SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 5945

Usporedba radnih okvira klijentske strane u razvoju web aplikacija

Antonio Kamber

Zagreb, svibanj 2019.

Zahvaljujem mentoru, doc.dr.sc. Alanu Joviću,
na pomoći, savjetima, strpljenju i razumijevanju.

Sadržaj

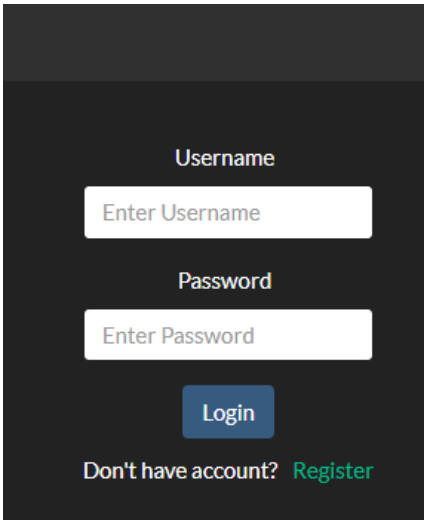
1. Uvod

Zahtjevi na razvoj programske potpore iz godine u godinu neprestano rastu. Kako bi proizvođači programskih sustava bili konkurentni na tržištu mnogo pažnje se ulaže u automatiziranost razvoja s ciljem poboljšanja kvalitete i pouzdanosti sustava. Pri tome valja uzeti u obzir troškove i dostupne resurse. Kako bi se resursi što bolje iskoristili, uobičajeno je koristiti radne okvire prilikom procesa implementacije programske podrške. Radni okvir je alat koji pruža automatiziranu podršku razvoju aplikacija. Nudi skup učestalo korištenih funkcionalnosti koje služe za rješavanje raznih problema prilikom procesa razvoja. Radni okviri klijentske strane za razvoj web aplikacija su većim dijelom izvedeni u obliku JavaScript i JSON[1] konfiguracijskih datoteka. Okviri pružaju sintaksu koja je programerima jednostavnija, nude iskustveno dokazano najbolju strukturu za implementaciju i sugeriraju optimalni način rada. Svaki okvir nudi niz funkcija za upotrebu, čija su ispravnost i pravovremenost izvršavanja detaljno ispitana. Programeri ne moraju nužno pisati sav kod kako bi implementirali funkcionalne zahtjeve sustava kojega razvijaju već je dovoljno pozivati funkcije radnog okvira ili koristiti propisanu sintaksu koju okvir razumije te ju potom pravilno parsira i samostalno obavlja složene operacije. Kako se upotrebom okvira smanjuje pisani kod, to za posljedicu ima smanjenje grešaka, te budući da su funkcionalnosti pouzdane, nije ih nužno provjeravati pa se smanjuju i potrebni testovi. Zbog navedenoga programeri imaju uštedu vremena i resursa u procesu implementacije i mogu više pažnje posvetiti kvalitetnoj razradi problema koji su specifični za razvijani sustav. Osim što radni okviri nude funkcionalne usluge, većina ih nudi veliki broj već postojećih komponenti koje se mogu koristiti. Postoji niz vanjskih alata čijom se instalacijom dodatno podupire proces razvoja. Kako je danas razvijeno mnoštvo operacijskih sustava i web preglednika, razvoj web aplikacija je postao izuzetno zahtjevan, jer je potrebno voditi računa o specifičnostima svakog pojedinog uređaja na kojemu se aplikacije koriste, te o načinu prikazivanja sustava u pojedinim web preglednicima. Radni okviri koji su razmotreni u radu također garantiraju pravilno izvođenje aplikacije neovisno o okruženju izvođenja, te pružaju mogućnost razvoja neovisno o platformskim komponentama. Glavna uloga okvira

klijentske strane se odnosi na predstavljanje aplikacije korisniku. Dakle u slučaju web aplikacija, tu se radi o web stranici, onome što korisnik vidi kada otvori web preglednik. Naglasak je na interakciji i pružanju ugodnog iskustva korisnicima. Okviri nam omogućavaju automatsko generiranje elemenata web stranice, obavještavanje prilikom akcija korisnika, te automatizirani odgovor korisniku. Prema statistikama za 2018. godinu najpopularniji radni okviri klijentske strane su Angular, ReactJS i VueJS. Zbog navedenog sam se za razvoj ogledne aplikacije odlučio na primjenu upravo navedenih okvira.
Sva tri radna okvira se koriste za izradu jednostraničnih aplikacija (engl. Single-page-application, kraće: SPA[2]) . Kod takvih aplikacije se sav rad omogućuje putem jedne web stranice. Korisnik dobiva iluziju straničenja dinamičkom manipulacijom glavnog HTML dokumenta. Po potrebi se elementi ubacuju ili izbacuju u korisničko sučelje ovisno o akcijama korisnika.

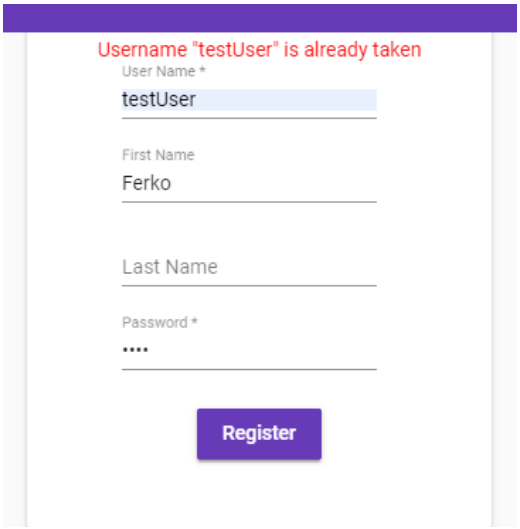
2. Aplikacija

Cilj ogledne aplikacije je implementirati funkcionalnosti koje su zajedničke većini današnjih web aplikacija. Prilikom pokretanja aplikacije, korisnika se usmjerava na stranicu za prijavu. Ukoliko korisnik nije registriran potrebno je kliknuti na link za registraciju pri čemu se prikazuje obrazac registracije (slika 1) . Korisnik se registrira jedinstvenim korisničkim imenom i odabire zaporku.

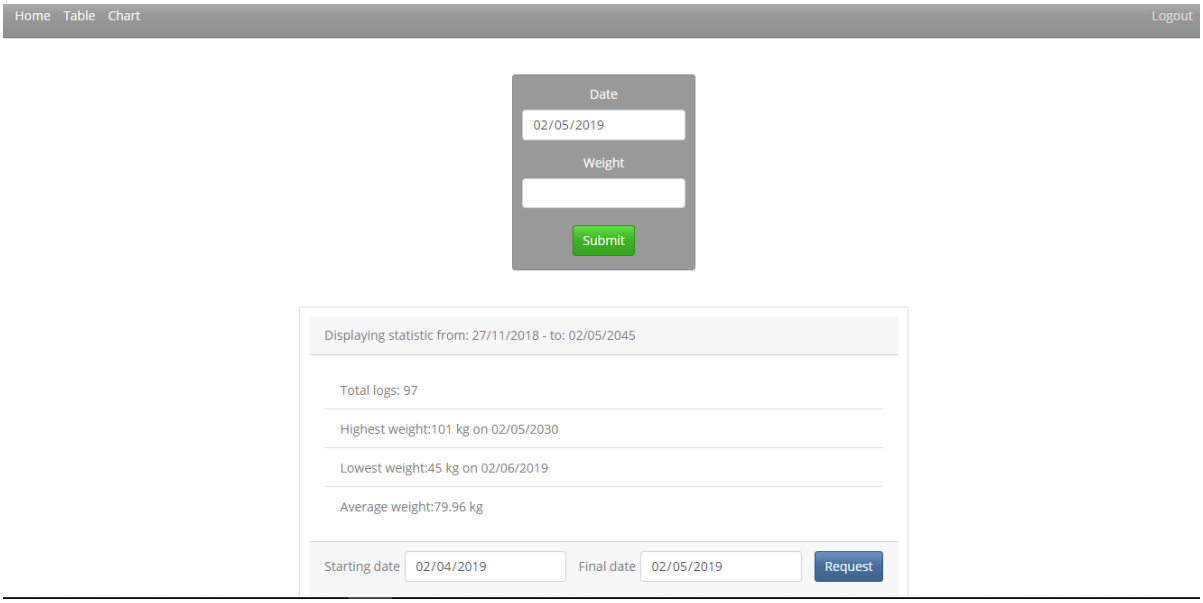


Slika 1. ReactJS - prijava korisnika

Nakon pokušaja registracije provedena je validacija, te ukoliko je korisničko ime već zauzeto, korisnik dobiva odgovarajuću obavijest (slika 2). Nakon uspješne registracije korisnik dobiva povratnu informaciju o uspješnosti te se može prijaviti i koristiti aplikaciju. Jednom kada se korisnik prijavi u sustav, na naslovnoj strani ima mogućnost unosa dnevne tjelesne kilaže koja se odnosi na trenutačni datum, no ukoliko želi, korisnik može promijeniti datum i unijeti zapis (slika 3).

