Zbornik radova Fakulteta tehničkih nauka, Novi Sad

UDK: 004.42:004.738.12
DOI: https://doi.org/10.24867/01BE44Pavicic

POREĐENJE ARHITEKTURA KLIJENTSKIH WEB APLIKACIJA

COMPARISON OF WEB FRONT-END ARCHITECTURES

Dalibor Pavičić, *Fakultet tehničkih nauka, Novi Sad*

Oblast – Primenjene računarske nauke i informatika

Kratak sadržaj –*Izbor odgovarajuće arhitekture i radnog okvira za novi ili postojeći projekat može da predstavlja izazov zbog velikog broja faktora koje je potrebno uzeti u obzir. U ovom radu je izvršeno poređenje poznatih arhitektura za klijentske web aplikacije. Definisan je konceptualni model evaluacije klijentskih radnih okvira koji je potom primenjen na evaluaciju tri popularna radna okvira: Angular, React i Elm. U svakom od navedenih radnih okvira implementirana je i testna aplikacija. Na kraju su date smernice za odabir najadekvatnije arhitekture i radnog okvira u zavisnosti od konteksta.*

Ključne reči: *Evaluacija, Poređenje, Klijentski radni okvir , JavaScript, SPA*

Abstract *–Choosing the right architecture and framework for a new or existing project may be a challenge due to the many factors that one has to consider. This paper contains a comparison of popular architectures for client web applications. The conceptual model of the front-end frameworks evaluation was defined and applied to the evaluation of three popular options: Angular, React and Elm. A demo application has been implemented by using each of them. The paper also provides guidelines for selecting the most suitable architecture and framework for the given context.*

Keywords: *Evaluation, Comparison, Front-end framework, JavaScript, SPA*

1. UVOD

U poslednjih nekoliko godina došlo je do naglog razvoja klijentskih web aplikacija i prelaska sa tradicionalnog višestraničnog modela (*multi-page*) na jednostranični model (*single-page*). Jedan od glavnih nedostataka višestraničnog modela jeste loš odziv web stranica. SPA (*single page applications*) aplikacije u kombinaciji sa AJAX tehnologijom predstavljaju web aplikacije koje staju na jednu stranicu i imaju za cilj da obezbede fluidnije korisničko iskustvo.

Porast popularnosti SPA aplikacija doveo je do toga da se više procesiranja odvija na klijentskoj strani što je uslovilo razvoj programskih jezika za pisanje klijentskih aplikacija, prvenstveno JavaScript-a.

NAPOMENA:

Ovaj rad proistekao je iz master rada čiji mentor je bio doc. Milan Segedinac.

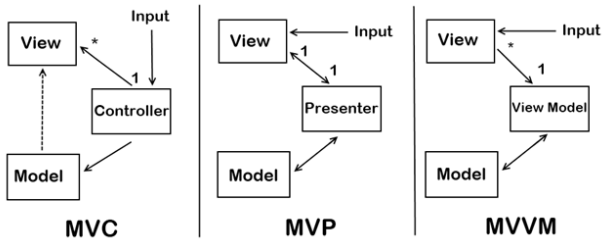
Nastali su novi radni okviri sa ciljem da olakšaju implementaciju i strukturiranje aplikacija i da obezbede gotova optimizovana rešenja za razvoj kompleksnih funkcionalnosti. Međutim, intenzivan razvoj JavaScript radnih okvira doveo je do tzv. „JavaScript zamora“ koji se javlja kada je od mnoštva alata potrebno izabrati jedan koji bi bio najpogodniji za specifičan projekat. Odatle i motivacija za ovaj rad koji ima za cilj da definiše smernice i olakša izbor odgovarajuće arhitekture i radnog okvira uzimajući u obzir sve relevantne faktore.

2. ARHITEKTURA KLIJENTSKIH WEB APLIKACIJA

Pod pojmom arhitektura klijentske web aplikacije podrazumeva se izbor suštinskih elemenata koji čine jednu klijentsku web aplikaciju. Ovde će biti reči o sledećim arhitekturama: MV* (*Model View**), FLUX i MVU (*Model View Update*).

2.1. MV*

U MV* arhitekture spadaju: MVC, zatim MVP (*Model View Presenter*) i MVVM (*Model View ViewModel*) (Slika 2.1).



