na taj način treba razlikovati ove dvije vrste aplikacije.

Pred mikro servisnu arhitekturu se često stavljaju slijedeći zahtjevi:

- Automatizirani i brzi sistemski build, test i isporuka uz odgovarajuće CI/CD procese
- Zero-downtime deployment, neprekinuti rad sustava
- Sloboda isporuke na bilo koju okolinu:
  - Cloud (Azure, AWS, …)
  - Edge (što uključuje On-Premises i više)
- Iako kompleksan, sistem se mora lako nadgledati
- Sustav mora biti otporan na vanjske utjecaje (eng. Resiliency)
- Zahtijeva se neovisnost o operativnom sustavu
- Laka konfiguracija
- Poželjno je da sustav radi u Serverless okruženju. Serverless okruženje je ono u kojem davatelj Cloud usluge brine o alociranju hardverskih resursa prema definiranim zahtjevima. Sam korisnik Cloud usluge se u ovom slučaju fizički ne brine o serverima, diskovima, memoriji, i sl.

Određeni zahtjevi, iako su tehnički mogući, u praksi mogu biti nerealistični prema mojem iskustvu. Navedimo neke od njih:

- Implementacija različitih servisa u različitim tehnologijama
  - Teško je u timu održavati široki spektar znanja tijekom životnog vijeka servisa
  - TCO (Total Cost of Ownership) može biti vrlo visok u tom slučaju
  - Može biti korisno jedino ako vidimo komponentu kao „black box“
- Decentralizacija, kao moguća posljedica mikro servisne arhitekture
  - Zapravo se ne događa u praksi
  - Svi servisi su pokrenuti na jednom Cluster serveru
  - Svi servisi se nalaze iza jednog API Gateway-a
- Više timova može raditi istovremeno na sustavu
  - To je efikasan način rada, ali je cijena često problem
  - Izvedivo za velike proizvode i/ili kompanije

U detaljnijem analiziranju specifične situacije, te odlučivanju oko najprikladnije arhitekture za naš sustav, može nam pomoći SWOT analiza. Parametri koje bi pritom razmatrali su slijedeći:

- Strengths / Weaknesses
  - Snaga Dev tima (broj/znanje/tehnologije)
  - Snaga DevOps tima (broj/znanje/tehnologije)
- Opportunities / Threats
  - Proračun / investicije
  - Vremenska ograničenja

U nastavku ćemo, za svaku razvojnu platformu, naglasiti uvjete i strukturu tima koji bi najviše profitirao od njenog korištenja prema ovoj listi parametara.

3. SLOJEVI MIKRO SERVISNE ARHITEKTURE

Dodajmo i ovo. Kako bi u cijelosti razumjeli poruku ovog rada, imajmo na umu da mikro servisnu arhitekturu možemo podijeliti u slojeve:

- Hosting platforma (Kubernetes, Docker Swarm)
- Networking (Service Meshes)
- Auto-Scaling (Keda, Knative)
- Razvojna platforma (ASF, ASF Mesh, DAPR)

U ovom je radu svakako naglasak na razvojnim platformama, međutim, spominjati ćemo i ostale slojeve i pripadajuće alate na način da ih stavimo u relaciju sa procesom razvoja i razvojnim platformama, te pojasniti kako i kome pomažu u određenim fazama.

4. RAZVOJNE PLATFORME I ALATI

Pogledajmo sada kako su se platforme za razvoj mikro servisa povijesno pojavljivale, te koje smo sve opcije razvoja koristili.

4.1. Azure Service Fabric (ASF)

ASF je jedna od prvih platformi koja se pojavila na tržištu. Prvo je korištena od strane Microsoft-a na njihovim internim projektima. Azure Cloud je izgrađen na ASF-u, čak i dijelovi Kubernetesa na Azure-u (AKS) koji se tretira kao njegova direktna konkurencija. Nije primarno građen za mikro servise, nego radije za distribuirane aplikacije. .NET Dev tim bez puno DevOps znanja može profitirati od ove platforme. Bliskost sa .NET-om će im omogućiti da postanu familijarni sa platformom u kratkom vremenu, a također će moći koristiti prednosti ugrađenih alata na nivou same platforme, primjerice za nadgledanje sustava. Neke od loših strana koje se vežu uz ovu platformu su spori sistem start i debugging u lokalnoj razvojnoj okolini, te nespretno rukovanje projektima.

4.2. Azure Service Fabric Mesh (ASF Mesh)

ASF Mesh je naprednija, Serverless opcija ASF-a. Ova platforma bi mogla biti odlična opcija za manje .NET timove koji nemaju dovoljno DevOps znanja. Migracija sa ASF-a na ASF Mesh nije zahtjevna. U jednostavnijim slučajevima dovoljno je samo zamijeniti pripadajući Cluster server. Loša je vijest da smo sa ASF Mesh-om strogo vezani samo na Microsoft Azure cloud. Također, ASF Mesh je u Preview fazi već jako dugo, bez naznaka da će stvarno ući u fazu produkcije.

Ako ćemo biti iskreni, ASF i ASF Mesh gube trku sa ostalim platformama, i otpasti će vremenom. Osobno sam se nadao drugačijem ishodu, ali to izgleda više nije realna opcija.

4.3. Prilagođeni razvoj mikro servisa

Servise i sve što ide oko njih, naravno, uvijek smo mogli i možemo i sada razvijati u potpunosti sami (koristeći .NET Core, Node, Python, i druge programske jezike), ne oslanjajući se na određenu mikro servisnu radnu okolinu ili platformu. Tako razvijene servise možemo pokretati uz pomoć Kubernetes-a i/ili Service Mesh alata. U slučaju da smo se odlučili za ovu opciju, tada je puno posla tijekom razvoja na nama samima i moramo se pobrinuti implementirati slijedeće:

- Data components: storage, service bus, secret store
- Health checks (shallow / deep health checks)
- Automatski deployment (Ansible, Terraform, Helm)
- API Gateway (setup i konfiguracija)
- I drugo.

Prije nekoliko godina, ovo je bio jedini način kako su se mikro servisi razvijali. Danas, kako bi povećali svoju produktivnost, koristimo što je više moguće gotovih alata i komponenti. U prilagođeni razvoj sustava ići ćemo samo u rijetkim situacijama i to samo na dijelovima sustava gdje želimo zadržati potpunu kontrolu.

4.4. Kubernetes (K8s)

Kubernetes spada u hosting platforme. To je sistem orkestrator za automatiziranje isporuke, skaliranje i upravljanje aplikacija sadržanih unutar kontejnera. Kako smo već spomenuli, servise koje sami razvijemo možemo pokretati unutar Kubernetes-a.

Da bi se koristili Kubernetes-om trebamo imati puno DevOps znanja. Vrijeme koje je potrebno da se razvije poslovna funkcionalnost aplikacije jednaka je vremenu koje je potrebno da se razvije infrastruktura u ovom slučaju. Također, razvoj infrastrukture treba započeti od prvog dana razvoja, paralelno sa razvojem aplikacije. Kubernetes se koristi naširoko. Prema nekim statistikama, 90% kompanija ili već ima ili želi imati Kubernetes kao dio svoje mikro servisne infrastrukture.

Postoje alati s kojima se Kuberenetes može pokretati u Serverless okruženju.

4.5. Service Meshes

U ovu grupu spadaju različiti alati odgovorni za organizaciju i komunikaciju mikro servisa na nivou infrastrukture. Neki od poznatih alata iz ove grupe su:

- Linkerd
- Conduit
- Istio

Funkcionalnosti koje ovi alati pokrivaju su primjerice: load balancing, service discovery, circuit breaking, dynamic request routing, HTTP proxy integration, retries, deadlines, TLS, transparent proxying, distributed tracing, instrumentation itd. Mogu biti korišteni u lokalnoj okolini, sa kontejnerima, unutar Kubernetes-a, i sl. Uglavnom ih koriste DevOps eksperti, a ne toliko developeri, zato dajem samo njihov kratki pregled.

U slijedećim točkama ćemo spomenuti platforme koje stvarno donose revoluciju u moderni razvoj mikro servisa.

4.6. DAPR (Distributed Application Runtime)

DAPR je relativno novi alat. Ove godine (02/2021) je ušao u fazu produkcije. Ako krećete u razvoj novog sustava, obavezno proučite i razmotrite korištenje ove platforme. DAPR olakšava mnoge izazove s kojima smo se suočavali u razvoju mikro servisa u prošlosti. Njegovi građevni blokovi (building blocks) apstrahiraju kritične dijelove sustava, poput:

- Service Invocation
- State Management
- Publish/Subscribe Mechanism
- Bindings
- Actors
- Observability
- Secret Store

Komponenta je implementacija određenog građevnog bloka. Komponenta je konfigurabilna, i za nas tijekom razvoja predstavlja crnu kutiju. Najbolja karakteristika ovakvog načina rada je u tome što konačni odabir spremišta podataka ili stanja sustava, te service bus komponente možemo odgoditi do trenutka isporuke, a možemo ih i mijenjati tijekom razvoja ili produkcije ako je to potrebno.

Kao orkestrator u kombinaciji sa DAPR-om možemo koristiti Tye u lokalnoj okolini, koji također može poslužiti kao alat za isporuku u Kubernetes, bilo da ostajemo u lokalnoj okolini, bilo da migriramo u Cloud.

Napomenimo još da se DAPR funkcionalno preklapa sa Service Mesh alatima, ali to ne znači da oni ne mogu biti korišteni paralelno. Dapače, ako nam zajedno mogu pomoći, u redu ih je koristiti u kombinaciji.

4.7. OAM (Open Application Model)

OAM je standard otvorenog tipa uveden od strane Microsoft-a i Alibaba Cloud-a, a sve zajedno pod okriljem OAM Foundation. To je specifikacija neovisna o tehnologiji koja definira mikro servisnu arhitekturu i pomaže tijekom kreiranja platforme s primarnim fokusom na aplikaciju. Specifikacija omogućava razvojnom timu pravilno definirati koherentni model koji predstavlja aplikaciju i odvaja opis aplikacije od same isporuke. Zapravo OAM razlikuje tri uloge u timu koji razvija aplikaciju:

- Razvojni inženjeri – razvijaju poslovnu logiku aplikacijskih komponenata,
- Aplikacijski operateri – konfiguriraju, instaliraju i upravljaju komponentama i aplikacijom,
- Operateri infrastrukture – upravljaju infrastrukturom, često na nivou kompanije.

Aplikacijski model uzima u obzir ove uloge i kreira arhitekturu na način da je moguće jasno odvojiti aktivnosti vezane na njih u procesu razvoja, pokretanja i održavanja aplikacije. U budućnosti je za očekivati da će alati omogućavati sve bogatija grafička sučelja preko kojih će biti moguće napraviti sve ono što danas rade operateri.

Prva implementacija OAM-a se zvala Rudr (na Kubernetes platformi), međutim kako se sve ovdje odvija vrlo brzo, ova implementacija je već sada zastarjela. Jedna od trenutno aktivnih implementacija se zove KubeVela. OAM i pripadajuće implementacije svakako treba pratiti u budućnosti.

Važno je naglasiti da DAPR nema namjeru ulaziti u polje aplikacijske arhitekture. DAPR će ostati programski model, te tako djelovati na nižim razinama apstrakcije. Iz tog razloga moguće ga je kombinirati paralelno sa nekom od OAM implementacija.

5. COMPOSITE UI / MODULAR WEB

Na kraju spomenimo i ovo. Kada smo unutar našeg sustava uložili već toliko truda u kreiranje kvalitetne mikro servisne arhitekture, tada svakako ne želimo imati monolitni UI. Monolitni UI je teško održavati i debuggirati. Sve prednosti koje imamo na razini servisa želimo imati i na korisničkom sučelju. U tu svrhu kreirati ćemo UI koje će biti modularan ili kompozitan, modul svakog servisa će se razvijati i učitavati unutar glavne UI ljuske kao odvojena cjelina. Nazivi koji se koriste za takav tip sučelja su Composite UI ili Modular Web.

U slučaju da želite slijediti ovu preporuku dizajniranja korisničkog sučelja, a ne želite počinjati od nule, u pomoć možete pozvati neki od gotovih radnih okolina, ovisno o tehnologiji koju koristite i sloju na kojem implementirate. Pa tako imamo slijedeće radne okoline dostupne na tržištu:

- Klijentska strana: Piral, OpenComponents
- Serverska strana: Mosaic, PuzzleJs
- Biblioteke: Single SPA

Ono što mogu naglasiti kao potencijalno interesantno za .NET developere je radna okolina Oqtane, podržana od strane .NET Foundation. Ona omogućava kreiranje kompozitnog sučelja ili modularnog web-a u odnosu na Blazor tehnologiju.

6. KRATKA VIZIJA BUDUĆNOSTI

Aplikacije budućnosti biti će zasigurno fleksibilne, moći će se pokretati u bilo kojoj radnoj okolini, na bilo kojem Cloudu, operativnom sustavu, ili uređaju. Sve to će biti upakirano u jedno rješenje. Time se otvaraju nove mogućnosti distribucije. Zamislimo samo slijedeću situaciju. Do sada smo imali App Store vezan uvijek za davatelja Cloud usluge. Ta situacija se sada može promijeniti. Postoji prostor za neovisne App Store davatelje usluga koji će nuditi aplikacije dovoljno fleksibilne da se instaliraju na bilo koju Cloud ili Edge okolinu.

7. ZAKLJUČAK

Ako krećete u razvoj novog softverskog proizvoda, razmišljate o njegovoj arhitekturi, birate najbolje radne okoline, platforme i alate, budite otvoreni prema novim trendovima, ali ih proučite kritički. Oni donose dobrobiti ako se koriste ispravno i u skladu sa okolnostima. U suprotnom, donose gubitke. Analizirajte detaljno svoju situaciju prije konačne odluke. Dobra komunikacija sa investitorima ili menadžmentom je također ključna.

U praksi sam svjedočio projektima na kojima zbog loše komunikacije menadžera i razvojnih timova menadžment nije u konačnici zadovoljan proizvodom. Očekivali su više implementirane poslovne funkcionalnosti, dok ih je razvojni tim uvjeravao da je napravljena stabilna baza na kojoj će se moći razvijati sustav budućnosti. Menadžment nije vidio niti cijenio napor razvojnog tima u takvim slučajevima.

Ako se zbog loše komunikacije, nejasno definiranih rokova ili proračuna dogodi raskorak između vidljivih poslovnih rezultata i uloženih sredstava, to svakako nije dobro i smanjuje povjerenje investitora u ishod projekta. Nemojte forsirati mikro servisnu arhitekturu u slučaju kada investitor želi vidjeti korisnu funkcionalnost što je prije moguće. Krenite sa monolitnom arhitekturom, te planirajte tranziciju na mikro servisnu. I u okviru monolitne arhitekture se može kreirati pravilna modularna arhitektura koju je lakše migrirati prema mikro servisnoj. Pritom, monolitna će vam arhitektura omogućiti da uštedite i do 50% vremena razvoja koji bi se inače potrošio na razvoj infrastrukture.

Ne zaboravite da i običan App Service na Azuru, može osigurati brzo, jeftino i kvalitetno rješenje za pokretanje i isporuku vašeg rješenja. Primjerice, uz uvođenje jednog slota po svakoj okolini (Development, Testing, Staging, Production) možete implementirati kvalitetan CI/CD proces. Pritom isporuka je više nego jednostavna. Generalno gledano, korištenje servisa bilo kojeg Cloud-a će vam omogućiti dobro planiranje i brzu tranziciju iz monolitne u mikro servisnu arhitekturu, već prema potrebi.

Napomenimo još jednu važnu činjenicu. Svijet u kojem živimo se ubrzano mijenja, nude se nove usluge iz dana u dan. Dostupni su nam alati koje prije nismo imali. Moramo budno pratiti sve te promjene, čak i kada završimo sa aktivnim razvojem naše aplikacije. Pod tim pritiskom, može se dogoditi da se arhitektura vaše aplikacije promijeni zbog cijene implementacije, cijene Cloud usluge ili cijene nekih drugih resursa. Trenutna arhitektura vaše aplikacije često neće biti tehnički najidealnija, ali će zato uvijek nastojati biti najjeftinija. Ne zaboravite na to tijekom planiranja razvoja vašeg softverskog proizvoda. Tako nećete dovesti u pitanje svoju produktivnost i konkurentnost.

Ako krećete u razvoj i sigurni ste da želite mikro servise, onda obavezno uzmite u obzir DAPR, OAM, Kubernetes i Composite UI.

Literatura:

1 DAPR Docs (https://docs.dapr.io/)

2 OAM Specification (https://oam.dev/)

3 Kubernetes Docs (https://kubernetes.io/docs/)

Informacije o autoru:

Vedran Zdešić, mag. ing. el.

Velike dimenzije d.o.o.

Jasinje 8, Velika Ostma

10370 Dugo Selo

e-mail: vzdesic@gmail.com

Vedran Zdešić, mag. ing. el., MCT, senior konzultant, softver arhitekt, solution developer, predavač i mentor na VERN-u. Zadnjih sedam godina angažiran na projektima velikih kompanija američkog i europskog tržišta.

OGLEDNI PRIMJER WEB APLIKACIJE VELERI-OI METEO SYSTEM RAZVIJENE ZA POTREBE OBRAZOVNOG PROGRAMA VELERI-OI IOT SCHOOL

EXAMPLE OF THE VELERI-OI METEO SYSTEM WEB APPLICATION DEVELOPED FOR THE NEEDS OF THE VELERI-OI IOT SCHOOL EDUCATIONAL PROGRAM

Alen Jakupović, Sanja Čandrlić, Sabrina Šuman, Martina Ašenbrener Katić, Danijela Jakšić, Lucia Načinović Prskalo, Vanja Slavuj, Vedran Miletić, Marin Kaluža, Vlatka Davidović, Davor Širola, Ozren Rafajac, Miran Pobar, Damir Malnar

SAŽETAK:

U okviru EU projekta "Razvoj internacionalnog obrazovnog programa Veleri-OI IoT School", kojeg je sufinancirala Europska unija iz Europskog socijalnog fonda, razvijen je novi internacionalni obrazovni program u području interneta stvari. Nastava novog obrazovnog programa se temelji na metodi "learning by doing" koja je implementirana kroz održavanje radnih sastanaka i definiranje radnih zadataka. Cjelokupan nastavnički i studentski rad je povezan s oglednim primjerom web aplikacije nad kojom studenti uče i čija rješenja analogijom primjenjuju na svoje projekte. Ogledni primjer web aplikacije izrađen je primjenom frontend Quasar frameworka i backend Google Firebase serverless platforme. Članak prikazuje dio procesa razvoja web aplikacije kojeg su nastavnici koristili u poučavanju studenata na radnim sastancima. Isti taj proces su kasnije studenti implementirali u svoje projekte.

Ključne riječi: Web aplikacija, learning by doing, Quasar, Google Firebase, Veleri-OI Meteo System

ABSTRACT:

As part of the EU project entitled "Development of the International Education Program Veleri-OI IoT School", co-financed by the European Union from the European Social Fund, a new international educational program, in the field of the Internet of things, has been developed. The classes of the new educational program are based on the learning-by-doing method and implemented through work meetings and defining work assignments. The entire work done by teachers and students is related to the illustrative example of a web application used as part of student training. The solutions found in the example are further used by the students who apply them analogously to their own projects. The illustrative example was built using the Quasar framework for front-end, as well as the Google Firebase serverless platform for back-end. This paper describes part of the web application development process used by the teachers in the program to teach students during work meetings. The same process was later implemented by the students in their own projects.

1. UVOD

Veleučilište u Rijeci kao nositelj i Sveučilište u Rijeci, Odjel za informatiku kao partner pokrenuli su 2018. godine trogodišnji EU projekt "Razvoj internacionalnog obrazovnog programa Veleri-OI IoT School", kojeg je sufinancirala Europska unija iz Europskog socijalnog fonda. Cilj projekta je bio razviti internacionalni obrazovni program iz područja Interneta stvari (IoT). Trenutno je projekt u fazi provedbe pilot obrazovnog programa koji se sastoji od sedam modula: Razvoj poslovne ideje (3 ECTS), Dokumentiranje korisničkih zahtjeva (3 ECTS), Uspostava razvojnog okruženja (3,5 ECTS), Nerelacijske baze podataka (5 ECTS), Razvoj web-aplikacija (6 ECTS), Razvoj hibridnih mobilnih aplikacija (3,5 ECTS), Arduino ugradbeni (IoT) sustavi (6 ECTS).

Nastava obrazovnog programa se temelji na metodi "learning by doing" koja od studenata traži aktivan angažman i stjecanje iskustva rada na konkretnim projektima. Kako bismo implementirali ovu metodu, ali i studentima približili proces rada u poduzećima, cjelokupnu nastavu smo organizirali u vidu tjednih radnih sastanaka. Na radnim sastancima nastavnik je studente poučavao koristeći ogledni primjer web aplikacije i ugradbenog sustava koji zajedno realiziraju IoT meteo sustav. Prilagodba i primjena rješenja iz oglednog sustava u studentskim projektima predstavljala je osnovu tjednih radnih zadataka. Na zakazane termine, studenti su prezentirali svoje implementacije i branili rješenja zadataka.

Ovaj članak donosi opis dijela procesa razvoja oglednog primjera web aplikacije kojeg su nastavnici koristili u poučavanju na radnim sastancima, a koje su kasnije studenti koristili u realizaciji radnih zadataka. Sam proces razvoja se sastoji iz niza poznatih faza razvoja softvera: analiza, dizajn, implementacija. Ovim fazama smo u obrazovnom programu dodali još dvije: prije analize, fazu u kojoj studenti pripremaju svoje projekte kako bi se upoznali s poduzetničkim pothvatom i prije implementacije, fazu u kojoj studenti pripremaju složeno razvojno okruženje u kojemu će implementirati svoje projekte.

Članak svojim poglavljima prati navedene faze razvoja (analiza, dizajn i implementacija).

U poglavlju "Specifikacija korisničkih zahtjeva" opisane su primijenjene metode i rezultati koji spadaju u fazu analize procesa razvoja softvera. Primijenjene su sljedeće metode: tekstualni opis sustava, korisničke priče i UML dijagram uporabe s opisom slučaja uporabe.

U poglavlju “Oblikovanje baze podataka i aplikacije” opisana je osnovna informacijsko-komunikacijska arhitektura cjelokupnog hardversko-softverskog sustava “Veleri-OI Meteo System”. Prikazan je model podataka kao relacijski i nerelacijski model te skice sučelja buduće aplikacije.

U poglavlju “Implementacija aplikacije” opisane su korištene tehnologije Quasar i Google Firebase. Prikazani su i objašnjeni neki dijelovi programskog koda.

2. SPECIFIKACIJA KORISNIČKIH ZAHTJEVA

Prvi je korak u razvoju sustava prepoznati korisničke zahtjeve. Njihovo ispunjavanje preduvjet je za uspjeh projekta. Kako bi se osiguralo da proizvod zadovoljava stvarne potrebe korisnika, zahtjevi koje korisnik ima od tog proizvoda trebaju se dogovoriti, oblikovati i dokumentirati. Prepoznavanje onoga što korisnik treba i traži te jasna komunikacija o tome presudna je za uspjeh svakog novog projekta. Zahtjevi koji trebaju biti iskazani jasno i jednoznačno zapisuju se u dokumentu specifikacije ili na neki drugi način, koristeći neku drugu metodu (poput korisničkih priča, slučajeva upotrebe ili drugih modela).

Ovako se osigurava da klijent-naručitelj i pružatelj usluga rade na postizanju istog cilja, kao i da će sustav raditi upravo ono što korisnici trebaju.

Problemska situacija koju projektom želimo riješiti vezana je uz izradu sustava, odnosno aplikacije za praćenje različitih meteoroloških podataka (temperatura, vlaga, brzina vjetra, padaline, …). Sustav omogućuje prijavu i autentikaciju administratora. Registracija je moguća korištenjem e-mail adrese, Facebook podataka ili gmail podataka. Korisnik može pregledavati podatke bez registracije.

Administrator može ažurirati (upisivati, mijenjati i brisati) podatke o meteo stanicama i senzorima koji vrše mjerenja kako bi se na temelju dozvoljenih vrijednosti senzora mogao aktivirati alarm na meteo stanici za slučaj mjerene veličine izvan predviđenog intervala. Administrator pri unosu ili izmjeni podataka meteo stanice ima mogućnost očitavanja trenutne geo lokacije uređaja (ukoliko je gps aktiviran preuzima lokaciju ili nakon 5 sekundi čekanja ponavlja traženje) pa se koordinate postavljaju na vrijednost geografska duljina i širina. Kod mobilne aplikacije administrator ima mogućnost u samoj aplikaciji pokrenuti kameru kako bi fotografirao meteo stanicu na lokaciji.

Korisnik može dohvatiti podatke o meteo stanicama na kojima se vrše mjerenja putem google maps prikaza odabirom markera vidljivog na karti, kako bi pregledao aktualne vrijednosti mjerenih veličina uz popratni opis stanice i prisutnih senzora kao i slike same meteo stanice.

Korisnik može za odabranu meteo stanicu odabrati grafički i/ili tablični prikaz mjernih vrijednosti unutar traženog perioda kako bi dobio uvid u dinamiku mjerenih veličina.

Na temelju ovog grubog, općeg iskaza korisničkih zahtjeva, u sljedećem se koraku korisnički zahtjevi rafiniraju. Detaljnije zapisani zahtjevi rezultat su dublje analize i razumijevanja do kojeg dolazi u suradnji s korisnicima. Iako se neki pristupi ograničavaju na primjenu samo određenih metoda, može se koristiti i hibridni pristup kojim se kombinira više njih. Metode odabrane za detaljniji zapis općih korisničkih zahtjeva su korisničke priče, slučajevi uporabe i dijagrami slučaja uporabe. Njihovom primjenom razvojni tim bolje spoznaje očekivanja i zahtjeve korisnika.

Korisničke priče fokusiraju se na cilj krajnjeg korisnika kako bi se iskazalo (i zapisalo) ono što će softver moći obavljati. Korisničkim pričama se potiče razgovor i rasprava o izvedivosti korisničkih zahtjeva, o obuhvaćenosti svih željenih funkcionalnosti i o potencijalnim problemima zadovoljenja zahtjeva. Korisničke priče su kratki, jednostavni opisi značajke ispričane iz perspektive osobe koja želi nove mogućnosti, obično korisnika ili kupca sustava. One obično slijede jednostavan predložak:

Kao <vrsta korisnika> želim <neki cilj> tako da <neki razlog>

Korisnička priča opisuje vrstu korisnika, što oni žele i zašto te pomaže u stvaranju pojednostavljenog opisa zahtjeva.

Korisničke priče za promatrani meteo sustav dane su u nastavku. Treba naglasiti da korisničke priče ne čini samo tekstualni zapis, već i proces kojim su one nastale, suradnjom u timu i razumijevanjem svih članova tima.

Slijede korisničke priče nastale temeljem ranije prikazanog općeg iskaza korisničkih zahtjeva:

- Administrator se prijavljuje u sustav kako bi mogao koristiti one funkcionalnosti sustava za koje ima ovlasti.
- Administrator može upisivati nove senzore.
- Administrator može mijenjati ranije podatke o senzorima kako bi podaci bili ažurni.
- Administrator može brisati podatke o senzorima kako senzor koji više nije aktivan ne bi bio vidljiv.
- Administrator treba za svaku vrstu senzora unijeti mjernu jedinicu u kojoj senzor mjeri vrijednosti kako bi podaci bili relevantni.
- Administrator može upisivati, mijenjati i brisati podatke u šifarniku vrsti senzora kako bi sve vrste senzora bile dostupne za odabir.
- Administrator može unositi, mijenjati i brisati podatke o meteo stanicama (GPS koordinate, naziv i opis) kako bi podaci o njima bili ažurni.
- Administrator ima mogućnost dodjele GPS koordinata stanici na temelju prethodno utvrđene lokacije uređaja ukoliko sama stanica ne posjeduje GPS kako bi podaci o lokaciji bili ažurni
- Administrator svakoj meteo stanici može dodijeliti sliku, tj. fotografiju meteo stanice kako bi se mogla utvrditi lokacija stanice relativno na okolni prostor..
- Administrator u mobilnoj aplikaciji može direktno pristupiti kameri uređaja i fotografirati meteo stanice kako bi se mogla vidjeti lokacija stanice u odnosu na okolni prostor.
- Administrator za svaki senzor može upisati donju (i/ili gornju) graničnu vrijednost kako bi se upalio alarm nakon što očitanje bude manje ili jednako donjoj, ili veće ili jednako gornjoj vrijednosti.
- Administrator za svaki senzor na meteo stanici upisuje opis senzora kako bi se iz njega mogli vidjeti dodatni podaci o senzoru koji nisu upisani u druga predviđena polja.

- Korisnik putem internetske stranice ili mobilne aplikacije može pristupiti sustavu bez prijave.
- Korisnik preko Google maps prikaza i markera vidljivih na mapi može pregledavati lokacije meteo stranica kako bi mogao pristupiti pojedinoj meteo stanici na temelju njene lokacije i pregledavati njene podatke.
- Korisnik preko Google maps prikaza može pregledati naziv meteo stanice i zadnje očitanje senzora kako bi znao izmjerenu vrijednost.
- Korisnik može pregledavati prethodna očitanja (unutar traženog perioda) nekog senzora u tabličnom ili grafičkom prikazu kako bi mogao pratiti trend.

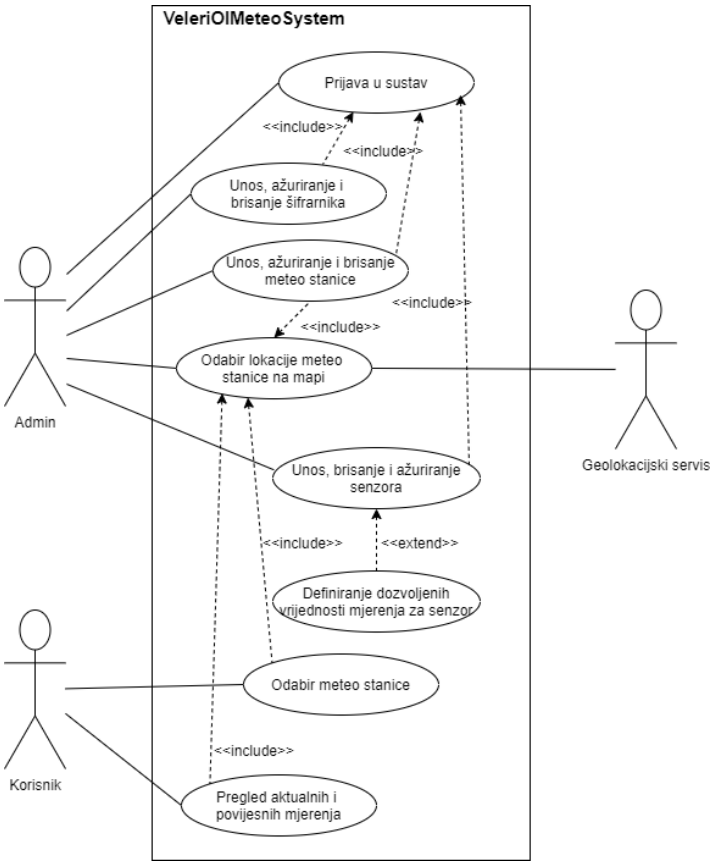
Korisnički zahtjevi mogu se zapisati i u formalnijem obliku, npr. u obliku slučaja uporabe (engl. use case). Slučaj uporabe predstavlja skup scenarija koji imaju isti korisnički cilj, ali različit tijek kontrole. Slučaj uporabe može se zapisati tekstualno, ali i grafički. Grafički prikaz slučaja uporabe, njihove međusobne povezanosti i povezanosti s akterima daje dijagram slučaja uporabe. Međutim, dijagram slučaja uporabe ne daje informacije o njegovoj semantici ga je potrebno dodatno opisati. Za to se koristi tekstualni zapis ili prikaz u obliku tablice.

U tablici 1 je dan prikaz jednog slučaja uporabe promatranog meteo sustava.

Tablica 1. Jedan od slučajeva uporabe METEO sustava

Pregled	
Naslov	**Unos, ažuriranje i brisanje senzora**
Akteri	Administrator
Početno stanje i preduvjeti	Uspješna prijava administratora u sustav Upisana meteo stanica
Tijek aktivnosti	
1. korak: Unos, ažuriranje i brisanje podataka o senzoru 2. korak: Dodjela mjerne jedinice senzoru u kojoj se mjeri vrijednost. 3. korak: Upis donje (i/ili gornje) granične vrijednosti – odstupanje aktivira alarm. 4. korak: Upis opisa senzora kako bi se iz njega mogli vidjeti dodatni podaci o senzoru koji nisu upisani u druga predviđena polja.	
Preduvjet za daljnji tijek:	
Korisnik može pregledavati aktualna ili povijesna mjerenja odabranog senzora.	