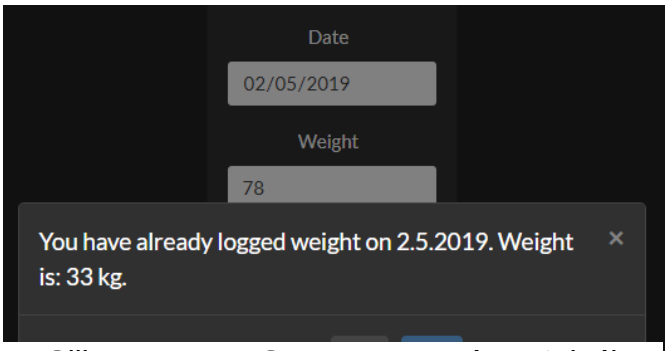


Slika 2. Angular - neuspješna registracija



Slika 3. VueJS - naslovna stranica

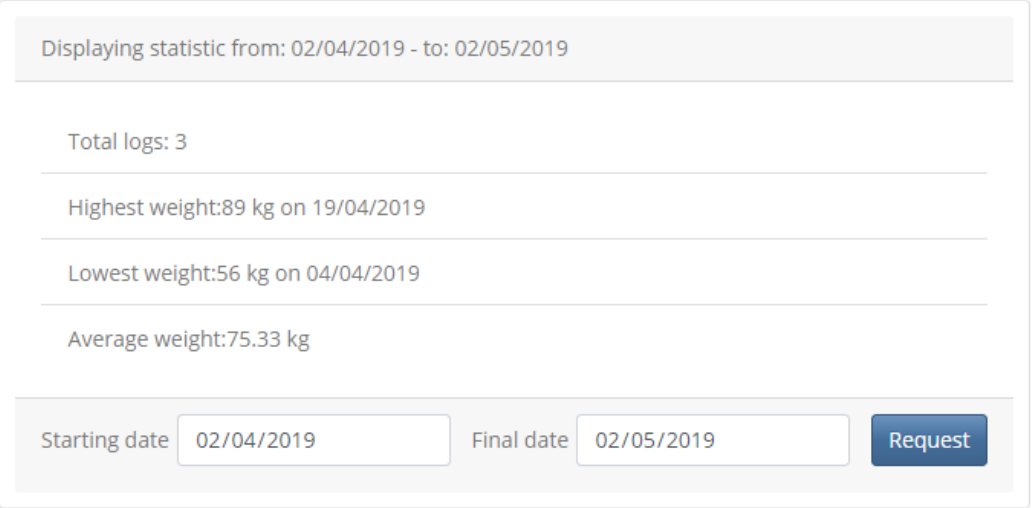
Svi se podaci pohranjuju u lokalnu relacijsku bazu podataka. Ukoliko korisnik pokuša unijeti kilažu za datum koji je već unio, aplikacija obavijesti korisnika te nudi mogućnost prihvaćanja novo



Slika 4. ReactJS - unos na već postojeći datum

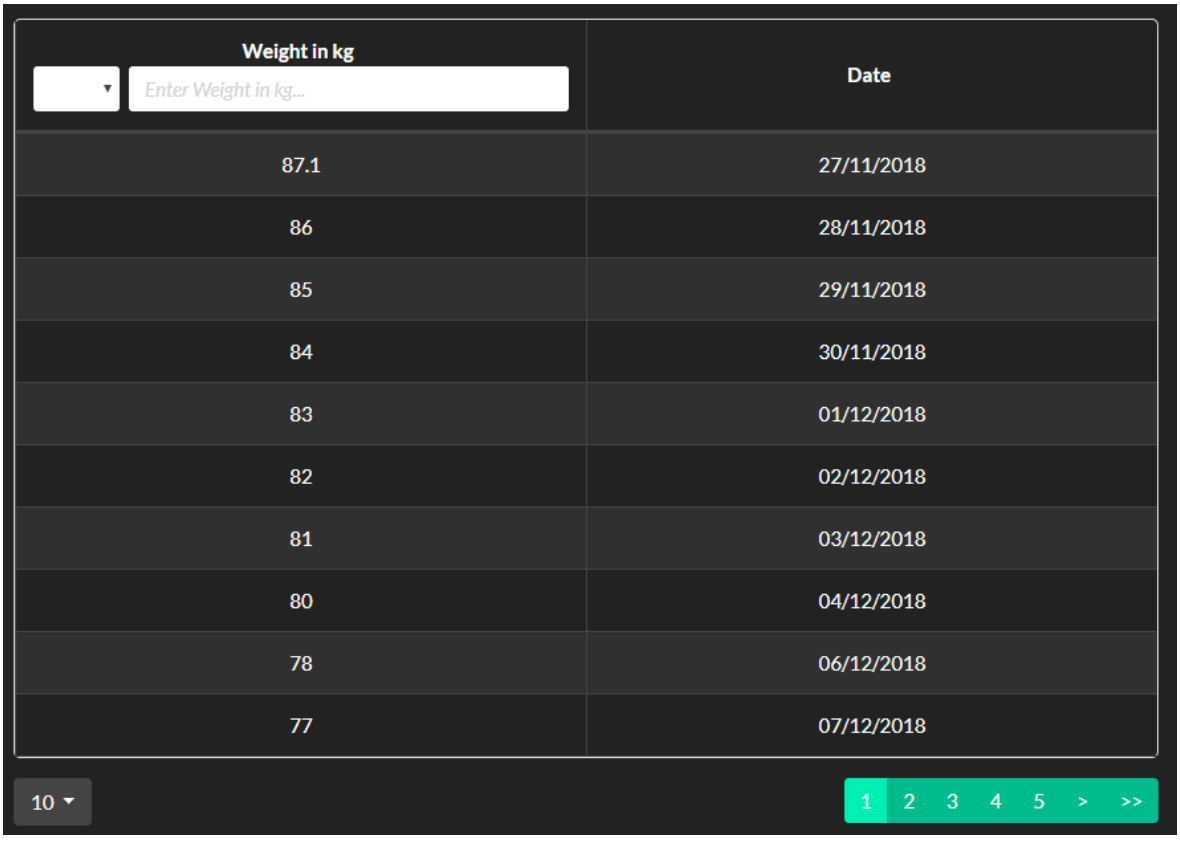
unesene vrijednosti ili odustajanje od unosa (slika 4).

Naslovna strana ispisuje najveću, najmanju i prosječnu kilažu korisnika premda je moguće i podesiti vremenski interval na kojemu se podaci promatraju (slika 5).



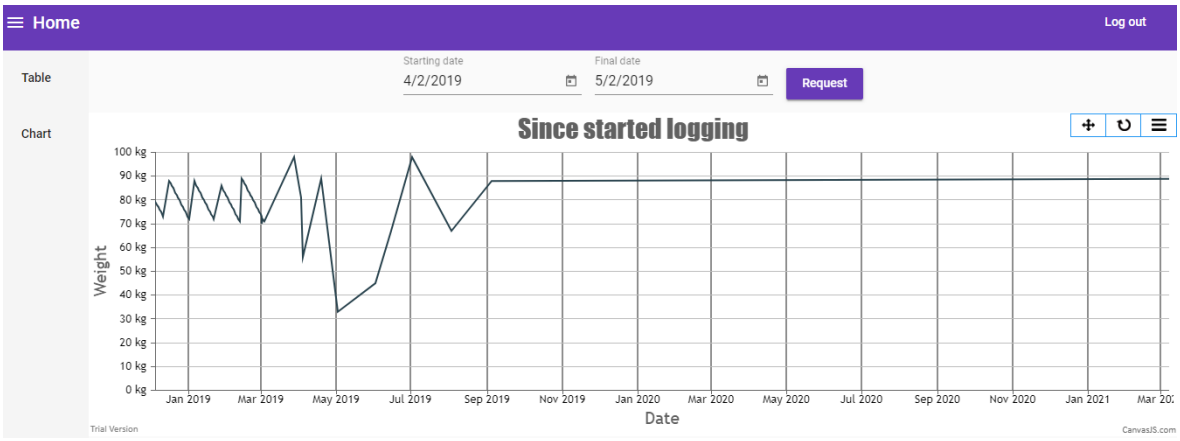
Slika 5. VueJS - statistika na ograničeni period

Korisnik može tablično pregledati unose kilaže na posebnoj stranici. Tablica omogućuje straničenje, sortiranje i pretraživanje (slika 6).



Slika 6. ReactJS - tablični prikaz zapisa

Osim tabličnog prikaza postoji i stranica koja nudi opciju grafičkog prikaza u obliku linijskog interaktivnog grafa. Graf je interaktivan te korisnik može povećati dijelove prikaza, kao i preuzeti graf u slikovnom obliku na računalo. Za implementaciju interaktivnog grafičkog prikaza je korišten ugradbeni dodatak za JQuery pod nazivom ChartJS. Za odabir tabličnog ili grafičkog prikaza korisnik može definirati početni i završni datum prikazivanja podataka (slika 7).



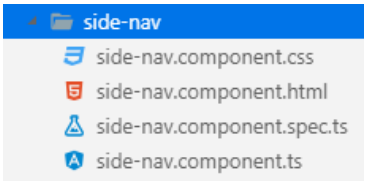
Slika 7. Angular - grafički prikaz zapisa

Aplikacija je implementirana kao klijentsko-poslužiteljska web aplikacija. Kao poslužiteljske tehnologije koriste se .NET core radni okvir i baza podataka SQL Server. Navedene funkcionalnosti su implementirane kroz tri nezavisne klijentske aplikacije izrađene uporabom radnih okvira Angular, ReactJS i VueJS. Za responzivno oblikovanje primijenjene su biblioteke Angular Material, Bootstrap 4 i Bootswatch. Kada se govori o pojmu stranica u kontekstu ogledne aplikacije, tu nije riječ o uobičajenim stranicama kao kod višestraničnih aplikacija. Svaka stranica je isti glavni korijenski HTML dokument čijom manipulacijom i dinamičkim umetanjem prikaza komponenata korisnik dobiva dojam straničenja. Kako danas zahtjevi na brzinu izvođenja aplikacija rastu, preporučeno je koristiti jednostranične aplikacije. Velika korist takvog načina rada je što se web preglednik ne osvježava učestalo, čime se postižu bolje vremenske performanse u odnosu na tradicionalne višestranične aplikacije.