Slika 2.1 *Šematski prikaz ključnih razlika između MVC, MVP i MVVM arhitektura*

MVC

MVC je verovatno najpopularnija arhitektura u poslednjih 30 godina koju koriste milioni programera u različitim projektima, nezavisno od programskog jezika. Glavna inovacija koju je MVC uneo u razvoj softvera jeste konačno razdvajanje podataka od njihove vizuelne reprezentacije što je do tada još uvek bio nedovoljno istražen koncept.

MVC definiše sledeće činioce:

- model (*Model*) – sadrži stanje aplikacije kao i domenske podatke
- prikaz (*View*) – predstavlja korisnički interfejs sa kojim korisnici vrše interakciju
- kontroler (*Controller*) – predstavlja vezu između modela i prikaza i obično je nadležan za orkestraciju toka komunikacije u aplikaciji

MVP

Ova arhitektura dolazi do izražaja u situacijama kada je potrebno koristiti iste prikaze ili ponašanja u različitim delovima iste aplikacije ili u potpuno različitim projektima. Naročito je korisna kod aplikacija koje treba da podrže različite platforme pri čemu je potrebno koristiti iste podatke, komunikacioni sloj i ponašanja ili u slučajevima kada se prikaz želi dinamički zameniti u toku izvršavanja ili kompajliranja aplikacije. Ključne razlike u odnosu na MVC sadržane su u sledećem:

- Umesto kontrolera se koristi prezenter.
- Veza između prezentera i prikaza nije 1 na prema više, kao što je slučaj sa kontrolerom kod MVC-a, nego je uvek 1 na prema 1.
- Najkorisnija implementacija MVP-a je u slučaju kada su prikazi pasivni jer je tada zamena prikaza jednostavna dokle god je ispoštovan ugovor između prezentera i prikaza.

MVVM

MVVM zadržava razdvajanje modela i prikaza kao i MVP, ali donosi i određene razlike u odnosu na druge dve MV* arhitekture. MVVM je sličan MVP-u s tim što za razliku od njega prikaz više nije pasivan.

Prva veća razlika u odnosu na MVP je to što umesto prezentera imamo model za prikaz (*ViewModel*). To je objekat koji služi kao most između podataka u modelu i prikaza. Ovaj objekat je u suštini odgovoran za izlaganje podataka iz modela na prikaz i pripremu podataka na način kako prikaz očekuje pa je logika modela za prikaz tesno povezana s onim što prikaz treba da prikaže.

Druga važna karakteristika modela za prikaz je činjenica da je veza između modela za prikaz i samih prikaza 1 na prema više. Dakle, jedan model za prikaz može da se koristi na više prikaza ili komponenti.

2.2. FLUX

Flux je arhitektura za klijentske web aplikacije koja je nastala u Facebook-u. Flux arhitektura definiše sledeće činioce: dispečer (*dispatcher*), skladišta (*stores*) i prikaze (*views*). Postoje i kontroleri, ali je njihova uloga nešto drugačija u odnosu na MVC. Kontroleri se obično nalaze na vrhu hijerarhije prikaza. Oni dobavljaju podatke iz skladišta i prosleđuju ih komponentama koje su ispod njih u hijerarhiji. Umesto MVC-a, Flux favorizuje jednosmerni tok podataka.

2.3. MVU

MVU (*Model View Update*), poznatiji kao "Elm arhitektura" predstavlja jednostavan, ali moćan šablon za strukturiranje web aplikacija. To je čisto funkcionalna arhitektura koja se vezuje za Elm, funkcionalni programski jezik za web, ali su ideje i koncepti koje ova arhitektura donosi primenljivi i u drugim programskim jezicima. Činioci koje definiše ova arhitektura su:

- model (*Model*) – sadrži stanje aplikacije kao i domenske podatke (slično MVC-u)
- prikaz (*View*) – funkcije koje generišu prikaz na osnovu modela
- akcije (*Actions*) – događaji koji nastaju kao posledica korisničkih interakcija
- ažuriranje (*Update*) – funkcije koje rukuju korisničkim interakcijama i generišu novo stanje aplikacije na osnovu prethodnog stanja i akcije

Novije arhitekture poput FLUX-a i MVU-a donose nove ideje i koncepte, ali se takođe oslanjaju na već poznate i dokazane principe iz prošlosti koji se susreću u MV* arhitekturama. Ove arhitekture su ugrađene u mnoge popularne radne okvire i biblioteke te stoga poznavanje ovih arhitektura može biti od velike koristi na realnim projektima, naročito prilikom donošenja odluke o tome koju arhitekturu odabrati za konkretan projekat.