Na slici 1 je prikazan dijagram slučaja uporabe za cijeli sustav.

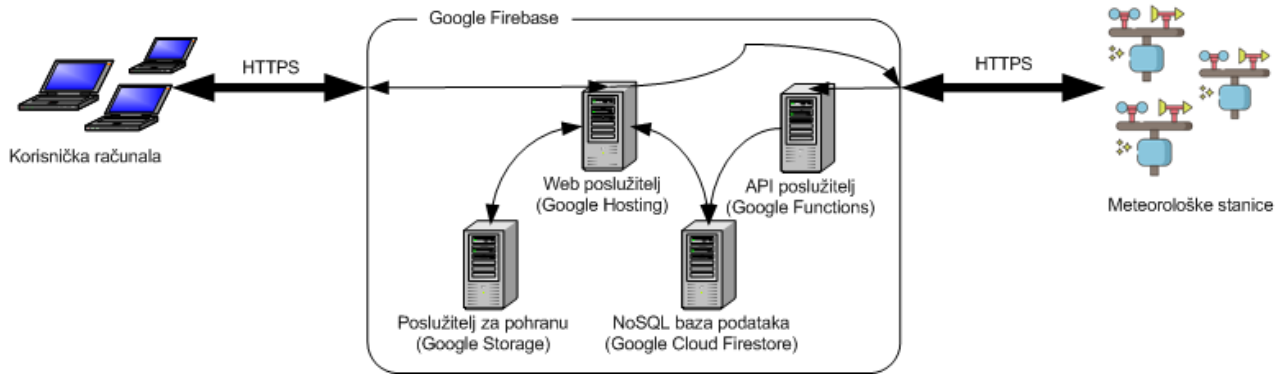


Slika 1. Dijagram slučaja uporabe za cijeli sustav

U sljedećem se koraku za ovaj sustav analiziraju podaci koji će se se pohraniti u bazi podataka.

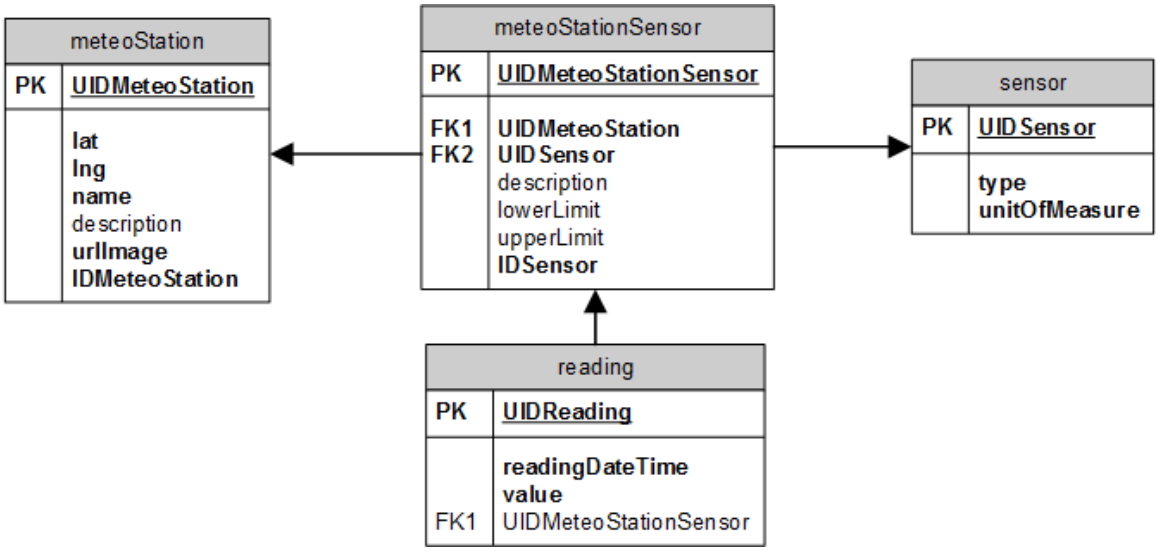
3. OBLIKOVANJE BAZE PODATAKA I APLIKACIJE

Cjelokupan ogledni hardversko-softverski sustav “Veleri-OI Meteo System” koji se koristi u poučavanju prikazan je na slici 2. Web aplikacija izrađena koristeći Quasar framework se nalazi na Web poslužitelju (Google Hosting) unutar Google Firebase platforme. Korisnička računala do aplikacije dolaze koristeći internet preglednik. Podaci se spremaju ili na poslužitelj za pohranu podataka (Google Storage) - tu se spremaju slike - ili u NoSQL bazu podataka (Google Cloud Firestore) - tu se spremaju podaci. Meteorološke stanice svoje podatke spremaju u NoSQL bazu podataka javno dostupnim API (engl. Application Programming Interface) metodama koje se nalaze na API poslužitelju (Google Functions) i koje su izrađene primjenom Node.js. Komunikacija između poslužitelja se odvija Google Firebase API metodama. Budući da NoSQL baza podataka Google Firestore ima mogućnost implementacije reaktivnosti, odnosno osluškivanja promjena na bazi, implementirana je i reaktivnost u web aplikaciji odnosno automatsko ažuriranje prikaza podataka na klijentskom sučelju. U slučaju da korisnik treba dostaviti podatke nekoj meteorološkoj stanici, tada se poziva API metoda te meteorološke stanice preko koje joj se dostavljaju podaci (npr. interval očitanja temeljem kojega se treba aktivirati alarm).



Slika 2. Prikaz arhitekture hardversko softverskog sustava “Veleri-OI Meteo System” (Izvor slike meteorološke stanice: Flaticon.com)

Na temelju ranije izrađene specifikacije korisničkih zahtjeva (korisničkih zahtjeva izrađenih u obliku korisničkih priča, slučajeva uporabe i dijagrama slučajeva uporabe) izrađen je relacijski model. Na slici 3 prikazan je relacijski model na kojem nisu istaknuti tipovi i duljine atributa/stupaca. Uvedeni su jedinstveni identifikatori za svaku od tablica koji će biti primarni ključevi tih tablica. Također, istaknuti su i vanjski ključevi.



Slika 3. Relacijski model sustava

Model nerelacijske dokumentne baze podataka za sustav prikazan je u tablici 2. Model se sastoji od tri kolekcije i jedne podkolekcije s pripadnim poljima.

Tablica 2. Kolekcije i podkolekcije za sustav

Kolekcija	**Polja u dokumentu**

meteostation	UID: {
	UIDMeteoStation
	IDMeteoStation
	name
	description
	geoPoint: {Latitude, Longitude}
	urlImage
	meteostationsensor: [UID: {UIDMeteoStationSensor, UIDMeteoStation, IDSensor, description, lowerLimit, upperLimit, sensor: UID: {UIDSensor, type, unitOfMeasure}}]
	}

reading	UID: {
	UIDReading
	UIDMeteoStation
	UIDMeteoStationSensor
	readinDateTime
	value
	}

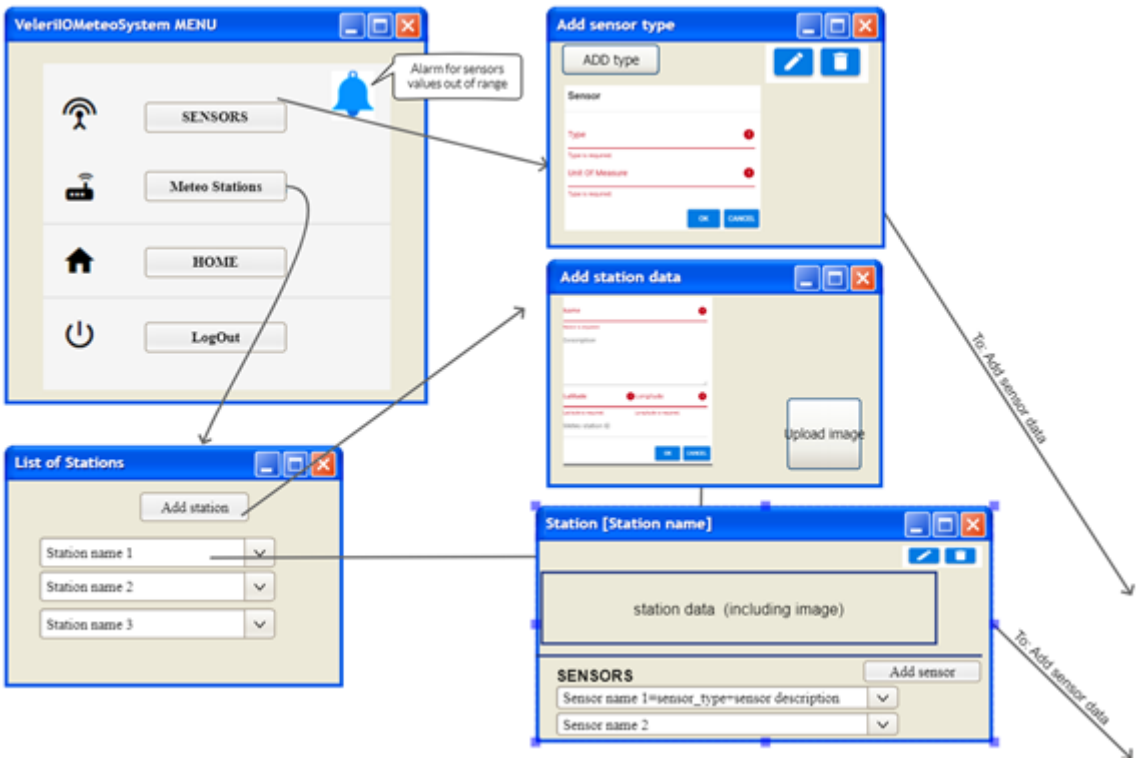
sensor	UID: {
	UIDSensor
	type
	unitOfMeasure
	}

Za podatke koji su u analizi prepoznati kao bitni za budući sustav potrebno je naći mjesto na ekranima putem kojih će korisnik ostvariti interakciju sa sustavom, unoseći, mijenjajući ili čitajući podatke koje su mu od značaja. Korisnik će interakciju sa sustavom ostvariti putem korisničkog sučelja. Međutim, nije samo izgled ekrana bitan za korisnika, treba razmišljati i o načinu na koji će korisnik koristiti sustav, odnosno uz sučelje dizajnirati i iskustvo korištenja (UI/UX). Pritom se mogu koristiti grube skice (engl. wireframe) samo s osnovnim elementima koje daju temelj aplikacije predstavljajući njen glavni sadržaj, strukturu i raspored vizualnih elemenata, a one mogu biti izrađene korištenjem nekog alata za crtanje, ili samo na papiru. Međutim, može se izraditi i detaljnija skica (engl. mockup) koji prikazuje više vizualnih detalja (boje, tipografiji) i osnovne funkcionalnosti. Na mockupu se prikazuju gumbi i stvarni tekst uz uređen raspored i razmak elemenata sučelja, a definirana je i navigacija. Mockup olakšava svim sudionicima, uključujući i klijente, da bolje razumiju budući proizvod i rano otkriju greške koje se mogu ispraviti već u sljedećoj iteraciji. Za još detaljniji prikaz budućeg sustava može se koristiti prototip koji ima ugrađene i određene funkcionalnosti i omogućava interakciju sa sustavom.

Prilikom osmišljavanja sučelja potrebno je razumjeti poziciju korisnika, njegove potrebe i očekivanja od sustava, što je već dokumentirano tijekom specifikacije. Međutim, treba razumjeti i način na koji će korisnici koristiti sustav jer jedino tako možemo kreirati intuitivan i upotrebljiv sustav.

Za izradu mockupa za web aplikaciju koristit će se besplatan alat ***Pencil*** (dostupno na**:** https://pencil.evolus.vn/). Alat između ostalog omogućuje izradu više različitih stranica (opcija *Add page*) za prikaze različitih ekrana aplikacije. Pritom se za izradu pojedinih ekrana nudi veliki broj različitih oblika (kartica *Shapes*) te mogućnost izrade vlastitih kategorija oblika (stencils) odabirom izbornik-Tools-Developer Tools-Stencil Generation. Na taj se način mogu u jednu kategoriju importirati već postojeće slike. Pri kreiranju mockup elemenata koristimo standardne operacije kod crtanja kao što je podešavanje ispune i obruba, karakteristika teksta, grupiranja, poravnanja, raspoređivanja elemenata (front/back) i sl. Također, preporučuje se koristiti osnovnim grafičkim načelima za odabir odgovarajućih kombinacija boja pozadine i elemenata. Potrebno je voditi računa o tome da je mockup vizualni alat koji prikazuje elemente buduće aplikacije i oni moraju biti ugodni oku i intuitivno organizirani. Pokušajmo razmisliti tko će koristiti web aplikaciju i što će budućim korisnicima biti važno. Koji cilj ima budući korisnik aplikacije? Postoje li slične aplikacije ili web stranice? Što njima nedostaje? Što nude kao dobro rješenje? Iskoristimo detalje definirane u prethodnom koraku kad smo analizirali potrebe korisnika i zapisali ih kao korisničke priče i kao dijagrame slučaja uporabe. Kako će neki korisnik koristiti sustav? Osim razmišljanja o izgledu ekrana svakako treba razmisliti o akcijama koje će korisnik pokretati koristeći sustav, a njegovo kretanje kroz aplikaciju može se prikazati i grafički.

Primjer skice nekoliko ekrana web aplikacije dan je na slici 4. Uz prikaz skica ekrana prikazana je i interakcija korisnika u sustavu. Vidi se kako se, počevši u glavnom izborniku, mogu pokretati razne akcije sa sustavom: pregledavati i unositi stanice, pregledavati i unositi senzori, itd.



Slika 4. Primjeri skica web aplikacije

4. IMPLEMENTACIJA APLIKACIJE

Američki poduzetnik i investitor Paul Graham, poznat po svom radu na Lispu i osnivanju startup akceleratora Y Combinator, napisao je u travnju 2007. godine esej pod ciljano šokantnim naslovom “Microsoft je mrtav” (Graham, 2007). Najvažnija teza koju u eseju postavlja i brani je da razvoj weba, specijalno asinkronog JavaScripta i XML-a (engl. Asynchronous JavaScript and XML, kraće AJAX), omogućuje razvoj vrlo moćnih web aplikacija. Takve web aplikacije mogu nuditi razinu funkcionalnosti i ergonomičnosti korištenja koje su ranije bile rezervirane za desktop aplikacije, zbog čega je moguće aplikacije isporučiti korisnicima putem weba i zaobići Microsoftov monopol na desktop. Graham dalje tvrdi kako će tu mogućnost iskoristiti ne samo web mail servisi, već i “sve aplikacije do Photoshopa”, što se i dogodilo 2011. kad je Adobe otvorio Creative Cloud (Anderson, 2011).

Od Grahamove objave eseja “Microsoft je mrtav” prošlo je gotovo 14 godina i u međuvremenu su se webu pridružile i mobilne platforme, a desktop se (jedva) drži i to uglavnom za specijalizirane aplikacije kao što je visokofrekventno trgovanje dionicama (Bolton, 2020). I upravo smo iz tog razloga, kod odabira stoga za implementaciju aplikacije, prije svega zahtijevali višeplatformsko rješenje koje uključuje web i mobilne platforme, a potencijalno i desktop.

Moderne web aplikacije odvajaju prednji dio (engl. frontend) od stražnjeg dijela (engl. backend) korištenjem arhitekture Representational state transfer (REST). REST je arhitekturni stil za distribuirane sustave poput interneta, koji standardizira način rada s različitim resursima te njihovu razmjenu. Prvi ga je spomenuo Roy Fielding u svojoj doktorskoj disertaciji Architectural Styles and the Design of Network-based Software Architecture 2000. Osnova arhitekture REST su resursi od kojih svaki ima svoj identifikator prema kojem ga se može naći i reprezentaciju koja ga opisuje. U toj arhitekturi stražnji dio web aplikacije može se realizirati kao jedan ili više mikroservisa koji odgovaraju na HTTP zahtjeve. Prednji dio web aplikacije šalje te HTTP zahtjeve i prima odgovor na njih te mijenja prikaz na ekranu ovisno o primljenim podacima, a korištenje istih mikroservisa omogućeno je i mobilnim i desktop aplikacijama.

Osim navedenog, REST pristup je također prikladan i za organizaciju i realizaciju upravljačkog softvera (firmware) hardverskog dijela meteo stanice. Specifičnosti upravljanja aktuatorima i senzorima, priprema i prilagodba formata podataka i komunikacija prema vanjskim sustavima se realizira po principu server-klijent interakcije čime hardver sustava postaje apstraktna stvar - “Thing” na Internetu čiji su resursi dostupni putem URL-a.

Drugi zahtjev je bio jednostavnost razvoja. Specijalno, u prednjem dijelu web aplikacije već se koriste HTML, CSS i JavaScript pa smo htjeli okvir za razvoj stražnjeg dijela aplikacije koji također koristi JavaScript kako bismo ukupno smanjili broj programskih jezika koje je potrebno poznavati. Osim toga, jednostavno korištenje u postojećim razvojnim okruženjima kao što su Visual Studio Code i JetBrainsovi alati. (Uočili smo u radu sa studentima kako rado koriste Code i kad to kolegij ne zahtijeva i bez obzira što kao studenti imaju opciju besplatnog korištenja JetBrainsovih alata pa nam je podrška unutar Code-a bila važnija.)

Treći zahtjev je bila popularnost. Na webu je u posljednjih 10-ak godina nastalo i palo u zaborav na stotine biblioteka, okvira i drugih tehnologija. Dijelovi stoga koje biramo nisu morali biti trenutno najpopularnije tehnologije u svom području, ali su morali jasno pokazati da postoji zajednica korisnika i razvijatelja koja će osigurati njihovu održivost u narednim godinama.

Naposlijetku smo se odlučili za sljedeći stog:

- (Nerelacijska) baza podataka: Firestore (dio platforme Google Firebase)

- Backend: Node.js, jezik JavaScript
- Frontend: Vue.js i Quasar, jezici HTML, CSS i JavaScript

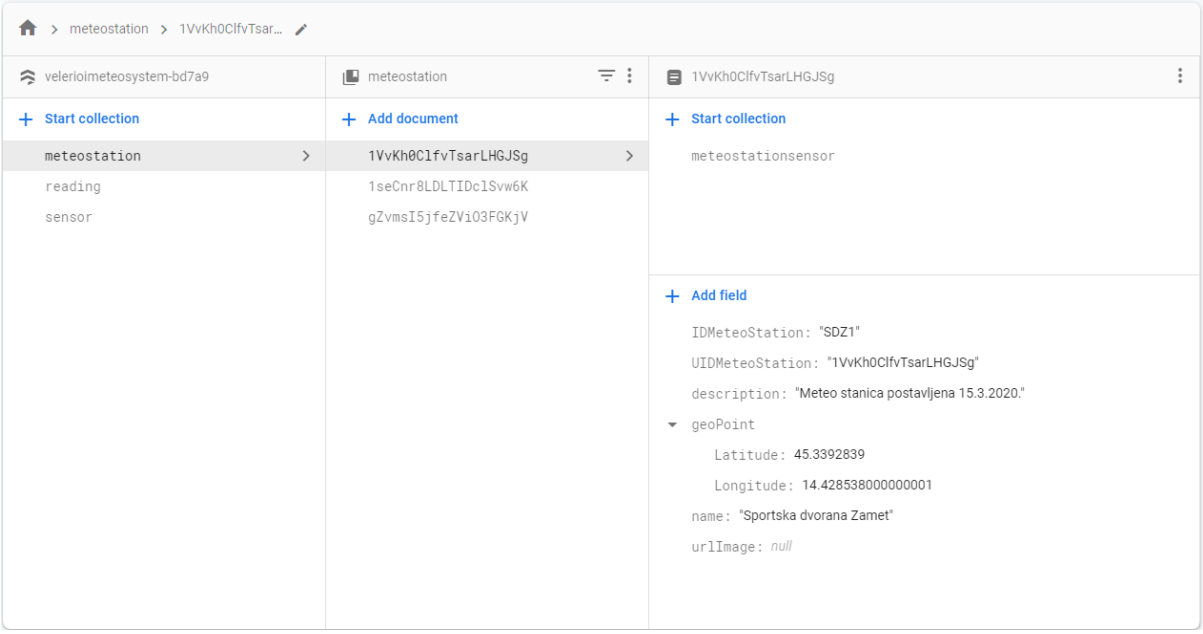
U ovom projektu korištene su usluge koje pruža platforma Google Firebase. Ona omogućuje smještaj klijentskog i poslužiteljskog dijela web aplikacije, kao i NoSQL baze podataka i datoteka (slika). Korišteni su: Cloud Firestore baza za smještaj podataka, Cloud Functions za kreiranje API-ja, te Cloud Storage za pohranu slika. Autentikacija je riješena korištenjem Firebase Authentication usluge koja se može koristiti uz jedinstveni Google račun. Firebase također nudi mogućnost kreiranja više projekata što je olakšalo fazu učenja i testiranja.

Cloud Functions je *serverless* razvojni okvir koji se izvršava na Cloud platformi. Serverless je arhitekturni stil koji developerima smanjuje brigu oko infrastrukture i resursa na kojima se izvršava kod, te se bazira na autoskaliranju resursa prema potrebi. Unutar tog razvojnog okvira developeri definiraju funkcije koje se pozivaju i izvršavaju kad se dogodi očekivani događaj. Ovaj razvojni okvir je korišten za definiranje REST API-ja. U slučaju projekta Veleri-OI Meteo System, senzori su opisani podacima poput identifikatora, lokacije, temperature i predstavljeni su kao resursi na internetu. Kreiran je API koji je omogućio osnovne operacije za spremanje podataka u Firestore bazu i manipuliranje tih podataka sa različitih senzora.

4.1. Firestore

Google Cloud Firestore je fleksibilna i skalabilna dokumentna nerelacijska baza podataka u stvarnom vremenu, koja se nalazi u oblaku. Dovoljno je jednostavna za brzo prototipiranje, a opet dovoljno prilagodljiva i fleksibilna da raste do bilo koje veličine. Dizajnirana je tako da pojednostavljuje razvoj aplikacija uz sinkronizaciju uživo, podršku izvan mreže i ACID transakcije u stotinama dokumenata i zbirki. Integrirana je s Google Cloud Platform (GCP) i Google Firebase, Googleovom platformom za mobilni razvoj (Firebase Guides, 2021).

Prilikom odabira platforme za razvoj nerelacijske baze podataka, u početku su i Google Firebase i MongoDB bili opcija. Obje su nerelacijske baze podataka sa sličnim JSON-ovim modelima podataka i shemama dokumenata, te su obje izgrađene za lakši razvoj aplikacija i horizontalnu skalabilnost (Comparing Firebase vs MongoDB, 2021). Iako MongoDB ima nekoliko prednosti u odnosu na Firebase, naposljetku smo se odlučili za Google Firebase zbog njegovog brzog i kvalitetnog rješenja u oblaku (htjeli smo izbjeći to da studenti moraju postavljati poslužitelj) te usmjerenosti k razvoju mobilnih aplikacija.



Slika 5. Nerelacijska baza podataka Veleri-OI Meteo System

Na temelju modela nerelacijske dokumentne baze podataka za Veleri-OI Meteo System, izrađena je NoSQL baza podataka na Google Firebase platformi (Google Cloud Firestore). Baza podataka prikazana je na slici 5. U bazi podataka, sukladno modelu podataka, nalaze se tri kolekcije i jedna podkolekcija s pripadnim poljima.

4.2. Node.js

Node.js je jedno od popularnijih okruženja (2020 Developer Survey, 2021) za razvoj aplikacija na poslužiteljskoj strani. Osnovne karakteristike su mu:

- omogućuje asinkrono programiranje
- visoko je skalabilan
- može opsluživati velik broj dolaznih zahtjeva
- podržan je od strane brojnih cloud-based hosting providera
- ima preko 600000 modula (Mixed Case Package Names, 2021)

Node.js se često koristi za izradu REST API servisa. Ima više od 76900 zvjezdica i više od 19300 forkova na GitHubu (Node, 2021).

4.3. Vue.js i Quasar

Vue.js je progresivni okvir za razvoj korisničkog sučelja prednjeg dijela web aplikacije. Njegova progresivnost očituje se u mogućnosti postepenog uvođenja korištenja pojedinih dijelova okvira sve do primjene čitavog okvira kod izrade jednostranične web aplikacije. Moguće ga je jednostavno integrirati s ostalim bibliotekama za prikaz pojedinih elemenata sučelja, npr. audiovizualnog sadržaja, geografskih karata ili stupičastih grafova. Zbog toga se može smatrati jednostavnijim za početnike od većih okvira kao što su Angular i React.

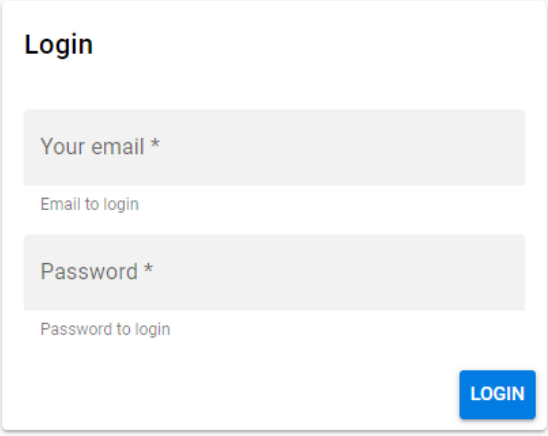
Vue.js ima više od 179000 zvjezdica i više od 28100 forkova na GitHubu (Vue, 2021).

Quasar je okvir temeljen na Vue.js-u koji omogućuje razvoj responzivnih web aplikacija koje se zatim isporučuju korisnicima u raznim oblicima, npr. jednostranične web aplikacije, aplikacije čiji se prikaz stvara na strani web poslužitelja, progresivne web aplikacije, mobilne aplikacije za Android i iOS te višeplatformske desktop aplikacije. Quasar uključuje gotove stilove, rasporede i komponente koje je moguće koristiti u vlastitoj aplikaciji, a moguće je i dodatno proširiti funkcionalnost korištenjem priključaka (engl. plug-ins).

Quasar ima više od 17600 zvjezdica i više od 2100 forkova na GitHubu (Quasar, 2021).

4.4. Primjeri programskog koda

Budući da u članku nije moguće prikazati i opisati detaljno sve dijelove web aplikacije, ovdje će se prikazati samo neki koji ocrtavaju karakteristike korištenja okvira Quasar i Express i Google Firebase API metoda. Kao primjer će poslužiti vizualna komponenta za prijavu korisnika u sustav. Njezino sučelje je prikazano na slici 6.



Slika 6. Sučelje za prijavu korisnika u sustav

Sučelje se sastoji od dvije komponente za unos podataka (nazivi polja su “Your email*” i “Password*”) i jedne komponente za pokretanje akcije prijave (gumb “LOGIN”). U pravilu se komponente u Quasar frameworku sastoje od tri dijela: template, script i style. U template dijelu se specificira kako će komponenta vizualno izgledati (primjena HTML-a), u script dijelu se definira logika ponašanja komponente (primjena Javascripta) i u style dijelu se mogu definirati stilovi koji će se primijeniti na vizualni dio komponente (primjena CSS-a). Template dio komponente se sa skriptnim dijelom povezuje reaktivnim varijablama i pozivima metoda. Reaktivne varijable su obostrano povezane sa script i template dijelom (engl. Two-way data binding) što znači da bez obzira gdje se varijabla promijeni (u script ili template dijelu) ona će automatski biti ažurirana na svim mjestima gdje se koristi. Metode koje se pozivaju u template dijelu su definirane u script dijelu komponente.

Sljedeći programski kod, napisan koristeći Quasar framework, prikazuje dio komponente koji se odnosi na unos elektroničke pošte i gumba za prijavu:

```
1.  <template>
2.  ...
3.     <q-form @submit="onLogin">
4.     ...
5.        <q-input
6.          filled
7.          type="email"
8.          v-model="email"
9.          label="Your email *"
10.         hint="Email to login"
11.         lazy-rules
12.         :rules="[ val => emailPattern.test(val) || 'Please type valid email']"
13.        />
14.    ...
15.        <q-btn
16.          label="Login"
17.          type="submit"
```

```
18.          color="primary"
19.        />
20.     ...
21.     </q-form>
22. ...
23. </template>
24. <script>
25. export default {
26.   ...
27.   data () {
28.     return {
29.       ...
30.       email: null,
31.       ...
32.     }
33.   },
34.   ...
35.   methods: {
36.     onLogin () {
37.       this.$auth.signInWithEmailAndPassword(this.email, this.password)
38.         .then(response => {
39.           ...
40.         })
41.         .catch(error => {
42.           ...
43.         })
44.     }
45.   }
46. }
47. </script>
48.
```

Iz programskog koda se može primijeniti kako je varijabla email deklarirana unutar script dijela komponente (linija 33) i kako je povezana s komponentom za unos podataka u template dijelu (linija 8). Također se može primijetiti kako je akcija koja se izvodi kada se klikne na gumb “LOGIN” (linija 18) zapravo ona koja je definirana na događaju @submit (linija 3), a koji poziva metodu onLogin deklariranu u script dijelu komponente (linija 39).

U metodi onLogin poziva se Google Firebase API signInWithEmailAndPassword kojoj se dostavljaju podaci koje je korisnik unio preko sučelja i koji su se postavili u povezane varijable email i password (linija 37). Jasno je da je ranije u aplikaciji bilo potrebno napraviti povezivanje s Google Firebaseom koristeći korisničke podatke za prijavu na tu platformu. Ova metoda, koristeći Promise (metoda kontrole asinkronih poziva u Javascriptu), ulazi u blok then ako je korisnik uspješno prijavljen, odnosno u blok catch ako nije prijavljen uspješno.

U sljedećem programskom kodu prikazan je primjer API metode izrađene primjenom Node.js koju koriste meteorološke stanice kako bi poslale svoja očitanja:

```
1.  ...
2.  exports.setReading = functions.https.onRequest((request, response) => {
3.    if (request.method === 'POST') {
4.      const data = request.body
5.      ...
6.      const meteoStationCollection = db.collection('meteostation')
7.      return            meteoStationCollection.where('IDMeteoStation',            '==',
8.  data.IDMeteoStation).get()
9.        .then(meteoStations => {
10.         ..,
11.         const meteoStation = meteoStations.docs[0].data()
12.         const                    meteoStationSensorCollection                    =
13. meteoStationCollection.doc(meteoStation.UIDMeteoStation).collection('meteostationse
14. nsor')
15.         return          meteoStationSensorCollection.where('IDSensor',          '==',
16. data.IDSensor).get()
17.           .then(meteoStationSensors => {
18.             ...
19.             const meteoStationSensor = meteoStationSensors.docs[0].data()
20.             const reading = {
21.               UIDReading: null,
22.               UIDMeteoStation: meteoStation.UIDMeteoStation,
23.               UIDMeteoStationSensor: meteoStationSensor.UIDMeteoStationSensor,
```

```
24.                 readingDateTime: new Date(),
25.                 value: data.value
26.               }
27.               return db.collection('reading').add(reading)
28.                 .then((doc) => {
29.                   var docRef = db.collection('reading').doc(doc.id)
30.                   return docRef.update({ UIDReading: doc.id })
31.                     .then(() => {
32.                       return response.status(200).send('OK.')
33.                     })
34.                     .catch((error) => {
35.                       console.error(error)
36.                       return response.status(500).send('Error updating UID in reading
37.  collection.')
38.                     })
39.                 })
40.                 .catch((error) => {
41.                   console.error(error)
42.                   ...
43.                 })
44.             })
45.             .catch(error => {
46.               console.error(error)
47.               ...
48.             })
49.         })
50.         .catch(error => {
51.           console.error(error)
52.           ...
53.         })
54.     }
55.     ...
56.  })
57.
58.
```

Definirana je API metoda “setReading” koja reagira samo na POST poziv (linija 2 i 3). Iz zahtjeva se preuzimaju podaci (linija 4) unutar kojih se nalazi identifikacija meteorološke stanice (IDMeteoStation), identifikacija senzora na toj meteorološkoj stanici (IDSensor) i samo očitanje koje se treba spremiti u bazu podataka (value). Da bi se podaci spremili u bazu podataka, koriste se Google Firestore API metode. Tako se prvo dolazi do reference na kolekciju u kojoj se čuvaju meteorološke stanice (linija 6) - kolekcija meteostation. Iz te kolekcije se dolazi do dokumenta koji predstavlja meteorološku stanicu koja je poslala očitanje (linija 7 do 11) primjenom dostavljenog IDMeteoStation. U tom dokumentu se čuva i podkolekcija dokumenata koja predstavlja sve senzore koji se nalaze na toj meteorološkoj stanici te se dolazi do reference na tu podkolekciju (linija 12 do 14) - podkolekcija meteostationsensor. Iz podkolekcije se dolazi do dokumenta koji predstavlja senzor za koji se dostavlja očitanje (linija 15 do 20) primjenom dostavljenog IDSensor. Potom se priprema dokument reading u koji će se ubaciti podaci o identifikaciji meteorološke stanice, senzora, datuma i vremena očitanja i vrijednosti očitanja (linija 21 do 28) te će se taj dokument spremiti u kolekciju u kojoj se spremaju očitanja s meteorološke stanice (linija 29 do 41) - kolekcija reading.