3. Angular

3.1. Podrška razvoju

Osnivač radnog okvira Angular je Miško Hevery koji je pokrenuo prvu stabilnu verziju 2016. godine. Hevery danas održava radni okvir zajedno s timom ljudi koji rade za firmu Google. Angular je radni okvir otvorenoga koda zahvaljujući čemu se aplikacije mogu razvijati na bilo kojoj sklopovskoj platformi. Okvir podržava razvoj aplikacija za web, desktop i mobilne uređaje. Jezici u kojima se piše kod su TypeScript, HTML i CSS. TypeScript podržava provjeru tipova podataka kao i uporabu objektno orijentiranih mehanizama poput sučelja, razreda i nasljeđivanja. Sintaksa je slična ostalim objektno orijentiranim jezicima, npr. C# koji je prema statistici za 2018. godinu najčešći jezik u kojem se razvija poslužiteljska strana aplikacije u kombinaciji s Angularom za klijentsku stranu. Razvojni tim je uložio mnogo truda u automatiziranu podršku procesu razvoja. Za većinu danas korištenih razvojnih okolina postoji veliki broj proširenja i alata koji se mogu instalirati. Instalirani alati omogućuju automatsku dopunu koda, uvid u dokumentaciju unutar razvojne okoline, razna upozorenja koja ukazuju na potencijalne probleme te veliki broj već kreiranih predložaka koda koji su besplatni i javno dostupni za upotrebu. Radni okvir nudi vlastiti naredbeni redak čijom uporabom možemo kreirati komponente, servise, module i druge strukturalne cjeline aplikacije. Kako bih započeo projekt dovoljno je bilo pokrenuti naredbu: > *ng new angular angular-client.* Okvir je time generirao početni kod aplikacije unutar kazala *angular-client*. Prilikom razvoja sam najčešće koristio naredbu: > *ng g c <ime_komponente>* koja stvori kazalo sa svim potrebnim datotekama za potpuni razvoj i testiranje komponente. Generirana struktura je podijeljena s obzirom na funkcionalnost što olakšava naknadnu izmjenu komponente. Naredba: > *ng g c side-nav* će stvoriti strukturu prema slici 8.



Slika 8. Naredba generate component

3.2. Organizacija aplikacije

Angularov projekt je organiziran kao niz modula u jeziku TypeScript koji se uvoze u aplikaciju. Koncepti koje okvir koristi su komponente, servisi, moduli, direktive i filteri. Komponente i servisi su razredi u TypeScriptu. Semantička razlika je u tome što su komponente građevni dijelovi aplikacije, a servisi obavljaju funkcionalnosti aplikacije. Direktiva je HTML-ov atribut koji manipulacijom sadržaja elementa omogućuje zahtijevano funkcionalno ponašanje. Filteri primaju podatke iz cjevovoda, obrađuju ih i prosljeđuju ih u cjevovod. Složeni građevni dijelovi aplikacije su moduli. Angularov modul predstavlja niz komponenti grupiranih u jednu organizacijsku jedinicu na temelju zajedničkih karakteristika čime se povećava kohezija i olakšava naknadna izmjena. Komponente međusobnom interakcijom razmjenjuju podatke i obavljaju potrebne funkcionalnosti. Svaka Angularova aplikacija ima korijenski modul koji služi za inicijalizaciju i povezivanje. Dekorator za modul je *@NgModule* kojega je potrebno uvesti iz *@angular/core* biblioteke i predati mu objekt (slika 9: retci 24-52) koji okvir koristi kako bi pravilno povezao TypeScriptov razred sa mehanizmima koje pruža modul. Osim komponenti, unutar modula se navode usluge i drugi moduli koji se koriste. Često je korisno radi preglednosti napraviti poseban modul koji uvozi i izvozi niz komponenti. Budući da takvi moduli nemaju funkcionalnosti osim uvoza i izvoza, nazivaju se strukturalni moduli. Dovoljno je u drugom modulu uvesti strukturalni modul umjesto da se uvozi svaka komponenta zasebno, time se postiže veća modularnost. Modularnost Angular-a je značajna za velike projekte čija struktura može postati znatno složena. Unutar korijenskog modula se deklarira objekt koji sadrži svojstvo *bootstrap* (slika 9: redak 51), čija vrijednost služi radnom okviru da zna koja komponenta je točka pokretanja. Prema slici će ogledna aplikacija prilikom pokretanja u web preglednik dinamički umetnuti sadržaj komponente *AppComponent.* Preporuka službene dokumentacije je početnu komponentu nazvat *AppComponent* , jer se tu radi o komponenti koja predstavlja aplikaciju.

```
24  @NgModule({
25    declarations: [···
39    imports: [···
47    providers: [···
51    bootstrap: [AppComponent]
52  })
53  export class AppModule { }
```

Slika 9. Korijenski modul

Tok aplikacije uvjetuje početna komponenta time što sadrži niz drugih komponenti. U korijenskom modulu je nužno uvesti glavni modul *BrowserModule* iz biblioteke *@angular/platform-browser* koji sadrži sve ugrađene funkcionalnosti poput objašnjenih direktiva i filtera koji dolaze uz radni okvir. Sve funkcionalnosti modula definiranih u korijenskom kazalu se potom mogu koristiti kroz cijelu aplikaciju. Komponente imaju vlastiti logički i vizualni karakter. Pogled je termin koji se odnosi na predstavljanje komponente korisniku. TypeScriptovi razredi koji imaju funkcionalnost komponenti ili pak usluga, moraju biti označeni posebnim dekoratorima. Dekoratori su ključni za povezivanja svih dijelova aplikacije u funkcionalnu cjelinu. Označavaju razred dajući mu semantičko značenje, sadrže metapodatke potrebne okviru za dodjeljivanje nužnih funkcionalnosti, npr. dekorator *Injectable* će dekoriranom razredu dodijeliti sve funkcionalnosti usluge. Preporuka je tok aplikacije implementirati na način da komponente pozivaju servise koji potom obavljaju logički dio. Metapodatci usluge služe okviru kako bi ih mogao dinamički umetnuti u konstruktore komponenata prilikom njihove inicijalizacije, čime komponenta dobiva instancu usluge i može koristiti njezine metode. Naredba *platformBrowserDynamic().bootstrapModule(AppModule)* smještena u datoteci *main.ts* je metoda koja pokreće cijelu aplikaciju unutar web preglednika. Interakcija korisnika sa preglednikom uvjetuje daljnji tok aplikacije.

3.3. Komponente

Angularove komponente su kombinacija komponentne i model-pogled-upravljač (engl. model-view-controller, kraće: MVC[6]) . Upravljač je TypeScript razred koji kontrolira podatke i može manipulirati pogledom, model su vrijednosti varijabli razreda, a pogled sadržava HTML predložak koji okvir veže uz razred. Kako bi smo naznačili da se radi o komponenti potrebno je koristiti dekorator *@Component*. Dekoratoru je potrebno predati objekt koji sadrži svojstvo *selector*, čija vrijednost predstavlja naziv pomoću kojega će se komponenta koristiti unutar aplikacije, svojstvo *templateUrl,* čija je vrijednost relativna putanja do HTML predloška komponente, i atribut *styleUrls,* čija vrijednost sadrži listu putanja do CSS datoteka kojima se uređuje grafički prikaz komponente. Okvir enkapsulira sav kod dekoriranog razreda dajući mu funkcionalnosti Angularove komponente. Slika

10 prikazuje komponentu *Table* čija je funkcionalnost u oglednoj aplikaciji da prima podatke te ih tablično prikazuje nudeći dodatne funkcionalnosti poput pobrojavanja, pretraživanja i sortiranja. S obzirom na vrijednost svojstva *selector* prema slici 10 druge ju komponente mogu koristiti kao HTML element <*app-table*>. Na temelju vrijednosti *templateUrl* i *styleUrls* (slika 10: retci 9-10) svojstava se može vidjeti kako se predložak u HTML-u i datoteka u CSS-u nalaze unutar istoga kazala kao i datoteka koja sadrži kod razreda. Takvom strukturom se postiže veća preglednost i bolja funkcionalna kohezija.

```
7   @Component({
8     selector: 'app-table',
9     templateUrl: './table.component.html',
10    styleUrls: ['./table.component.css']
11  })
```

Slika 10. Dekorator komponente Table

Okvir nudi mogućnost stvaranja strukture kazala i datoteka komponente *Table* naredbom pokrenutom iz naredbenog retka: > *ng generate component table.* Sučelje Angularove komponente je izvedeno u obliku ulaznih svojstva koja se označuju dekoratorom @*Input* i izlaznih akcija koje komponenta generira prilikom zadovoljenja uvjeta okidanja, označenih dekoratorom @*Output* .

3.4. Mehanizmi komunikacije komponenti

Odnos komponenti može biti roditelj-dijete, rođaci i nepovezane. Ako komponenta koristi drugu komponentu onda su u odnosu roditelj-dijete. Ako roditeljska komponenta koristi nekoliko drugih komponenti, one su u odnosu rođaci. U ostalim slučajevima su komponente nepovezane. Roditelj unutar svoga pogleda poziva dijete kao HTML-ov element čija je oznaka ime djeteta. Kako bi dijete povezalo internu varijablu s vrijednošću atributa koju joj definira roditelj, potrebno je koristiti dekorator *@Input(<ime_atributa>)*. Internoj varijabli djeteta je moguće kao vrijednost dodijeliti referencu na funkciju čiji će poziv roditelj registrirati. Kako bi se to postiglo potrebno je varijablu dekorirati s *Output(<ime_događaja>)* . Vrijednost varijable je instanca razreda *EventEmitter* (slika 11:redak 14) koji sadrži metodu *emit.* Pozivom metode dijete komponenta objavljuje događaj na koji registrirani roditelj može reagirati. Na slici 11 je prikazan isječak koda komponente

PeriodPicker koja omogućuje odabir početnog i završnog datuma prema kojemu se potom pokreće dohvat podataka, odabirom datuma se objavljuje događaj *request* (slika 11: redak 14). Njezina funkcionalnost se koristi unutar nekoliko drugih komponenti aplikacije. Time smo smanjili dupliciranje koda, rabeći temeljni princip programskog inženjerstva ponovne upotrebe. U varijablu *from* (slika 10: redak 11) se pohranjuje vrijednost koju komponenta *PeriodPicker* primi od roditelja. Varijable *from* i *to* komponenta koristi kao i ostale interne varijable razreda.

```
11    @Input('from') from: Date;
12    @Input('to') to: Date;
13
14    @Output('request') request = new EventEmitter<PeriodPickerRequest>();
```

Slika 11. Svojstva komponente *PeriodPicker*

Putem *@Input* dekoratora se postiže mehanizam prijenosa podataka roditelja djetetu. Zahvaljujući dekoratoru *@Output* komponenta objavljuje događaj te prenosi podatke putem argumenata pozivane metode. Okidanjem događaja *request* se aktivira funkcija roditelja *periodRequest* (slika 12: redak 2) kojoj se kao argument prenosi *$event* objekt, čija je vrijednost određena unutar djeteta kao argument metode *emit*.

```
2     <app-period-picker (request)="periodRequest($event)"></app-period-picker>
3   </div>
4   <div class="row">
5     <div class="table">
6       <app-table [data]="tableData"></app-table>
```

Slika 12. Upotreba komponente *PeriodPicker*

Objekt koji se prenosi roditelju pod referencom *$event* sadrži vrijednost svojstva *from* i *to* (slika 13: retci 34-37) definiranih unutar razreda komponente djeteta.

```
33    onRequest() {
34      this.request.emit({
35        from: this.from,
36        to: this.to
37      });
38    }
```

Slika 13. Objava događaja request

Roditelj tada može koristiti podatke djeteta jer ih ima na raspolaganju unutar funkcije *periodRequest*. Opisanim mehanizmom imamo mogućnost prijenosa podataka djeteta roditelju.