Ne postoji univerzalno pravilo koje definiše kako treba da izgleda struktura projekta, ali svaka od opisanih arhitektura ima svoju namenu, kao i svoje prednosti i mane. "Arhitekturu softvera treba više shvatiti kao putovanje nego kao odredište, više kao tekući proces nego kao krut predmet za upotrebu" [**1**].

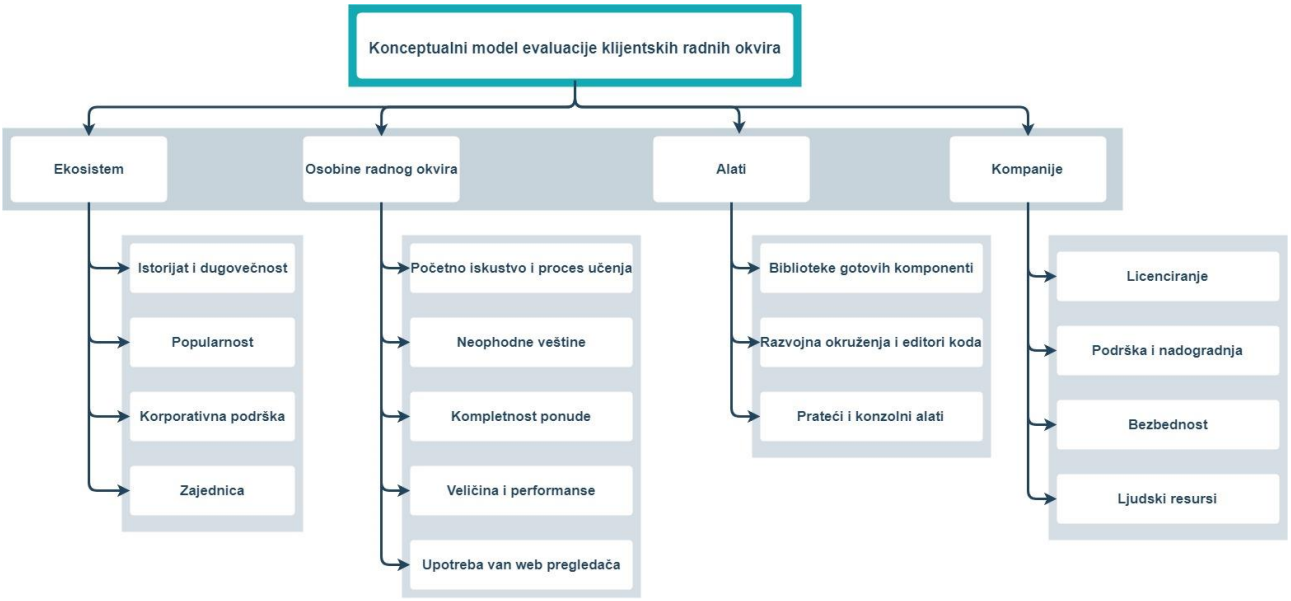
3. EVALUACIJA IZABRANIH BIBLIOTEKA I RADNIH OKVIRA

Bilo da je u pitanju potpuno novi projekat ili je potrebno migrirati ili modernizovati postojeću web aplikaciju, web programerima se nameće potreba za evaluacijom radnih okvira za web. Od mnoštva dostupnih i kvalitetnih radnih okvira ovde će se evaluacija ograničiti na sledeća tri:

- Angular
- React
- Elm

Postoji mnogo studija i članaka koji se bave neformalnim poređenjem klijentskih radnih okvira od kojih se većina, poput [**2**] i [**3**], bazira na subjektivnom mišljenju autora. Nešto formalnije poređenje, ali ipak ograničeno na specifičan domen je izvršeno u [**4**]. Ipak, postoje i pokušaji da se definiše formalna metodologija i opštiji kriterijumi koji su primenljivi u širem kontekstu kao što je slučaj sa [**5**] i [**6**]. U ovom radu su iskorišćeni, sistematizovani i sumirani neki od rezultata prethodno navedenih studija i istraživanja i na osnovu njih je definisan konceptualni model evaluacije klijentskih radnih okvira (Slika 3.1). Predloženi model je primenjen na analizu tri popularna radna okvira, Angular-a, React-a i Elm-a, uz pokušaj da se da odgovor na pitanje koji od njih koristiti za postizanje najboljih rezultata u razvoju date web aplikacije koju određena kompanija ili programer želi da napravi.

Prikazani konceptualni model se satoji od faktora i kriterijuma koji se mogu podeliti na kvantitativne u koje spadaju: popularnost, zajednica, ekosistem, veličina i performanse i kvalitativne kao što su: lakoća učenja, korporativna podrška, podrška za razvojna okruženja, licenciranje i bezbednost. Predloženi model ne daje numeričke ocene evaluiranim radnim okvirima jer je težinu i važnost pojedinačnih faktora kao i izbor kriterijuma potrebno prilagoditi jedinstvenom kontekstu kompanije ili razvojnog tima kao i projektu za koji se radi evaluacija.



Slika 3.1 *Konceptualni model evaluacije klijentskih radnih okvira*

3.1. Ekosistem

Ekosistem određene biblioteke uključuje sledeće faktore: istorijat razvoja, zajednicu, popularnost, uključenost korporacija kao i poznate aplikacije koje koriste tu biblioteku ili su napravljene uz pomoć nje [**7**].

Istorija i starost su korisni faktori jer daju jasniju sliku o tome kako je biblioteka započeta i kolike su šanse da će opstati u doglednoj budućnosti. Angular je najstariji među evaluiranim kandidatima ako se računa prva verzija koja je izašla 2010. godine pod nazivom AngularJS. Međutim, 2016. godine je izašla nova verzija koja je u potpunosti promenjena i iznova napisana. Migracija sa starije na noviju verziju gotovo da zahteva pisanje aplikacija ispočetka što je jedna od velikih zamerki Angular-u. Prva verzija React-a je izašla 2013. godine i od tada nije bilo značajnijih izmena API-ja što ukazuje na stabilnost i pouzdanost. Slična je situacija i sa Elm-om koji je nastao 2012. godine.

Kvantitativne mere poput broja zvezdica na GitHub platformi, broja skidanja sa npm registra biblioteka kao i zastupljenost na StackOverflow-u (Tabela 1) ukazuju na popularnost analiziranih radnih okvira. Iako daju uvid u to koliko se svaki od njih koristi, ovi brojevi ne prave razliku između stvarne upotrebe u praksi i upotrebe u svrhu učenja ili isprobavanja pa ih treba razmotriti u kombinaciji s primerima konkretnih aplikacija koje ih koriste u praksi.

Tabela 1 *Popularnost biblioteka izražena kvantitativnim merama*

	GitHub	npm	StackOverflow
A	~31000	~1,93 miliona	~87000
R	~76000	~5.85 miliona	~67000
E	~4500	~65000	~1200

Sve tri biblioteke imaju neki oblik korporativne podrške. Zaposleni u Google-u aktivno nadziru i održavaju Angular, dok je s React-om slična situacija sa zaposlenima u Facebook-u. Iza Elm-a ne stoji velika korporacija kao što je slučaj sa druge dve biblioteke. Međutim, projekat su podržale kompanije Prezi i NoRedInk, a uspostavljena je i fondacija za Elm softver (*Elm Software Foundation*) čija je misija da promoviše, zaštiti i unapredi programski jezik Elm.

Iako je teško kvantitavno izmeriti veličinu zajednice, mogu se razmotriti određeni faktori koji govore o aktivnosti ekosistema oko određene biblioteke. Jedan od tih faktora je broj npm biblioteka i paketa dostupnih programerima (Tabela 2).