5. ZAKLJUČAK

U članku je prikazan proces razvoja web aplikacije koja je dio šireg hardversko-softverskog sustava “Veleri-OI Meteo System”, kojeg osim web aplikacije čine i modularne meteo postaje razvijene na osnovu ESP32 razvojnog modula i promjenjivog broja senzora. ESP32 moduli imaju ugrađene WiFi mogućnosti za lako povezivanje na internet, kao i mogućnost programiranja korištenjem Arduino IDEa što ih čini pogodnim za razvoj IoT uređaja u kontekstu edukacije. Meteo postaje putem REST APIja omogućuju dohvat informacija o dostupnim senzorima, konfiguraciju i očitavanje, a također se mogu konfigurirati da na isti način periodički šalju zahtijevane podatke u web aplikaciju. Opisani sustav se koristi kao ogledni primjer u nastavi obrazovnog programa “Veleri-OI IoT School”. Nastava se odvija primjenom metode “learning by doing” čija je karakteristika snažniji angažman studenata u stjecanju iskustva. Stjecanje iskustva im je omogućeno kroz provedbu radnih sastanaka i radnih zadataka te analognom primjenom rješenja oglednog primjera na konkretne studentske projekte.

Obrazovni program “Veleri-OI IoT School” je internacionalni obrazovni program nastao u sklopu trogodišnjeg EU projekta “Razvoj internacionalnog obrazovnog programa Veleri-OI IoT School” čiji je nositelj Veleučilište u Rijeci, a partner je Sveučilište u Rijeci, Odjel za informatiku. Projekt je započeo u drugoj polovici 2018. godine i trenutno prva generacija polaznika pohađa nastavu novog obrazovnog programa. Nastavnici uključeni u nastavu i dalje poboljšavaju nastavne materijale koji će do završetka projekta biti prevedeni na engleski jezik te će biti javno dostupni.

U ovoj zadnjoj godini projekta izradit će se poslovni model i poslovni plan koji će osigurati održivost obrazovnog programa na tržištu. Također će se pripremiti dokumentacija za formalno priznavanje ovog obrazovnog programa od strane

odgovarajućih Agencija. Definirat će se i metoda evaluacije obrazovnog programa koja treba osigurati njegovu kvalitetu i usklađenost s potrebama tržišta rada.

Literatura:

1 2020 Developer Survey (2021), https://insights.stackoverflow.com/survey/2020#technology-other-frameworks-libraries-and-tools-professional-developers3

2 Anderson T. (2011). Adobe announces Creative Cloud, acquires PhoneGap, https://www.theregister.com/2011/10/03/adobe_max_announcements/

3 Bolton D. (2020). Is Desktop Development Dead? Or Still Worth It?, https://insights.dice.com/2020/03/04/desktop-development-dead-still-worth-it/

4 Comparing Firebase vs MongoDB (2021), https://www.mongodb.com/firebase-vs-mongodb

5 Firebase Guides (2021), https://firebase.google.com/docs/guides

6 Graham P. (2007). Microsoft is Dead, http://www.paulgraham.com/microsoft.html

7 Mixed Case Package Names (2021), https://github.com/nice-registry/mixed-case-package-names

8 Node (2021), https://github.com/nodejs/node

9 Quasar (2021), https://github.com/quasarframework/quasar

10 Vue (2021), https://github.com/vuejs/vue

Podaci o autorima:

izv. prof. dr. sc. Alen Jakupović
e-mail: alen.jakupovic@veleri.hr

Alen Jakupović je diplomirao 1997. godine na Pedagoškom fakultetu u Rijeci (profesor matematike i informatike), magistrirao na Fakultetu organizacije i informatike u Varaždinu 2006. godine (magistar znanosti iz područja društvenih znanosti, polja informacijskih znanosti, grane informacijskih sustava i informatologije) i doktorirao na Filozofskom fakultetu u Zagrebu 2010. godine (doktor znanosti iz područja društvenih znanosti, polja informacijskih i komunikacijskih znanosti, grane informacijskih sustava i informatologije).
Od 1998. do 2008. je bio zaposlen u poduzeću za informacijsko i programsko inženjerstvom i u tom periodu je obavljao poslove programera, projektanta, voditelja projekta, predstavnika prodaje, financijskog konzultanta i direktora poduzeća.
Od 2008. godine radi na Veleučilištu u Rijeci gdje je nositelj niza kolegija iz područja programiranja i projektiranja.
Od 2014. do 2018. godine je kao vanjski suradnik bio angažiran u nastavi na Fakultetu za odgojne i obrazovne znanosti, Sveučilišta Jurja Dobrile u Puli.
Od 2012. godine je kao vanjski suradnik angažiran u nastavi doktorskog studija na Odjelu za informatiku Sveučilišta u Rijeci.
Izabran je u nastavno zvanje profesora visoke škole u trajnom zvanju i u znanstveno-nastavno zvanje izvanredni profesor. Sudjelovao je u više stručnih i znanstvenih projekata. Područja znanstvenog i stručnog interesa su mu: razvoj metrika i metoda u izgradnji informacijskih sustava, umjetna inteligencija (formalizmi prikaza znanja), razvoj inteligentnih sustava, oslonjivost informacijskih i poslovnih sustava, te ICT u obrazovanju.
Aktivan je i u gospodarstvu gdje obavlja poslove konzultanta u primjeni informacijske tehnologije i projektanta informacijskih sustava. Suosnivač je startup poduzeća METREAN d.o.o.

izv. prof. dr. sc. Sanja Čandrlić
e-mail: sanjac@inf.uniri.hr

Sanja Čandrlić je izvanredna profesorica na Odjelu za informatiku Sveučilišta u Rijeci. Magistrirala je 2005. te doktorirala 2012. godine u polju informacijskih i komunikacijskih znanosti na Filozofskom fakultetu Sveučilišta u Zagrebu. Nositeljica je nekoliko kolegija na preddiplomskom, diplomskom i doktorskom studiju na Odjelu za informatiku Sveučilišta u Rijeci. Sudjelovala je u više nacionalnih i međunarodnih znanstvenih i stručnih projekata, kao voditeljica i suradnica. Područja njezina znanstvenog interesa su: softversko inženjerstvo i timski razvoj softvera, dizajn korisničkog iskustva i sučelja, analiza poslovnih procesa i modeliranje podataka te primjena IKT-a u obrazovanju.

dr.sc. Sabrina Šuman
e-mail: ssuman@veleri.hr

Rođena u Rijeci, diplomirala na Filozofskom fakultetu, smjer matematika i informatika. Doktorirala na Odjelu informatike Sveučilišta u Rijeci. Zaposlena je kao viši predavač na Veleučilištu u Rijeci gdje radi od 2005. godine te je nositelj kolegija: Informacijski sustavi za potporu upravljanju i odlučivanju, Grafika tekst i multimedija, Upravljanje kvalitetom informacijskih sustava. Sudjelovala je i sudjeluje na nekoliko znanstvenih projekata. Autor je niza recenziranih nastavnih materijala te jednog udžbenika. Objavila je više od 20 znanstvenih i stručnih članaka. Područja interesa: metode umjetne inteligencije, procesiranje prirodnog jezika, rudarenje podataka i teksta, poslovna inteligencija.

doc.dr.sc. Martina Ašenbrener Katić
e-mail: ssuman@veleri.hr

Martina Ašenbrener Katić je docentica na Odjelu za informatiku Sveučilišta u Rijeci. Doktorirala je 2017. godine u polju informacijskih i komunikacijskih znanosti na Odjelu za informatiku Sveučilišta u Rijeci. Sudjelovala je na više znanstvenih i

stručnih projekata. Kao autor ili koautor, objavila je više znanstvenih i stručnih radova. Njeno područje istraživanja uključuje metodologije razvoja informacijskih sustava, informacijske sustave, softversko inženjerstvo, modeliranje podataka, prikaz znanja.

doc.dr.sc. Danijela Jakšić
e-mail: danijela.jaksic@inf.uniri.hr

Danijela Jakšić je docentica na Odjelu za informatiku Sveučilišta u Rijeci. Doktorirala je 2016. godine u polju informacijskih i komunikacijskih znanosti na Odjelu za informatiku Sveučilišta u Rijeci. Njeno područje istraživanja uključuje modeliranje podataka, baze podataka, skladištenje podataka, poslovnu inteligenciju, vizualizaciju podataka, znanost o podacima, informacijsko-komunikacijsku tehnologiju (IKT) u obrazovanju i šire utjecaje IKT-a na društvo.

doc.dr.sc. Lucia Načinović Prskalo
e-mail: lnacinovic@inf.uniri.hr

Lucia Načinović Prskalo docentica je na Odjelu za informatiku Sveučilišta u Rijeci. Diplomirala je informatiku i engleski jezik i književnost na Filozofskom fakultetu u Rijeci 2008., a doktorirala na Poslijediplomskom studiju iz informacijskih i komunikacijskih znanosti na Sveučilištu u Zagrebu 2016. godine. Od 2009. radi na Odjelu za informatiku Sveučilišta u Rijeci te sudjeluje u pripremi i izvedbi nastave na preddiplomskom i diplomskom studiju informatike. Područje njenih znanstvenih istraživanja uključuje računalnu obradu prirodnih jezika, strojno prevođenje, analizu i vizualizaciju podataka, izradu web aplikacija, prozodiju jezika, sintezu govora, računalno potpomognuto učenje jezika.

doc.dr.sc. Vanja Slavuj
e-mail: vslavuj@inf.uniri.hr

Vanja Slavuj je docent na Odjelu za informatiku Sveučilišta u Rijeci gdje je nositelj više kolegija iz područja multimedije i metodike informatike. Doktorirao je 2017. godine u polju informacijskih i komunikacijskih znanosti na Odjelu za informatiku Sveučilišta u Rijeci. Njegovo područje znanstvenog interesa uključuje metodiku poučavanja informatike, poučavanje (jezika) upotrebom informacijske i komunikacijske tehnologije, te inteligentne i prilagodljive obrazovne sustave (jezika).

doc.dr.sc. Vedran Miletić
e-mail: vmiletic@inf.uniri.hr

Vedran Miletić radi kao viši predavač na Odjelu za informatiku Sveučilišta u Rijeci, gdje je nositelj više kolegija iz područja računalnih mreža i dinamičkih web aplikacija. Član je Povjerenstva za superračunalne resurse Sveučilišta u Rijeci. Doktorirao je računarstvo na FER-u u Zagrebu, a poslijedoktorsko usavršavanje u području računalne biokemije proveo je na Heidelberškom institutu za teorijske studije u Heidelbergu, Njemačka. Doprinio je nekolicini slobodnosoftverskih projekata razvojem otvorenog koda, a voditelj je razvoja slobodnog softvera otvorenog koda za pristajanje, visokoprotočni virtualni probir i predviđanje načina vezivanja biomolekula RxDock.

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.

Vlatka Davidović
e-mail: vlatka.davidovic@veleri.hr

Diplomirala je na Filozofskom fakultetu, smjer matematika i informatika. Na Veleučilištu u Rijeci je zaposlena kao viši predavač. Nositeljica je kolegija: Objektno orijentirane tehnologije I i II, Izgradnja objektno orijentiranih aplikacija te Razvoj web aplikacija. Doktorandica je na poslijediplomskom sveučilišnom doktorskom studiju Informatika na Odjelu za informatiku Sveučilišta u Rijeci. Trenutno područje interesa joj je duboko učenje i procesiranje prirodnog jezika.

dr.sc. Davor Širola
e-mail: davor.sirola@veleri.hr

Magistrirao je 2006., a doktorirao 2017. godine u području ekonomije, grana marketinga. Od 2008. godine u ulozi je predavača na Veleučilištu u Rijeci, danas u zvanju profesora visoke škole. U prethodnih 18 godina rada u privatnom sektoru stekao je desetljeće iskustava na menadžerskim pozicijama u Elektromaterijalu d.d., Rijeka i sklonost području marketinga i poduzetništva u praksi i znanosti.

doc.dr.sc. Ozren Rafajac
e-mail: ozren.rafajac@veleri.hr

Magistrirao je 2008., a doktorirao iz 2013. iz znanstvenog područja društvenih znanosti polja ekonomija. Njegovo područje istraživanja usmjereno je na organizacijsku inteligenciju, integralnu komunikaciju, strateški menadžment,

liderstvo i organizacijski razvoj, menadžment ljudskih potencijala, turizam, e-poslovanje i menadžment prodaje. Od 2016. godine do danas, član je uređivačkog odbora stručnog časopisa za liderstvo, menadžment i organizacijski razvoj qLife. Sudjelovao je u više stručnih i znanstvenih projekata.

doc.dr.sc. Miran Pobar
e-mail: mpobar@inf.uniri.hr

Diplomirao je 2007. godine na Tehničkom fakultetu Sveučilišta u Rijeci (dipl. ing elektrotehnike), a 2014. godine na Fakultetu elektronike i računarstva u Zagrebu stječe akademsko zvanje doktora znanosti u području tehničkih znanosti, polje računarstva. Od rujna 2008. godine zaposlen je na Odjelu za informatiku Sveučilišta u Rijeci kao asistent na kolegijima iz područja računarskih i informacijskih znanosti. U suradničko zvanje višeg asistenta iz znanstvenog područja društvenih znanosti izabran je 2014., a 2017. u znanstveno zvanje znanstvenog suradnika. Trenutno je zaposlen na radnom mjestu docenta. Područje njegovog znanstveno-istraživačkog rada je umjetna inteligencija, raspoznavanje uzoraka i računalni vid.

Damir Malnar
e-mail: dmalnar@veleri.hr

Diplomirao je elektrotehniku, smjer automatizacija postrojenja i mehatronika pri Tehničkom fakultetu Sveučilišta u Rijeci 2008. godine. Nakon diplome radi kao razvojni inženjer sklopovskih i programskih rješenja uređaja za mjerenje kvalitete električne energije, a krajem 2009. postaje znanstveni novak i asistent na Tehničkom fakultetu u Rijeci gdje sudjeluje u nastavi više kolegija iz područja elektrotehnike i računarstva kao i na nekoliko znanstvenih projekata. Trenutno je doktorand poslijediplomskog studija Elektrotehnika pri Tehničkom fakultetu, a zaposlen je kao predavač na preddiplomskom stručnom studiju Telematika Veleučilišta u Rijeci. Njegovo područje interesa je digitalna obrada signala, optimizacijske metode i ugradbeni sustavi.

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

Odjel za informatiku
Sveučilište u Rijeci
Radmile Matejčić 2, 51000 Rijeka
tel: +385 51 584 700
fax: +385 51 584 749
web: www.inf.uniri.hr

USPOREDBA UDALJENOSTI SQL DIJALEKTA I MOGUĆNOSTI RDBMS-A OD SQL NORME (ISO/IEC 9075:2016)

Sanja Žmarić, dr.sc. Marin Kaluža

SAŽETAK:

Ovim radom napravljena je i opisana usporedba udaljenosti SQL dijalekta u tri relacijske baze podataka u odnosu na propisanu i trenutno aktualnu SQL normu (ISO/IEC 9075:2016). RDBMS-ovi analizirani u radu su MySQL, Microsoft SQL Server, te Oracle. Ovi RDBMS-ovi odabrani su iz razloga što su trenutno prilično rasprostranjeni i često korišteni RDBMS-ovi, te i najčešće spominjani na raznim Internet portalima. Kod analize i usporedbe fokus je postavljen na SQL dijalekt koji se koristi u navedenim bazama podataka, te provjeru koliko se SQL dijalekt podudara ili odstupa od propisane norme. Izrada baze podataka je važan korak u procesu razvoja aplikacija te je važno moći saznati koji od svih, na tržištu dostupnih, RBDMS-ova pruža što sličniji dijalekt dijalektu SQL norme. Iako postoje razne sličnosti između promatranih RDBMS-ova, oni nisu isti te tako svaki od njih pruža različite mogućnosti svojstvene samo sebi. Zbog toga je napravljena analiza odabranih RDBMS-ova i definiran je postupak kojim se može saznati koliko je RDBMS usklađen s normom, te u slučajima nesuklada postoji li zamjena za promatranu funkcionalnost.

ABSTRACT:

This paper describes the comparison of the distance of the SQL dialect in three relational databases in relation to the prescribed and current SQL standard (ISO / IEC 9075: 2016). The RDBMS analyzed in the paper are MySQL, Microsoft SQL Server, and Oracle. These RDBMSs were chosen because they are currently quite widespread and frequently used RDBMSs, and are most often mentioned on various Internet portals. In the analysis and comparison, the focus is placed on the SQL dialect used in the specified databases and in checking how much the SQL dialect matches or deviates from the prescribed standard. Creating a database is an important step in the application development process and it is important to be able to find out which of all the available RBDMSs on the market provides the most similar dialect to the SQL standard dialect. Although there are various similarities between the observed RDBMSs, they are not the same and each of them provides different features unique to themselves. Therefore, an analysis of selected RDBMS was made and a procedure was defined to find out how much the RDBMS complies with the standard and in cases of non-compliance whether there is a substitute for the observed functionality.

1. UVOD

Cilj rada je istražiti i usporediti mogućnosti i funkcije dostupne u tri besplatna alata za razvoj i upravljanje relacijskim bazama podataka te zaključiti koji se od analiziranih alata pokazao kao najbolji. Svrha usporedbe ovih alata se odnosi na provjeru udaljenosti od SQL dijalekta i mogućnosti RDBMS-a od SQL norme (ISO/IEC 9075:2016) što će dokazati koji RDBMS je po svim ispitanim parametrima najadekvatniji za upravljanje i razvoj baza podataka. Potrebno je pronaći odgovarajući algoritam za provjeru SQL konstrukata od SQL norme koji se može primijeniti na bilo kojem DMBS-u, a ne samo na odabranima.

RDBMS-ovi koji će se analizirati ovim radom su alati koji se najčešće pojavljuju na vrhu različitih popisa kao jedni od najboljih za rad s bazama podataka, a to su MySQL, Microsoft SQL Server i Oracle. Postupak opisan ovim radom moguće je provesti na bilo kojem RDBMS-u, na način da se kao i u radu uspoređuju upiti koji se koriste za rad s bazom podataka.

Usporedba će se raditi s više strana, počevši od tipova podataka koje određeni RDBMS nudi kao i analiza svih SQL, T-SQL i PL/SQL konstrukata baze podataka koji su definirani SQL normom.

2. PRETHODNA ISTRAŽIVANJA

SQL (strukturirani jezik upita) standardiziran je od međunarodnih standardnih tijela kao što su ISO i ANSI. Korištenjem standardnog SQL-a lakše je premještati aplikacije između različitih sustava baza podataka bez potrebe za prepisivanjem značajne količine koda. Često se dešava da korisnici koriste određeni RDBMS-u jer njihov SQL kod nije u skladu sa SQL standardom.

Prva SQL norma izašla je 1987. godine pod nazivom ISO 9075:1987 te je od tada nadograđivana 8 puta (1989., 1992., 1999., 2003., 2006., 2008., 2011. i 2016. godine). Većina promjena vezanih uz normu nastaje upravo zbog promjena i razvoja tehnologija što vodi i unaprjeđenju pravila za korištenje SQL-a za rad i razvoj baza podataka.[‡‡]

[‡‡] https://en.wikipedia.org/wiki/ISO/IEC_9075 (21.12.2020)

Jedan od alata za provjeru SQL upita u bazi podataka je Mimer SQL. Mimer SQL Validator nudi se kao besplatni mrežni alat kako bi se programerima baza podataka pomoglo u pisanju SQL-a koji odgovara SQL standardu. [§§]

3. POSTUPAK

Postupak usporedbe od SQL norme će se vršiti u tri dijela, a to su SQL dijalekt i SQL konstrukti i moguće alternative konstrukata te zadnji dio koji će se odnositi na tipove podataka u određenom RDBMS-u.

Izraz konstrukt predstavlja tekstualni niz SQL izraza iz popisa stavki podataka i doslovnih vrijednosti. Direktiva koja specificira oblikovanje može po želji prethoditi svakoj stavci podataka ili argumentu doslovne vrijednosti.

3.1. Opis algoritma uspoređivanja udaljenosti od norme

Algoritam za provjeru će se vršiti tako što će se provjeravati SQL dijalekt određenog RDBMS-a te zapravo na koji način se u određenom RDBMS-u pišu upiti za razvoj baze podataka. SQL dijalekt je definiran SQL normom te će se razlika utvrđivati po ključnim riječima koje se pišu u upitima te da li su iste propisane ili ne propisane normom. podataka u potpunosti slaže sa normom što znači da koristi ključne riječi koje su propisane.

Ocjene za odstupanje od norme prikazane su u sljedećoj tablici:

Tablica 1. Prikaz ocjena

OCJENA	OPIS	PRIKAZ
1	Ni malo se ne podudara sa normom	
2	Djelomično	
3	U potpunosti se podudara s normom	

Izvor: Autorica

SQL konstrukti koji će se analizirati za udaljenost od SQL norme su:

3.1.1. Data Definition Statements (DDL)

ALTER ,CREATE, DROP, RENAME, TRUNCATE

3.1.2. Data Manipulation Statements (DML)

CALL, DELETE, HANDLER, SELECT, UPDATE

3.1.3. Transactional and Locking Statements

START TRANSACTION, COMMIT, ROLLBACK, SET TRANSACTION, SAVEPOINT

3.1.4. Tipovi podataka

STRING – String je vrsta podataka koja se za predstavljanje teksta. Sastoji se od skupa znakova koji također mogu sadržavati razmake i brojeve. String tipovi podataka promatrani u ovom radu su: CHAR, VARCHAR, BINARY, VARBINARY, BLOB

NUMERIC – Numerički tipovi podataka su brojevi pohranjeni u stupcima baze podataka. Ti se tipovi podataka obično grupiraju prema točnim numeričkim vrstama ili vrijednostima. Numeric tipovi podataka promatrani u ovom radu su: BIT, BOOLEAN, SMALLINT, INT, INTEGER, BIGINT, FLOAT, DOUBLE, DECIMAL

DATE AND TIME - Tip DATE koristi se za vrijednosti s datumskim dijelom, a TIME se koristi za vrijednosti koje se sastoje od vremena u satima, minutama, sekundama, neobaveznih djelića sekunde i neobavezne vremenske zone. Date and time tipovi podataka promatrani u ovom radu su: DATE, TIMESTAMP, TIME, YEAR[***]

3.2. Provedba postupka na RDBMS-u

Provedba postupka vršiti će se kroz pregled upita tj. da li je upit u određenom RDBMS-u napisan onako kako to opisuje SQL standard te su tako kod provedbe postupka bitni SQL konstrukti te SQL dijalekt. Svaki od promatranih RDBMS-ova biti će ocijenjen za svaki od promatranih konstrukata. Ocjene će se na kraju prebrojati te će se dobiti konačna ocjena koja govori koliko određeni DMBS odstupa ili ne odstupa od norme.

Postupak koji će se provoditi na odabranim bazama podataka može se na isti način provoditi na bilo kojoj bazi podataka. Postupak je univerzalan te jedino što je bitno za provedbu su upiti u bazu podataka koji se koriste za upravljanje i manipulaciju podacima, a svojstveni su za svaki RDBMS. Upiti se trebaju provesti kroz tablice navedene u postupku te bi se svakom trebala dodijeliti ocjena za podudaranje tj. odstupanje od norme. U konačnici, po dodijeljenim ocjenama je vidljivo da li RDBMS jako odstupa od norme ili ne što pomaže kod odabira baze podataka.

[§§] https://developer.mimer.com/features/sql-standard/ (21.12.2020)
[***] https://www.w3schools.com/sql/sql_datatypes.asp (08.01.2021)

Tipovi podataka

Tablica 2. BINARY tip podatka

BINARY			
RDBMS	Podržava li RDBMS?	Zamjena	Ocjena
MySQL	Da	Nije potrebna zamjena.	3
Microsoft SQL Server	Da	Nije potrebna zamjena.	3
ORACLE	Ne	BINARY_INTEGER	2

Izvor: Autorica

Tablica 3. DOUBLE tip podatka

DOUBLE			
RDBMS	Podržava li RDBMS?	Zamjena	Ocjena
MySQL	Da	Nije potrebna zamjena.	3
Microsoft SQL Server	Ne	FLOAT/REAL	2
ORACLE	Da	Nije potrebna zamjena.	3

Izvor: Autorica

Tablica 4. DATE tip podatka

DATE			
RDBMS	Podržava li RDBMS?	Zamjena	Ocjena
MySQL	Da	Nije potrebna zamjena.	3
Microsoft SQL Server	Da	Nije potrebna zamjena.	3
ORACLE	Da	Nije potrebna zamjena.	3

Izvor: Autorica

DDL konstrukti

Tablica 5. ALTER TABLE - ADD COLUMN

ALTER TABLE statement: ADD COLUMN clause			
RDBMS	Upit	Ocjena	Opis
MySQL	ALTER TABLE table_name ADD COLUMN column_name VARCHAR(15);	3	Upit se u potpunosti podudara sa normom.
Microsoft SQL Server	ALTER TABLE table_name ADD column_name VARCHAR(15);	2	Upit se razlikuje po ključnoj riječi „COLUMN“. Pisanje upita bez ključne riječi se može samo ubrzati. Funkcionalnost upita je ista.
ORACLE	ALTER TABLE table_name ADD column_name VARCHAR(15);	2	Upit se razlikuje po ključnoj riječi „COLUMN“. Pisanje upita bez ključne riječi se može samo ubrzati. Funkcionalnost upita je ista.

Izvor: Autorica

Tablica 6. ALTER TABLE - ALTER COLUMN

ALTER TABLE statement: ALTER COLUMN clause			
RDBMS	Upit	Ocjena	Opis
MySQL	ALTER TABLE table_name MODIFY COLUMN column_name datatype;	2	Upit se razlikuje po ključnoj riječi MODIFY. Funkcionalnost upita je ista.
Microsoft SQL Server	ALTER TABLE table_name ALTER COLUMN column_name datatype;	3	Upit se u potpunosti podudara sa normom.
ORACLE	ALTER TABLE table_name MODIFY column_name datatype;	2	Upit se razlikuje po ključnoj riječi MODIFY te po ključnoj riječi COLUMN koja se ne piše. Pisanje upita bez ključne riječi se može samo ubrzati. Funkcionalnost upita je ista.

Izvor: Autorica

Tablica 7. ALTER TABLE - DROP CONSTRAINT

ALTER TABLE statement: DROP CONSTRAINT clause			
RDBMS	Upit	Ocjena	Opis
Unique constraint			
MySQL	ALTER TABLE table_name DROP INDEX index_name;	1	Upit se ne podudara s normom. Umjesto DROP CONSTRAINT navodi se naziv indexa tablice na kojoj se izvršava ALTER naredba. Upit svejedno funkcionira bez greške.
Microsoft SQL Server	ALTER TABLE table_name DROP CONSTRAINT constraint_name;	3	Upit se u potpunosti podudara sa normom.
ORACLE	ALTER TABLE table_name DROP CONSTRAINT constraint_name;	3	Upit se u potpunosti podudara sa normom.

Izvor: Autorica

Tablica 8. ALTER column data type

ALTER column data type			
RDBMS	Upit	Ocjena	Opis
MySQL	ALTER TABLE table_name MODIFY column_name new_data_type(size);	3	Upit se u potpunosti podudara sa normom.
Microsoft SQL Server	ALTER TABLE table_name ALTER COLUMN column_name new_data_type(size);	3	Upit se u potpunosti podudara sa normom.
ORACLE	ALTER TABLE table_name MODIFY column_name new_data_type(size);	3	Upit se u potpunosti podudara sa normom.

Izvor: Autorica

Tablica 9. ALTER transform statement

ALTER transform statement			
RDBMS		Ocjena	Opis
MySQL	/	1	Promatrani RDBMS nema mogućnost izvršavanja naredbe ALTER TRANSFORM. Umjesto promatrane naredbe sadrži druge naredbe poput CONVERT-a pomoću koje se radi konverzija iz jednog tipa podatka u drugi.
Microsoft SQL Server	/	1	Promatrani RDBMS nema mogućnost izvršavanja naredbe ALTER TRANSFORM. Umjesto promatrane naredbe sadrži operator PIVOT koji jedinstvene vrijednosti u jednom stupcu pretvara u više stupaca na outputu i izvodi agregiranje svih preostalih vrijednosti stupaca.
ORACLE	/	1	Promatrani RDBMS nema mogućnost izvršavanja naredbe ALTER TRANSFORM. Umjesto ove naredbe sadrži naredbu CONVERT koja radi konverziju iz jednog tipa podatka u drugi.

Izvor: Autorica

Tablica 10. DROP TABLE – RESTRICT clause

DROP TABLE statement: RESTRICT clause			
RDBMS	Upit	Ocjena	Opis
MySQL	DROP TABLE [IF EXISTS] table_name [RESTRICT | CASCADE];	3	Upit se u potpunosti podudara sa normom.
Microsoft SQL Server	DROP TABLE [IF EXISTS] { database_name.schema_name. Table_name [,...n];	1	Upit se razlikuje po dijelu koji se odnosi na RESTRICT te taj dio nema.
ORACLE	DROP TABLE schema_name.table_name [CASCADE CONSTRAINTS | PURGE];	1	Upit se razlikuje po dijelu koji se odnosi na RESTRICT te umjesto toga PL/SQL koristi naredbe CASCADE CONSTRAINTS ili PURGE.

Izvor: Autorica

Tablica 11. TRUNCATE TABLE - identity column restart option

TRUNCATE TABLE: identity column restart option			
RDBMS	Upit	Ocjena	Opis
MySQL	/	1	Promatrani RDBMS ne podržava opcije CONTINUE IDENTITY and RESTART IDENTITY. Brojač (AUTO_INCREMENT) vraća se na nulu.
Microsoft SQL Server	DBCC CHECKIDENT (table_name [, { NORESEED | { RESEED [, new_reseed_value] } }])	1	Promatrani RDBMS ne podržava opcije koje se odnose na TRUNCATE TABLE i AUTO_INCREMENT. Umjesto toga koristi drugačiju naredbu.
ORACLE	ALTER TABLE tbl_name MODIFY (ID GENERATED BY DEFAULT ON NULL AS IDENTITY START WITH LIMIT VALUE);	1	Promatrani RDBMS ne podržava AUTO_INCREMENT već umjesto toga koristi sekvence.

Izvor: Autorica

DML konstrukti

Tablica 12. CALL statement

CALL statement			
RDBMS	Upit	Ocjena	Opis
MySQL	CALL productpricing();	3	Upit se u potpunosti podudara s normom.
Microsoft SQL Server	{CALL productpricing}	2	Upit se razlikuje po dijalektu od norme, ali ima iste funkcionalnosti.
ORACLE	CALL productpricing();	3	Upit se u potpunosti podudara s normom.

Izvor: Autorica

Tablica 13. HANDLER

HANDLER			
RDBMS	Upit	Ocjena	Opis
MySQL	HANDLER tbl_name OPEN [[AS] alias];	3	Upit se u potpunosti podudara s normom.
Microsoft SQL Server	BEGIN Try <statements> END Try BEGIN Catch <Statements> END Catch	1	Promatrani RDBMS ne podržava HANDLER opcije. Slično HANDLER-u u T/SQL-u je TRY…CATCH opcija.

ORACLE	BEGIN
	EXCEPTION
	WHEN e THEN
	-- exception_handler1
	WHEN OTHERS THEN
	--other_exception_handler
	END;
		1	Promatrani RDBMS ne podržava HANDLER opcije. Slično HANDLER-u u PL/SQL-u je EXECPTION opcija.

Izvor: Autorica

Tablica 14. INSERT - DEFAULT VALUES

INSERT statement: DEFAULT VALUES clause			
RDBMS	Upit	Ocjena	Opis
MySQL	/	1	Promatrani RDBMS ne podržava umetanje default vrijednosti u tablicu te nema zamjenu za takav upit.
Microsoft SQL Server	INSERT INTO tbl_name DEFAULT VALUES;	3	Upit se u potpunosti podudara s normom.
ORACLE	/	1	Promatrani RDBMS ne podržava umetanje default vrijednosti u tablicu te nema zamjenu za takav upit.