Odnosom roditelj-dijete podaci se nadalje mogu prenositi duž lanca komponenti i tako postići izmjena podataka među nepovezanim komponentama. Problem nastaje kada se podaci žele dijeliti između komponenti koje su vrlo udaljene u hijerarhiji, te bi bilo potrebno prenositi podatke putem svojstava između većeg broja komponenti s ciljem propagacije do odredišta. To je vrlo loša praksa, jer svako nepotrebno prenošenje dodatno usporava rad sustava. Kako bi se izbjegao problem, Angular uvodi mehanizam zvan *Subject. Subject* je razred koji može biti definiran bilo gdje unutar aplikacije. Kako bi se koristio potrebno ga je uvesti iz biblioteke *rxjs.* Razred *Subject* sadrži funkcije koje mogu generirati događaje čiji će argumenti biti podaci koji se žele prenositi između udaljenih komponenti. Bilo koja komponenta se može registrirati na objave događaja generiranih od strane instance razreda *Subject*. Prema preporukama Angularovog razvojnog tima, ovaj mehanizam se koristi na način da se napravi usluga koja kao vrijednost varijable sadrži instancu razreda *Subject.* Komponenta koja želi pokrenuti događaj, izvorišna komponenta, unutar svojega konstuktora zahtijeva umetanje servisa, što radni okvir dinamički odrađuje. Komponenta pokreće događaj pozivom metode *next* nad instancom razreda *Subject* pohranjenom u varijabli usluge. Komponenta koja je zainteresirana za događaj će također unutar konstrukora zatražiti umetanje usluge. Potrebno je prijaviti odredišnu komponentu na događaje koji se objavljuju, što se izvodi pozivom metode *subscribe* nad instancom razreda *Observable*. Kako bi smo iz razreda *Subject* dobili instancu razreda *Observable* potrebno je prijaviti metodu *<naziv_instance_subject>.asObservable()*. Time će bilo koja komponenta registrirana kao zainteresirana za događaj biti obavještena kada se događaj okine i podaci koji se događajem prenose će joj biti dostupni, neovisno o udaljenosti izvora i odredišta. Razred *Observable* sve funkcionalnosti obavlja asinkrono te ne

blokira rad aplikacije. Navedenim mehanizmom Angular nudi razmjenu podataka između više nepovezanih komponenti istovremeno.

3.5. Stanje, podaci i prikaz komponente

Stanje komponente u pojedinom trenutku je jedinstveno određeno njezinim pogledom i modelom. Kako korisnik obavlja razne akcije nad grafičkim sučeljem okvir se brine za pravovremenu aktivaciju povezanih komponenti, osvježavanje podataka, pogleda, te kontinuiranu interakciju s korisnikom uporabom asinkronih mehanizama. Komponenta *Record* ogledne aplikacije implementira funkcionalnost unosa kilaže korisnika. Njezin pogled sadrži gumb čijim se klikom pohranjuje datum i iznos kilaže koju je korisnik definirao ispunivši obrazac. Klikom na gumb se okine događaj *click* koji je sastavni dio okvira te se tada poziva funkcija komponente *onSave* (slika 14). Tada možemo unutar pozvane funkcije promijeniti vrijednosti varijabli razreda komponente. Ukoliko su događaji uokvireni oblim zagradama radi se o događajima koje radni okvir generira. Dakle događaj *click* (slika 14) nije isti kao i uobičajeni JavaScriptov događaj *onClick*.

```
<button *ngIf="!activeLog.id" mat-raised-button (click)="onSave()" [disabled]="form.invalid"
```

Slika 14. Okidanje događaja *click*

Kada se promijeni pogled funkcije, odnosno ispuni obrazac i klikne gumb mijenjaju se podaci komponente unutar pozvane metode *onSave*. Povezivanje se može obaviti i u drugom smjeru na način da se na temelju vrijednosti varijabli mijenja prikaz komponente. Prema slici 14 vidimo kako se atribut *disabled* gumba na formi komponente *Record* postavlja na vrijednost svojstva *invalid* interne varijable *form*. Kako bi okvir ispravno povezao vrijednost interne varijable sa atributom, potrebno je atribut uokviriti uglatim zagradama. Ukoliko se promijeni vrijednost varijable, okvir mijenja pogled, što je drugi smjer jednosmjernog povezivanja.
Prava moć okvira dolazi do izražaja ukoliko koristimo dvosmjerno povezivanje. Uporabom dvosmjernog povezivanja programer ne mora pisati kod koji će prilikom promjene podataka osvježavati pogled, te kod koji će interakcijom korisnika sa grafičkim sučeljem mijenjati podatke već se osvježavanje vrijednosti podataka i

pogleda odvija automatski. Ako atribut nekog HTML elementa uokvirimo uglatim pa zaobljenim zagradama *[(<ime_atributa>)] = "<naziv_varijable>"* tada signaliziramo radnom okviru da želimo da referencira vrijednost atributa s vrijednošću varijable. Kada se vrijednost varijable promijeni, automatski se mijenja vrijednost atributa HTML-elementa, što rezultira promjenom prikaza u web pregledniku. Kada korisnik interakcijom s grafičkim sučeljem uzrokuje promjenu atributa HTML-elementa, vrijednost varijable razreda komponente će se automatski promijeniti. Okvir izmjene obavlja asinkrono te ne blokira kontinuirani rad aplikacije. Dvosmjerno povezivanje omogućuje smanjenje koda, povećava automatiziranost i omogućuje programeru brži razvoj. Značajno se smanjuje mogućnost pogreške u usporedbi s eksplicitnim mijenjanjem podataka na temelju pogleda i obratno. Mehanizam smanjuje potrebu ispitivanja, jer možemo biti sigurni kako će radni okvir uspješno izvršiti zahtijevanu funkcionalnost.
Osim pridruživanja vrijednosti atributima HTML elementima, moguće je direktno u tijelo elemenata umetnuti dinamički nastale podatke, što se naziva interpolacija. Spomenuta komponenta *Record* prije nego što pošalje podatke poslužitelju provjerava pokušava li korisnik unijeti novi zapis sa datumom koji već postoji. Ukoliko se uvjet ispuni komponenta šalje obavijest korisniku. Potom se nudi mogućnost promjene vrijednosti kilaže i datuma već postojećega zapisa ili pohrana zapisa koji se pokušao spremiti. U grafičkom sučelju se prikazuje informativna obavijest korisniku koju Angularova komponenta pohranjuje u vrijednost varijable *userMessage*. Tekstualni sadržaj elementa *<mat-card-subtitle>* (slika 15), koji je dio pogleda komponente, će zahvaljujući interpolaciji biti vrijednost varijable *userMessage* ako ona postoji. Kako bi se u HTML-ovom predlošku prikazala vrijednost, potrebno je koristi dvostruke vitičaste zagrade unutar kojih se navodi naziv varijable ili složeni JavaScript izraz koji generira neku vrijednost.

```
<mat-card-title>Log your weight</mat-card-title>
<mat-card-subtitle *ngIf="userMessage">{{userMessage}}</mat-card-subtitle>
```

Slika 15. Prikaz dinamički nastalog sadržaja

3.6. Oblikovanje

Problem velikog broja CSS-ovih datoteka s kojim se programeri susreću je ispreplitanje i konflikti zbog istoimenih CSS-ovih razreda. Kako se unutar web preglednika preuzima niz CSS-ovih datoteka, može se dogoditi da se definiraju različita svojstva za isti CSS-ov selektor. Primjerice, na početnoj stranici HTML-elementu pridijelimo razred *main* čiju vrijednost odredimo unutar CSS-datoteke. Prilikom razvoja se može dogoditi da ponovno definiramo drugačije vrijednosti za *main* klasu jer na sekundarnoj stranici koristimo element kojemu također dodijelimo razred *main* te želimo da njegov element bude drugačiji. Ukoliko se vratimo na početnu stranicu, HTML element sa razredom *main* će poprimiti oblikovanje definirano unutar CSS-datoteke koja se posljednja preuzela i sadrži navedeni razred, što nije ono što smo htjeli. Primjenom radnog okvira Angular, CSS-datoteke će dobiti vlastiti doseg, što znači da se neće dogoditi ispreplitanje. Radni okvir će svakoj CSS-datoteci dodijeliti identifikator kako bi ju znao razlikovati od ostalih datoteka. Oblikovanje definirano za razred *main* unutar CSS-datoteke naslovne stranice će se odnositi isključivo i samo na taj razred iako se kasnije definira istoimeni razred. Time okvir sprječava mogućnost pogrešaka koje su vrlo česte kada na projektu radi velik broj ljudi.