Tabela 2 *Broj dostupnih paketa za svaku od izabranih biblioteka*

Biblioteka	Broj paketa
Angular 2+	~18,322 npm
React	~8,174 npm
Elm	~992

3.2. Osobine radnog okvira

Proces učenja svakog od analiziranih radnih okvira u velkoj meri zavisi od prethodnog iskustva. Angular zahteva učenje TypeScript-a i specifične sintakse koja se koristi u UI šablonima. Većina dostupnog React koda koristi ES6 sintaksu i JSX koje je neophodno savladati za ozbiljniji rad sa React-om. Elm je zaseban jezik koji se kompajlira u JavaScript, a pri tom je i čisto funkcionalan pa je pored specifične sintakse potrebno savladati i koncepte funkcionalnog programiranja što može dodatno da oteža početno iskustvo u radu s Elm-om.

Angular u osnovnom paketu sadrži neke od neophodnih alata za pravljenje složenih aplikacija: upravljanje korisničkim interfejsom, upravljanje stanjem aplikacije, rutiranje i testiranje i zahteva manje dodatnih biblioteka u odnosu na React i Elm. To može da bude prednost jer je početni razvoj znatno brži i zahteva manje odluka, ali isto tako i mana jer je tada i fleksibilnost znatno manja.

Kada su performanse u pitanju nema značajnijih odstupanja između kandidata. Neki *benchmark* testovi [**8**] pokazuju da je Angular najsporiji, React nešto brži i Elm najbrži, ali razlike nisu velike.

U poslednje vreme klijenti sve češće očekuju i zahtevaju da aplikacije koje rade u web pregledačima rade i na mobilnim uređajima pa čak i nativno na desktopu. Za svakog od kandidata postoje gotova rešenja ili makar pokušaji za rešavanje opisanog problema. Dve popularne opcije za Angular su: NativeScript i Ionic. Za React su to React Native i react-native-renderer. Postoji i projekat pod nazivom elm-native-ui kao pokušaj da se Elm koristi za mobilne aplikacije.

3.3. Alati

Za sva tri evaluirana radna okvira postoje odlične biblioteke UI komponenti kao što su Angular-Material, Material-UI, React-Bootstrap, elm-bootstrap kao i veliki broj gotovih individualnih komponenti.

Od razvojnih okruženja treba istaći Visual Studio Code i WebStorm koji pružaju odličnu podršku za Angular i React uključujući isečke i šablone koda kao i moćne alate za refaktorisanje dok je podrška za Elm prisutna nešto slabija. Pored razvojnih okruženja postoje i dodatni alati za automatizaciju zadataka kao što su: kreiranje skeleta projekta, provera stila koda i pokretanje jediničnih testova, server za lokalni razvoj uz automatsko osvežavanje aplikacije tokom pisanja koda, alati za pakovanje i stavljanje koda u produkciju itd. Ovde se blaga prednost može dati Angular-u, ali i druga dva kandidata raspolažu solidnim alatima za navedene zadatke.

3.4. Kompanije

Poslednja grupa evaluacionih faktora tiče se direktno kompanija. Tu spadaju: licenciranje, podrška, bezbednost i mogućnost zapošljavanja programera sa odgovarajućim veštinama [**7**]. Ovi faktori se mogu primeniti i na manje organizacije, ali oni u slučaju kompanija mogu biti presudni prilikom donošenja odluke o biblioteci ili proizvodu koji ispunjava sve ostale kriterijume.

Kompanije koje koriste alate otvorenog koda za pravljenje softvera treba da pažljivo analiziraju licence pod kojima se ti alati održavaju i da se osiguraju da se te licence ne kose sa upotrebom alata u kompaniji. Evaluirani radni okviri su pod MIT licencom [**9**] sa izuzetkom React-a pre verzije 16 (BSD + Patenti [**10**]) koja može uzrokovati potencijalne zakonske probleme *startup* kompanijama.

Ni za jedan od analiziranih radnih okvira nisu prijavljeni veći bezbednosni problemi, a ako ih je i bilo uglavnom su brzo rešavani i distribuirani. To je važan podatak za kompanije koje se odluče za neki od njih s obzirom na svakodnevni porast poslova koji se obavljaju na mreži.