Izvor: Autorica

Transakcije

Tablica 15. START TRANSACTION

START TRANSACTION statement			
RDBMS	Upit	Ocjena	Opis
MySQL	START TRANSACTION;	3	Upit se u potpunosti podudara s normom.
Microsoft SQL Server	BEGIN TRANSACTION;	2	Upit se razlikuje po ključnoj riječi BEGIN. Funkcionalnost upita je ista.
ORACLE	/	1	Promatrani RDBMS nema mogućnost puštanja naredbe START TRANSACTION. Transakcija kod PL/SQL-a započinje sa prvim SQL upitom koji se pušta nakon spajanja na bazu.

Izvor: Autorica

4. DISKUSIJA

Fokus rada stavljen je na usporedbu udaljenosti SQL dijalekta RDBMS-a od SQL norme te se kroz analizu da zaključiti da svaki od promatranih RDBMS-ova prati normu u većini slučajeva. Za svaki konstrukt provjeren je način pisanja upita odnosno SQL dijalekt u određenom RDBMS-u te je tako provedena usporedba i utvrđivanje udaljenosti od norme. Udaljenosti od norme su se ocjenjivale sa 3 ocjene, a to su potpuno podudaranje s normom, djelomično podudaranje s normom i kompletno odstupanje od norme. Ovim ocjenama je ocijenjen svaki od RDBMS-a za određeni upit po određenim konstruktima. Razlike i odstupanja su se svela na nedostajanje ključne riječi što je bilo ocijenjeno sa djelomičnim odstupanjem, a ocjena za kompletno odstupanje je dodijeljena ukoliko RDBMS nema implementiranu zamjenu za određeni upit ili ukoliko se upit u potpunosti razlikuje sa propisanom normom. Potpuno podudaranje s normom se dodijelilo ukoliko se upit u RDBMS-u u potpunosti podudara s normom i koristi sve propisane ključne riječi.

U početnom dijelu postupka usporedbe se odnose na tipove podataka koji su propisani normom te je većina promatranih tipova podataka dostupna u sva tri RDBMS-a. Djelomična odstupanja su vidljiva kroz par tipova podataka, ali za svaki postoji zamjena te se ne gube funckionalnosti određenog tipa podatka.

Sljedeći dio postupka koji odnosi se na DDL konstrukte, RDBMS-ovi se većinom podudaraju s normom ili se djelomično podudaraju. Iako po većini točaka RDBMS-ovi prate normu, ima i točaka u kojima uopće nemaju implementirano rješenje ili zamjenu za funkionalnost koja je propisana normom. Jedan od primjera odstupanja u kojem RDBMS ne zahtjeva ključnu riječ je jednostavni upit koji se odnosi na ALTER TABLE ADD COLUMN gdje je normom riječ COLUMN propisana kao ključna riječ, a u T-SQL-u i PL/SQL-u upit se može napisati bez ključne riječi uz istu funkcionalnost. Najviše odstupanja od norme kod DDL konstrukata javlja se kod upita koji se odnose na domenu, transformaciju i autoinkrement pošto niti jedan od RDBMS-ova nema implementirane te specifične mogućnosti. Manje razlike su se javile i kod DROP CONSTRAINT naredbe gdje MySQL ne koristi ključnu riječ CONSTRAINT već za brisanje index-a koristi ključnu riječ INDEX, za primarni ključ PRIMARY KEY i za vanjski ključ FOREIGN KEY.

U sljedećem dijelu provedena je usporedba DML konstrukata te se skoro po svim promatranim točkama pisanje upita tj. SQL dijalekt podudara s normom. Najveća odstupanja u ovom dijelu rada se javlja kod HANDLER opcija gdje dva od tri promatrana RDBMS-a nemaju opciju korištenja HANDLERA već pružaju drugačije mogućnosti sličnih funkcionalnosti. Odstupanja su također vidljiva kod INSERT-a zadanih vrijednosti te tu mogućnost ima implementiran samo jedan promatrani RDBMS. U ostalim ispitivanjima RDBMS-ovi se poklapaju s normom ili se pojavljuje minimalna razlika u dijalektu.

Zadnji dio provjere se odnosi na upravljanje transkacijama gdje su se odstupanja pokazala samo kod pozivanja transakcija gdje jedan od RDBMS-a koristi drugačiju ključnu riječ, a drugi uopće nema mogućnost pozivanja transakcija pošto transakcije započinju sa prvim SQL upitom koji se pušta nakon spajanja na bazu. Po ostalim točkama svi promatrani RDBMS-ovi se u potpunosti podudaraju s normom.

5. ZAKLJUČAK

Uspredbom udaljenosti SQL dijalekta od SQL norme može se zaključiti da svaki od promatranih RDBMS-ova ima svoje prednosti i nedostatke. U većini slučajeva funkcionalnosti koje RDBMS nudi se u potpunosti slažu sa normom što je dobar znak da se prati i implementira ono što je propisano normom. Vidljivo je i kako svaki RDBMS ima svoje specifične funkcionalnosti koje zamjenjuju ono što je propisano normom pa se tako ne gubi na mogućnostima i funckionalnostima upravljanja podacima u bazi podataka. RDBMS koji ima najviše podudaranja s normom je MySQL koji se u samo 10 promatranih točaka ne podudara s normom. Iza MySQL-a slijedi Microsoft SQL server koji se po 14 točaka ne podudara s normom dok je zadnji ORACLE koji se u 16 točaka ne podudara s propisanom normom. Iako se RDBMS-ovi ne podudaraju s normom u određenim točkama, svaki od njih ima implementiran drugačiji način za postizanje istih mogućnosti što ne znači da u budućnosti neće uskladiti svoj SQL dijalekt sa normom. U promatranim slučajevim, nedostatak ključne riječi može samo ubrzati pisanje upita što ne mora nužno biti nedostatak već prednost.

Provedbom ove analize vidljivo je kako svaki od RDBMS-a većinom prati normu, ali također sadrže i funkcionalnosti koje još nisu ni propisane normom te se zapravo niti ne mogu uskladiti s istom. Svaki od RDBMS-ova nudi jedinstvene funckionalnosti koje su specifične samo za njih te se kod biranja baze podataka treba fokusirati na ono što je korisniku potrebno. Postupak bi se dalje mogao razvijati kada bi se promatrali i ostali propisani konstrukti što bi još dodatno povećalo usporedbu i mogućnosti promatranih RDBMS kao i ostalih RDBMS-a koji nisu spomenuti ovim radom. Postupak

bi se također mogao još detaljnije razviti ukoliko bi se usporedba vršila na nerelacijskim bazama podataka i to konkretno na Column-family nerelacijskim bazama podataka.

Literatura:

1 https://en.wikipedia.org/wiki/ISO/IEC_9075 (21.12.2020)

2 https://developer.mimer.com/features/sql-standard/ (21.12.2020)

3 shorturl.at/kquvB (03.01.2021)

4 https://www.tutorialspoint.com/sql/sql-indexes.htm (03.01.2021)

5 https://www.geeksforgeeks.org/sql-ddl-dql-dml-dcl-tcl-commands/ (05.01.2021)

6 https://www.tutorialspoint.com/dbms/dbms_transaction.htm (07.01.2021)

7 https://www.w3schools.com/sql/sql_datatypes.asp (08.01.2021)

8 https://dev.mysql.com/doc/ (11.01.2021. – 29.01.2021)

9 https://docs.microsoft.com/en-us/sql/?view=sql-server-ver15 (11.01.2021. – 29.01.2021)

10 https://docs.oracle.com/en/ (11.01.2021. – 29.01.2021)

Podaci o autorima:

Sanja Žmarić, bacc. ing. inf.

Studentica 5. godine stručnog spec. studija poslovne informatike na Veleučilištu u Rijeci s dvogodišnjim iskustvom rada s Oracle bazama podataka i izgradnji informacijskih poslovnih sustava za općine i gradove diljem Hrvatske.

dr. sc. Marin Kaluža, profesor visoke škole

e-mail: mkaluza@veleri.hr

Marin Kaluža je profesor 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
web: www.veleri.hr

USPOREDBA STATE MANAGEMENT-A NA JAVASCRIPT FRONT-END RAZVOJNIM OKVIRIMA

Morena Pavlović, dr.sc. Marin Kaluža

SAŽETAK:

U razvoju aplikacija sa front-end razvojnim okvirima postoji potreba za upravljanjem stanjima raznih dijelova korisničkog sučelja. Često stanje jednog elementa korisničkog sučelja ovisi o stanju drugog ili više drugih elemenata. U radu su obrađene knjižnice stanja: Vuex kao knjižnica za upravljanje stanjima za Vue.js razvojni okvir, Ngrx za Angular razvojni okvir, te Redux knjižnica koja se može koristiti na većini Javascript razvojnih okvira, a primarno je nastala za React. Služeći se praktičnom primjenom svih navedenih knjižnica u izradi jednostavnih elemenata korisničkog sučelja koji imaju funkcije unosa, ispisa i brisanja, prikazana je usporedba state managementa. Prikazani su uzorci koda za definiranje store-a, mutacija/reduktora i akcija triju navedenih knjižnica. Izvršena je analiza mogućnosti i pokazane su njihove međusobne sličnosti i različitosti.

ABSTRACT:

In the application development with front-end development frameworks, there is a need to manage the states of various parts of the user interface. Often the state of one UI (user interface) depends on the state of another or more other elements. The paper deals with state libraries: Vuex as a state management library for Vue.js development framework, Ngrx for Angular development framework and Redux library, which can be used for most of Javascript development frameworks, primarily made for React. Using a practical example of all the above libraries in creating simple UI elements that have input, print and delete functions, a comparison of state management is presented. Sample codes for defining store, mutations/reductors and actions of the three mentioned libraries are presented. An analysis of the functionalities was performed and their mutual similarities and differences were shown.

1. UVOD

Prilikom izgradnje web korisničkih sučelja (UI – user inteface) sve su popularniji okviri (framework) koji se mogu definirati kao softverski alati za izgradnju softverskih programa koji će se pokretati na Internetu. Riječ je o skupu klasa i sučelja koji definiraju elemente, strukturu i ponašanje UI-a, koje je svakim danom zastupljenije u svijetu izrade web rješenja pa je tako predmet ovog rada usporedba upravljanja stanjima (state management) Angular, React, Vue.js framework-a koji se nalaze na listi najzastupljenijih Javascript (JS) front-end okvira. State management odnosi se na upravljanje stanjem jedne ili više kontrola korisničkog sučelja, poput tekstualnih polja, gumba, poruka o pogreškama gdje stanje i ponašanje jednog UI elementa ovisni o stanju drugog ili više UI elemenata.

Svrha istraživanja je analiza i prikaz sustava za upravljanje stanjima NGRX, Vuex i Redux koje koriste navedeni okviri, njihovih značajki i utjecaja na korisnička sučelja, te usporedba suptilnih razlika ili sličnosti upravljanja stanjima i načina na koje se iste može primijeniti. Njihovom usporedbom će se staviti naglasak na definiranje vrijednosti stanja koja sadrže podatke aplikacije, definiranje akcijskih objekata koji opisuju događaje u aplikaciji i na funkcije reduktora koji izračunavaju ažuriranje stanja na temelju određenih radnji, a sukladno tome se za cilj ove aktivnosti postavlja definiranje ključnih značajki sustava za upravljanje stanjima, načina i benefita njihova korištenja za tri navedena okvira.

Cilj će biti postignut praktičnom primjenom state managementa za Angular, React i Vue.js okvira tako što će se izraditi određeni broj UI elementa sa pripadnim stanjima, ponašanjima i ovisnostima.

2. PRETHODNA ISTRAŽIVANJA

Upravljanje stanjem odnosi se na upravljanje stanjem jedne ili više kontrola korisničkog sučelja poput tekstualnih polja, gumba, radio gumba i slično, u grafičkom korisničkom sučelju. U ovoj tehnici programiranja korisničkog sučelja stanje jedne UI kontrole ovisi o stanju drugih UI kontrola. Na primjer, kontrola korisničkog sučelja poput gumba, bit će u omogućenom stanju kada polja za unos imaju valjane ulazne vrijednosti, a gumb će biti u onemogućenom stanju kada su polja za unos prazna ili imaju nevaljane vrijednosti. Kako aplikacije rastu, to na kraju može postati jedan od najsloženijih problema u razvoju korisničkog sučelja.

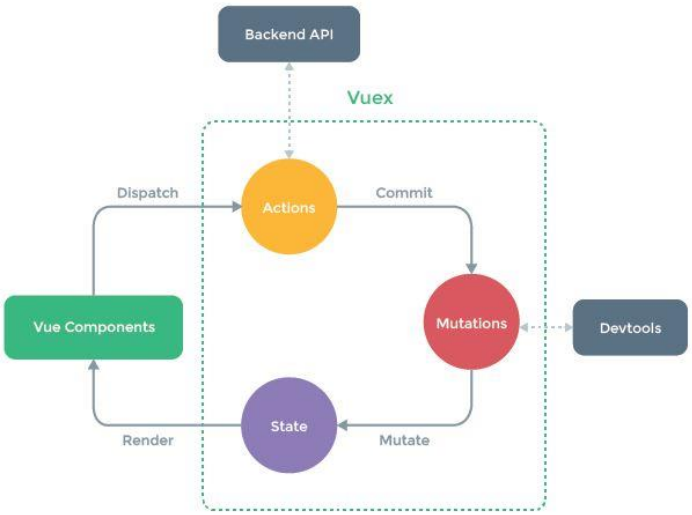
Primjeri knjižnica stanja za upravljanje uključuju Vuex kao knjižnicu za upravljanje za Vue.js Javascript okvir. Angular uključuje vlastitu knjižnicu NgRx, a Redux je općenita knjižnica za upravljanje stanjima koja se može koristiti s bilo kojim od gore navedenih okvira ili drugim knjižnicama, ali se vrlo često koristi s React knjižnicom. (https://en.wikipedia.org/wiki/State_management)

Komponente koje čine Vuex, NgRx i Redux su stanja (store), mutacije (mutations)/reduktori (reducers) i akcije (actions). **Stanje** je objekt koji sadrži svojstva koja treba dijeliti unutar aplikacije, a jedini način da se stvarno promijeni stanje u

store-u je vršenje mutacije. Vuex-ove **mutacije ili** Redux-ovi i Ngrx-ovi **reduktori** vrlo su slični događajima. One se mogu ažurirati i mutirati/reducirati kroz stanja, sinkrone su, događaju se jedna za drugom. **Akcije** su one koje čine mutacije i mogu sadržavati proizvoljne asinkrone operacije, što znači da se možda neće događati onim redoslijedom u kojem se pojavljuju u kodu. (https://medium.com/js-dojo/vuex-2638ba4b1d76, https://ngrx.io/docs)

2.1. VUEX

Vuex je obrazac upravljanja uzorcima stanja i knjižnica za aplikacije Vue.js. Služi kao centralizirano spremište (store) za sve komponente u aplikaciji s pravilima koja osiguravaju da stanja mogu mutirati na predvidljiv način. (https://vuex.vuejs.org/guide/actions.html#dispatching-actions)



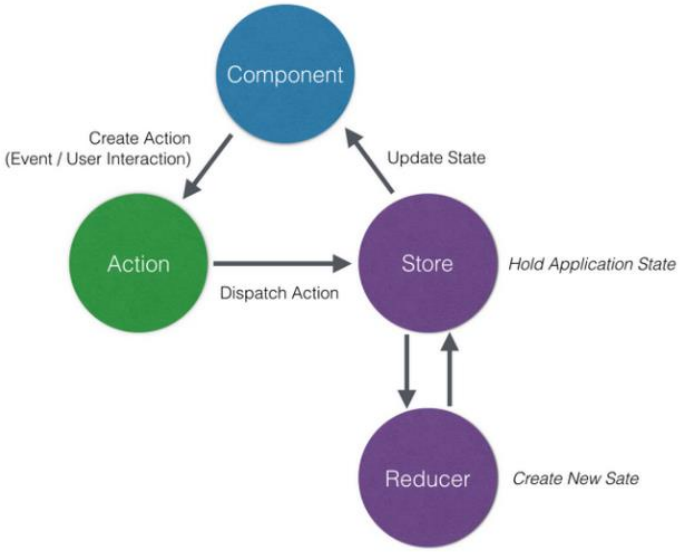
Slika 11: Vuex arhitektura, Izvor: https://vuex.vuejs.org/

Vuex ne ograničava strukturiranje koda, već umjesto toga provodi niz principa na visokoj razini tako da su stanja na razini aplikacije centralizirana u store-u, jedini način mutiranja stanja je izvršavanjem mutacija, a asinkrona logika treba biti inkapsulirana i može biti sastavljena od akcija. U slučaju ako store datoteka postane prevelika, akcije i mutacije može se dijeliti u zasebne datoteke. (https://vuex.vuejs.org/guide/structure.htm)

2.2. REDUX

Redux je predvidljivi spremnik stanja za Javascript aplikacije. Pomaže u pisanju aplikacija koje se dosljedno ponašaju, pokreću u različitim okruženjima i lako ih je testirati. Povrh toga, pruža izvrsno iskustvo za programere, poput uređivanja koda uživo u kombinaciji s programom za uklanjanje pogrešaka kroz vrijeme. Redux se može koristiti zajedno s Reac-tom ili s bilo kojom drugom knjižnicom. Mali je (2 kB, uključujući ovisnosti), ali ima velik ekosustav dodataka. (https://redux.js.org/introduction/getting-started)

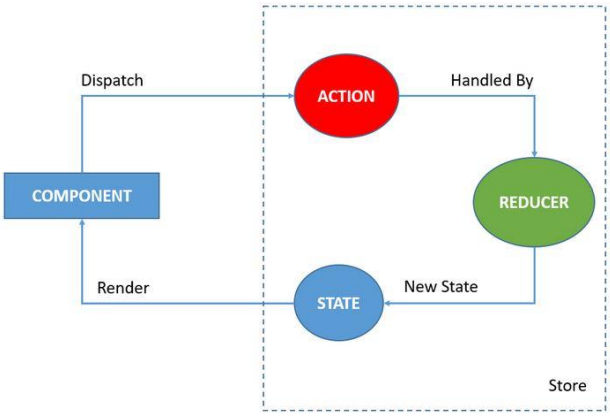
Redux održava stanje cijele aplikacije u jednom nepromjenjivom stablu stanja (objektu), koje se ne može izravno promijeniti. Kada se nešto promijeni, stvara se novi objekt (pomoću akcija i reduktora). (https://www.smashingmagazine.com/2016/06/an-introduction-to-redux/)



Slika 12: Redux arhitektura, Izvor: https://medium.com/codingthesmartway-com-blog/learn-redux-introduction-to-state-management-with-react-b87bc570b12a

2.3. NGRX

Inspiriran Redux-om, NgRx pruža način održavanja podataka u Angular aplikaciji kao jedinstvenog izvora istine. NgRx koristi streamove za interakciju s pohranom podataka. Ova se pohrana podataka povezuje s komponentama i uslugama i na kraju pojednostavljuje cjelokupni postupak upravljanja podacima aplikacije. NgRx store rješava višestruka pitanja kada se odgovornost za neke vidljive podatke dijeli između različitih komponenata. (https://medium.com/angular-in-depth/how-to-start-flying-with-angular-and-ngrx-b18e84d444aa)



Slika 13: NgRx arhitektura, Izvor: https://www.codemag.com/article/1811061

2.4. VUEX, NGRX, REDUX DOSTUPNE USPOREDBE

Usporedba Redux, Vuex i Ngrx knjižnica za upravljanje stanjima dostupna na Internetu je najčešće na temelju osobnih doživljaja programera s obzirom na njihovo programersko iskustvo, što ne predstavlja činjenična stanja, na temelju broja skidanja paketa i korištenja knjižnice u određenom vremenskom razdoblju ili su praktične usporedbe bez dodatne rasprave, pojašnjenja, zaključka, već samo prikazuju dvije iste aplikacije izrađene najčešće redux-om i vuex-om sa uzorcima koda. U nastavku je usporedba Redux-a sa Vuex-om i Redux-a sa Ngrx-om na temelju nekih osobnih doživljaja i praktičnih korištenja pronađeno na nekoliko izvora dostupnih na Internetu.

2.4.1. Vuex naspram Redux-a

Vuex je početnicima lakše naučiti od Redux-a. Vuex ima ugrađeno asinkrono rukovanje i jednostavniju implementaciju. Da bi se u Redux-u implementiralo pokretanje asinkronog zahtjeva koji mora rezultirati promjenom stanja potrebno je dodati dodatne knjižnice (među-opremu), dok je Vuex-u asinkrona podrška ugrađena u akcije što pokretanje i upravljanje pozivima čini trivijalnima. Vuex ima čišću integraciju u okvir i iako se i Redux i Vuex mogu koristiti neovisno od Vue-u i React-u, Vuex nudi uslužne programe zbog kojih se čini da se prirodnije uklapa u Vue nego Redux u React. (https://www.quora.com/Vuex-vs-React-Redux-which-one-is-better)

Bez obzira što su i Redux i Vuex nevjerojatne knjižnice i obrasci upravljanja stanjima mnogima je Vuex draži za rad jer je s mutacijama lakše raditi nego s reduktorima, asinkrone akcije su organiziranije, jednostavniji je za postavljanje i Vuex dobro iskorištava jedinstvenost Vue-a. (https://www.codementor.io/@petarvukasinovic/redux-vs-vuex-for-state-management-in-vue-js-n10yd7g2f)

2.4.2. Ngrx naspram Redux-a

Akcije u Redux-u ili Ngrx-u ista su stvar, a iako je tip potrebno definirati u oba, oznaka Action se razlikuje jer u Redux-u objekt Action osim tipa ovisi o programeru, a kod Ngrx-a može se dodati samo opcionalni ključ prijenosa podataka. Što se tiče reduktora, još jednom Redux i Ngrx imaju isti koncept, reduktori su čiste funkcije koje vraćaju nove reference stanja ukoliko se je dogodila promjena. Osim toga, imaju isti koncept otpreme tako da je jedini način pokretanja promjene stanja koristeći dispečer. (https://julienrenaux.fr/2017/02/16/from-redux-to-angular-ngrxstore/)

3. POSTUPAK

Služeći se praktičnim primjerom izrade jednostavnih elemenata korisničkog sučelja koji imaju funkcije unosa, ispisa i brisanja koji će zajedno činiti „todo“ listu izrađenih korištenjem Vue.js i Vuex, React i Redux, Angular i Ngrx okvira i knjižnica prikazati će se primjena state managementa koji će se u idućem poglavlju koristiti za usporedbu.

3.1. HTML

Prikaz elemenata i njihove funkcije za potrebe rada iste su za sve okvire i knjižnice, pa se tako jednostavna „todo“ aplikacija sastoji od naslova, polja za unos, gumba za potvrdu unosa, liste unosa sa gumbom za brisanje i označavanje stavke kao završene.

```
<section>
 <h1>TO DO LIST</h1>                        //naslov
  <div>
    <header>
      <input>                               //polje za unos
      <button>ADD</button>                  //potvrda unosa
    </header>
    <section>
      <ul>                                          //početak liste
        <li>                                        //jedan element liste
          <div>                                     //koji sadrži
            <label>"1. ToDo"</label>        //unos
            <button>Delete</button>         //gumb za brisanje unosa
               <button>Toggle</button>              //gumb za označavanje kao završeno
          </div>
        </li>
      </ul>                                         //završetak liste
    </section>
  </div>
</section>
```

Slika 4: struktura aplikacije – HTML

3.2. KORIŠTENJE KNJIŽNICA

Za upravljanje stanjima koristeći bilo koji od navedenih okvira potrebno je uvesti i dodatne knjižnice koje su usko povezane, što znači da neke od njih međusobno moraju komunicirati kako bi radile. U nastavku su prikazani uzorci koda za početak korištenja Vuex, Redux i Ngrx sa objašnjenjima.

3.2.1. Vuex uvoz knjižnica

Vuex je usko povezan samo sa Vue knjižnicom i jedina mu je potrebna za nesmetan rad. Oni moraju biti povezani jer se Vue okvirom definiraju elementi korisničkog sučelja, a Vuex upravlja stanjima tih elemenata.

```
import Vue from 'vue';                          //pozivanje vue
import Vuex from 'vuex';                        //pozivanje vuex
```

Slika 5: Vuex uvoz knjižnica

3.2.2. Redux uvoz knjižnica

Redux nije usko povezan samo sa Reactom, a za uspješno upravljanje stanjima potrebno je uvesti Provider i Connect. Provider čini Redux trgovinu dostupnom svim ugniježđenim komponentama koje su umotane u funkciju connect, a Connect pruža funkciju povezivanja React komponenti sa store-om.

```
import React from 'react';                  //pozivanje react
import { Provider } from "react-redux";           //čini redux dostupnim svim ugnježđenim
                                                  komponentama
import { Connect } from "react-redux";            //povezivanje komponenata sa storeom
```

Slika 14: Redux uvoz knjižnica

3.2.3. Ngrx uvoz knjižnica

Ngrx usko komunicira sa Angularom. Angularom se definira infrastruktura klasa za komponente, hijerarhije pogleda, prikaz podataka, a Ngrx kontrolira te komponente, podatke i slično. Također komunicira sa Rxjs Observable i Subscription, skupom povratnih poziva i njihova izvršavanja, a map operator prenosi svaku izvornu vrijednost kroz funkciju transformacije kako bi se dobilo odgovarajuće izlazne vrijednosti.

```
import { Component, OnDestroy, OnInit } from '@angular/core';
                                          //funkcionalnosti angulara
import { Store, select, Action } from '@ngrx/store';
                                          //spremnik kontroliranih stanja
import { Observable, Subscription } from 'rxjs';
                                               // zbirke pull i push vrijednosti
import { map } from 'rxjs/operators';    // manipulacija zbirkama
```

Slika 15: Ngrx uvoz knjižnica

3.3. STORE I STANJA

U središtu svake aplikacije izrađene korištenjem Vuex, Redux ili Ngrx nalazi se store. Store je spremnik koji sadrži stanja aplikacije koja se koriste za upravljanje UI elementima.

3.3.1. Vuex definiranje store-a i stanja

Vuex store je reaktivan, što znači da kada Vue komponente iz njega dohvaćaju stanja, reaktivno se stanja unutar store-a mijenjaju. Vuex store unutar više dokumenata ili jednog dokumenta sadrži sva stanja, akcije i mutacije, a u slijedećem uzorku koda prikazano je stvaranje novog store-a.

```
Vue.use(Vuex)  //inicijalizacija Vuexa i naredba Vue-u da ga koristi


export default new Vuex.Store({  //dodjeljivanje store-a varijabli zbog izvoza
  state: {                           //inicijalizacija stanja
    todos: []
  }
});
```

Slika 16: Vuex definiranje store-a i stanja

3.3.2. Redux definiranje store-a i stanja

Store se koristeći Redux stvara pozivom redux zbirke za stvaranje store-a. Stvoreni store dodjeljuje se varijabli koja poprima vrijednost naziva store-a i čuva naredbu da predmetni store koristi redux. Potom je potrebno inicijalizirati stanja i stvoriti reduktore.

```
import { createStore } from 'redux'         //uvoz zbirke za kreiranje store-a


//dodjela store-a varijabli, dodjela naziva store-a i naredba da se koristi redux
const store = createStore(todos, ['Use Redux'])


var initialState = [];                            //inicijalizacija stanja
```

Slika 17: Redux definiranje store-a i stanja

3.3.3. Ngrx definiraje store-a i stanja

Za stvaranje store-a i stanja koristeći Ngrx kreira se model koji sadrži elemente i tipove podatka liste koji se potom poziva u druge datoteke. Kada se stanja inicijaliziraju koriste se u obliku liste u slijedećem primjeru.

```
export default class ToDo {                 //stvaranje store-a/modela
    Title: string;
}
```

```
import ToDo from './todo.model';            //uvoz modela


const initialState: Array<Todo> = [];             //inicijalizacija stanja za listu modela
```

Slika 18: Ngrx definiranje store-a i stanja

3.4. MUTACIJE I REDUKTORI

Vuex koristi mutacije, Redux i Ngrx koriste reduktore koji navode na koji način se stanja aplikacije mijenjaju i njihova primjena jedini su način da bi se promjena nekog elementa zaista dogodila.

3.4.1. Vuex mutacije

Jedini način da se stvarno promijeni stanje u Vuex store-u je izvršavanje mutacije. U primjeru dodavanja novog objekta podaci se prosljeđuju mutaciji komponente koja ju je izvršila, novi podaci se dodaju u listu funkcijom push. Prilikom označavanja objekta izvršenim pronalazi se objekt u nizu koristeći se jedinstvenim identifikacijskim brojem i označava se kao završen, a prilikom brisanja objekta iz liste prosljeđuje se indeks objekta i mijenja se sadržaj niza.

3.4.2. Redux reduktori

Reduktori održavaju i stanje i njihove modifikacijske metode. Na početku potrebno je definirati globalnu varijablu sa početnom vrijednosti jedinstvenog identifikacijskog broja po kojem će se vršiti promjene (reduktori) s obzirom na akcije. U slučaju dodavanja novog objekta listi uzima se trenutno stanje liste kojoj se dodaje id uvećan za jedan, vrijednost unesenog teksta i označava kao nedovršeni objekt u listi. Prilikom označavanja objekta dovršenim uzima se trenutno stanje liste, vraća se objekt koji odgovara id-u objekta i označava završenim, te na posljetku prilikom brisanja objekta iz liste filter metodom se stvara novi niz sa svim objektima čija se vrijednost id-a razlikuje od id-a pronađenog objekta.

```
mutations: {
        addItem(state, payload) {    //prosljeđivanje podatka mutaciji od
                        komponente koja ju je izvršila (dodavanje
       novog elementa u todo listu)
          state.todos.push({id:GLOBAL_ID++, title: payload, completed: false};
        }                                    //ova funkcija šalje (gura/push) novi todo objekt u
                                            niz koji se nalazi u
                                             store-u
,
        togglecompletion(state, id) {       //pronalazak todo objekta u nizu
                              putem  jedinstvenog id-a
          state.todos.forEach( item => {    //za svako stanje todo store-a
            if(item.id === id)                     // ako je id objekta jednak   pronađenom id-u
              item.completed = !item.completed;           //todo završen
          })
        },
        removeItem(state, index) {               //prosljeđivanje indexa koji se želi obrisati iz
                                                 liste
          state.todos.splice(index, 1);   //mijenja sadržaj niza, uklanja
                                     objekt
        }
      }
```

Slika 19: Vuex mutacije

```
 var GLOBAL_ID = 1;                  //dodjeljivanje id-a za objekt liste
 const todos = (state = initialState, action) => { //parametar stanja koji se
                                     prosljeđuje funkciji todos
  switch (action.type) { //za obradu više radnji (dodaj, završeno, obriši)
    case "ADD_ITEM":  //u slučaju dodavanja objekta u listu
      return [        //vrati
        ...state,            //trenutno stanje i dodaj
        {
          id: GLOBAL_ID++,                  //definirani id povećaj za jedan
          title: action.title,              //uneseni tekst
          completed: false                  //označi kao nezavršeno
        }
      ];
    case "TOGGLE_COMPLETION":               //u slučaju označavanja sa dovršeno
      console.log('action', action);        //bilježenje akcije
      return state.map(todo =>                     //prima stanje todo liste i
        todo.id === action.id ? { ...todo, completed: !todo.completed } : todo
      );                                           // vraća objekt koji odgovara id-u, označava ga
                                                  kao završen

    case "REMOVE_ITEM":                            //u slučaju brisanja objekta
      return state.filter(todo => todo.id !== action.id); //metoda filter
                                     stvara novi niz sa svim elementima
                        koji ne odgovaraju pronađenom id-u
    default:
      return state;                         //vraćanje novog stanja
  }
};

export default todos;                       //izvoz nove liste za korištenje
```

Slika 20: Redux reduktori

3.4.3. Ngrx reduktori

Kao i kod Reduxa, Ngrx koristi reduktore kao modifikacijske metode, postavlja se id koristeći uuid (univerzalni jedinstveni identifikator), postavlja se parametar trenutnog stanja od kojeg se dalje vrše modifikacije liste. U slučaju dodavanja objekta u listu uzima se postojeća stanja i dodaje novi id i vrijednost unesenog teksta, u slučaju brisanja objekta stvara se novi niz bez onog objekta čiji id odgovara pronađenom koristeći filter metodu, a stanje dovršenog objekta označava se map metodom, odnosno prima se trenutno stanje „todo“ liste, pronalazi objekt koji odgovara id-u i vraća novonastalo stanje kroz listu.

```
import * as uuid from 'uuid';                //univerzalni identifikator
export const todoReducer = createReducer(initialState, //parametar stanja za..
//u slučaju/akciji Add u stanja se dodaje id i uneseni tekst
    on(Add, (state, action) => ([...state, { id: uuid.v4(), text: action.text, todo: true }])),
//u slučaju brisanja stvara se novi niz bez objekta koji odgovara pronađenom id-u
    on(Remove, (state, action) => state.filter(i => i.id !== action.id)),
   //u slučaju označavanja sa dovršeno prima se stanje todo liste, vraća objekt koji odgovara id-
u i novonastalo stanje liste
       on(Toggle, (state, action) => state.map(i => i.id === action.id ? {...i, todo: !i.todo} :
i)),
  )
```

Slika 21: Ngrx reduktori

3.5. AKCIJE

Kako bi se reduktori i mutacije prikazane u prethodnom poglavlju dogodili moraju postojati akcije. Akcije se ne moraju događati onim redoslijedom kojim se pojavljuju u kodu. Svaka akcija dodjeljuje se određenom elementu korisničkog sučelja koja potom poziva mutacije ili reduktore koje stvaraju definirane promjene. Za primjer „todo“ liste kreirano je onoliko akcija koliko je kreirano mutacija/reduktora.