4. ReactJS

4.1. Podrška razvoju

Osnivač ReactJS-a je Jordan Walke, programski inženjer Facebooka. Razvojni timovi koji se brinu o održavanju su oni od tvrtki Facebook i Instagram. Prva stabilna verzija je objavljena u ožujku 2013. godine. Motivacija za razvoj je bila prevelika složenost vlastitog proizvoda. Nakon što je biblioteka objavljena većina programskog sustava je ponovno implementirana uporabom ReactJS-a. Danas je prema statističkim podacima Stack Overflow-a ReactJS najomiljeniji alat za razvoj klijentske strane web aplikacija. Iako omogućuje samo razvoj web aplikacija, razvojni tim je objavio inačicu React Native koja se može koristiti za izradu

mobilnih aplikacija. U službenoj dokumentaciji je napisano kako je ReactJS biblioteka JavaScripta za razvoj korisničkog sučelja. Iako je riječ o biblioteci, zbog velike popularnosti je zajednica otvorenoga koda razvila mnoštvo paketa koji proširuju biblioteku kako bi se omogućile funkcionalnosti koje nedostaju. Fleksibilnost biblioteke ju čini kompatabilnom s nizom alata čijom se integracijom postiže potpuna funkcionalnost radnoga okvira. Zbog toga u praksi ReactJS većina smatra radnim okvirom. Kako bih se pokrenuo projekt dovoljno je preuzeti skriptu koja sadrži sav kod biblioteke. To je moguće putem mreže za dostavu sadržaja (engl. content delivery network, kraće: CDN[3] navođenjem izvora koji dostavlja skriptu prilikom preuzimanja glavnog HTML-dokumenta u web pregledniku. U praksi se češće koristi instalacija i preuzimanje skripte putem naredbenog retka. Za potrebe razvoja ogledne aplikacije koristio sam generiranje iz naredbenog retka. Potrebno je imati instaliran node.js[4] i npm[5] upravitelj paketa na računalu. Potom se iz naredbenog retka izvrši naredba: > *npx create-react-app react-client* koja generira strukturu i predložak projekta unutar kazala *react-client*. Za generiranje komponenti i automatiziranu podršku ne postoje naredbe u izvornom naredbenom retku kao što je to omogućeno Angularovim radnim okvirom. Zajednica je razvila paket npm čijom se instalacijom ugrađuju naredbe u naredbeni redak na lokalno računalo koje omogućuju automatiziranu podršku generiranja komponenti okvira. Službena dokumentacija je vrlo jasno napisana i jednostavna za upotrebu te postoji veliki broj primjera za razne scenarije kao i veliki broj besplatnih tutoriala.

4.2. Organizacija aplikacije

Kako je ReactJS biblioteka koja je implementirana komponentnom arhitekturom aplikacije su organizirane kao skup komponenti koje zajedničkim radom izvršavaju potrebne funkcionalnosti sustava. JavaScriptove datoteke u izvornom obliku se nazivaju skripte, no primjenom novih mehanizama koje uvodi EcmaScript (ES6) nekadašnje skripte poprimaju modularna obilježja. Ukoliko u skripti uvozimo ili izvozimo varijable, funkcije, objekte ili druge module onda se za skriptu koristi termin modul. U glavnom HTML-dokumentu ReactJS-ove aplikacije potrebno je definirati korijenski element koji će poslužiti kao prostor unutar kojega se dinamički

umeću novi HTML elementi kako bi se sagradilo korisničko sučelje. U oglednoj aplikaciji sam prema preporukama definirao korijenski element *div* s atributom *id* = '*root*'. Glavni modul indeks.js uvozi razred *React* iz biblioteke *react* i razreda *ReactDOM* iz biblioteke *react-dom.* Metoda *render* (slika 16) razreda *ReactDOM* je metoda koja se prva poziva kako bi se pokrenula aplikacija u web pregledniku. Pozivom ReactJS pokreće proces dinamičkog umetanja početne komponente unutar korijenskog elementa glavnoga HTML-dokumenta.

```
ReactDOM.render(<App />, document.getElementById('root'));
```

Slika 16. Pokretanje aplikacije

Kao prvi argument metode *render* predaje se početna komponenta. Drugi argument je referenca na korijenski element. Slično kao i kod Angulara za opis komponenti se koristi termin pogled za predstavljanje unutar web preglednika. Pogled početne komponente se sastoji od niza drugih komponenti koje ona koristi. Na taj način ostvaruje se stablasta struktura komponenti koja čini korisničko sučelje. Budući da je ReactJS aplikacija jednostranična, potrebne su učestale manipulacija HTML-dokumentom. Manipulacije su vrlo zahtjevne za performanse web preglednika i zbog toga se ne preporučaju, jer ne pružaju zadovoljavajuće iskustvo za korisnike aplikacije. Virtualni dokument jest mehanizam koji nudi rješenje problema. Svaka ReactJS-ova komponenta ima svoj HTML-predložak i njegovu kopiju, virtualni dokument. Prilikom promjene stanja komponente pamti se sadržaj trenutačnog HTML-predloška. Umjesto da se izmijeni svaki element njezinog HTML- predloška mijenja se virtualni dokument. Izmjena virtualnog dokumenta je mnogo brža u odnosu na izmjenu HTML- dokumenta. Nakon što se promijene svi elementi virtualnog dokumenta obavlja se usporedba sačuvanog stanja HTML-predloška komponente i virtualnog dokumenta te se utvrđuje koji su se točno elementi izmijenili. Potom se ne mijenjaju svi elementi HTML-predloška već samo oni koji su promijenjeni u virtualnom dokumentu. Kao zadnji korak izmjene stanja komponente je dinamičko umetanje izmijenjenog HTML predloška unutar korijenskog elementa. Korisnik interakcijom s web preglednikom uzrokuje dinamičko umetanje ili uklanjanje pojedinih komponenti iz stablaste hijerarhije komponenti. Kako bih postigao što veću funkcijsku koheziju funkcionalnosti koje

komponente obavljaju implementirao sam ih pomoću usluga. Pogodno je izraditi usluge kako bi se omogućila ponovna upotreba učestalo korištenih metoda. Zahvaljujući modulima u ES6 svaka komponenta može uvesti uslugu koja je izvedena iz nekog drugog modula. Logika aplikacije je sadržana u funkcijama servisa, a predstavljanje u pogledima komponenti. Osim bolje kohezije, razdvajanjem logičkog od vizualnog dijela aplikacije se osigurava lakša izmjena sustava. Preporuka je koristiti ReactJS u svim aplikacijama u kojima se javlja potreba za izradom složenog interaktivnog korisničkog sučelja s ciljem poboljšanja korisničkog iskustva.

4.3. Komponente

Konceptualna ideja biblioteke je zasnovana na temeljnom principu programskog inženjerstva „podijeli pa vladaj“. Cilj je složenu funkcionalnost olakšati razbijajući ju na jednostavnije cjeline koje su zapravo Reactove komponente. Postoje dva načina deklariranja komponenti ReactJS-a. Jednostavnije komponente se mogu definirati kao JavaScriptove funkcije koje vraćaju kod pisan JSX-sintaksom, što postaje vrlo nepregledno ukoliko komponenta ima mnogo funkcionalnosti. Tada je praksa da se koristi ES6-sintaksa definirajući komponentu ključnom riječju *class.* Nužno je da razred koji predstavlja komponentu naslijedi razred *Component* (slika 17: redak 9) koji se uvozi iz biblioteke *react*. Naziv komponente mora započinjati velikim slovom, jer u suprotnom će ReactJS smatrati da se radi o standardnom HTML elementu. Komponentama je moguće definirati svojstva (engl. props). Svojstva komponente služe samo za prikaz podataka te ih komponenta samoinicijativno nikada ne bih trebala mijenjati. Budući da se vrijednosti varijabli razreda i stanje komponente mijenjaju kroz životni vijek instance nije dovoljno da komponenta ima samo svojstva. Zbog navedenog se uvodi koncept stanja (engl. state). Stanje je slično svojstvu, no ono je privatno i pod kontrolom je isključivo komponente, dok su svojstva komponente javna i mogu se kontrolirati iz drugih izvora. Razred komponente ima metodu *render* koja se poziva svaki put kada se dogodi bilo kakva izmjena vezana uz komponentu. Kao povratnu vrijednost metoda *render* vraća predstavljanje komponente, što je zapravo kod pisan JSX-sintaksom. Vrlo je važno da se prilikom programskog uništenja komponente

oslobode svi zauzeti resursi. Zbog navedenog, biblioteka generira pozive unaprijed definiranih metoda u ključnim trenutcima životnog ciklusa pojedine komponente. Tako primjerice možemo koristiti metodu *componentWillUnmount,* koja se poziva prije nego se komponenta uništi, kako bi smo oslobodili zauzetu memoriju. Još jedan primjer upotrebe životnog ciklusa je potreba za dohvatom podataka. Na slici 17 je prikazana *TablePage* komponenta koja koristi ugrađenu metodu ReactJS-a *ComponentWillMount* kako bi pravovremeno dohvatila podatke s poslužitelja. *TablePage* je složena komponenta čiji pogled nudi više mogućnosti. Pogled komponente koji se prikazuje korisniku sadrži tablični prikaz zapisa kilaže, obrazac koji nudi mogućnost unosa, brisanja i izmjene zapisa, te obrazac kojim se odabire period koji vremenski ograničava podatke za prikaz. Komponenta *TablePage* koristi druge komponente aplikacije: *Table, PeriodPicker* i *Record* kako bi obavila potrebne funkcionalnosti. Iz primjera se vidi kako je zahvaljujući manjim komponentama problematika podijeljena na više dijelova što olakšava implementaciju složenijih funkcionalnost, jer se problem dijeli na manje komponente. Unutar ugrađene metode *componentWillMount* (slika 17: redak 10) šalje se zahtjev poslužitelju za podacima te, kada oni stignu kao odgovor, osvježava se stanje komponente. Metoda se poziva u trenutku kada započne proces umetanja pogleda komponente u grafičko sučelje. Pristigli podaci se prosljeđuju komponenti *Table* koja je zadužena za grafički prikaz.

```
9   class TablePage extends Component {
10
11    constructor(props){
12      super(props);
13      this.recordChild = React.createRef();
14
15      this.state = {
16        tableData: []
17      }
18    }
19
20    componentWillMount() {
21      logService.getAllLogs()
22                .then(res => {
23                  let logs = [];
24                  res.data.forEach(log => {
25                    logs.push(new Log(log.id,log.value,(new Date(log.date).toLocaleDateString())))
26                  });
27                  this.setState({tableData: logs})
28                })
29                .catch(err => window.location.href='error');
30    }
```