Kada se bira web radni okvir za pravljenje preduzetničkih aplikacija u softverskim kompanijama potrebno je razmotriti i ljudski faktor i osigurati da trenutno zaposleni programeri mogu da ga nauče kao i to da je moguće angažovati dodatne ljudske resurse ukoliko se za to javi potreba. Aktuelne statistike [**8**] pokazuju da je React trenutno najpopularniji dok popularnost Angular-a opada. Interesovanje za Elm raste, ali je daleko manje u odnosu na druga dva kandidata.

4. STUDIJA SLUČAJA

U svakom od analiziranih radnih okvira, Angular-u, React-u i Elm-u razvijena je istovetna CRUD aplikacija koja predstavlja registar kompanija i njihovih zaposlenih. Ova aplikacija sadrži većinu elemenata složene klijentske web aplikacije uključujući dobavljanje podataka preko HTTP-a, rutiranje, generičke komponente za tabelarne prikaze, forme i validaciju podataka. Razvoj testne aplikacije je omogoućio i praktičnu proveru zaključaka i tvrdnji vezanih za arhitekture i tehnologije koji su istaknuti u ovom radu.

4.1 Komponente

Tipične komponente koje su implementirane u testnoj aplikaciji su: tabela koja prikazuje podatke uz mogućnost filtriranja, sortiranja i straničenja, navigacioni meni, tekstualno polje i dugme za akcije. React i Elm su se pokazali odličnim za kreiranje malih pasivnih komponenti koje nemaju interno stanje već se definišu kao klasične funkcije koje dobijaju ulazne podatke i vraćaju HTML. Kreiranje komponenti u Angular-u je nešto složenije zbog sintakse, ali i komplikovanijeg mehanzima za komunikaciju sa klijentskim komponentama.

4.2 Sintaksa

Implementacija svake od testnih aplikacija zahtevala je korišćenje specifične sintakse. Kod Angular-a su to TypeScript i direktive, kod React ES6+ i JSX a kod Elm-a sintaksa i koncepti samog jezika. Moglo bi se reći da se trud uložen u učenje ES6 sintakse verovatno najviše isplati s obzirom da se stečeno znanje može upotrebiti u različitim kontekstima, čak i za razvoj serverskih aplikacija.

4.3 UI šabloni

Kod pisanja UI šablona uočena je sličnost između React-a i Elm-a koji su uneli inovacije i doveli do preispitivanja dugogodišnjih „najboljih“ praksi. JSX sintaksa koju koristi React kao i Elm-ove *view* funkcije ponovo uvode mešanje UI šablona sa programskom logikom iako su programeri decenijama ranije pokušavali da ih razdvoje smatrajući takav pristup lošom praksom. Zagovornici ovakvog pristupa smatraju da je razdvajanje šablona od logike samo razdvajanje tehnologija, ali ne i nadležnosti. Stoga savetuju da programeri treba da razvijaju komponente umesto UI šablona jer su to jedinice koje se mogu ponovo upotrebiti, kombinovati i testirati u izolaciji [**11**]. Angular je negde između ova dva pristupa jer su HTML šabloni i dalje prisutni s tim što su obogaćeni direktivama.

4.4 Upravljanje stanjem i povezivanje podataka

Pored komponenti, stanje predstavlja važan koncept u izgradnji web korisničkih interfejsa. Komponente opisuju stanje korisničkog interfejsa u bilo kom trenutku vremena. Posmatranje korisničkog interfejsa kao “funkcije” stanja je u manjoj ili većoj meri zastupljeno u sve tri implementacije testne aplikacije. Tok podataka kroz React i Elm aplikacije je jednosmeran, dok je kod

Angular-a dvosmeran. Pristup koji koriste prva dva radna okvira jeste nešto opširniji za pisanje, ali omogućava bolju kontrolu, lakše upravljanje stanjem, pa čak i bolje performanse. Nasuprot tome, kod Angular-ovog dvosmernog povezivanja sam radni okvir interno menja stanje modela kada se neki element poput polja za unos ažurira što rezultuje čistijim kodom ali je u kompleksnijim slučajevima teže ispratiti tok podataka.