3.5.1. Vuex akcije

Svaka Vuex akcija prosljeđuje argumente stanjima i poziva onu mutaciju koja se treba dogoditi kada se određena akcija dogodi. Primjerice, za dodavanje novog objekta kreira se akcija naziva 'addItem' koja prosljeđuje argumente stanjima (state, naziva todos) i poziva mutaciju koja se događa prilikom dodavanja novog objekta, naziva 'addItem' (mutacija prikazana u poglavlju 3.4.1.).

```
actions: {
        addItem({commit}, todos){                  //prosljeđivanje argumenata
                                        na akciju dodavanja objekta
                      listi
          commit(‘addItem', todos)                 //pozivanje mutacije koja se
                                        događa prilikom dodavanja
                      objekta
        },
        togglecompletion({commit}, todos){         //isto kao iznad
          commit(‘togglecompletion', todos)
        },
        removeItem({commit}, todos){
          commit(‘removeItem', todos)
        }
      }
```

Slika 22: Vuex akcije

3.5.2. Redux akcije

Isto kao i kod Vuex primjera akcija i kod Reduxa se stvaraju akcije kojima se dodjeljuje naziv koje pozivaju definirane reduktore (reduktor prikazan u poglavlju 3.4.2.). Redux za dohvaćanje reduktora koristi axios HTTP.

3.5.3. Ngrx akcije

Ngrx unutar store komponente ime već definirane funkcije za stvaranje akcija i definiranje novih podataka. Pozivanjem tih funkcija kreiraju se akcije sa pripadnim nazivom koje aktiviranjem određene komponente pozivaju reduktore da učine definirane promjene (reduktor prikazan u poglavlju 3.4.3.).

```
import axios from 'axios';          //HTTP za dohvaćanje spremanje podataka

  export const addItem = title => {         // akcija dodavanja
    return {
      type: "ADD_ITEM",                             // poziva mutaciju
      title
    };
  };

  export const toggleCompletion = id => { // isto kao iznad
    return {
      type: "TOGGLE_COMPLETION",
      id
    };
  };

  export const removeItem = id => {
    return {
      type: "REMOVE_ITEM",
      id
    }
  };
```

Slika 23: Redux akcije

```
//funkcije koje vraćaju objekt u obliku sučelja i definiranje dodatnih metapodataka
import {createAction, props} from '@ngrx/store';

//stvaranje akcije nad komponentom liste koja poziva mutaciju add i stvara novi podatak oblika
string
  export const Add = createAction('[Todo Component] Add', props<{text: string}>());
//isto kao iznad
  export const Remove = createAction('[Todo Component] Remove', props<{id: str ing}>());
  export const Toggle = createAction('[Todo Component] Toggle', props<{id: string}>());
```

Slika 24: Ngrx akcije

3.6. IMPLEMENTACIJA STORE-A U APLIKACIJU

Izrađeni store sa njegovim pripadnim mutacijama/reduktorima i akcijama potrebno je implementirati u aplikaciju, gumbima dodijeliti metode kada će reagirati na korisnikovu akciju i kako. Na elemente gumba dodane su funkcije dodavanja objekta, brisanja objekta i označavanja objekta završenim.

3.6.1. Vuex implementacija store-a

Aplikacija za pokretanje dodanih funkcija na elemente gumba koristi podatke store-a koristeći metode. Tako izrađene metode šalju zadatke u akcije nakon čega se izvršavaju mutacije. Primjerice, kod dodavanja nove stavke u listu, funkcija/naziv metode „addItem“ dodaje se svojstvu gumba „ADD“, ta metoda šalje zadatak u akciju „addItem“ koja izvršava „addItem“ mutaciju.

3.6.2. Redux implementacija store-a

Aplikacija za pokretanje funkcija dodanih elementima gumba koristi propove. Propovi ukazuju na vrijednosti funkcija odnosno na dispečere koji omogućuju pokretanje akcija koje pokreću reduktore. Primjerice, kod dodavanja nove stavke u listu „this.props.addItem“ funkcija koja je dodana gumbu „ADD“ ukazuje na dispečer „addItem“, poziva akciju „ADD_ITEM“ koja pokreće reduktor „ADD_ITEM“.

```
<button @click="addItem">ADD</button>  //dodavanje addItem funkcije na gumb
<button @click="removeItem">Delete</button> //dodavanje RemoveItem funkcije na
                                gumb
<button @click="toggleItem">Toggle</button> //dodavanje toggleItem funkcije na
                                gumb

export default {                                    //korištenje todos store-a
  name: 'app',
  data() {
    return {
      todos: [],
      holder: ''
    };
  },
//svojstvo methods sadrži sve metode koje korisnik može zatražiti od neke komponente
      methods: {
//ova metoda šalje trenutnu stavku zadatka u akciju koja se naziva addItem. Ova radnja izvršava
addItem mutaciju
        addItem: function(){
          this.$store.dispatch(addItem, this.holder);
          this.holder = '';
        },

//ova metoda šalje trenutnu stavku zadatka u akciju koja se naziva toggleItem. Ova radnja
izvršava toggleCompletion mutaciju
        toggleItem: function(todo){
          this.$store.dispatch('toggleCompletion', todo);
        }
      },

//ova metoda šalje trenutnu stavku zadatka u akciju koja se naziva removeItem. Ova radnja
izvršava removeItem mutaciju

       removeItem: function(todo){
          this.$store.dispatch('removeItem', todo);
        }
      },
    }
    </script>
```

Slika 25: Vuex implementacija store-a

```
//dodavanje funkcija gumbu
<button onClick={this.props.addItem(item.id)}>ADD</button>
<button onClick={this.props.removeItem(item.id)}>Delete</button>
<button onClick={this.props.toggleCompletion(item.id)}>Toggle</button>

const mapStateToProps = (state) => { //čini state dostupan u aplikaciji
  return { todos: state };
}

//prosljeđivanje funkcija akcijama koje pokreću reduktore
//povezivanje događaja na stranici s radnjama u store-u
const mapDispatcherstoProps = dispatch =>  { //omogućuje pokretanje akcija
  return {
//funkcije koje pokreću akcije
    toggleCompletion: (id) => dispatch(TOGGLE_COMPLETION(id)),
    removeItem: (id) => dispatch(REMOVE_ITEM(id)),
    addItem: (title)=> dispatch(ADD_ITEM(title)),
  }}
```

Slika 26: Redux implementacija store-a

3.6.3. Ngrx implementacija store-a

Aplikacija za pokretanje dodanih funkcija na elemente gumba koristi pripadajuće komponente i podatke store-a korištenjem promatrača i konstruktora. Promatrač promatra što se događa nad „todo“ listom, a konstruktor čini state dostupnim svim radnjama. Komponente dodane gumbima šalju događaje akcijama pomoću dispečera koje potom pokreću reduktore. Primjerice, kod dodavanja stavke u listu „addItem“ funkcija koja je dodana gumbu „ADD“ ukazuje na „addItem“ komponentu koja poziva „Add“ akciju koristeći dispečer, te ta akcija poziva reduktor i dodaje se uneseni tekst u listu.

```
//dodavanje komponenta elementima gumba
<button (click)="addItem()">ADD</button>
<button (click)="removeItem(todo.id)">Delete</button>
<button (click)="toggleItem(todo.id)">Toggle</button>

//sadržaj komponenti/mogućih radnji nad gumbima
 export class AppComponent {
        todos$: Observable<Todo[]>; //promatrač radnji
        newTodoText: string = "";
//čini state dostupnim svim radnjama
        constructor(private store: Store<{ todoState: Array<Todo> }>) {
          this.todos$ = store.select(state => state.todoState);
        }
//ova komponente šalju događaje akcijama koje potom pokreću reduktore
        addItem() {
          this.store.dispatch(Add({ text: this.newTodoText}));
          this.newTodoText = '';
        }

        removeItem(id) {
          this.store.dispatch(Remove({ id }));
        }

        toggleItem(id) {
          this.store.dispatch(Toggle({ id }));
        }
      }
```

Slika 27: Ngrx implementacija store-a

4. DISKUSIJA

Kroz prethodno poglavlje napravljene su tri iste „Todo“ aplikacije, koristeći Vuex, Redux i Ngrx. Prikazani su i opisani uzorci koda podijeljeni na način da svako čini jedan dio state managementa od svake knjižnice za upravljanje stanjima. Sva potpoglavlja služe za usporedbu triju knjižnica za upravljanje stanjima pa sukladno tome napravljena je sinteza elemenata izrađenih u prethodnoj cjelini. Postupkom iznad i analizom uzoraka koda slijedi tablični prikaz suptilnih razlika i sličnosti Vuex, Redux i Ngrx knjižnica za upravljanje stanjima.

Tablica 16: Usporedba Vuex, Redux i Ngrx knjižnica

PITANJA	VUEX	REDUX	NGRX
	KORIŠTENJE KNJIŽNICA		
Koliko je knjižnica neophodnih za rad?	2	3	4
Je li knjižnica stanja usko povezana sa FW od kojeg je nastala?	DA	NE	DA
	STORE I STANJA		
Je li store reaktivan?	DA	DA	DA
Koriste li se dodatne zbirke prilikom kreiranja store-a?	NE	DA	NE
Je li kod boilerplate (ponavljajući)?	DA	DA	DA
U koliko dokumenata se nalazi store?	1 ili više	1 ili više	1 ili više
Je li potrebno inicijalizirati stanja?	DA	DA	DA
	MUTACIJE I REDUKTORI		
Koristi li mutacije ili reduktore?	mutacije	reduktori	reduktori
Koristi li dodatne funkcije za dodavanje elemenata u listu?	push	NE	NE
Koristi li definirani jedinstveni identifikacijski broj za manipulaciju podacima?	DA	DA	DA
Koje dodatne metode koristi za pronalazak podataka u store-u?	forEach	map	map
Stvara li novu listu prilikom brisanja objekta iz liste?	NE	DA	DA
Koje dodatne metode koristi za brisanje objekta iz liste?	splice	filter	filter
	AKCIJE		
Koristi li akcije za pokretanje mutacija/reduktora?	DA	DA	DA
Je li potrebno definirati naziv stanja za prosljeđivanje akcije?	DA	NE	NE
Koje metode koristi za primanje i slanje argumenata akcija i mutacija	commit metoda	axios HTTP zbirka	funkcije createAction i props definirane u ngrx/store zbirci
Koliko mutacija poziva svaka akcija?	1	1	1
Događaju li se akcije samo kada se dogodi određena reakcija korisnika?	DA	DA	DA
	IMPLEMENTACIJA STORE-A U APLIKACIJU		
Što koristi za pokretanje funkcija elemenata?	propove	propove	promatrače i konstruktore
Što čini state dostupan u aplikaciji?	data() (objekt podataka koji pretvara svojstva u gettere i settere) ili mapState	mapStateProps (metoda za primanje stanja store-a)	constructor (metoda za primanje stanja store-a)
Koja metoda omogućuje pokretanje akcija?	dispatch	dispatch	dispatch
Je li točno da svaka funkcija gumba pokreće jednu akciju koja potom pokreće/poziva jednu metodu?	DA	DA	DA

Iz tablice proizlazi sljedeće:

- Uska povezanost knjižnica sa okvirom od kojeg su nastale utječe na moguće povezivanje knjižnica stanja sa drugim Javascript okvirima, pa se tako Redux može povezati s bilo kojim Javascript okvirom ili knjižnicom dok to nije slučaj kod Vuex-a i Ngrx-a koji se mogu koristiti samo sa onim okvirima od kojih su nastali.
- Velike sličnosti između Redux-a i Ngrx-a i razlike sa Vuex-om vidljive su kada se počne govoriti o načinu promjene stanja, odnosno o mutacijama i reduktorima. Vuex se od ostale dvije knjižnice vidljivo razlikuje zbog toga što za promjene stanja koristi mutacije kod kojih su stanja promjenjiva, dok su kod reduktora ona nepromjenjiva. Nepromjenjivost Reduxa i Ngrx-a pridonosi bržem odzivu aplikacije i dovodi do jednostavnijeg

programiranja jer je podatke koji se nikad ne mijenjaju lakše razumjeti od podataka koji se proizvoljno mijenjaju u cijeloj aplikaciji.

- Zbog toga što se korištenjem reduktora, koji su čiste funkcije, zapisuju stanja i radnje lako je razumjeti pogreške kodiranja, mrežne pogreške i druge oblike softverskih pogrešaka koji se mogu pojaviti tijekom razvoja i implementacije sustava, što rezultira i njihovim lakšim otklanjanjem.
- Osim toga pojavljuje se mogućnost implementacije „putovanja kroz vrijeme“, odnosno kretanja među prethodnim stanjima i pregled rezultata u stvarnom vremenu. „Putovanje kroz vrijeme“ nije moguće kod Vuex-a jer su njegova stanja i radnje promjenjive, a kretanje među stanjima naprijed-natrag zahtijeva čiste funkcije bez popratnih pojava zbog pravilnog prelaza između različitih stanja, ali i zbog uklanjanja mogućih pogrešaka.
- Kada je riječ o dodavanju elemenata u listu Redux i Ngrx ne koriste se dodatnim funkcijama baš zbog upotrebe reduktora za promjene stanja, dok Vuex koristi metode mutacije niza zbog toga što mijenja postojeći niz i tako stvara novi. Za uklanjanje elemenata iz liste Vuex koristi splice koji omogućava uklanjanje elementa iz postojećeg određenog niza, dok Redux i Ngrx koriste filter koji ne dodiruje ulazni niz već stvara i vraća novi što je također posljedica korištenja reduktora.
- Pronalazak podataka u store-u razlikuje se zbog toga što Vuex u tom slučaju uvijek vraća nedefinirane vrijednosti i rezultat njegove forEach() metode ne daje izlaz, dok Redux i Ngrx koristeći map() petlju svakom elementu niza dodjeljuje memoriju i pohranjuje povratne vrijednosti.
- Prilikom prosljeđivanja akcija Vuex komponente uvijek pristupaju izravno store-u, dok Redux i Ngrx komponente nikad ne pristupaju izravno već šalju radnje komponenata prema zadanim postavkama, a povezana komponenta može sama poslati radnje korištenjem props i dispatch argumenata.

Provedenim koracima za usporedbu Vuex, Redux i Ngrx knjižnica za upravljanje stanjima pokazala se velika sličnost između Redux-a i Ngrx-a, ali i očita razlika njih i Vuex-a. Velike razlike jesu rezultat promjena stanja koristeći mutacije ili reduktore. S obzirom da u ovom slučaju Vuex koristi mutacije za promjene stanja, a Redux i Ngrx reduktore zbog čega ih je moguće promatrati zajedno, u nastavku slijedi tablični prikaz njihove usporedbe.

Tablica 17: Sinteza usporedbe

PITANJA	VUEX	REDUX I NGRX
Jesu li definirana stanja nepromjenjiva?	NE	DA
Čini li brži odziv aplikacije?	NE	DA
Čini li podatke lako razumljivima?	NE	DA
Čini li pronalazak, razumijevanje i otklanjanje grešaka razumljivima?	NE	DA
Omogućava li putovanje kroz vrijeme?	NE	DA
Vraća li prilikom pronalaska podataka u store-u definirane vrijednosti i izlaze?	NE	DA
Pristupaju li UI komponente izravno store-u?	DA	NE

5. ZAKLJUČAK

Ovim radom prikazane su i analizirane knjižnice za upravljanje stanjima i njihova usporedba na temelju store-a, mutacija/reduktora i akcija. Objašnjene su njihove karakteristike, uzorci koda i prodiskutirane uočene sličnosti i razlike. Prema rezultatima analize i sinteze kroz poglavlje diskusije, knjižnice koje koriste reduktore za promjene stanja pokazale su se pogodnijima za korištenje.

Redux i Ngrx, pokazali su se pogodnijima jer ostvaruju brži odziv aplikacije zbog toga što su reduktori čiste funkcije. Osim što reduktori utječu na lako razumijevanje podataka čine pronalazak i otklon grešaka razumljivijima i lakšima zato što se podaci ne mijenjaju proizvoljno u cijeloj aplikaciji. Redux i Ngrx omogućavaju kretanje kroz prethodna stanja i pregled rezultata u stvarnom vremenu, dok ovakva kretanja kod knjižnica koje imaju promjene nastale mutacijama nisu moguća. Ipak, bez obzira na primijećene velike sličnosti Redux-a i Ngrx-a, Redux se je istaknuo jer se može povezati s bilo kojim Javascript okvirom ili knjižnicom dok to nije slučaj ni kod Ngrx-a, ni kod Vuex-a koji se mogu koristiti samo sa onim okvirima od kojih su nastali.

Kako bi prikaz usporedbe bio vjerniji i dosljedniji u daljnjem istraživanju preporučeno je ovaj rad proširiti sa još jednom state management knjižnicom čija su stanja mutirajuća odnosno promjenjiva kao što je npr. Mobx.

Literatura:

1 13.11.2020.; State management; URL: <https://en.wikipedia.org/wiki/State_management >; 15.11.2020.

2 Harry Beckwith; 24.06.2019.; Vuex – how to use state; URL: < https://medium.com/js-dojo/vuex-2638ba4b1d76 >; 15.11.2020.

3 What is Ngrx?; URL: < https://ngrx.io/docs > ; 15.11.2020.

4 Actions; URL: < https://vuex.vuejs.org/guide/actions.html#dispatching-actions >; 21.12.2020.

5 Aplication Structure; URL: < https://vuex.vuejs.org/guide/structure.htm >; 21.12.2020.

6 Getting Started with Redux; URL: < https://redux.js.org/introduction/getting-started >; 21.12.2020.

7 Alex Bachuk; 28.06.2016.; Redux – An Introduction; URL: < https://www.smashingmagazine.com/2016/06/an-introduction-to-redux/ >; 23.12.2020.

8 Andrew Evans; 19.03.2019.; How to Start Flying with Angular and Ngrx; URL: < https://medium.com/angular-in-depth/how-to-start-flying-with-angular-and-ngrx-b18e84d444aa >; 23.12.2020.

9 Kevin Ball; 11.05.2018.; Vuex vs. React Redux, which one is better?; URL: < https://www.quora.com/Vuex-vs-React-Redux-which-one-is-better >; 03.01.2021.

10 Petar Vukašinović; 26.09.2018.; Redux vs Vuex for state management in Vue.js; URL: < https://www.codementor.io/@petarvukasinovic/redux-vs-vuex-for-state-management-in-vue-js-n10yd7g2f >; 03.01.2021.

11 Julien Renaux; 16.02.2017.; From Redux to Angular ngrx/store; URL: < https://julienrenaux.fr/2017/02/16/from-redux-to-angular-ngrxstore/ >; 04.01.2021.

12 Connect: Dispatching Actions with mapDispatchToProps; URL: < https://react-redux.js.org/using-react-redux/connect-mapdispatch >; 14.01.2021.

Podaci o autorima:

Morena Pavlović, bacc. ing. telem.

Morena Pavlović 2019. godine stekla je naziv stručna prvostupnica inženjerka telematike, a trenutno je studentica II. godine specijalističkog diplomskog stručnog studija Informacijske tehnologije u poslovnim sustavima na Veleučilištu u Rijeci. Od 2014. godine zaposlenik je poduzeća Helada d.o.o. na radnom mjestu informatičar gdje obavlja poslove održavanja informatičke opreme, izrade i održavanja web stranica i snimanja dronom.

dr. sc. Marin Kaluža, profesor visoke škole

e-mail: mkaluza@veleri.hr

Marin Kaluža je profesor 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

web: www.veleri.hr

JAVASCRIPT I MODERNI RAZVOJ APLIKACIJA

mag. inf. Raul Sušanj Samolov, izv. prof. dr. sc. Marina Ivašić-Kos

SAŽETAK:

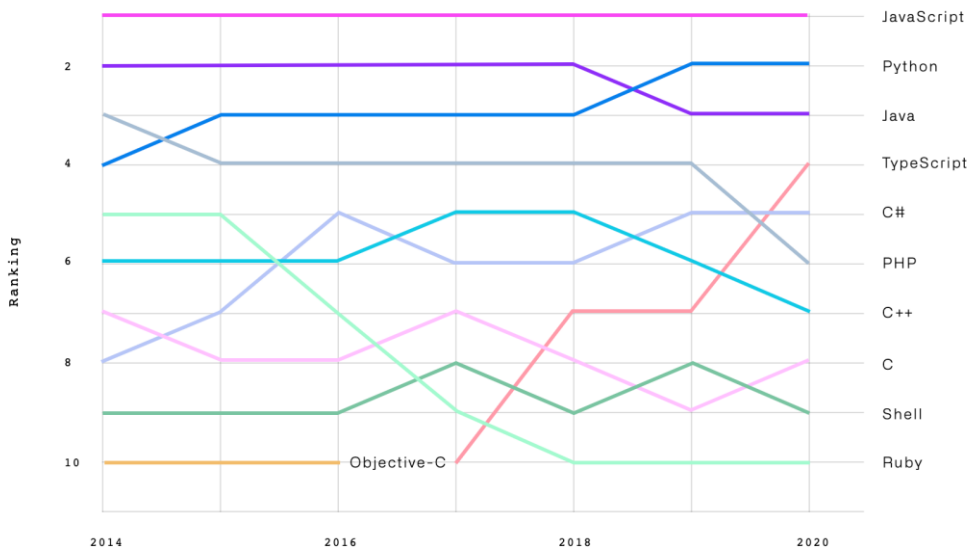
U radu je detaljan opis razvoja modernih aplikacija koje su napisane u JavaScript programskom jeziku i imaju za cilj da se mogu pokretati na raznovrsnim uređajima i platformama. Rad se bavi opisivanjem tehničkog i ekonomskog dijela razvoja aplikacija s detaljnim opisivanjem načina rada JavaScripta s naglaskom na njegove prednosti i mane te pomoćne alate koji su kompatibilni za korištenje. U zaključku se daje osvrt na trenutnu poziciju JavaScripta u IT zajednici i predviđanjima za budućnost primjene

ABSTRACT:

The paper describes in detail the development of modern applications written in the JavaScript programming language and aimed at running on a variety of devices and platforms that use the JavaScript programming language. The paper deals with the description of the technical and economic part of the application development with a detailed description of how JavaScript works, with emphasis on the advantages and disadvantages and the auxiliary tools that are compatible for its use. The conclusion gives an overview of the current position of JavaScript in the IT community and predictions for the future of the application.

1. UVOD

Razvoj modernih aplikacija kao što su web aplikacije je danas jedan od najprofitabilnijih poslova na svjetskom tržištu. Gotovo svaka tvrtka koja pruža neke servise ili proizvode posjeduje neku vrstu web aplikacije poput online trgovine ili mobilnu aplikaciju za program lojalnosti ili neku drugu svrhu. Razvoj web aplikacija ne zahtijeva skupu opremu ili sirovinu, već znanje programiranja, računalo s operacijskim sustavom i poslužitelj za podizanje sustava tako da trenutno najbrže posao dobivaju programeri sa znanjem JavaScript programskog jezika. JavaScript je posljednjih godina jedan od najpopularnijih programskih jezika sa velikim brojem korisnika, ne samo programera, već i tvrtki koje razvijaju aplikacije, Slika 1.



Slika 1. Najpopularniji jezici posljednjih godina na Githubu, https://octoverse.github.com/

JavaScript je omogućio razvoj na širokom spektru uređaja i platforma, a samim time je povećao vrijednost onih ljudi koji ga razumio i znaju koristiti. Tako se npr. JavaScript osim za razvoj web aplikacija može koristiti i za razvoj u mobilnih aplikacija, što je njegova velika prednost jer danas gotovo sve što želimo možemo odraditi pomoću pametnih telefona i aplikacija koje su instalirane na njima. Korištenjem mobilnih aplikacija danas nas do naručivanje hrane, odjeće i obuće, kozmetike, taksi službe, plaćanja računa i razgovora putem video poziva s prijateljima dijeli svega nekoliko dodira na ekranu.

2. POVIJEST JAVASCRIPTA "OD NASTANKA DO DANAS"

JavaScript (https://www.javascript.com/) je razvijen 1995. godine od strane programera Brendan Eicha koji je radio za Netscape Communications, tvrtku koja je tada imala monopol nad tržištem web preglednika. Eich je kreirao prvu verziju programskog jezika u svega desetak dana, a inspiraciju za strukturu i specifikaciju jezika dobio je od Java programskog jezika. Sredinom 1995. godine, Microsoft započinje preuzimanje tržišta web preglednika sa svojim web preglednikom pod nazivom Internet Explorer. Netscape Communications i Microsoft su tada postali glavni suparnici u utrci za najkvalitetnijim i najbržim web preglednikom pa je Netscape u nastojanju da spriječi sustizanje Microsoftovog Internet Explorer poduzeo slijedeće korake. Prvo su krenuli s procesom standardizacije programskog jezika kako Microsoft ne bi preuzeo kontrolu nad jezikom koji mu je tada bio interesantan kao već gotovo i provjereno rješenje za web preglednike. Standardizacija je nazvana ECMAScript te s koristi i danas u svrhu generalnog programskog standarda za razvoj jezika poput JavaScripta, a propisuje ga organizacija Ecma International (European Computer Manufacturers Association). Osim procesa standardizacije, Netscape ulazi u partnerstvo s tvrtkom Sun koja je od 1990. godine razvijala i promovirala Java programski jezik kao najbolji alat za softverska rješenja pametnih kućanskih aparata, a kasnije kao najbolje rješenje za web platformu, dijelila je zajednički interes s Netscapeom da spriječi Microsoftov monopol. Sun je pomoću partnerstva s Netscapeom imao benefit promocije svojeg programskog jezika u kombinaciji s još uvijek najpopularnijim web preglednikom, dok je Netscape imao benefit promocije JavaScripta kao službenog kompatibilnog programskog jezika s Java programskim jezikom, drugim riječima JavaScript je dobivao na popularnosti zbog postojećeg popularnog jezika. Iz istog razloga je JavaScript i preimenovan iz originalnog naziva Mocha u JavaScript.

Netscape je nudio i svoj proizvod pod nazivom LiveWire koji je omogućavao razvoj u istom programskom jeziku na klijentskoj i poslužiteljskoj strani sustava, funkcionalnost koju je i Sun razvijao s Javom. S obzirom na limitiranost tadašnjeg weba u usporedbi s kompleksnom Javom, Sun nije gledao na JavaScript kao stvarnu prijetnju te se partnerstvo između te dvije tvrtke i dalje nastavilo. Iako se počelo "čuti" za JavaScript i dalje je bio jedan od nepoželjnih jezika za programere. Kratko vrijeme inicijalnog razvoja jezika, slaba promocija, novi i prvi put viđen dizajn programskog jezika i generalno slab interes za razvoj sustava za web preglednik, rangirali su JavaScript daleko niže od ostalih programskih jezika na ljestvici. Averzija i ismijavanje prema jeziku nije postojala samo od strane programera već i od njegovog začetnika i kreatora Brendan Eicha koji je priznao da mu je žao što se odlučio na neke elemente i dizajn programskog jezika.

Kroz godine JavaScript više nije bio u središtu pozornosti sve dok se nije pojavio sintaktički podskup samog jezika pod nazivom JSON (JavaScript Object Notation) kojeg je razvio Douglas Crockford te je samim time osvježio interes za JavaScriptom. Web je postajao sve veći i interesantniji populaciji, a samim time je bila sve veća potreba za razvojem web platforma tj. web stranica i programera koji će razvijati u JavaScriptu. U 2000-tim godinama popularnost i korištenje JavaScripta postaje sve veća, a jedan od svojih vrhunaca dostiže 2006. godine kada se pojavljuje jedna od najpopularnijih biblioteka za JavaScript, a to je jQuery (https://jquery.com/), biblioteka koja je pomogla velikom broju programera da razumije do tada slabo shvaćen i proučen programski jezik.

Potencijal ovog programskog jezika su primijetile i druge vodeće tvrtke te su pokušale razvijati svoje podskupe i supersetove jezika, međutim većina ih nije dospjela do većeg broja programera osim Typescript (https://www.typescriptlang.org/). Typescript je razvio Microsoft i postavio ga je kao softver otvorenog koda kako bi ga popularizirao i pridobio zainteresirane za supersetom JavaScripta koji se danas koristi u velikom broju mobilnih i web aplikacija, ali opet ne koliko i sam JavaScript. Najveći porast korištenja JavaScripta se pojavljuje 2009. godine kada je objavljena peta verzija ECMAScripta, a najveće izmjene i nadopune se događaju 2015. godine kada izlazi šesta verzija ECMAScripta poznatija kao ES6 (te pod novim nazivljem ECMAScript 2015). Kako se standard nije izmjenjivao gotovo šest godina ECMA international odlučuje u buduće objavljivati svake godine novu verziju ECMAScripta kako ne bi opet došlo do velikog broja izmjena i nadopuna koje otežavaju učenje postojećim, ali i novim JavaScript programerima.

Danas JavaScript pod standardom ECMAScript 2020 (ES11) stoji kao jedan od najpopularnijih i najkorištenijih programskih jezika u kojem su napisane brojne moderne web i mobilne aplikacije, te se svake godine nadograđuje s novim funkcionalnostima i unaprjeđuje dizajn [1].

3. MOGUĆNOSTI JAVASCRIPTA

JavaScript se godinama tretirao kao skriptni jezik namijenjen za razvoj klijentskih aplikacija za okruženje web preglednika, međutim kroz vrijeme se njegova primjena proširila i na razvoj poslužiteljskih aplikacija te na veći broj okruženja i uređaja.

Istina je da se i danas većinom JavaScript koristi za razvoj klijentskih aplikacija, ali se sve veći broj IT tvrtki i njihovi razvojni inženjeri odlučuju za JavaScript kao jezik za razvoj poslužiteljskih servisa poput API-a (Application Programming Interface), CRON poslova i sl. Nakon Netscapeovog programa Live Wire koji je pružao mogućnost pisanja JavaScripta na poslužiteljskoj strani softvera, 2009. godine se pojavljuje Node.js (https://nodejs.org/en/) tzv. JavaScript runtime koji omogućava isto, ali dobiva na sve većoj popularnosti te se danas tretira kao glavno okruženje za pisanje JavaScripta bez potrebe web preglednika. Nakon popularizacije i sve većem broju napisanih servisa u Node.js-u 2010. godine se pojavljuje i upravitelj paketima pod nazivom npm (https://www.npmjs.com/get-npm) koji je olakšao programerima objavljivanje i preuzimanje dijelova otvorenog koda za Node.js.

Danas npm broji preko 350.000 paketa otvorenog koda te olakšava razvoj klijentskih i poslužiteljskih aplikacija

4. KOMPONENTE I NAČIN RADA JAVASCRIPT-A

JavaScript se kroz godine značajno unaprijedio, unaprijedili su mu se koncepti i dizajni jezika. Kao i svaki programski jezik, JavaScript se sastoji od nekoliko glavnih komponenata i često ga se u IT zajednici tretira kao poprilično neobičan programski jezik jer ponekad dozvoljava kršenje nekih pravila što zna rezultirati s uspješnim kodom, ali zna i predstavi problem programerima. Kako bi se izbjegle greške i kako ne bi došlo do nelogičnosti potrebno je dobro poznavanje

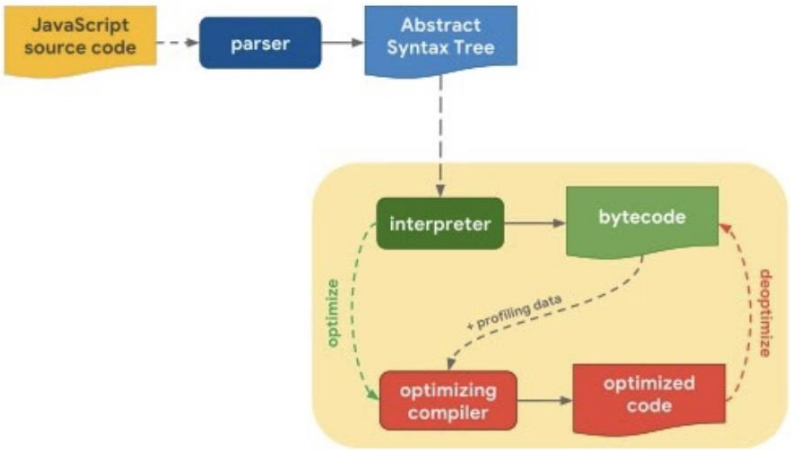
JavaScripta kao programskog jezika, programiranja općenito ali i znanje o funkcioniranju programskog jezika “ispod haube” tj. kako JavaScript procesuira kod koji pišemo u njemu kako bi postigli određenu funkcionalnost.