Slika 17. Ugrađena metoda *componentWillMount*

4.4. Mehanizmi komunikacije komponenti

Glavni mehanizam komunikacije ReactJS-ovih komponenata je putem svojstava komponenti. Svojstva mogu biti bilo koja vrsta podataka, objekti, liste ili funkcije. Kako bih se komponenta koristila unutar druge komponente, koristimo sintaksu HTML oznaka. Na slici 18 prikazan je prijenos podataka uporabom svojstava. Komponenta *Route* se uvozi iz biblioteke *react-router-dom.* Komponenta na temelju URI-a web preglednika donosu odluku o tome koja će se komponenta prikazivati. Kada je putanja koja određuje URI jednaka */register* poziva se komponenta *SignForm.* Komponenta prima podatke pohranjene u varijabli *registerProps* putem svojstva *data* koji je sintaksno definiran kao atributa HTML-elementa.

```
26  <Route path='/register' render={() => (
27    <SignForm data = {registerProps}/>)
28  } />
```

Slika 18. Prijenos podataka putem svojstava

Kako se registracija i prijava korisnika u sustav razlikuje samo na temelju podataka koji se prikazuju unutar obrasca koji korisnik treba ispuniti, obje funkcionalnosti je moguće obaviti pomoću iste Reactove komponente *SignForm.* Komponenta *SignForm* će zahvaljujući mehanizmu prijenosa podataka putem svojstava primiti objekt prema slici 19. Ukoliko se radi o obrascu koji je potrebno ispuniti prilikom registracije korisnika, komponenta će napraviti obrazac prema vrijednostima atributa *formControls* objekta *registerProps*. Nakon što korisnik ispuni sve podatke, komponenta poziva uslugu kojoj predaje vrijednost svojstva *postUrl* objekta *registerProps* te podatke koje je korisnik unio ispunivši obrazac. Usluga tada upućuje HTTP-zahtjev poslužitelju. Ukoliko želimo da

```
26  export const registerProps = {
27      formControls: {
28        firstName: {
29          label:'First Name',
30          value: '',
31          type: 'text'
32        },
33        lastName: {
34          label:'Last Name',
35          value: '',
36          type: 'text'
37        },
38        username: {
39          label:'Username',
40          value: '',
41          type: 'text',
42          required: true
43        },
44        password: {
45          label: 'Password',
46          value: '',
47          type: 'password',
48          required: true
49        }
50      },
51      error: {
52        message: null
53      },
54      postUrl: apiUrl+'/users/register',
55      registerMode : true,
56      action: 'Register'
57    };
```

Slika 19. Podaci za prijenos

komponenta generira obrazac za prijavu korisnika dovoljno je izmijeniti svojstva objekta koji se šalje komponenti *SignForm* putem svojstva *data* kako bih se postigla željena funkcionalnost. Opisano je primjer prijenosa podataka roditelja djetetu.

Kako bi se podaci prenosili od djeteta prema roditelju, roditelj može djetetu putem svojstva predati funkciju koju dijete potom poziva te šalje podatke kao argumente predane funkcije, vrlo slično mehanizmu kojega u Angularu pruža uporaba dekoratora *@Output*. Dakle, pomoću svojstava je vrlo lako prenositi podatke između komponenti koje su u hijerarhijskom odnosu. Kada je riječ o komponentama koje su nepovezane, može se koristiti propagacija podataka putem svojstava kroz lanac komponenti, no u slučaju velike složenosti to nije primjereno. Za komunikaciju udaljenih komponenata ReactJS podržava uporabu mehanizma konteksta (engl. Context). Kontekst predstavlja prostor za definiranje dijeljenih podataka, a za instanciranje se koristi naredba *React.CreateContext()*. Naredba stvara razred koji koristi komponente *Provider* i *Consumer.* Komponenta *Provider* putem svojstva prima vrijednosti koje se žele prenositi između udaljenih komponenti. Komponenta *Consumer* služi kao omotač oko komponenti koje će imati dostupne podatke koje komponenta *Provider* ima definirane putem svojstva. Tijelo čini anonimna funkcija čiji su argumenti dijeljeni podaci, a kao povratnu vrijednost vraća kod pisan JSX-sintaksom koji sadrži komponente koje se žele koristiti. Nedostatak konteksta dolazi do izražaja kada složenost aplikacije raste, jer ne postoji automatski mehanizam sinkronizacije, već je nužno pisati kod koji čita i postavlja vrijednosti instance razreda *Context*. Učestala izmjena vrijednosti čini kod ranjivim na pogrješke, pogotovo ako se u sustavu koriste asinkroni događaji. *Context* ne podržava objavljivanje i reagiranje na događaje. Za tu svrhu preporučeno je koristiti mehanizme koji nisu sastavni dio biblioteke, npr. JavaScriptove događaje. Ideja je da određena komponenta generira događaj, te zainteresirana komponenta sluša okidanje događaja. Uporabom JavaScriptovog događaja postiže se slična funkcionalnost koju Angular implementira uporabom razreda *Subject*. Razvojni tim ReactJS-a kao najbolje rješenje komunikacije nepovezanih komponenti predlaže uporabu Reduxa koji je također biblioteka JavaScripta, a čija je svrha upravljanje stanjem cijele aplikacije.

4.5. Stanje, podaci i prikaz komponente

Stanje komponente je predstavljeno objektom koji se definira u konstruktoru. Komponenta *ChartPage* ogledne aplikacije unutar svog konstruktora inicijalno postavlja vrijednost stanja na objekt čiji je atribut *chartData* prazna lista (slika 20: retci 11-12). Funkcionalnost komponente jest ostvariti grafički prikaz podataka unosa kilaže korisnika. Unutar konstruktora ReactJS automatski povezuje svojstva koja su definirana u komponenti od strane roditelja. Kako bih se pravilno povezala svojstva komponente nužno ih je proslijediti pozivajućem konstruktoru nadrazreda *Component* (slika 20: redak 10). Budući da ReactJS radi asinkrono, važno je povezati funkcije komponente s trenutnim kontekstom (slika 20: redak 14). To se radi tako da se unutar konstruktora nad funkcijom pozove metoda *bind* koja kao argument prima kontekst označen ključnom riječju *this*.

```
8   class ChartPage extends Component {
9     constructor(props){
10      super(props);
11      this.state = {
12        chartData: []
13      }
14      this.mapChartData = this.mapChartData.bind(this);
15    }
```

Slika 20. Konstruktor komponente *ChartPage*

Kako bi se podaci zapisa kilaže korisnika prikazali, potrebno ih je dohvatiti s poslužitelja šaljući HTTP-zahtjev. Jednom kada stigne odgovor, poziva se funkcija *mapChartData* koja kao argument prima listu unosa kilaže korisnika primljenih s poslužitelja i naslov koji opisuje grafički prikaz (slika 21: redak 22). Unutar funkcije se prilagođavaju podaci iz baze za prikaz korisniku. Jednom kada je ulazna lista prilagođena, stanje komponente se mijenja pozivom funkcije *this.setState* (slika 21: redak 30) pri čemu se *this* odnosi na ispravan kontekst, jer je povezan s funkcijom unutar konstruktora komponente (slika 20: redak 14). Funkciji *setState* se predaje objekt koji postaje vrijednost stanja komponente.

```
22    mapChartData(dataToMap, title) {
23      let dataPoints = [];
24        dataToMap.forEach( log => {
25          dataPoints.push({
26            x: new Date(log.date),
27            y: log.value
28          });
29        });
30        this.setState({chartData: dataPoints, chartTitle: title});
31    }
```

Slika 21. Promjena stanja komponente

Važno je istaknuti kako ReactJS, za razliku od Angular-a i VueJS-a ne pruža mehanizam dvosmjernog povezivanja, objašnjenog u poglavlju 3.6. Biblioteka dvostruko povezivanje imitira dinamičkim podešavanjem stanja. Iako se funkciji *setState* predaje objekt, stanje se ne postavlja na predani objekt već se ažuriraju atributi postojećega stanja. Ukoliko stanje komponente sadrži atribute *a,b,c i d* te se u nekom trenutku promijeni stanje pozivom funkcije *setState* kojoj se preda objekt s atributima *a* i *b*, atributi *c* i *d* prethodno postavljenog stanja se neće izgubiti, dok će vrijednosti atributa *a* i *b* zamijeniti novima. Iako se stanje definira kao varijabla komponente, ono nije klasična varijabla te ako se proba eksplicitno mijenjati stanje kao što se mijenja vrijednost ostalih varijabli, npr. *this.state = {…},* metoda *render* se neće pozvati što rezultira time da se zapravo na vizualnom prikazu komponente ne vidi promjena unutarnjeg stanja. Kako bih se unutar HTML-ovog predloška prikazala vrijednost varijable komponente, tj. postigla interpolacija, slično kao i u Angularovom okviru, potrebno je navesti ime varijable čiju vrijednost se želi prikazati i uokviriti ju unutar jednostrukih vitičastih zagrada, dok Angular za potrebe interpolacije koristi dvostruke vitičaste zagrade.

4.6. Oblikovanje

ReactJS ne koristi posebne mehanizme poput Angulara za jednoznačno povezivanje CSS-datoteka s grafičkim prikazom komponente kako bi se spriječili konflikti CSS-selektora. Kako bi se oblikovanje napisano u CSS-u primijenilo na komponentu, potrebno je navesti relativnu putanju do datoteke unutar modula ES6 u kojemu je pisan kod komponente. Budući da je česta praksa da se JSX-ov kod koji zamjenjuje HTML-ov kod piše direktno unutar metode *render* određenog razreda, prikladno je koristiti linijski stil definiranja CSS-oblikovanja. Potrebno je

definirati objekt (slika 22: retci 87-90) koji se potom predaje kao vrijednost atributa *style* HTML-elementa (slika 22: redak 93) na koji se želi primijeniti definirano oblikovanje.

```
86   render() {
87     const formStyle = {
88       marginTop: '40px',
89       textAlign: 'center'
90     }
91       return (
92         <div className="d-flex justify-content-center">
93           <form style={formStyle} onSubmit={this.submit}>
```

Slika 22. Linijsko definiranje CSS stila

Linijskim stilom definiranja oblikovanja prividno se sakrivaju CSS-konflikti, jer se na komponentu uvijek primjenjuje stil s najvećim prioritetom. Najveći mogući CSS-ov prioritet imaju upravo oni stilovi koji su definirani linijski.