4.5 Fleksibilnost

Kada je u pitanju fleksibilnost, prednost se može dati React-u. React se može koristiti kao standardna JavaScript biblioteka jednostavnim uključivanjem *script* taga na stranicu. Pored toga, React je više biblioteka nego radni okvir i zagovara suprotnu filizofiju u odnosu na Elm i Angular koji nude sveobuhvatan radni okvir. React omogućava zamenu malih delova aplikacije boljim bibliotekama, umesto čekanja da sam radni okvir evoluira i uvede novine.

5. ZAKLJUČAK

U ovom radu su opisane neke od popularnih arhitektura za klijentske web aplikacije. Definisan je konceptualni model evaluacije klijentskih radnih okvira koji je primenjen na evaluaciju tri popularna radna okvira po svakom od uspostavljenih kriterijuma. Implementirana je i testna aplikacija u cilju istraživanja i sticanja praktičnog iskustva sa svakim od analiziranih radnih okvira.

Odgovor na pitanje koju arhitekturu i radni okvir ili biblioteku izabrati za konkretan projekat nije jednostavan i umnogome zavisi od konteksta i brojnih faktora na koje je ovaj rad nastojao da ukaže. Što se arhitektura tiče, blaga prednost se može dati novijim arhitekturama poput FLUX-a i „Elm arhitekture“ kao i njihovim varijacijama jer sadrže ugrađeno prethodno znanje i iskustvo, a ujedno donose i nove ideje i koncepte koji su prilagođeni potrebama savremenog web razvoja. O analiziranim radnim okvirima su izvedeni sledeći zaključci:

- Ukoliko je potrebno rešenje koje ima ugrađeno sve što je potrebno za pravljenje kompleksnih aplikacija, Angular je najbolji izbor.
- Ako se cilja na biblioteku iza koje stoje veliki korporativni sponzori, Angular i React su dobre opcije.
- Razvojni timovi koji dobro poznaju Java i C# programske jezike i preferiraju sisteme sa statičkim tipovima će se najlakše snaći s Angular-om.
- Za razvojne timove sastavljene od iskusnih JavaScript “gurua” i ljubitelja funkcionalnog programiranja React i Elm su odlične opcije.
- Organizacije koje su u razvoju i žele da unajme programere koji su motivisani za rad sa najnovijim tehnologijama, React i Elm su takođe dobar izbor.

6. LITERATURA

[1] Robert Cecil Martin, *Clean Architecture*.: Prentice Hall, 2017.

[2] Jens Neuhaus. Medium. [Online]. https://medium.com/unicorn-supplies/angular-vs-react-vs-vue-a-2017-comparison-c5c52d620176

[3] Cory House. freeCodeCamp. [Online]. https://medium.freecodecamp.org/angular-2-versus-react-there-will-be-blood-66595faafd51

[4] Jaakko Voutilainen, "Evaluation of Front-end JavaScript Frameworks for Master Data Management Application Development," Metropolia University of Applied Sciences, Bachelor’s Thesis 2017.

[5] Julia Plekhanova, "Evaluating web development frameworks: Django, Ruby on Rails and CakePHP," Temple University, Philadelphia, IBIT Report 2009.

[6] Changpil Lee, "An Evaluation Model for Application Development Frameworks for Web Applications," The Ohio State University, Columbus, Thesis 2012.

[7] Brandon Satrom. (2018, Jan.) Telerik. [Online]. https://www.telerik.com/whitepapers/kendo-ui/choosing-the-right-javascript-framework-for-your-next-web-application

[8] Stefan Krause. (2017) Results for js web frameworks benchmark – round 7. [Online]. http://www.stefankrause.net/js-frameworks-benchmark7/table.html

[9] Wikipedia. [Online]. https://en.wikipedia.org/wiki/MIT_License

[10] Open Source Initiative. [Online]. https://opensource.org/licenses/BSDplusPatent

[11] Pete Hunt. (2013) YouTube. [Online]. https://www.youtube.com/watch?v=x7cQ3mrcKaY&t=11s

[12] The State Of JavaScript. [Online]. https://stateofjs.com/2017/front-end/results/

Kratka biografija:

Dalibor Pavičić rođen je u Novom Sadu 1992. god. Master rad na Fakultetu tehničkih nauka iz oblasti Elektrotehnike i računarstva – Primenjene računarske nauke i informatika odbranio je 2018.god.
kontakt: dalibor.pavicic92@gmail.com