4.1. Javascript Engine (JavaScript pogon)

Prilikom kompajliranja JavaScript programskog jezika dolazi u uporabu tzv. JavaScript Engine koji je sastavni dio web preglednika, ali ga možemo pronaći i u prije navedenom runtime okruženju Node.js. Proces je sličan u oba slučaja pa ćemo ovdje dati primjer kako funkcionira JavaScript Engine Googleovog Chrome web preglednika pod nazivom V8 Engine jer se on koristi i kao JavaScript Engine u Node.js-u. Drugi web preglednici imaju svoje JavaScript Engine koji se minimalno razlikuju, neki od njih su npr. Spider Monkey za Firefox Mozillu, JavaScript Core za Safari, Chakra za Microsoft Edge.

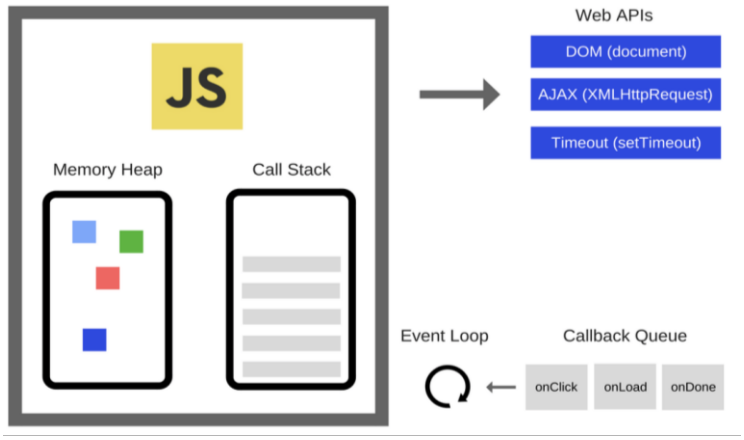
JavaScript Engine može čitati i pokretati kod, zato je prvi korak čitanje izvornog koda koji je programer napisao, taj dio procesa još nazivamo parsiranje. Prilikom parsiranja koda, JavaScript engine ga pretvara u Stablo Apstraktne Sintakse (Abstract Syntax Tree, AST) te ga prosljeđuje dalje u alatni lanac kompajlera (Slika 2). Neki od poznatijih kompajlera jesu Acron, Esprima i cheroot. Prvo kroz što se provodi AST, u daljnjem tekstu kod, je interpreter koji pretvara kod na dva sljedeća načina. Prvi način je da uzima jednostavne dijelove koda tj. one dijelove koji su optimizirani i one dijelove koda koji se ne ponavljaju više od jednog puta te ih direktno pretvara u binarni kod, a drugi način je da uzima složenije dijelove koda i tzv. vruće kodove tj. kodove koji su identični i primaju iste parametre (npr. funkcija koja se poziva na više mjesta), zatim ih optimizira pomoću TurboFan kompajlera (https://v8.dev/docs/turbofan) za optimizaciju uz pomoć mapiranih podataka prethodnog koda koji je već pretvoren prethodno u binarni kod te na kraju takav optimizirani kod pretvara u binarni kod.

Proces je vrlo jednostavan te ga se skraćeno može opisati bez dodatnih pojašnjenja oko pojedinih pojmova. Pojednostavljeno bi JavaScript Engine opisali kao alat koji prihvaća izvorni kod, zatim ga pretvara u stablo apstraktne sintakse i šalje interpreteru koji jednostavni kod pretvara u binarni kod, a složeniji i ponavljajući kod šalje najprije kompajleru za optimizaciju, a ovaj ga nakon optimizacije pretvori u binarni kod. Cijeli opisani proces se u V8 JavaScript enginu naziva JIT (justi-in-time compilation) [2].



Slika 2. Prikaz optimizacije izvornog koda (https://mathiasbynens.be/notes/prototypes)

Cjelokupan proces rada JavaScripta ne oslanja samo na JavaScript Engine, već su potrebni dodatni alati koje pružaju web preglednici kao što su hrpa memorije, stog poziva i slično, Slika 3.



Slika 3. JavaScript runtime (https://thecodest.co/blog/asynchronous-and-single-threaded-javascript-meet-the-event-loop/)

4.2. Memory Heap (Hrpa memorije)

Svaka varijabla, funkcija ili objekt koji definiramo mora se pohraniti negdje u radnu memoriju. Memory Heap je komponenta u web pregledniku koja je zadužena za alociranje memorije u JavaScript kodu te ona za programera odrađuje taj dio posla. Ono na što moramo paziti kada je riječ o memoriji, je to da ju ne preopteretimo Memory Heap, tj. da ne dođe do curenja memorije (engl. memory leak) jer je memorija uvijek limitirana. Prednost JavaScript jest ta da nam pruža ugrađeni kolektor smeća (garbage collector) koji je zadužen da oslobodi dijelove radne memorije od neiskorištenog koda, odnosno od onih varijabli, funkcija i objekata čija se referenca više ne koristi u programu [3].

4.3. Stog poziva (Call Stack)

Stog poziva je važna komponenta JavaScript obzirom da se radi o tzv. programskom jeziku s jednom niti (engl. one threaded programming language) koji izvršava liniju po liniju koda tako da se sljedeća linija koda se ne može pozvati niti izvršiti dok god prethodna linija nije izvršena. Takve programske jezike i samo programiranje još nazivamo sinkrono. Kako bi se osiguralo da se kod izvršava sinkrono, koristi se Stog poziva kao sastavni dio runtime memorije koji se sastoji od okvira za pozive. Svaki okvir za poziv se sastoji od lokalnih varijabli, argumenata i parametara te povratne adrese rezultata. Kada želimo izvršiti neku funkciju, Stog poziva postavlja okvir na vrh te ga nakon izvršenja izbacuje van. Stog poziva koristi metodu pozivanja FIFO, prvi unutra prvi van.

Ono što ne želimo da se dogodi je tzv. Stack overflow koji je u pojednostavljenim riječima događaj kada se Stog poziva napuni s pozivima preko svog limita što dovodi do rušenja aplikacije. Stack overflow možemo demonstrirati s neispravno napisanom rekurzivnom funkcijom koja nema uvjet prekida izvršavanja [2].

4.4. Web API

Kako bi se izbjeglo ograničenje JavaScripta da se naredbe ne mogu izvršavati istovremenu, uvedena je još komponenta koja ne pripada sastavno JavaScript programskom jeziku već web preglednicima, a to je Web API. Bez Web API-a većina aplikacija bi danas bila vrlo spora ili se neke funkcionalnosti ne bi mogle niti implementirati pa tako npr. ne bi mogli poslati korisniku obavijest dok on pretražuje neku listu u aplikaciji koja dohvaća podatke s poslužitelja, već bi morali čekati da korisnik prekine pretraživanje ili bi ga prekinuli u akciji slanjem obavijesti i prikazom iste na sučelju aplikacije.

Web API sadrži većinu funkcionalnosti koje zahtijevaju asinkronu izvedbu te dok se JavaScript program izvršava Web API preuzima asinkroni dio programa. Neke od najkorištenijih funkcija Web API-a jesu fetch, setTimeout, setInterval i potrebna je implementacija istih od strane programera.

Cjelokupni tok neke asinkrone radnje se odvija tako da se definirane asinkrone funkcije pozivaju i obrađuju od strane Web API-a, tako da Web API-i izvršava funkciju, dok se ostatak funkcija koje su sinkrone dalje šalju u Red događanja (Event queue). Kada se funkcija iz Web API-a izvrši, tada šalje funkciju isto u Red događanja [2]. Jedan primjer bi bio s funkcijom setTimeout, koja prima dva parametra, prvi parametar je povratna funkcija, a drugi je vrijeme čekanja za poziv povratne funkcije u milisekundama. Kada engine primijeti da se radi o asinkronoj funkciji, automatski ju šalje Web API-u koji ju izvršava na zasebnoj niti i to u C++ programskom jeziku.

4.5. Red događanja, red poslova i petlja događanja

Red događanja (Event queue) je red u kojem čekaju funkcije za slanje u Stog poziva na izvršavanje. Od ECMAScript 2015 uvedena je dodatna komponenta tzv. red poslova (Job queue) koji sadrži JavaScript Promise sa resolverima i rejectorima koji imaju prednosti izvršenja nad ostatkom funkcija, što znači da će petlja događanja slati prvo funkcije iz reda poslova u Stog poziva, a onda tek iz reda događanja. Petlja događanja (Event loop) je petlja koja se beskonačno izvršava i provjerava red poslova i događanja te Stog poziva [2]. Petlju događanja možemo zamisliti kao petlju koja konstantno pita da li je Stog poziva prazan, da li ima kakva funkcija u redu poslova i događanja, te ako je stog prazan, šalje funkciju iz redova u stog. Nakon cijelog procesa se funkcije pozivaju u Stogu poziva.

5. RAZVOJ POSLUŽITELJSKIH APLIKACIJA

Pojavom Node.js-a i sve češćim korištenjem u razvoju modernih web aplikacija kao alternativa za poslužiteljske servise, povećao se i broj JavaScript programera zbog toga što se pružila mogućnost razvoja vještina programiranja u domeni klijentskog i poslužiteljskog razvoja.

Veliki broj ozbiljnih, produkcijskih poslužiteljskih servisa se danas razvija u Node.js-u iako po performansama nije najbolji izbor [4]. Pravi razlog korištenja JavaScripta na poslužiteljskoj strani je ušteda resursa i radne snage, jer dok je prije bilo obavezno imati jednog programera za klijentsku stranu i jednog programera za poslužiteljsku stranu, danas ta dva posla može odraditi jedan programer u JavaScriptu. Svakako je preporučljivo u industriji da se ipak zapošljava programere za svako područje zasebno, ali tada imamo drugu prednost JavaScripta u razvoju cjelokupnog sustava, a to je da oni programeri koji rade na klijentskoj strani aplikacije uvijek mogu pomoću programerima na poslužiteljskoj strani i obrnuto. Osim navedenog, postoji visoka razina kompatibilnosti poslužiteljske i klijentske strane te veliki broj dodatnih biblioteka i alata koji jednako dobro funkcioniraju.

Postoji nekoliko oblika poslužiteljskih servisa, ali onaj najčešći u razvoju web aplikacija je API (Application Programming Language). API služi za dohvaćanje, slanje i manipulaciju podataka te je glavna poveznica između baze podataka i klijentske strane aplikacije. API najčešće odrađuje funkcije kreiranja, čitanja, izmjene i brisanja zapisa iz ili u bazu podataka ali može sadržavati i dodatne funkcionalnosti ne vezane za bazu podataka [5].

6. RAZVOJ KLIJENTSKIH APLIKACIJA

Istina je da razvoj poslužiteljskih servisa zahtjeva veliko znanje poznavanja principa rada baze podataka, mrežnog sustava i protokola, kibernetičke sigurnosti i drugih složenijih koncepata i dugi niz godina se smatralo navedene vještine i razvoj poslužiteljskih servisa zahtjevnijim i ozbiljnijim dijelom razvoja aplikacija. Danas je situacija nešto drugačija i kompleksnost razvoja klijentskih aplikacija se gotovo pa izjednačio s razvojem poslužiteljskih servisa, moglo bi se ponekad zaključiti da je čak krivulja učenja teža u razvoju klijentskih aplikacija nego poslužiteljskih. Kompleksnost razvoja na klijentskim aplikacijama možemo zahvaliti novim i modernim bibliotekama i razvojnim okruženjima čija je svrha ubrzati i pojednostaviti razvoj. Iako smo naveli kako je kompleksnost veća, ta kompleksnost rješava mnoge probleme s kojima su se programeri prije susretali, a kompleksnost je sve veća samo zato što se svakoga dana pronalaze nova i bolja rješenja, nove biblioteke i alati. JavaScript u ovom području dominira u usporedbi s drugim programskim jezicima i pruža najveću paletu alata za razvoj klijentskih aplikacija. Gotovo je moguće svaku aplikaciju danas napisati u JavaScriptu, radilo se o web, mobilnoj ili nekoj drugoj platformi. JavaScript se u razvoju klijentskih aplikacija toliko razvio da je u nekim oblicima čak zamijenio HTML i CSS te je moguće cijelu aplikaciju napisati samo u JavaScriptu [6]. Danas pomoću JavaScripta možemo kreirati HTML elemente i manipulirati s njima, a isto tako možemo dodavati stilove i stilske klase koje definiramo kao JavaScript objekte. Iako nam je potreban samo JavaScript prilikom razvoja, na kraju se ipak kod napisan u JavaScript translatira u dobri stari HTML i CSS, tako da je ključno i jako dobro poznavanje tih dviju tehnologija kako bi se mogle razvijati moderne i kvalitetne klijentske aplikacije.

6.1. React.js

U prošlosti je najpopularnija JavaScript biblioteka bila jQuery koja je omogućavala olakšanu manipulaciju nad DOM-om. DOM je stablasta struktura koja opisuje izgled i cjelokupnu strukturu aplikacija, a sastoji se od HTML elemenata. React.js biblioteka, (u daljnjem tekstu samo React), je preuzela ulogu jQueryja nakon 2013. godine kada je objavljen kao otvoreni kod, a do tada je bila korištena samo interno od strane Facebooka gdje je i razvijena. Razvio ju je programer Jordan Walke, po uzoru na XHP biblioteku za komponente u PHP programskom jeziku. Ciljevi Reacta su olakšati manipulaciju nad DOM-om i ubrzani razvoj korisničkog sučelja. Osnovni princip rada Reacta je tzv. *Lego princip* razvijanje korisničkog sučelja od komponenata (Components). Moguće je ugnježđivati već zadane komponente međusobno, te se tako dobivaju nove kompleksnije komponente. Zadane komponente primaju vrijednosti za zadana svojstva (props) te se putem njih proširuje funkcionalnost same komponente, a komponente koje smo sami razvili primaju samo svojstva koja definiramo. Princip prosljeđivanja podataka iz komponente u komponentu se odvija kroz svojstva, a privremeno memoriranje neke vrijednosti može se ostvariti drugim konceptom pod nazivom stanje (state). Stanje je poseban JavaScript objekt, specifičan za React u kojem se mogu pohranjivati vrijednosti u obliku ključ-vrijednost. Izmjene nad stanjem su moguće samo uz pomoć jedne funkcija setState(), koja za argument prima objekt sa sadržajem željenog ključa koji želimo izmijeniti i pripadajuću novu vrijednost. Moguće je razvijati komponente sa stanjem kao što su komponenta razreda (Class component) i bez stanja, funkcionalne komponente.

Sredinom 2018. godine uvedene su dodatne metode u React koje su omogućile postavljanje stanja i u funkcionalne komponente, takve metode se nazivaju kuke (hooks). Jedan od dodatnih koncepata Reacta jesu metode životnog ciklusa (lifecycle methods) koje isto tako nisu bile dostupne u funkcionalnim komponentama do pojave kuka. Metode životnog ciklusa jesu metode koje se pokreću u određenom trenutku životnog ciklusa aplikacije te unutar njih možemo izvršiti neku željenu funkciju. Neke od njih se pokreću prije prikaza komponente, neke prilikom izmjene na komponenti, a neke kada se komponente više ne prikazuje. Kuka useEffect je uvela jedinstvenu metodu koja objedinjava sva tri scenarija, samo je potrebno definirati da li ta metoda ovisi o nekom parametru komponente u kojoj je definirana [8].

6.2. React Native

Ideja hibridnih aplikacija postoji već dugi niz godina i postojale su neke biblioteke i neka razvojna okruženja koja su omogućavala razvoj mobilnih aplikacija pisanjem izvornog koda u web tehnologijama, konkretno u JavaScriptu. Nažalost rijetko su se programeri usudili eksperimentirati s takvim alatima sve dok se 2015. godine nije pojavio React Native. Dodatna biblioteka koja radi u kombinaciji sa Reactom je zainteresirala širu JavaScript zajednicu i s obzirom na to da je React do tada poprimio poprilično dobru reputaciju nije postojalo toliko straha da se isproba nešto novo. Prednost koju je većina web programera vidjelo u React Nativu jest mogućnost razvoja mobilnih aplikacija u programskom jeziku kojeg već dobro poznaju. Osim navedene prednosti dodatna prednost je bila što se pisanjem jednog izvornog koda pokrivalo razvoj aplikacije za čak tri platforme Android, iOS, Windows, a danas broji i četvrtu Symphony OS. Implementacija izvornog koda u React Nativu je identična onoj u React.js-u s nekoliko razlika. Ključne razlike su da React Native ne koristi iste elemente kao React, npr. dok u Reactu koristimo div u React Nativu se koristi View, a za elemente poput p, h1, h2 i sl. za prikaz teksta se koristi komponenta Text. React Native ima prednost pred drugim sličnim bibliotekama, da može pristupiti matičnim modulima (native modules) poput kamere, bluetootha, nfc-a i cijelom skupu senzorike pametnih mobitela.

Osim razvoja mobilnih aplikacija, React Native je moguće koristiti za razvoj aplikacija za stolna i prijenosna računala, nosive uređaje poput pametnih satova i narukvica, pa čak i pametne televizije [9].

7. TESTIRANJE

Jedan od najvrjednijih procesa prilikom razvoja modernih aplikacija je testiranje. Segment testiranja je često zanemaren tako da se samo površno testiraju aplikacija od strane programera, a ako neka tvrtka ima djelatnike samo za testiranje, QA inženjere, često se upotrebljava samo koncept E2E (end to end) testiranja. E2E testiranje je kada netko iz razvojnog tima testira aplikaciju tako da ju koristi kako bi je i krajnji korisnik koristio. Osim takve verzije E2E testiranja, moguće je napisati automatske E2E testove. Postoji nekoliko vrsta testiranja kao što su E2E, unit testovi, integracijski testovi i produkcijski testovi.

Kključni testovi su E2E i unit testovi, dok E2E simulira ponašanje krajnjeg korisnika, unit testovi testiraju tj. provjeravaju da li neka funkcija vraća ispravan rezultat u nekom određenom scenariju korištenja aplikacije. Preporuka je da se uz E2E testiranje s ljudskim faktorom implementiraju i automatizirani E2E i unit testovi. Osim što je preporuka pisanje testova, postoji i princip razvoja aplikacija vođen testovima, što znači da se prvo pišu testovi za potencijalnu funkciju, a zatim funkcija koja mora za svaki dati test vratiti željeni rezultat. Zbog kompleksnosti nekih testova i velikog broja linija koda koje je potrebno napisati, koriste se neka razvojna okruženja za testiranje poput MochaJS-a, Jesta, Jasmine, Karme i drugih [7].

8. PODIZANJE SUSTAVA

Pod sustav se smatra cjelokupna aplikacija koja se može sastojati od baze podataka, poslužiteljskih servisa i klijentskih aplikacija. Sustav možemo podignuti na više različitih okruženja, a podignuti sustav se ovisno o okruženju može razlikovati u verziji. Najčešće razvojne agencije podižu sustav na tri različita okruženja. Postoji testno okruženje koje se nalazi na udaljenom poslužitelju i dostupno je samo razvojnim programerima, zatim se postavlja *stagging* okruženje koje stoji na raspolaganju voditeljima projekta i klijentima i zadnje okruženje koje se podiže je produkcijsko. Prije podizanja sustava potrebno je podesiti okruženja prema kriterijima koje zahtjeva JavaScript aplikacija. Potrebno je prvo podignuti bazu podataka za koju je preporučljivo da se podiže na zaseban poslužitelj zbog zauzeća memorije i dodatne sigurnosti. Nakon uspješno podignute baze podataka, podižu se poslužiteljski servisi i klijentske aplikacije. Oboje se može nalaziti na istome poslužitelju, ali i ne mora. Kada je riječ o JavaScriptu najbolji izbor kreiranja softverskog poslužitelja je *nginx* poslužitelj (https://www.nginx.com/) unutar čije se konfiguracije mora podesiti na kojim ulazima će se pokretati poslužiteljski servis, a na kojem klijentska aplikacija. Unutar nginx konfiguracije se definira i domena putem koje će se moći pristupiti aplikaciji i servisima. Izvorni kod sustava se može ručno prenijeti na poslužitelje ili putem git alata dohvatiti iz repozitorija, što je i preporučeni način. Nakon cijelog postupka potrebno je pokrenuti nginx poslužitelj, zatim poslužiteljske servise, a klijentsku aplikaciju se treba prvo "izgraditi" (build) i zatim pokrenuti uz pomoć pripadajućih skripti. Kako se gotovo uvijek mora s vremena na vrijeme ponovo pokrenuti poslužitelj zbog izmjena u izvornom kodu ili ako dođe do neke greške u sustavu ili samom poslužitelju, dobra ideja je postaviti alate za automatsko ponovno pokretanje sustava u slučaju greške i praćenje sustava. Jedan od najjednostavnijih u osnovnoj verziji besplatnih alata je PM2 (https://pm2.io/docs/runtime/guide/startup-hook/). Kada se radi o mobilnim klijentskim aplikacijama proces podizanja je drugačiji. Mobilne aplikacije je potrebno podignuti na trgovine za aplikacije za svaku platformu zasebno, tako imamo Google Play za Android platformu, App Store za iOS platformu i App Gallery za Symphony OS. Pomoću zadanih naredbi koje pokrećemo unutar projekta kreiramo uvez aplikacije (app bundle) koji je potrebno podignuti na navedene trgovine putem koje krajnji korisnici mogu preuzeti aplikaciju. U trgovinama aplikacija nije potrebno nikakvo dodatno konfiguriranje osim unosa osnovnih podataka o aplikaciji [4].

9. DOKUMENTACIJA RAZVOJA I PROJEKTA

Ključan dio nakon svake završene faze razvoja, da li se radilo o krajnjoj fazi ili nekoj ranijoj je dokumentacija razvoja i projekta. Dokumentacija razvoja dokumentira tehničke aspekte sustava kao što su model baze podataka, model entiteta i veza, strukturu projekta, biblioteke koje se koriste, nejasne dijelove izvornog koda i sl., dok se dokumentacija projekta bavi dokumentiranjem sustava, najčešće klijentske aplikacije. Dokumentacija projekta služi kao uputstvo za krajnjeg korisnika i sadrži tko je sudjelovao na projektu, koliko je sati na kojem djelu sustava utrošeno, kako se koristi sustav i druge korisne informacije.

10. ZAKLJUČAK

S JavaScript programskim jezikom danas možemo razvijati moderne aplikacije za gotovo sve uređaje iako je nastao od skriptnog jezika napisanog u svega 10 dana. Prednosti JavaScripta jesu jednostavna implementacija, brzo učenje jezika, pokretanje na velikom broju uređaja i platforma, velika raznovrsnih biblioteka i razvojnih okruženja i velika zajednica programera.

Danas kad spomenemo modernu web ili mobilnu aplikaciju, možemo s velikom sigurnošću reći da je barem klijentska aplikacija razvijena u JavaScript programskom jeziku. Najčešći oblik implementacije JavaScripta na klijentskim aplikacijama je razvojno okruženje React.js i React Native, dok poslužiteljske servise implementiramo pomoću Node.js-a.

Svaki programski jezik ima svoje uspone i padove u popularnosti, pa tako i JavaScript kojem popularnost trenutno još uvijek raste, ali se usprkos tome on se neprestano razvija, prilagođava novim tehnologijama i unaprjeđuje.

Literatura:

1 Ben Aston, "A brief history of JavaScript", https://medium.com/@_benaston/lesson-1a-the-history-of-javascript-8c1ce3bffb17 (pristupljeno 08.07.2020.).
Alexander Zlatkov , "How JavaScript works: an overview of the engine, the runtime, and the call stack", https://blog.sessionstack.com/how-does-javascript-actually-work-part-1-b0bacc073cf (pristupljeno 11.08.2020).

2 Marijn Haverbeke, *Eloquent JavaScript, 3rd edition*, No Starch Press, San Francisco 2018.
Brad Traversy, "Node.js Deployment", https://gist.github.com/bradtraversy/cd90d1ed3c462fe3bddd11bf8953a896 (pristupljeno 21.11.2020).

3 Adrian Senecki, "GraphQL vs REST – the battle of API designs", https://tsh.io/blog/graphql-vs-rest/ (pristupljeno 21.08.2020).
Jon Ducket, *HTML & CSS, Design and Build Websites*, John Wiley & Sons, Inc. Indianapolis 2011.

4 Jash Unadkat, “Top 5 JavaScript Testing Frameworsk”, https://www.browserstack.com/guide/top-javascript-testing-frameworks (pristupljeno 12.11.2020).

5 Alex Banks, Eve Porcello, *Learning React, Functional web development with React and Redux*, O’Reilly Media, Inc., Sebastopol 2017.
Bonnie Eisenman, *Learning React Native, Building native mobile apps with JavaScript*, O’Reilly Media, Inc., Sebastopol 2016.

Podaci o autorima:

mag. inf. Raul Sušanj Samolov

e-mail: raulsusanj95@gmail.com

izv. prof. dr. sc. Marina Ivašić-Kos

e-mail: marinai@uniri.hr

Odjel za informatiku
Sveučilište u Rijeci
Radmile Matejčić 2, 51000 Rijeka
tel: +385 51 584 700
fax: +385 51 584 749
web: www.inf.uniri.hr

ARM ARHITEKTURA NAPREDUJE – NO POSTOJE IZAZOVI

Zlatko Sirotić

SAŽETAK:

Procesori ARM ISA arhitekture ne koriste se više "samo" za mobitele i tablete, nego i za serverska računala (pa i superračunala), te laptop / desktop računala.

I Apple je kod Macintosh računala prešao sa Intel arhitekture na ARM arhitekturu. Veliki izazov kod tog prelaska bio je - kako omogućiti da se programski kod pisan za Intelovu ISA arhitekturu, izvršava na procesoru M1 koji ima ARM arhitekturu.

Općenito, želja je da se programi pisani za jednu ISA arhitekturu mogu sa što manje napora izvršavati na računalima koja imaju drugačiju ISA arhitekturu. Problem je u tome što različite ISA arhitekture mogu imati dosta različite memorijske modele.

ABSTRACT:

ARM ISA processors are no longer used "only" for mobile phones and tablets, but also for server computers (even supercomputers) and laptops / desktops.

Apple has also switched from Intel to ARM on Macintosh computers. The big challenge in that transition was - how to enable program code written for Intel's ISA architecture to run on an M1 processor that has an ARM architecture.

In general, the desire is that programs written for one ISA architecture can be executed with as little effort as possible on computers that have a different ISA architecture. The problem is that different ISA architectures can have quite different memory models.

1. UVOD

Procesori ARM ISA arhitekture ne koriste se više "samo" za mobitele i tablete, nego i za serverska računala (pa i superračunala), te laptop/desktop računala. Koriste se i u industriji, u proizvodnim procesima, a i kao ugradbeni čipovi u ostale proizvode, npr. za ugradnju u IOT uređaje i u vozila (posebno autonomna vozila).

U 2. točki ukratko se prikazuje što radi firma Arm, što su ARM ISA arhitekture i mikroarhitekture, kakva je podjela posla u proizvodnji ARM čipova i na njima zasnovanih krajnjih uređaja (npr. mobitela).

U 3. točki prikazuje se gdje se koriste ARM čipovi. Posebno se naglašava da i Apple Macintosh sada koristi ARM čip - M1, kojega je dizajnirao sam Apple. Apple je napravio i novi dinamički binarni translator Rosetta 2, koji omogućava da se većina softvera pisanog za Macintosh računala sa Intel procesorom, uglavnom bez problema (i dovoljno brzo) izvršava na novim Macintosh računalima. Ali, ne bi trebalo misliti da je taj prelazak uvijek lako izvesti. Problem je u tome što različite ISA arhitekture mogu imati dosta različite memorijske modele.

U 4. točki (koja je napravljena uglavnom na temelju literature [1]) prikazuju se memorijski modeli kod različitih ISA arhitektura, te načini rješavanja mogućih problema kod dva naizgled različita zahtjeva:

- korektna emulacija softvera na različitim ISA arhitekturama, u paralelnom načinu rada
- (automatska) paralelizacija sekvencijalnih programa.

U 5. točki prikazuju se poboljšanja specifikacije ARM arhitekture (naročito memorijskog modela). Spominje se i najava da će ARM arhitektura uvesti hardversku transakcijsku memoriju (HTM), kod čije realizacije su se "opekli" Intel i IBM. Na kraju, daje se i (vjerojatni) odgovor na pitanje: Kako je Apple uspio da programi pisani za Intel dobro rade na njegovom M1 čipu ARM arhitekture?

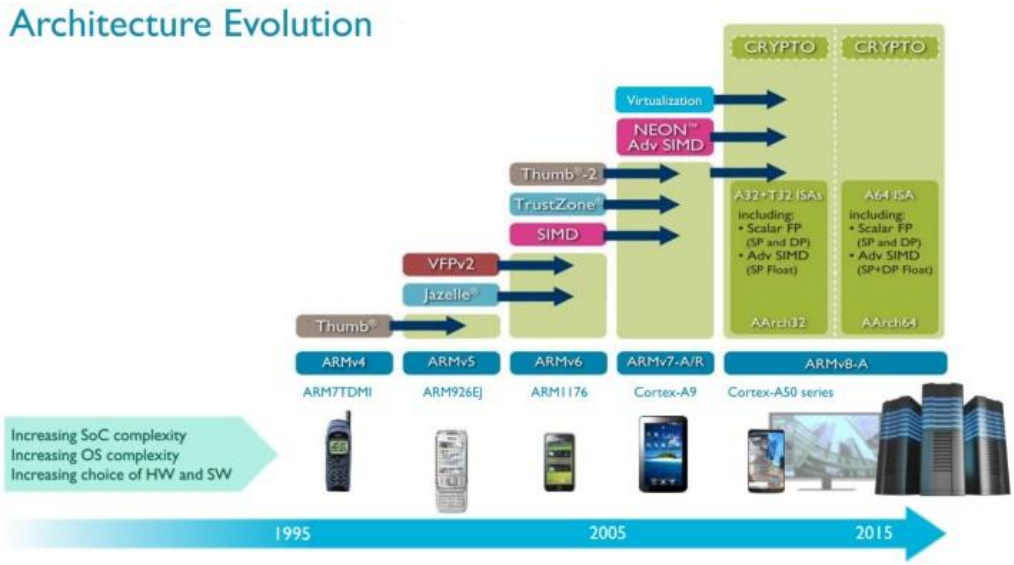
2. ARM ISA ARHITEKTURE I MIKROARHITEKTURE

Gotovo svi današnji pametni telefoni koriste procesorske čipove temeljene na jednoj od ARM ISA (Instruction Set Architecture) arhitektura. Napomena: budući da skraćenica ISA već sadrži slovo A, koje znači Arhitektura, donekle je redundantno reći *ISA arhitektura*, ali je često razumljivije.

Britanska firma Arm Ltd. **dizajnira ARM procesorske arhitekture** (množina!) iz određene ARM porodice,npr. arhitektura ARMv8.2-A iz porodice Cortex-A (64-bit).Firma Arm **dizajnira i procesorske jezgre (tj. mikroarhitekture)** temeljene na određenoj arhitekturi, npr. jezgra Cortex-A78 iz arhitekture ARMv8.2-A.

Slika 1. Logo firme Arm ltd. (lijevo) i ARM arhitekture (desno); Izvor: https://en.wikipedia.org/wiki/Arm_Ltd.

Firma Arm je jedan od "preživjelih potomaka" firme Acorn. Akronim ARM je prvo korišten 1983. i tada je značio "Acorn RISC Machine", a od 1990. znači "Advanced RISC Machines" (kako se zvala i firma Arm od 1990. do 1998.).



Slika 2. Evolucija ARM arhitekture; Izvor: https://www.androidauthority.com/developing-arm-everything-need-know-389402/

Druge firme na temelju licenci koje kupuju od firme Arm (te su licence za sada relativno povoljne, npr. iznose oko 1% prodajne cijene čipa) dizajniraju i vlastite inačice ARM jezgri i **SoC (System on a Chip)** čipove (tako rade npr. Apple, Samsung i Qualcomm), ili dizajniraju samo SoC čipove (tako rade npr. Huawei i MediaTek).

Te čipove često proizvode treće firme. Najbolju (5 nm) tehnologiju proizvodnje procesorskih čipova za sada imaju tajvanski TSMC ("svjetski prvak") i Samsung Electronics.

Na temelju SoC čipova, četvrte firme mogu dizajnirati npr. mobitele (ali, ti se čipovi mogu koristiti i za druge namjene), a pete firme mogu proizvoditi mobitele.