5. VueJS

5.1. Podrška razvoju

Osnivač VueJS-a je Evan You. Prilikom svoga rada u Googleu dobio je ideju za razvoj vlastitog radnog okvira. Bio je nezadovoljan AngularJS-om kojega je tada aktivno koristio. VueJS zbog toga sintaksom podsjeća na AngularJS, no brži je po vremenskim performansama i zauzima manje memorije. Prva verzija je objavljena u veljači 2014. godine. Jezik u kojem se piše je poput i ReactJS-a najčešće JavaScript, koristeći sintaksu ES6. Proizvođači su omogućili i upotrebu TypeScripta. Danas okvir održava njegov osnivač Evan You zajedno sa svojim mnogobrojnim timom. Prema statistikama sa Stack Overflow-a, VueJS je postigao najveći skok u uporabi i popularnosti gledajući razdoblje od 2016. do 2018. godine. Kako bi se krenulo s razvojem, nužno je imati dostupnu skriptu s kodom implementacije. To se može, kao i kod ReactJS-a, napraviti preuzimanjem skripte putem CDN-a, ali se može i koristiti naredbeni redak koji preuzima skriptu na lokalno računalo unutar projekta. Za početak razvoja korištenjem naredbenog retka, potrebno je imati instaliran upravitelj paketa npm i node.js. Upotrebom

funkcionalnosti npmovog paketa *vue/cli* imamo mogućnost generiranja strukturalnih dijelova aplikacije. Naredba > *vue create vue-client* generira preporučenu strukturu, sastavljenu od niza datoteka i kazala pozicioniranih unutar korijenskog kazala *vue-client*. Dobiven je vrlo pregledan predložak koji je pogodan za nastavak razvoja. Zbog jednostavnosti sintakse komponenti, mnogi ih programeri vlastoručno implementiraju bez upotrebe dodatnih alata, što sam osobno primjenjivao prilikom izrade ogledne aplikacije. Okvir nudi biblioteku koja sadrži veliki broj već stvorenih komponenti. Razvijene su od istog tima koji održava okvir, javno su dostupne i besplatne za upotrebu. Oblikovane su prema specifikacijama vizualnog jezika Material Design te su objavljene pod nazivom Vuetify. Njihovom uporabom omogućuje se razvoj moderno oblikovanih web aplikacija. U svrhu boljega razumijevanja funkcionalnosti i arhitekture okvira, za oglednu aplikaciju sam se odlučio za izradu vlastitih komponenti. Jedina korištena komponenta Vuetify biblioteke jest *VDataTable* čija je funkcionalnost tablični prikaz podataka. Kako bih prikazao zapise kilaže korisnika ogledne aplikacije, koristio sam upravo komponentu *VDataTable*. Dodatnim proširenjem funkcionalnosti i implementacijom postigao sam straničenje podataka, pretraživanje i sortiranje stupaca.

5.2. Organizacija aplikacije

VueJS ne nameće određeni oblik organiziranja repozitorija, kao ni strukturu aplikacije. Preporuka razvojnog tima je da se koriste mehanizmi modula koje donosi ES6, te da se za optimizaciju koristi alat poput Webpacka koji sažima više datoteka u jednu, optimizirajući njezinu veličinu. Stvaranjem aplikacije pomoću naredbenog retka dobivamo generički predložak. U glavnom modulu *main.js* uvozi se razred *Vue* iz biblioteke *vue,* te razred *VueRouter* iz biblioteke *vue-router*. Kako bi se aplikacija pokrenula, potrebno je stvoriti instancu razreda *Vue* pozivom konstruktora kojemu se predaje objekt koji sadrži atribut *render*. Ovaj atribut kao vrijednost sadrži metodu koja slično kao i u ostalim okvirima prima početnu komponentu. Uobičajeno ju je nazvati *App* (slika 23: redak 36). Metoda dinamički umeće HTML-predložak početne komponente unutar korijenskog elementa. U glavnom HTML-dokumentu nalazi se element s atributom *id = 'app'* koji se

referencira unutar poziva funkcije *$mount('#app')* (slika 23: redak 38), pri čemu je *'#app'* selektor koji traži element s vrijednošću atributa *id 'app'.*
Objektu koji je argument konstruktora *Vue* razreda također se postavlja atribut *router*, čija je vrijednost instanca razreda *VueRouter* koji je jedan od osnovnih razreda okvira. *VueRouter* na temelju URI-a web preglednika manipulira korisničkim sučeljem i odlučuje koja će se komponenta prikazivati.

```
35  new Vue({
36    render: h => h(App),
37    router
38  }).$mount('#app')
```

Slika 23. Pokretanje aplikacije

Kako je riječ o jednostraničnim aplikacijama glavni dokument je ujedno i jedini dokument koji nije statičan već zahvaljujući automatiziranosti okvira dinamički mijenja sadržaj kojega grade pogledi pojedinih komponenta.

5.3. Komponente

VueJS koristi JavaScriptove objekte i module pisane u ES6 kako bi povezao strukturne dijelove komponenti, za razliku od Angulara koji za to koristi dekoratore. Svaka komponenta je instanca već spomenutoga razreda *Vue*. Komponenta unutar datoteke .vue treba imati tri definirana dijela. Na slici 24 je prikazan isječak koda koji povezuje temeljne elemente gradeći komponentu *ChartPage.* Unutar oznaka <*template*> piše se HTML-kod koji predstavlja pogled komponente u web pregledniku. Kao i ReactJS, VueJS ne mijenja glavni HTML-ov dokument prilikom svake interakcije korisnika već koristi mehanizam virtualnog dokumenta. Unutar oznaka <script> piše se kod koji obavlja glavnu funkcionalnost komponente. Budući da je razred anoniman, ne definirati se prema klasičnoj sintaksi već upotrebom ključnih riječi *export class {…}*. Umjesto naziva razreda, anonimni razred sadrži objekt name čija vrijednost služi za kreiranje instance komponente. Unutar oznaka <style>, piše se CSS-dokument koji se odnosi na oblikovanje komponente.

```
1  <template src="./chart-page.html"></template>
2  <script src="./chart-page.js"></script>
3  <style scoped src="./chart-page.css"></style>
```

Slika 24. Povezivanje građevnih dijelova komponente

Osim spomenutog objekta *name*, značajni su i objekti *components*, *props*, *data* itd. Objekt *Components* ima vrijednost imena svih komponenti koje se koriste unutar komponente koju implementiramo, objekt *props* služi za prijenos podataka implementiranoj komponenti, dok *data* predstavlja objekt koji sadrži sve varijable razreda komponente.

Angular i VueJS su radni okviri koji, za razliku od ReactJS-a, podržavaju direktive. Direktiva omogućuje automatiziranu manipulaciju HTML-elementima. Primjeri direktiva VueJS-a su *v-if, v-else, v-for, v-on, v-bind, v-model* i mnoge druge. Direktive se sintaksno koriste kao atributi HTML-elemenata. Na slici 25 je prikazan kod datoteke .html komponente *SignForm* čija je funkcionalnost objašnjena u poglavlju 4.5. Komponenta stvara obrazac s poljima koje korisnik popunjava. Umjesto da se za svako polje piše kod, koristi se direktiva *v-for* . Primjenom direktive može se iterirati nad elementima liste, svojstvima i atributima objekata. Komponenta *SignForm* koristi direktivu kako bi iterirala po atributima objekta *formControls* (slika 25: redak 3). Direktiva daje referencu na vrijednost atributa, generira slijedni broj i jedinstveni identifikator *key*. Prikazani kod će generirati polja onoliko koliko objekt ima atributa, te će kao vrijednost tekstualnog opisa polja, sadržaj *label* HTML-elementa, iskoristiti vrijednost *value* atributa objekta *formControls*. Primjerom je pokazano kako se vrlo jednostavno može implementirati opsežan obrazac bez da se nužno pišu svi elementi obrasca.

```
2   <form class="form-login" @submit="submit">
3     <div class="form-group" v-for="(value, key, index) in formControls" :key="index">
4       <label :for="key">{{value.label}}</label>
5       <input :type="value.type" class="form-control" :name="key"
6       :aria-describedby="key" :placeholder="'Enter '+value.label"
7               v-model="value.value"  :required="value.required"
8               v-on:change="error = null"
9       />
```

Slika 25. Direktiva *v-for*

Valja naglasiti kako i Angular i VueJS nude velik broj sličnih direktiva koje se koriste za vrlo efikasne pokrate. Osim pokrata, direktive pružaju sigurnost, jer okvir garantira realizaciju funkcionalnosti direktive i time smanjuje učestalost pogrešaka.