Na ARM čipu mogu pisati informacije kao na slici 3. (ovisno o konkretnom proizvođaču i čipu): koja je to arhitektura (ARMv8-A), CPU jezgre (Cortex-A57 i A53), GPU jezgre (ARM Mali-T624) … ali i ime proizvođača čipa (TSMC), koji često nije isti kao firma koja je dizajnirala čip.

Slika 3. Primjer ARM čipa na matičnoj ploči; Izvor: https://en.wikipedia.org/wiki/ARM_architecture

Koliko nam je poznato, jedina firma koja na temelju ARM arhitekture sama radi i (re)dizajn jezgri i dizajn SoC čipova, proizvodi čipove, te dizajnira i proizvodi mobitele, je Samsung Electronics. To ne znači da Samsung uvijek tako radi. Npr., osim vlastito dizajniranih procesorskih čipova, Samsung koristi (naročito za tržište SAD-a) čipove koje dizajnira firma Qualcomm. Također, Samsung ne proizvodi sve čipove, već mu dobar dio čipova proizvodi firma TSMC (bilo zbog ograničenih kapaciteta, bilo zbog toga što Samsung još nije ovladao određenim tehnološkim procesom).

Tablica 1. pojednostavljeno prikazuje od kojih se glavnih faza sastoji proizvodnja npr. mobitela i tko sudjeluje u tome.

Tablica 1. Podjela posla u proizvodnji ARM čipova i mobitela; Izvor: obrada autora

	Redizajn ARM jezgre	**Dizajn SoC čipa**	**Proizvodnja SoC čipa**	**Dizajn mobitela**	**Proizvodnja mobitela**
Samsung	Da Exynos	Da Exynos	Da	Da	Da
Apple	Da Firestorm i Icestorm	Da M1 - za Macintosh	Ne	Da	Ne
Huawei	Ne	Da Kirin	Ne	Da	Da
Qualcomm	Da Kryo	Da Snapdragon	Ne	Ne	Ne
MediaTek	Ne	Da Dimensity	Ne	Ne	Ne

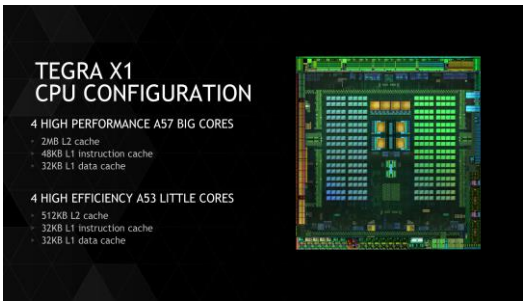
3. ARM ČIPOVI SE NE KORISTE "SAMO" U MOBITELIMA

Donedavno su se procesori ARM arhitekture u razmišljanjima većine ljudi povezivali "samo" s mobitelima. No činjenica je da se ARM arhitektura već dosta davno probila i na servere.

Npr. **najjače današnje superračunalo Fugaku** (po TOP500 rangiranju), koje je dizajnirao i proizveo japanski Fujitsu, ima 158 976 komada 48-jezgrenih čipova A64FX,koje je (kao i njegove jezgre) također dizajnirao Fujitsu, na temelju arhitekture ARMv8.2-A (čip je proizveo TSMC, u 7 nm tehnologiji). Fujitsu je prije toga za svoja računala i superračunala koristio vlastiti čip SPARC64 V.

Inače, vlasnik firme Arm je od 2016. japanski konglomerat SoftBank, koji ju je tada kupio za 32 milijarde $. Sada želi prodati 90% za 40 milijardi $, jer je pritisnut financijskim gubicima na drugim područjima (npr. fijasko s ulaganjima u firmu WeWork). Apple je izjavio da nije zainteresiran (vjerojatno zato što je pretpostavio da mu to nikako ne bi moglo proći kod regulatornih agencija), ali Nvidia jeste. **Mnogi se tome protive, bojeći se monopola,** između ostalog i Google, Microsoft i Qualcomm.

I Nvidia radi čipove na temelju ARM arhitekture. Slika 4. prikazuje Tegra X1 čipa, koji se koristio za Nintendo Switch igraću konzolu. Čip nije imao performanse kao konkurencija.



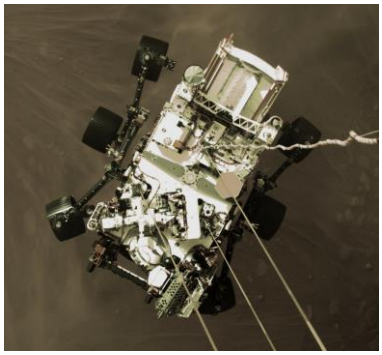
Slika 4. Nvidia Tegra X1 čip; Izvor: https://www.anandtech.com/show/8811/nvidia-tegra-x1-preview

Nvidia je najjači igrač kod GPU čipova, koji se ne koriste samo za igranje i rudarenje kriptovaluta (na čemu Nvidia dobro zarađuje), nego i za "suradnju" s CPU čipovima u serverskim računalima (naročito u cloudu) i superračunalima. Posjedovanje ARM arhitekture omogućilo bi da Nvidia ima u svojim rukama obje tehnologije (za GPU i CPU), i vjerojatno bi postala teško nadmašivi lider u području umjetne inteligencije.

Postoje razmišljanja da će se, ako Nvidia uspije kupiti firmu Arm i ako značajno poveća cijenu licenci ili na drugi način oteža rad drugih dizajnera ARM jezgri i čipova, desiti "prebacivanje" težišta s ARM arhitekture na open source RISC V arhitekturu. Naravno, takav prelazak nikada nije brz i lagan (nije riječ "samo" o hardveru, nego i o pratećem softveru), ali nužda tjera na velike promjene.

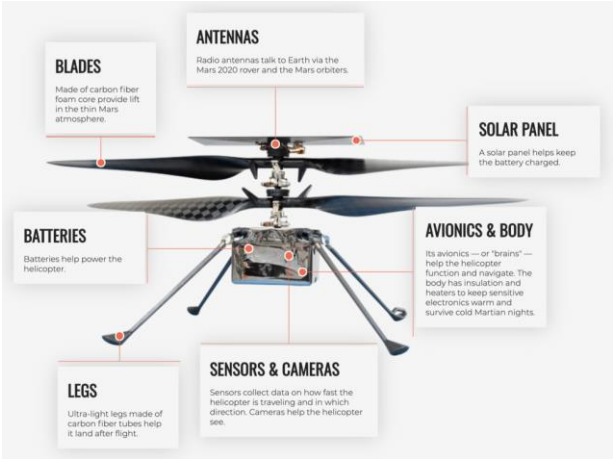
Procesori ARM arhitekture koriste se i u industriji, u proizvodnim procesima, a i kao ugradbeni čipovi u ostale proizvode, npr. za ugradnju u IOT uređaje i u vozila (posebno autonomna vozila).

Nedavno smo (18.02.2021.) imali priliku gledati spuštanje rovera Perseverance na Mars (slika 5.). Roverovo računalo koristi BAE Systems RAD750 čip (koji je ojačan štitom protiv radijacije; počeo se proizvoditi 2001.), koji je temeljen na PowerPC 750 32-bitnom mikroprocesoru (počeo se proizvoditi 1997.).



Slika 5. Spuštanje rovera Perseverance na Mars; Izvor: https://en.wikipedia.org/wiki/Perseverance_(rover)

No na Mars je ipak došao i ARM čip. Roverov helikopter (slika 6.) koristi 32-bitni čip Qualcomm Snapdragon 801 (čip je ARMv7, iz 2014.).



Slika 6. Helikopter rovera Perseverance ima ARM čip; Izvor: https://commons.wikimedia.org/wiki/File:Anatomy_of_the_Mars_Helicopter.png

Veliko zanimanje za ARM čipove u laptop/desktop korisničkom svijetu uslijedilo je tek nedavno, kad je Apple uspješno prešao sa Intelove arhitekture na ARM arhitekturui kod računala Macintosh. Prije toga je Apple koristio ARM arhitekturu "samo" za mobitele i tablete.

Zanimljivo je da je ovo Appleov peti prijelaz s jedne na drugu procesorsku arhitekturu (četvrti prijelaz kod Macintosh računala). Apple je za Apple II koristio Intel 6502 procesor. Za Machintosh, koristio je prvo Motoroline 68000 procesore, pa IBM PowerPC (od 1994.), pa Intelove (od 2006.).

U Macintosh se ugrađuje SoC M1 (proizvodi ga TSMC, u 5nm tehnologiji), kojega je (kao i dosadašnje čipove za mobitele) dizajnirao sam Apple. M1 čip ima CPU jezgre koje je također dizajnirao Apple, na temelju arhitekture ARMv8.6-A, i to četiri jače jezgre Firestorm i četiri štedljivije jezgre Icestorm.

Uz 8 CPU jezgri, M1 sadrži i 7 ili 8 GPU jezgri,te 16 Neural Engine jezgri. M1 SoC je upakiran zajedno s2 DRAM čipa (8 ili 16 GB) u SiP (system-in-a-package).

Slika 7. Apple M1 SoC; Izvor: https://www.apple.com/mac/m1/

Uspjehu Appleovog prelaska na novu arhitekturu sigurno je doprinio i Appleov novi dinamički binarni translator Rosetta 2. Prvu verziju, imena Rosetta, Apple je napravio 2006., kod prelaska sa IBM PowerPC procesora na Intelove procesore.

Rosetta 2 omogućuje da se binarni kod pisan za Intelovu ISA arhitekturu x86-64, izvršava na procesoru M1 koji ima ARM arhitekturu.

Pritom se translacija binarnog koda pokušava napraviti odmah kod instaliranja programa, tj. pokušava se napraviti **AOT translacija (ahead-of-time)**, kako bi program radio brzo i kod prvog pokretanja. U slučaju da AOT translacija (dijela) koda nije moguća, koristi se JIT translacija (just-in-time). Nažalost, postoje i programi koji se moraju preraditi da bi radili.

No iako je Rosetta 2 omogućila da se većina softvera pisanog za Macintosh računala sa Intel procesorskom arhitekturom uglavnom bez problema (i dovoljno brzo) izvršava na novim Macintosh računalima sa ARM arhitekturom, **ne bi trebalo misliti da je taj prelazak uvijek lako izvesti.**

ARM arhitektura, u odnosu na Intelovu, koristi memorijski model koji je više relaksiran. Zato nije lako raditi emulatore / translatore koji omogućavaju izvođenje programa pisanih za "konzervativnije" arhitekture (npr. x86-64) na računalu koje ima više relaksirani (slabiji) memorijski model, kao što ju ima ARM arhitektura.

4. EMULACIJA SOFTVERA NA RAZLIČITIM ISA ARHITEKTURAMA

Zbog nastojanja za povećanjem performansi programa, većina današnjih procesora implementira **relaxed memory consistency model** (memorijski model relaksirane konzistencije).

To znači da **procesori mogu mijenjati redoslijed instrukcija**. To je hardversko mijenjanje redoslijeda instrukcija u toku izvođenja, a ne softversko! Podsjetimo se da i optimizatori (kod kompajliranja), mogu softverski mijenjati redoslijed instrukcija u odnosu na onaj koji je napisao programer.

Različiti proizvođači imaju različite modele konzistencije memorije. Zapravo, čak i isti proizvođač može imati više modela, jer oni ovise o ISA arhitekturi.

Najintuitivniji model konzistencije memorije je **sekvencijalna konzistentnost** (Sequential Consistency), kod koje se naredbe procesora izvršavaju **striktno atomarno (atomically) i u redoslijedu koji je specificiran programom**.

Sekvencijalna konzistentnost se u praksi izbjegava, jer onemogućava hardversku optimizaciju programa. Slika 8. prikazuje različite modele konzistencije memorije, kod različitih današnjih procesora (tj. ISA). Kvačica uz određeno ograničenje (W -> R order, W -> W order, R -> RW order) pokazuje da je to ograničenje relaksirano, tj. da ga se procesor ne mora pridržavati. Vidi se da (poslije Sequential Consistency arhitekture) **najmanje je relaksirana x86 arhitektura, a najviše su relaksirane SPARC-RMO, POWER i ARM**.

Relaxation	W → R order	W → W order	R → RW order
SC			
x86-TSO	✓		
SPARC-TSO	✓		
SPARC-PSO	✓	✓	
SPARC-RMO	✓	✓	✓
POWER	✓	✓	✓
ARM	✓	✓	✓

Slika 8. Relaxed memory consistency model kod različitih ISA; Izvor: [1]

Slika 9. daje indikativan primjer dvije OS dretve koje sadrže dvije jednostavne instrukcije (gornji dio slike), te različite moguće rezultate izvršavanja kod x86 i POWER arhitekture (donji dio slike). Varijable x i y na početku imaju vrijednost 0. Vidimo da se kod POWER arhitekture (a tako bi moglo biti i kod ARM arhitekture) može javiti "nemogući rezultat", koji je ipak moguć, jer POWER arhitektura po specifikaciji dozvoljava da u ovom primjeru dođe do hardverskog mijenjanja redoslijeda naredbi. Kaže se "po specifikaciji", jer to nije obavezno, i često konkretne implementacije čipova to ipak ne rade.

Thread 0	Thread 1
`r1 = x` `y = 1`	`r2 = y` `x = 1`

Result	x86-TSO	POWER
r1 = 0, r2 = 0	✓	✓
r1 = 0, r2 = 1	✓	✓
r1 = 1, r2 = 0	✓	✓
r1 = 1, r2 = 1	✗	✓

Slika 9. "Nemogući rezultat" u 4. redu - moguć je kod npr. POWER arhitekture; Izvor: [1]

Može se postaviti pitanje: Čemu te komplikacije?

Odgovor je: Koristeći memorijski model relaksirane konzistencije, proizvođači procesora pokušavaju postići (automatsko) **paralelno izvršavanje sekvencijalnih programa,** u cilju povećanja performansi programa.

Ali, da bi tada omogućili korektno izvršavanje sekvencijalnih programa, primjenjuju (na softverskoj razini, najčešće kod kompajliranja) dvije različite tehnike.

Jedna je korištenje **instrukcija za sinkronizaciju ogradom** (**memory fence instructions,** skraćeno fences), koje može programer ručno dodati u program, ili ih (najčešće) automatski dodaje kompajler u fazi optimizacije.

Druga je primjena **hardverske transakcijske memorije (HTM)**, kod onih procesora (za sada rijetkih) koji podržavaju HTM, također ili ručno ili automatski (kod kompajliranja).

Zbog relaksiranog memorijskog modela, može doći do problema i kod emulacije softvera. Zapravo, kod emulacije softvera na istoj ISA arhitekturi, gdje virtualni stroj (gost) ima isti memorijski model kao i pravi stroj (domaćin), nema problema.

Problemi se mogu javiti kod **emulacije softvera na različitim ISA arhitekturama (cross-ISA system emulators).**

Ako domaćin ima manje relaksirani (jači) memorijski model u odnosu na gosta, opet nema (značajnih) problema. No ako domaćin ima više relaksirani (slabiji) memorijski model u odnosu na gosta (npr. ako se na procesoru POWER ili ARM arhitekture emulira x86 virtualni stroj), **moglo bi se desiti da se program izvršava nekorektno.** Naravno, to se ne smije dopustiti!

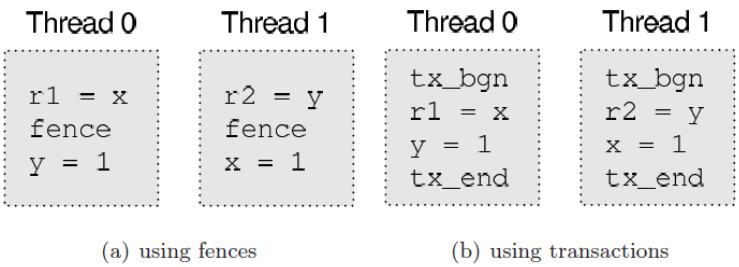
Jedno rješenje za postizanje korektnosti izvršavanja kod emulacije softvera na različitim ISA arhitekturama je **izbjegavanje paralelizacije.** Ako domaćin ima više relaksiran memorijski model u odnosu na gosta, programi koji su na izvornoj platformi (platformi gosta) višedretveni (pa se mogu izvršavati paralelno, na više hardverskih dretvi), izvršavaju se tada na domaćinu sekvencijalno, koristeći **timesharing**.

Treba napomenuti da su prije čak i emulatori između istih ISA arhitektura radili na sekvencijalan način. Naravno, time se gubi na performansi, jer se softver ne izvodi paralelno.

Današnji (paralelni) emulatori više ne rade tako, tj. oni koriste paralelizaciju kod emulacije. No često su **ograničeni na emulaciju samo unutar iste ISA arhitekture** (npr. x86 na x86), ili (u najboljem slučaju) **na emulaciju kod koje gost ima više relaksirani (slabiji) memorijski model** u odnosu na domaćina (npr. ARM na x86).

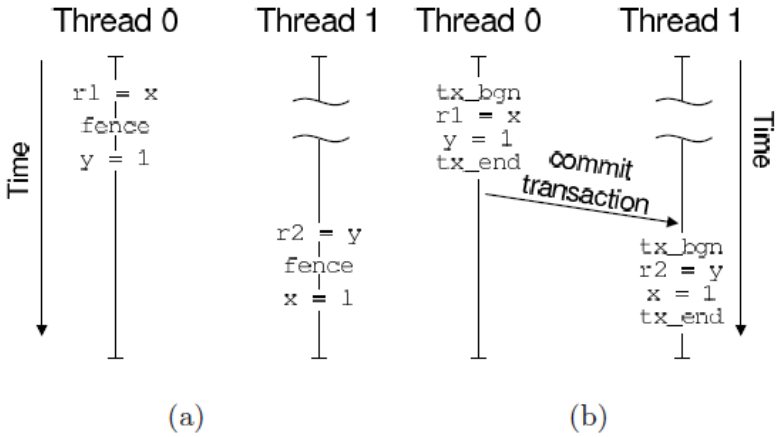
Postoje i (manje ili više uspješne) realizacije paralelizacije u slučaju kada gost ima manje relaksirani (jači) memorijski model u odnosu na domaćina. Kao i kod automatske paralelizacije sekvencijalnih programa, tada se isto može primijeniti sinkronizacija ogradom (fences) ili HTM, ali sugeriraju se i **hibridni pristupi** (primjena fences i HTM-a).

Slika 10. prikazuje korektnu implementaciju x86 emulacije na POWER arhitekturi (na primjeru sa slike 10.). Budući da je x86 arhitektura manje relaksirana od POWER arhitekture, **optimizator dodaje instrukcije za sinkronizaciju ogradom** (a) **ili za početak i kraj hardverske transakcije** (b).



Slika 10. Korektna implementacija primjera sa slike 9.; Izvor: [1]

Treba napomenuti da je optimizatoru jako teško utvrditi da li u nekom konkretnom slučaju (ne)treba ugraditi instrukcije za fence / transakciju. Npr. u primjeru sa slike 11., gdje su se dretve izvele sekvencijalno, instrukcije za fences / transakcije su se izvele nepotrebno.



Slika 11. Fences / transakcije su se o ovom slučaju izvele nepotrebno; Izvor: [1]

Rekapitulirajmo ukratko rezultate do kojih je došao R. Natarajan 2015. u [1]. On je analizirao primjenu fences ili transakcija (realiziranih pomoću HTM-a, konkretno na Haswell procesorima), te primjenu (vlastite) hibridne tehnike. Kako je već prikazano, analizirao je dva problema, koja je rješavao na sličan način:

- korektna emulacija softvera na različitim ISA arhitekturama, u paralelnom načinu rada
- (automatska) paralelizacija sekvencijalnih programa.

Njegovi rezultati pokazuju da je primjena transakcija ponekad bolja od primjene fences. Ako je veličina transakcije dovoljna da amortizira dodatne troškove kod transakcije (koji su relativno fiksni), i ako je učestalost konflikta kod dretvi relativno mala, tada su transakcije bolje od fences. Autor sugerira primjenu hibridne tehnike, koja primjenjuje fances ili transakcije, ovisno od karakteristika aplikacije.

5. POBOLJŠANJA SPECIFIKACIJE ARM ARHITEKTURE

R. Natarajan nije prvi koji je (2015. u [1]) analizirao probleme vezane za memorijski model relaksirane konzistencije. Jedan od značajnih radova je iz 2012. ([2]), pod naslovom:"A tutorial introduction to the ARM and POWER relaxed memory models". U tom radu, koji ima 50-ak stranica, tri autora sa sveučilišta (INRIA i University of Cambridge UK) analiziraju ponašanje prvenstveno ARM i POWER arhitektura, koje imaju više relaksirane memorijske modele u odnosu na Intel i SPARC arhitekture. Između ostalog, autori naglašavaju da bi **specifikacija** ponašanja kod memorijskog modela relaksirane konzistencije **u većini ISA arhitektura trebala biti puno preciznija, zasnovana na matematičkim modelima.**

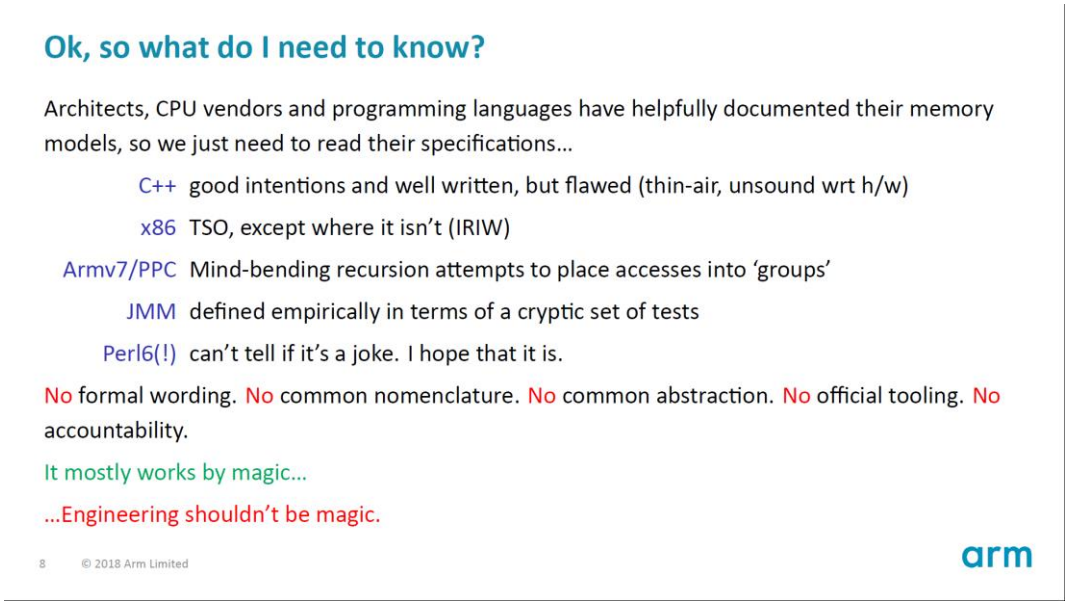
Autori rada [2] su nastavili raditi na tome. Jedan od značajnih radova je iz 2016. ([3]), pod naslovom: "Modelling the ARMv8 architecture, operationally: concurrency and ISA". Taj rad radila su prethodna tri autora, zajedno s još četiri autora sa sveučilišta i jednim arhitektom iz firme Arm. Iz samog naslova se vidi da je ovdje fokus bio specifično na ARMv8 arhitekturi. Vrlo pojednostavljeno (i neprecizno), autori su dali dva semantička modela koji služe za preciznije specificiranje ponašanja ARMv8 mikroarhitekture u konkurentnom radu:

- **Flowing Model**, koji je bliži konkretnoj mikroarhitekturi
- apstraktniji **POP (partial-order propagation) Model**
- te matematički dokaz da POP model dobro apstrahira FM.

Poboljšanju specifikacije ARM arhitekture naročito je doprinijelo istraživanje opisano u radu [4] iz 2018.: "Simplifying ARM concurrency: multicopy-atomic axiomatic and operational models for ARMv8", koje je radilo šestero autora, od kojih petoro onih koji su radili [3].

Kako je rečeno, jedan od autora rada [3], ali i [4], bio je i arhitekt iz firme Arm. U firmi Arm on je, između ostalog, koautor ARMv8 memorijskog modela. Taj Arm arhitekt je u prezentaciji [5] iz 2018., "Formalising the Armv8 memory consistency model", široj publici ("informatičkoj", a ne "matematičkoj"), prikazao **napore i uspjehe na poboljšanju specifikacije modela konzistencije memorije kod ARMv8 arhitekture.**

Između ostalog, istaknuo je da su mnoge dosadašnje specifikacije arhitektura manjkave i naglasio je da su znanstvena istraživanja omogućila kvalitetnije specificiranje ARMv8 arhitekture. Slika 12. prikazuje taj slajd iz njegove prezentacije.



Slika 12. Engineering shouldn't be magic; Izvor: [5]

Spomenimo još nešto vezano za hardversku transakcijsku memoriju (HTM). U wikipedijinom članku https://en.wikipedia.org/wiki/AArch64 pri kraju piše:

In May 2019, ARM announced their upcoming *Scalable Vector Extension 2 (SVE2) and* ***Transactional Memory Extension (TME)****.*

Podsjetimo se da je prvi visokotržišni procesor s HTM-om bio Intelov Haswell (2013.), kroz tehnologiju **Transactional Synchronization Extensions (TSX).**

Nažalost, Haswell i nasljednik Broadwell imali su bugove u TSX-u. Niti danas situacija nije sjajna, kako se vidi na: https://en.wikipedia.org/wiki/Transactional_Synchronization_Extensions :

However, Intel 10th generation Comet Lake and Ice Lake CPUs, which were released in 2020, ***do not support TSX/TSX-NI*** *… including both HLE and RTM.*

Međutim, možda ipak ima nade s Intelovim procesorima i HTM-om, jer pri kraju navedenog članka piše:

In Intel Architecture Instruction Set Extensions Programming Reference revision 41 from October 2020,[30] a new ***TSXLDTRK instruction set extension was documented and slated for inclusion in the upcoming Sapphire Rapids processors****.*

Nešto slično se možda desilo i s IBM PowerPC procesorima. Na https://en.wikipedia.org/wiki/Transactional_memory piše:

Available implementations ... IBM POWER8 and 9, ***removed in POWER10*** *(Power ISA v.3.1)*

Nadamo se da će (i) na temelju ovih iskustava firma Arm pažljivo i uspješno implementirati HTM u ARM arhitekturi!

Za kraj - kako je Apple uspio da programi pisani za Intel dobro rade na njegovom M1 čipu ARM arhitekture? Vjerojatno je točan odgovor onaj koji se može naći (i) u blogu na: https://www.infoq.com/news/2020/11/rosetta-2-translation/ :

… While byte ordering is not a problem for the transition from x86 to ARM, another issue related to memory, namely the memory consistency model total store ordering (TSO), could hamper performance in this case. ***To prevent this from happening, Apple added support for x86 memory ordering to the M1 CPU, as Robert Graham noted on Twitter.***

A ovo je izvorno rekao gore spomenuti Robert Graham na Twitteru (Nov 26, 2020):

So Apple simply cheated. They added Intel's memory-ordering to their CPU. ***When running translated x86 code, they switch the mode of the CPU to conform to Intel's memory ordering.***

6. ZAKLJUČAK

Procesori ARM ISA arhitekture ne koriste se više "samo" za mobitele i tablete, nego i za serverska računala (pa i superračunala), te laptop / desktop računala. Koriste se i u industriji, u proizvodnim procesima, a i kao ugradbeni čipovi u ostale proizvode, npr. za ugradnju u IOT uređaje i u vozila (posebno autonomna vozila).

I Apple je kod Macintosh računala prešao sa Intel arhitekture na ARM arhitekturu. Veliki izazov kod tog prelaska bio je - kako omogućiti da se binarni kod pisan za Intelovu ISA arhitekturu x86-64, izvršava na procesoru M1 koji ima ARM arhitekturu, tj. (ako je moguće) bez mijenjanja izvornog koda.

Općenito, želja je da se programi pisani za jednu ISA arhitekturu mogu sa što manje napora izvršavati na računalima koja imaju drugačiju ISA arhitekturu.

Problem je u tome što različite ISA arhitekture mogu imati dosta različite memorijske modele. Naročito je značajan različit stupanj relaksiranja konzistencije memorije.

Lakše je napraviti da program pisan za više relaksiranu arhitekturu (npr. ARM ili POWER) radi na manje relaksiranoj arhitekturi (npr. x86 ili SPARC). Obrnuto je teže, baš kao što je slučaj s Appleovim M1 procesorom ARM arhitekture, koji treba izvršavati (i) programe pisane za x86 arhitekturu.

Jedan od problema je da ISA arhitekture (ali ne samo one) nisu uvijek precizno definirane. Kako kaže ARM arhitekt u prezentaciji [5]: ***It mostly works by magic … Engineering shouldn't be magic.***

Srećom, u zadnjih nekoliko godina su znanstvena istraživanja, te timski rad matematičara i CPU arhitekata, omogućili kvalitetnije specificiranje ISA arhitektura, uključujući i ARMv8 arhitekture.

Literatura:

1 Natarajan, R. (2015):Leveraging hardware support for transactional execution to address correctness and performance challenges in software, doktorska disertacija, University of Minessota

2 Maranget, L., Sarkar, S., Sewell, P. (2012): A tutorial introduction to the ARM and POWER relaxed memory models, University of Cambridge UK, INRIA

3 Flur, S., Gray, K., Pulte, C., Sarkar, S., Sezgin, A., Maranget, L., Deacon W., Sewell P. (2016): Modelling the ARMv8 architecture, operationally: concurrency and ISA, University of Cambridge UK, University of St Andrews UK, INRIA, ARM Ltd.

4 Pulte, C., Flur, S., Deacon W., French, J., Sarkar, S., Sewell P. (2018): Simplifying ARM concurrency: multicopy-atomic axiomatic and operational models for ARMv8, University of Cambridge UK, University of St Andrews UK, ARM Ltd.

5 Deacon, W. (2018): Formalising the Armv8 memory consistency model, prezentacija, OpenSHMEM workshop, Baltimore MD

Podaci o autoru:

Zlatko Sirotić, univ.spec.inf.

ISTRA TECH d.o.o., Pula

e-mail: zlatko.sirotic@istratech.hr

Autor radi na informatičkim poslovima od 1984. godine, uglavnom u poduzeću ISTRA TECH d.o.o., Pula (ISTRA TECH je novo ime poduzeća Istra informatički inženjering, osnovanog 1990. godine). Oracle softverske alate (baza, Designer CASE, Forms 4GL, Reports, JDeveloper IDE, Java) koristi oko 25 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 - predavač, uglavnom na kolegijima Baze podataka 2 i Informatički praktikum.

embarcadero | Whitepaper

RAD STUDIO

GUIDE FOR MANAGERS

by Stephen Ball - January 2021

TABLE OF CONTENTS

Introduction

The world of software development thrives on innovative concepts like Object Orientated Programming (OOP), Agile Development, Continuous Integration (CI), DevOps, Low-Code, Enterprise & Micro Services, UI / UX design, and many more.

There is arguably one key concept behind many of these buzzwords, one that has seen a huge resurgence in recent times. It continues to heavily influence the modern tooling and processes used in software development today, and is now being claimed by a broader set of products covering a host of innovative approaches. That term is **Rapid Application Development (RAD)**.

With big software firms like Microsoft, Google, Apple, Amazon and Salesforce, to name just a few, all evangelising RAD approaches, this paper will explore how RAD is evolving, and how **RAD Studio®** today continues to innovate tooling and frameworks and supports the latest development practices and protocols.

Who Should Read This Paper

This paper will have a broad appeal to CTOs and leaders of software development teams, and is written for those following or evaluating market trends and possible solutions to use for both desktop and mobile applications.

If you have legacy **Delphi®** or **C++Builder®** code, this paper is especially important as it will help you understand how coding has evolved, and evaluate and harness the value in your existing codebase.

RAD Studio® in The Evolution of Software Development

From RAD and ASD to Agile

Rapid Application Development has seen continuous innovation and a resurgence recently. RAD has evolved a long way since its popularity exploded when groundbreaking developer tools like **Delphi®** and **Visual Studio** were launched in 1995 and 1997. These RAD tools, combined with **Adaptive Software Development (ASD)** practices that became popular at the time, enabled the acceleration of **Business Process Re-Engineering (BPR)** that coincided with Microsoft Windows' domination of the PC market.

Early exponents of RAD adopted ASD to benefit from shorter product life cycles, with early feedback from customers based on prototypes that reduced risk to overall delivery and enabled early versions to return value sooner into businesses that used them. The evolution of RAD-based development processes over the last 26 years, and the complementary assets created, for example, around software design, testing and source code control, has largely culminated in what we know today as **Agile Development**.

Market Trends Impacting Software Development

Alongside the evolution of development processes, the field of software development has navigated tectonic changes in the last 15 years.

Here are a few of them:

- New mobile platforms and hardware have changed the way we access and process information and collaborate
- Mobile has overtaken desktop in terms of Internet-connected devices
- App stores dominate the market for mobile application deployment

- 32-bit is giving way to 64-bit as both the Apple App Store and Google Play have switched to 64-bit-only for new apps
- Bring Your Own Device (BYOD) policies have brought about a shift in device ownership and data access. This has introduced new deployment and security challenges due to a wider mix of operating systems and devices (and their varied capabilities) that need to be addressed by developers.
- PaaS (Platform-as-a-Service) has enabled a new rethinking of hardware ownership and fault tolerance issues
- PaaS has also enabled more stable SaaS offerings with low-recurring-cost models
- Users today expect systems to have high interoperability. The drive towards Open Innovation is such that API's are expected to coexist or integrate with other systems enterprise customers have in place.
- Microservices have grown as a way to rapidly enable new capabilities within systems (e.g. language translation services, push notifications, weather data, etc.), with HTTPS as the default transport mechanism and JSON (first standardised in 2013) as the default data structure
- Containerization and DevOps have changed established approaches to the development and deployment of software, leading to rapid and continuous deployment models
- Data storage has exploded with the onset of NoSQL and traditional SQL data storage powering systems. As a result, data is often referred to as the new oil.

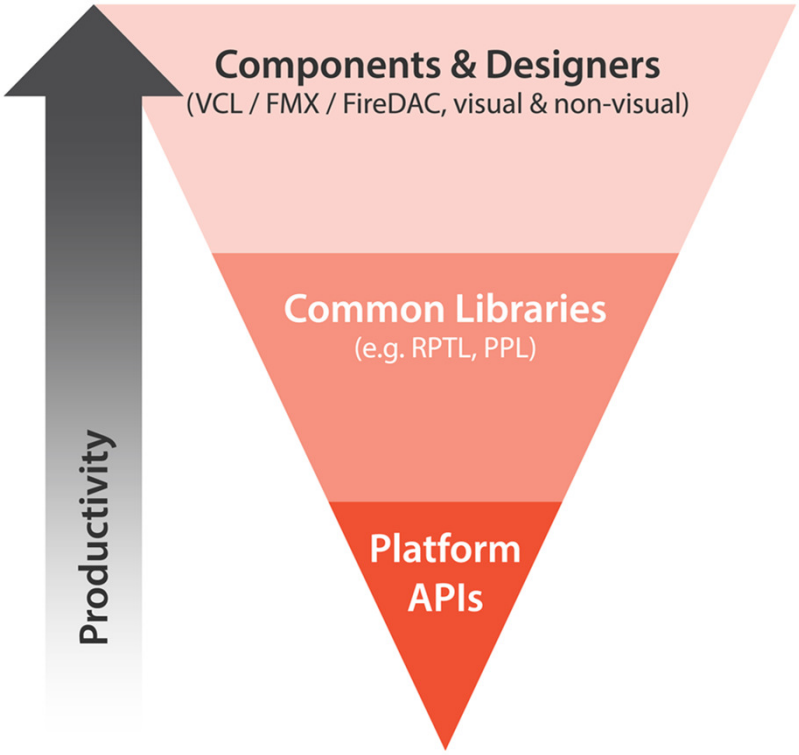
While Microsoft Windows still has a dominant place in today's landscape, this overview of market trends highlights the environment for software development today is more diverse than when RAD first emerge and Windows Desktop monopolized BPR requirements.

RAD Studio® Today

RAD Studio® uses a single code base that compiles to true native code on Windows, Linux, macOS, iOS and Android. This unique approach provides both speed and high flexibility in development, and rapid runtime speed. Time and again, the **RAD Studio®** approach proves to be **5x faster** than other languages and frameworks. With its single code base, **RAD Studio®** excels in ensuring faster times to market across all platforms through shorter development and testing times.

The original designs for the libraries and components in **Delphi®** have enabled **RAD Studio®** to provide long-term code investment security to developers for over 26 years. Indeed, the patterns are so strong they are easily identifiable in many languages and frameworks (like C# and .Net) that have followed since.

If you are not familiar with this approach, then suffice to say the true value components bring to RAD development is the abstraction of platform APIs into easy-to-use building blocks that take care of the low-level system calls. These components are underpinned by libraries that do the heavy lifting of enabling reusable core functionality (such as runtime libraries for file system, date time, screen info, etc.).



As market innovations and trends have emerged, the component and object orientated approach used in **RAD Studio®**, strongly supported by interfaces, has enabled the introduction of new features, platforms and capabilities, all while maintaining a high level of backward compatibility.

Cross-Platform Native Libraries by Design

One key contributor to the unique success of **RAD Studio®** in creating **true native code across multiple platforms** is the way interfaces are implemented in **Delphi®**.

The way a steering wheel can be used to steer a variety of different vehicles is an effective metaphor. If you know how it works, you can use the same tool to drive a truck or boat as easily as you drive a car.

In other words, the interface is the same for the driver, but what happens under the hood can be radically different. In the same way, developers can write code that checks for and employs common interfaces, (e.g. the accelerometer, compass, camera, etc) that are then provided with platform-specific implementations.

Now let's see how **RAD Studio®** has enabled developers to continuously ride the waves of **process** and **platform innovations** through examples

Process-Centric Innovation - The Example of Unit Testing

Unit testing enables continuous testing of code programmatically to help reduce regression issues and eliminate bugs from the code base. Through open-source projects that have integrated with the **Open Tools API** of the **RAD Studio®** IDE, developers the world over have been able to see how their code performs against tests in real time. Unit testing can also be run as part of a continuous build process, with the results going back into the external systems.

The introduction of unit testing around the year 2000 also encouraged a number of developers over time to adopt different design patterns such as **Model View Controller (MVC)** and **Model View-View Model (MVVM)** to enable easier testing for the logical parts of their systems. While these approaches haven't forced immediate changes to existing code, IDEs, languages or frameworks, they have driven the evolution of how code is written and also seen the introduction of additional frameworks such as **Spring4D** to support complimentary approaches such as dependency injection.

Product-Centric Innovation (It's More Than Just FMX)

While unit testing is about how code is written and which processes are followed, other innovations require major product enhancements. September 2011 saw the introduction of **FireMonkey (FMX)**, the cross-platform framework that at first glance mirrors in many ways the **Visual Component Library (VCL)** that enables RAD development for Windows.

Under the hood of FMX, the new components, using the existing updated runtime, support cross-platform coding with a single code base that works natively across all platforms. This is no small feat, and the FMX framework has evolved over the last decade to provide the most comprehensive framework for **single-code native development** on **iOS**, **macOS**, **Android**, **Linux**, and **Windows**. This has been supported with additional compilers added to the product, along with flexible build configurations to support compilation and packaging directly for app stores from inside the IDE..

Additionally, new features have been introduced, such as **Visual Live Bindings**, that enable the RAD binding of user interfaces to data and object models alike, while other features have evolved, like **FireDAC**, which enables broad database connectivity components that work cross-platform. The component approach used by **RAD Studio®** means database code that has worked on Windows for decades is now in reach of mobile platforms, enabling new life to be rapidly injected into existing code.

RAD Studio® is also using **Open Innovation** to enable fast-paced improvement. The new compilers are built using the LLVM project, providing the fastest runtime performance for each possible platform, and the IDE has recently introduced support for the **Language Server Protocol (LSP)**. This standards-based approach is also laying the foundation for a richer developer experience.

RAD Studio continues today to offer the **best available compilers**, the **deepest Window API integration** of any modern toolchain, and a proven history of enabling code to port rapidly to the latest platforms as they become available.

Innovation Through Partnership

Some innovations have been realized through partnerships. **RAD Studio®** was the first tooling anywhere in the world to enable the **Microsoft Desktop Bridge**. This opened the door to the Microsoft store, and also to the latest enterprise deployment mechanisms for traditional Windows applications. Since then, early support for Microsoft's new **Edge** browser has also been added along with the latest MSIX app packaging.

Innovation Through Acquisition

Embarcadero, now owned by **Idera Inc.**, is part of a rapidly growing technology group. This has opened the door to the adoption of many more tools to help developers work faster. Tools like **Ranorex**, **Sencha** and **Aqua Data Studio** were added to the **RAD Studio® Architect** product, enabling developers to automate user interface testing with the powerful Ranorex tool, develop faster with simplified access to manage a wide range of databases using Aqua Data Studio, and bring RAD to JavaScript through Sencha Architect.

In addition, a strong third-party ecosystem of component vendors create frameworks for using **RAD Studio®** to provide the compiled backend, with the same language and code to power web applications.

RAD Studio® continues to evolve with an open framework for enhancing code and an active third-party ecosystem providing components and add-ons that support the core features of the IDE, and is now in a renewed growth stage within an exciting developer-technology-centric group.

Modernize or Rebuild?

If you have existing code built in **Delphi®** or **C++Builder®**, the first questions must be about what use case it is being used for, and whether you will still be serving that use case going forward. If you will, then there is value in the code. The next question is how to get the logic from where it is today to where it can provide future value.

The key points when assessing your next steps are:

- Working out what your future system architecture will look like, (e.g. whether you need to enable remote/mobile access)
- Identifying and managing risk and cost
- Looking at timelines and the future cost of ownership
- What phases this will require

Let's look at a few of these points now.

Looking for Safe, Quick Wins That Buy You Time

One challenge for the continued use of legacy code bases is that often, coding practices in the company have evolved a long way since the first version was released. While the code and project works, there may be parts your team would approach in a different way if it were starting fresh today.

If part of the plan is to modernise your practices with code, then the safest approach is to use a phased update of the existing code. This keeps a customer-facing product always available with the latest options, while also enabling the update towards new practices.

User Interface Testing as a Migration Approach

Returning to code testing, one common desire is to add testing to the code to ensure it tests faster and more efficiently as part of the development process. While unit testing is one part (and can easily be added for new code or new standalone functions and classes), another popular approach is to look at **User Interface Testing**. This is where **Ranorex** (part of the Architect edition) truly complements **VCL** development today, and provides an easy-to-add foundation for checking future code changes. RAD developers create UI tests for their product to check their applications prior to UA testing. Because they work on the UI layer, they leave the developer free to completely rewrite the code underneath if they so desire, and verify everything by making sure the user experience hasn't changed.

These tests are also a great way to check the software when migrating from old versions of **RAD Studio®** to newer versions. Because they work on the Windows control handle, they even allow you to rearrange the UI without breaking the tests. Furthermore, common repeatable actions (such as logging into a system) can be saved and added to multiple tests to enable extended flexibility and speed up test creation. Because the tests are built onto the product after it's created, they can also be passed to contributors outside of the development team to create, freeing up valuable resources.

Adding in The Latest Windows 10 Features

Thanks to the component design of **RAD Studio®**, it is possible to rapidly enhance any existing application's UI with a few simple steps. Uniquely, the **RAD Studio®** components also make it possible to maintain support for older versions of Windows, ensuring you don't block specific users upgrading if they are stuck on versions older than Windows 10.

One quick win in **RAD Studio®** that enables Windows 10 support for high-DPI and multiple monitors running different DPI's is the replacement of the traditional **TImageList** with a new **TVirtualImageList** and **TImageCollection**. By encapsulating the latest per-monitor high-DPI support APIs, these components provide a no-code update that can serve the correct image based on the screen resolution for each monitor.

Today, RAD Studio offers the deepest integration of Windows 10 API's on the market, making it fast and easy for developers to access WinRT, COM and Win32 features.

Designed to Save Money and Time

Another big advantage of **RAD Studio®** is that it reduces manpower requirements, which means keeping fewer skill sets and code bases updated. This can be a great time and cost saver, especially when trying to build and deploy to multiple platforms at the same time. **RAD Studio®** is seeing a significant surge in interest because of this. An IoT boot camp was held in 2017 with developers from over 180 countries signing up to the week-long online conference. There are also great online resources today like **LearnDelphi.org** and the **EmbarcaderoAcademy.com**.

Although there is a large global **RAD Studio®** community, demand in certain areas can at times outstrip available developers. To cope with this, a growing number of companies are onboarding and upskilling C# developers to gain **RAD Studio®** experience. Two to four weeks typically suffices to get comfortable with **RAD Studio®** and the frameworks, depending on the developer. As C# was written by Anders Hejlsberg, who originally wrote **Delphi®**, there is a strong influence in the way C# mirrors the **Delphi®** approach, making this a feasible path.

The architecture of **RAD Studio®** has proven the test of time and continues to lead the way in innovative cross-platform solutions. Thanks to the component model that follows the best OOP practices, code is easily modernised to take advantage of market changes with minimal effort, providing exceptional long-term investment. Upskilling existing developers to **Delphi®** is easily achieved, and as **RAD developers are 5x more productive** than they are with other frameworks, it provides exceptional long-term productivity gains for any team.

Best of Both Worlds - Why RAD Studio®

Evolution of Cross-Platform Development Tools and Approaches

Earlier in this paper we looked at innovations impacting software development over the last 26 years. Time has shown that any emerging technology typically takes around 10 years to mature from the constant changes of the early growth cycle as standards gradually solidify. The early days of any technology carry higher risk, so typically providers look for the lowest-risk approach to delivering long term solutions. This early market mindset has led to the launch of a wave of web-centric solutions as a way to reach into new devices. With a web browser on every device, it was a quick win to test and establish full market needs.

An early version of **RAD Studio®** support for the Web came in the late 90's when IntraWeb was added to **RAD Studio®** to enable development for multiple web formats, including early browsers on phones and PDA's (anyone remember WAP?). Since then, web standards have improved, JavaScript has risen in popularity, and libraries have emerged to provide faster development and multi-device support that are far closer to the desktop browser's code standards.

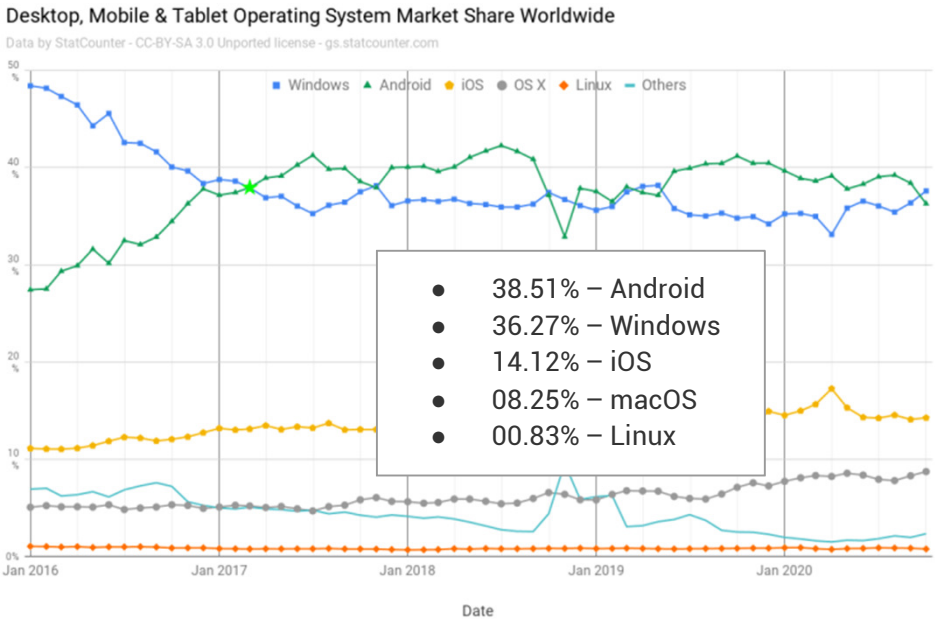
While **IntraWeb** and other **RAD Studio®** components like **TMS Webcore** have supported RAD web development in **Delphi®**, many standalone HTML and JavaScript frameworks have emerged as well. A leading example of this JavaScript-based approach in the RAD world is **Sencha**. Sencha offers **best-in-breed JavaScript-based web components** with properties to set and link to, enabling the rapid development of rich web applications. Sencha is used by and embedded with many solutions around the world, including some from the likes of Oracle, and since joining the Idera Group, it is included in the **RAD Studio® Architect** product.

The flexibility to both develop in web languages and compile native code (which is faster to execute) provides a rich set of capabilities for **RAD Studio®** users. However, as there are other options in the market as well, it would be useful at this point to summarise the differences between these approaches. Let us also look at the market each platform must reach.

Impact of Mobile on Business Processes

While mobile is not the primary platform for business applications, it's receiving intensive focus thanks to a new wave of **Business Process Reengineering** that is taking advantage of remote data capture capabilities. In a drive towards increased automation, many tasks that would traditionally have been completed centrally are being distributed to field workers to reduce paperwork and shorten process workflows. A good example of this would be that of an engineer making a site visit and completing a log entry, including photos. Mobile services are able to grab additional key data points, such as geo-coordinates.

With the aim of enabling this new wave of BPR, approaches such as BYOD have been adopted primarily to improve user adoption, but have also been embraced by companies as a way of reducing logistics around workforce enablement. This has become possible thanks to the broad market penetration of smartphones.



Operating System Market Share as of November 2020

Source: https://gs.statcounter.com/os-market-share

Although Android accounts for more users than iOS, the broad adoption of both frameworks combined with BYOD approaches means software needs to reach both iOS and Android for successful rollouts. To make things more complicated, successful products for each platform adhere to different user interface design guidelines. Failing to adhere to these guidelines can cause user acceptance friction that lowers the chance of a successful rollout.

In short, to enable a positive mobile-centric BPR capability today, software engineers need to be able to **reach iOS and Android at the same time**, with a look and feel that matches each platform's idiosyncrasies.

Reaching Multiple Platforms

While web technologies provide a quick way to reach a new device and platform, this approach is limited by browsers and their capabilities, causing constraints for BPR. Serving web pages is also slow, causing a poor user experience. For the best reliability, speed, user experience and access to device features (especially where security permissions are required), applications need to be installed on the device.

Two approaches are available to create mobile applications: those that are fully compiled and native, and hybrid applications that run a local web app inside a shell that provides a browser with enhanced access to features of the device.

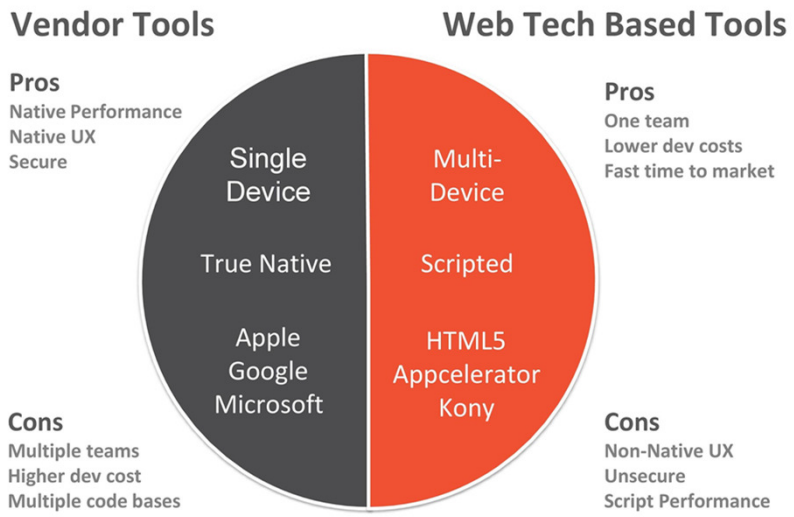
True Native Versus Hybrid Applications

True Native code is typically associated with vendor tools such as Xcode, Visual Studio and Android Studio. The upsides of True Native development are a native user experience, native speed and performance, and a high level of security as the code is compiled down to binary. Inversely, using vendor tools to achieve these benefits means having multiple code bases to develop and manage, and multiple skill sets and developers, which all lead to a higher development cost.

This **increased time**, **logistics** and **costs** are a reason many developers look to scripted/hybrid applications. Hybrid approach enables a single team of developers to create applications with a lower cost, that reach multiple platforms at the same time, but trade off security, performance and the native user experience.

The reason why **security and performance are low** in scripted/hybrid applications is because the scripted language needs to be interpreted at runtime. With the code executing at runtime, this provides a performance bottleneck and also creates a potential security hole for hackers to exploit. Additionally, the memory required for web applications is far higher than for native applications.

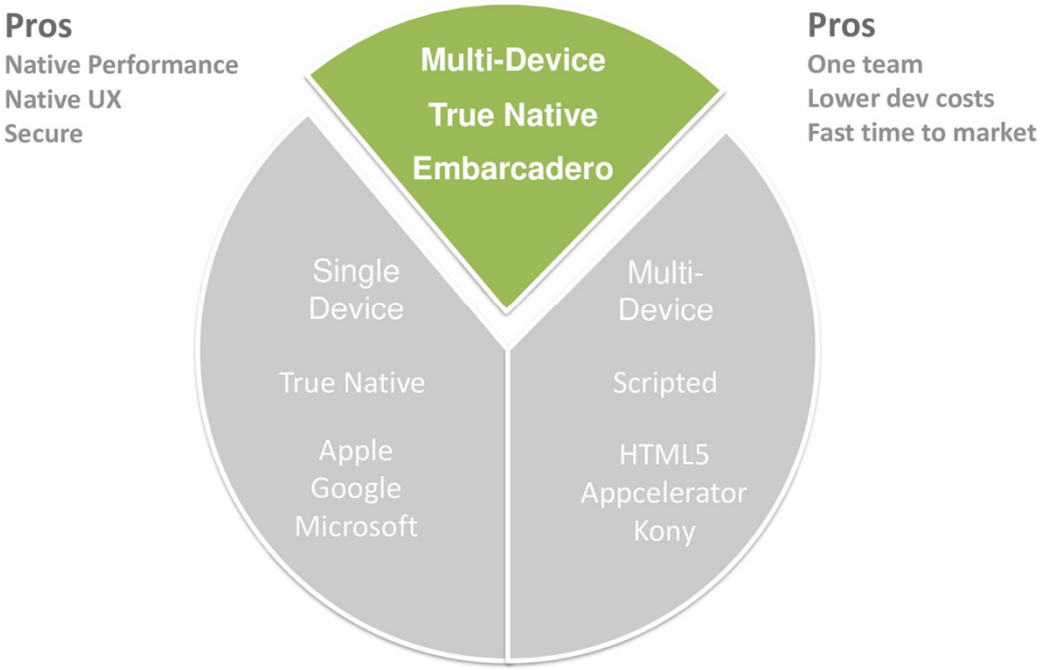
Finally, if your application is using a runtime such as .Net or JavaScript RunTime Webkit or JavaRuntime, this poses an additional security threat. Runtimes also come with considerable memory overhead, in part through the addition of a garbage collector for memory management.



No Compromise - The Best of Cross-Platform AND True Native

RAD Studio® brings a unique no-compromise approach to cross-platform development, enabling one development team to reach every targeted platform at the same time with a fully compiled True Native application, user experience, blazing performance, and the highest levels of security.

Using LLVM, every RAD Studio® application is compiled and highly optimized for each platform. This means RAD Studio® uses the same compiler for iOS and macOS that Xcode uses, and compiles to a lower level than Java to reach the same CPU and GPU access games developers target on Android while also avoiding the need to bloat memory with a garbage collector.



Launched in September 2011 as part of Delphi® XE2, the **FireMonkey (FMX) framework** has matured to offer the value of a single code base combined with the performance and security of a truly native user experience. Platform defaults for controls make it easy to achieve a platform-specific look and feel instantly, but still leave the developer in full control to customize where needed. High productivity features such as **FireUI** enable developers to immediately see (on any device) exactly how the UI design will look and behave as they develop, dramatically shortening development time.

Thanks to the use of a common compiler architecture, full language and framework support is available on all platforms. **RAD Studio®** FMX code can also run on Windows, macOS and Linux, providing even higher levels of productivity.

RAD Studio® additionally brings the ability to package applications ready for the notarization or deployment to the leading app stores directly inside the IDE, simplifying the entire build process.

How FMX Differs From Xamarin Forms

In May 2014, **Xamarin Forms** was launched in an attempt to provide an experience similar to FMX while using XAML. This provides Xamarin developers with a subset of controls for cross-platform development. Unfortunately, Xamarin falls short when compared with **RAD Studio®** in two key areas.

First, **language features are limited** depending on the platforms (e.g. generics can not be used on iOS meaning code often needs to be specifically written for each platform). Second, as Xamarin is based on .Net, it suffers from **inefficient memory management** due to a Garbage collector being added to iOS and a second one running on Android (both the .Net and a Java one).

Enterprise Data and Remote Data Connections

While **RAD Studio®** provides powerful mobile and desktop application development options, the complete stack requires serving data to remote devices.

Over the years, **RAD Studio®** has introduced a range of methods for enabling remote access to data. Today, the FireDAC Components enable local or remote connectivity to over 15 traditional SQL and NoSQL databases (plus ODBC for any others), and has also been expanded by partners to enable direct access to **180+ enterprise and big data systems** (such as Jira, Salesforce, Microsoft Teams, Google Drive, eBay, Facebook, Slack, Twitter, Amazon Marketplace) via standard SQL, with many included as part of the Enterprise product.

This ability to rapidly link to multiple databases and enterprise data sources makes **RAD Studio®** an ideal middle tier. With **WebServices**, **DataSnap** (based on Midas to enable a session's base connections) and **RAD Server** (a fully RESTful MEAP with Docker configurations and inbuilt usage analytics and reporting), there are options for creating and integrating with a range of systems, and exposing data rapidly, often with zero code.

Low-Code Application Platforms and RAD

Gartner defines a Low-Code Application Platform, or LCAP, as "an application platform that supports rapid application development, deployment, execution and management using declarative, high-level programming abstractions such as model-driven and metadata-based programming languages, and one-step deployments. LCAPs provide and support user interfaces (UIs), business processes and data services".

A growth in interest around LCAPs primarily goes back to the need to increase business automation through software, with a focus on business process review and the creation of apps to support a changing business. To achieve their rapid deployment and cross-platform support, LCAPs use a form of hybrid application based on web technologies to deliver their software.

Low-code options in the market are often sold based on their promise of enabling "citizen developers", where anyone can make an app with a little training. While the theory is good, the reality dictates a **steep learning curve** of a **bespoke limited-scope product** and a **heavy reliance on the vendor** to provide third-party integrations and maintain them. Some systems do provide advanced web-based RAD tooling for more seasoned developers to go beyond the simple tasks once they have been upskilled in the frameworks and models specific to the LCAP. A word of caution: the citizen developer route can also leave a business's software highly fragmented without a clear application strategy in place, producing a long-term technical debt that is hard to shake or manage.

For many of the low-code platforms, the **costs substantially mount up** through ongoing **subscriptions** charged per user for accessing the apps. Additionally, Gartner cautions that many low-code vendors earn substantial revenue from professional services, suggesting that professional developers who know the frameworks, rather than citizen developers, are required to support the tools. This means the true cost of LCAP ownership is substantially higher than initially expected for most.

With the history **RAD Studio®** brings to low-code development and support for business process review, rapid prototyping of applications and an open ecosystem for adding any connectivity required (either off-the-shelf or independently), the question for those looking at LCAPs is where the investment is better placed. Is upskilling a niche skill set belonging to a specific LCAP provider, or for enhancing existing team members with knowledge of **Delphi®**? With the latter able to deliver native applications that enable faster and more secure output across the leading platforms, it makes a compelling case to look at **RAD Studio®**.

RAD Studio® Today - Investing in Your Future

RAD and DevOps

The current industry trend for development process enhancement is DevOps. The understanding that developer and deployment teams need to work together to achieve a successful outcome is driving innovation around deployment configurations and setup.

The focus of DevOps could be a white paper in itself, but sometimes a picture does much more than words can. As we touched on earlier in the paper, because of the open ecosystem linking into **RAD Studio®**, many developer productivity tools are available in all stages of development. A subset of these tools is highlighted in the "**Power of RAD Studio®**" infographic on the next page.

embarcadero

The Power of RAD Studio

Code once, Natively target

CODE

Languages

Delphi, C++

UI Frameworks

VCL (for Windows), FireMonkey (Universal Cross Platform UI), Sencha (Web Components), HTML, JavaScript, Intraweb*, TMS WebCore*, uniGUI

Cross Platform RunTime Frameworks

AppTethering, Parallel Programming, XML, REST & JSON, RTTI, Graphics, RAD Server, Web Broker, DataSnap, SOAP

Data

FireDAC Univeral Database Access

Advantage Database, Firebird, IBM DB2, Informix Software, InterBase, MariaDB, Microsoft Access, Microsoft SQL Server, mongoDB, MySQL, Oracle Database, PostgreSQL, SQLite, SAP SQL Anywhere, ODBC Generic ODBC Driver, teradata.

Enterprise Connectors

70+ Sources included with Enterprise and Architect Edition. Up to 180+ sources via subscription*, the largest number of supported data sources in the industry.

PayPal, Facebook, MailChimp, Microsoft Teams, Twitter, Slack, Salesforce & Force.com, Wordpress, ebay, JIRA, DocuSign, Office, SurveyMonkey, Trello

BUILD

Database Management

AQUAFOLD Aqua Data Studio, Data Explorer, IBConsole

Platforms

Windows, Linux, macOS, iOS, Android, Web

Version Control Integration

Mecurial, Git, SVN, Github, Assembla

Build Systems

CMake, MSBuild, FinalBuilder*

TEST

Unit Testing (Automated)

Dunit, DUnitX, TestInsight* (IDE plugin), Sencha Test, Delphi Mocks*

UI / UA Testing

TestRail, Ranorex, TestComplete

Code & Runtime Analysis

TMS FixInsight*, CodeHealer*, AQTime Pro*, Pascal Analyzer*, Deleaker*

INTEGRATE

Monkey Builder*, Jenkins*, Continua CI*, RAD Studio IDE Integrated Deployment, Configuration Management, Azure Pipelines

DEPLOY

Microsoft App Store, Google Play Store, iOS App Store, macOS App Store, Docker

Web Server: APACHE, Microsoft IIS

OPERATE

FireDAC Database Services

Backup, Restore, Database Validation

MONITOR

RAD Server Analytics, CodeSite, FireDAC - Tracing and Monitoring with FDMonitor, EurekaLog*, MADExcept*, Grijjy Logger*, SmartInspect

* RAD Studio is also supported by a range of open-source and commercial offerings. A few are featured in this infographic for illustration purposes. Always check the feature matrix for the latest on what is included with RAD Studio. Features subject to change.

All product names, logos, and brands are the property of their respective owners. All companies and products used in this infographic are for identification purposes only. Use of these names, logos, and brands does not imply endorsement.

Investing for Both Present and Future Gains

RAD Studio® has arguably the best long-term return for code investment in any development tooling, and continues to focus on ensuring code is portable from the older version to the latest.

The roadmap for the future, regularly updated and shared with the **Embarcadero** community, shows the adoption of the latest innovations around Apple Silicon, Microsoft platform API's and AppStore changes, and a focus on developer experience.

Summary

As we have seen in this paper, the desire for RAD development is growing with many choices in the market for hybrid or native development. **RAD Studio®**'s unique True Native approach brings the fastest speed, performance and security to mobile development built on the same compiler technologies used by native-only tool vendors while providing the value of a single-source cross-platform code base.

RAD Studio® continues to develop and embrace market trends, providing code security that is second to none in the market, ensuring the highest possible return on investment for software development.

In response to the leading trends identified, **RAD Studio®** enables RAD development for middle tiers and microservices, deployment to the leading desktop and mobile app stores, and wide-ranging data connectivity to both SQL and NoSQL databases as well as leading enterprise systems.

RAD Studio®'s ability to offer a unique solution in the market, along with a framework that has already proven able to expand without compromise to Windows, macOS, iOS, Android and Linux, points to a bright and secure future for developers choosing **RAD Studio®**.

Beyond This Paper

While not strictly the scope of this paper, it is worth adding a few notes for how developer productivity has been enhanced in RAD Studio in recent years.

RAD Studio now includes compilers, enabling native app development into Windows (32-bit and 64-bit). Linux (64-bit), Android (32-bit and 64-bit) iOS (64-bit) and macOS (64-bit). All compiled from a single code base, reducing testing and management costs across all target platforms.

Developers connecting into the latest version of the IDE will benefit from many developer productivity updates in the IDE too, including:

- Integrated **source code repository support** for Git, SVN and Mercurial
- A **modernised IDE layout** and optional Dark Style
- **Larger memory support** to support compiling even larger projects
- **LSP (Language Server Protocol) support** that enables faster coding with background processing of Code Completion, and error insight tasks
- And more…

For further information, please contact an Embarcadero sales representative.

The Embarcadero Way

Single-source, native multi-platform software development

The RAD Studio IDE and frameworks enable you to write your code in modern Delphi or C++ languages, and compile your single code base to natively target Windows, Linux, macOS, iOS, and Android.

Design it, Build it, Run it - Today!

embarcadero
An Idera, Inc. Company