5.4. Mehanizmi komunikacije komponenti

Za razmjenu podataka između komponenti koristi se mehanizam svojstava, što je konceptualno isto kao i kod ReactJS-a, razlika je jedino u sintaksi. Kod ReactJS-a, komponenta dijete ne mora nužno definirati nazive svojih svojstava te roditelj može slobodno imenovati svojstvo koje predaje. Kako bi komponente VueJS-a koje su u odnosu roditelj-dijete komunicirale preko svojstava, nužno je da dijete definira nazive svojstava kao vrijednosti objekta *props*. Na slici 26 prikazana je komponenta *Modal* koja kao vrijednost objekta *props* prima nazive svojstava (slika 26: redak 45). Vrijednosti svojstava komponenta dijete može koristiti, poput varijabli razreda. Preduvjet da dijete koristi svojstva jest da im roditelj pridijeli vrijednost.

```
44  <script>
45  export default {
46      name:'modal',
47      props: {
48          closeFunc: Function,
49          updateFunc: Function,
50          message: String
51          },
```

Slika 26. Objekt *props*

Radni okvir ne stavlja ograničenje na tip podataka koji se prenosi, tako da se kao vrijednost objekta *props* mogu prenosi podaci u obliku jednostavnih vrijednosti (slika 26: redak 50), složenih objekata ili funkcija (slika 26: retci 48-49). Isto kao i kod ReactJS-a, roditelj koristi komponentu dijete rabeći sintaksu HTML oznaka, a podaci koji se prenose su definirani kao atributi. Kako bi okvir znao pravilno povezati vrijednosti sa svojstvima, potrebno je navesti dvotočku ispred atributa. Ako želimo koristiti komponentu *Chart* i predati joj podatke pohranjene u varijabli *chartData* putem svojstva *data*, to možemo postići sintaksom prema slici 27.

```
<chart :data="chartData"></chart>
```

Slika 27. Sintaksa upotrebe komponente

Pri tome je nužno da komponenta *Chart* bude definirana kao na slici 28.

```
export default {
    name: 'chart',
    props : {
        data: Object
    },
```

Slika 25. Definicija komponente *Chart*

Kada roditelj u atribut oznake komponente upiše vrijednost koja nije funkcija, dobiva se mehanizam slanja podataka roditelja djetetu. Kako bi dijete slalo podatke roditelju, potrebno je da roditelj kao vrijednost atributa oznake djeteta postavi vlastitu funkciju. Dijete tada poziva funkciju i putem argumenata šalje podatke, čime se dobiva mehanizam slanja podataka djeteta roditelju. Osim toga, slično kao i u Angularovom radnom okviru, VueJS podržava mogućnost da dijete generira događaj na koji se roditelj može registrirati. Tako će, prema slici 29, dijete *AppTable* generirati događaj *rowClicked.* Okvir će potom pozvati metodu roditelja *tableRowClicked* s argumentima koje je dijete odredilo. Kako bi okvir ispravno povezao događaj koji se okida, potrebno je da se naziv događaja navede poput HTML-ovog atributa kojemu prethodi znak @.

```
<app-table @rowClicked="tableRowClicked" :data="tableData"></app-table>
```

Slika 29. Okidanje događaja *rowClicked*

Za prijenos podataka između udaljenih komponenti koriste se globalne varijable. Za generiranje događaja koji se koriste između nepovezanih komponenti koristi se mehanizam globalne komponente. Iz glavne datoteke izvozi se instanca razreda *Vue* koju potom uvozi komponenta koja želi okinuti događaj te nad njom poziva funkciju *$emit.* Komponenta zainteresirana za događaj se prijavljuje na objavu događaja pozivom funkcije *$on* nad instancom globalne komponente. Prema slici 30, komponenta se prijavljuje na događaj *auth_changed* i prilikom svake objave događaja izvršava se funkcija predana kao drugi argument.

```
authEmitter.$on('auth_changed', val => this.isUserLogged = val)
```

Slika 30. Globalna komponenta

5.5. Stanje, podaci i prikaz komponente

Podaci komponente su pohranjeni kao vrijednosti atributa objekta *data*. Kada se vrijednosti objekta promijene, predstavljanje komponente se automatski osvježava, ukoliko njezin prikaz ovisi o internim podacima, bez potrebe za intervencijom programera. Dakle, baš poput Angulara, okvir VueJS podržava dvosmjerno povezivanje. Važno je istaknuti da se atributi i svojstva objekta *data* ne mogu naknadno dodati kao što je to uobičajeno za objekt JavaScripta, već se moraju definirani prilikom inicijalizacije komponente. Iinstanca razreda *Vue* koja predstavlja komponentu, sadrži niz drugih objekata čija imena vrlo precizno opisuju njihovu funkcionalnost. Tako radni okvir nudi mogućnost definiranja objekta *methods* čiji su atributi metode koje se mogu pozivati. Okvir nudi i naprednije mehanizme definirane pomoću objekata *computed* i *watch*. Atributi *computed* objekta odnose se na podatke čija se vrijednost izračunava na temelju vrijednosti nekih drugih varijabli razreda, dok su atributi objekta *watch* metode koje se automatski pozivaju kada se istoimene varijable promjene. Prikazani isječak koda (slika 31) se odnosi na komponentu koja grafički prikazuje unose kilaže korisnika ogledne aplikacije. Definira se atribut *data* objekta *watch* (slika 31: retci 6-11) čija je vrijednost metoda koja se poziva svaki puta kada se promijeni vrijednost istoimenog atributa *data* objekta *props* (slika 31: retci 3-5).

```
1  export default {
2      name: 'chart',
3      props : {
4          data: Object
5      },
6      watch: {
7          data: function() {
8              var chart = new CanvasJS.Chart("chartContainer", this.data)
9              chart.render();
10         }
11     }
12 }
```

Slika 31. Mehanizam *watch*

Zahvaljujući mehanizmima *watch* i *computed*, programer piše manje koda i time smanjuje mogućnost pogrešaka te ne mora brinuti o pravovremenom izvršavanju, jer je pravovremenost garantirana. Kako bih se unutar pogleda interpolirale vrijednosti, koristi se kao i u Angularu sintaksa s dvostrukim vitičastim zagradama.

5.6. Oblikovanje

Komponenti se prilikom uporabe može definirati atribut *class*, kao i svakom HTML-elementu. Okvir nudi mogućnost definiranja privatnog ili javnog dosega. Ukoliko se radi o privatnom dosegu, definirano oblikovanje unutar komponente se neće odnositi na druge HTML-elemente. U slučaju definiranja javnog dosega, cijelo oblikovanje bit će globalno i odnosit će se na svaki HTML-element aplikacije. Kako bih se koristio privatni doseg, sve što je potrebno jest postaviti atribut *scoped* (slika 32) unutar oznake <*style*>, kako bi radni okvir automatski spriječio prelijevanja oblikovanja na ostatak aplikacije.

```
3    <style src="./record.css" scoped></style>
```

Slika 32. Privatni CSS-doseg

6. Usporedba

6.1. Kvalitativna

Angular je znatno opsežniji i složeniji od svojih konkurenata, zbog čega pruža najbolju podršku automatiziranom razvoju programskog sustava. Glavne kritike su upravo na račun njegove složenosti. Prema statistikama, vrijeme potrebno za učenje je mnogo veće u usporedbi s konkurencijom. Mnogi se slažu kako je potrebno posvetiti dugi niz godina razvoju u Angularu kako bi se s pravom moglo tvrditi da je netko uistinu Angularov programer. VueJS je potpuna suprotnost što se tiče vremena učenja. Kako bih se shvatile osnove dovoljno je uložiti svega nekoliko sati ako programer razumije mehanizme i principe razvoja aplikacija unutar komponentno usmjerene arhitekture. Upravo je zbog svoje jednostavnosti

VueJS prema statistikama radni okvir koji se preporučuje početnicima ili ljudima koji trebaju vrlo brzo razviti jednostavniji sustav. Glavni razlog brzini učenja je činjenica da VueJS nema mnogo inovativnih koncepata koji su specifični za radni okvir, već je sve zasnovano na temeljnim konceptima komponentnog razvoja i jezika JavaScript. Koncepti poput dvosmjernog povezivanja, prijenosa podataka između komponenti, automatskog okidanja događaja i reakcija na događaj su sastavni dijelovi okvira Angular i VueJS te se oni obavljaju automatski bez potrebe pisanja koda. ReactJS predstavlja srednji izbor između dviju krajnjosti poput Angulara i VueJS-a po pitanju vremena koje je potrebno za učenje. Nije složen i zahtjevan poput Angulara, no zahvaljujući brzo rastućoj zajednici i mnoštvu jednostavnih alata s kojima se vrlo dobro uparuje, nudi gotovo jednaku automatiziranu podršku. Uspoređujući ga sa VueJS-om, može se utvrditi da je ipak ponešto zahtjevniji za učenje i teži za razumijevanje. Prvenstveno zbog novih koncepata i mehanizama koje uvodi. Nedostatak je i činjenica da nije pravi radni okvir, zbog čega se ništa ne odvija automatski, poput automatske izmjenu stanja kada je korisnik u interakciji s komponentom. Potrebno je pisati kod koji se izvršava kada korisnik okine događaj interakcijom nad grafičkim sučeljem komponente. Zbog navedenog, zahtijeva veći oprez i bolju razradu, što nije nužno loše jer u usporedbi s ostalim okvirima daje najveću slobodu. Omogućuje mnogo inovativnosti i kreativnosti prilikom implementacije te ne prisiljava programera da koristi točno određeni način rješavanja problema.

Kako bi se počelo raditi koristeći Angular, potrebno je poznavati TypeScript i objektno orijentirano programiranje. Za razliku od ReactJS-a i VueJS-a, ne nudi mogućnost pisanja u JavaScriptu što mu je veliki nedostatak i još jedna od niza karakteristika koje doprinose vremenu učenja. Zbog svoje objektne orijentiranosti Angular sugerira razbijanje problematike na većoj razini apstrakcije, što je korisno ukoliko se radi o složenim projektima, no također ponekad i nepotrebno. Za pokretanje projekta se treba instalirati niz paketa i alata, te podesiti mnogo konfiguracija. Kako bi se započelo s projektom upotrebom ReactJS-a ili VueJS-a, dovoljno je unutar glavnog HTML-dokumenta uključiti nekoliko skripti koje su javno dostupne putem CDN-a. Također, oba radna okvira podržavaju pisanje koda u JavaScriptu i u TypeScriptu što ih čini fleksibilnijima te daje programeru mogućnost slobodnog odabira stila pisanja.

Angular u svojemu osnovnom obliku podrazumijeva mnogo funkcionalnosti koje vrlo često nisu potrebne u svakoj aplikaciji te ih nije moguće izbaciti kako bi se smanjila zauzeće memorije. Budući da ima vlastiti mehanizam učitavanja modula, ponekad se nepotrebno učitava velika količina koda koja je „sakrivena“ unutar modula koji su nužni za uvoz kako bi aplikacija ispravno funkcionirala. Dodatno, mehanizmi kao što su filteri, dekoratori, strukturalne direktive, dinamičko umetanje usluga također sadrže veliku količinu koda, čineći cijeli radni okvir glomaznim i sporijim prilikom inicijalnog pokretanja. ReactJS i VueJS pružaju bolju mogućnost izbora, jer sadrže manje obaveznih funkcionalnosti. Kako bi se koristile dodatne mogućnosti, potrebno je instalirati vanjske pakete, koji su u slučaju Angulara sastavni dio okvira. Angular također zahtijeva određenu strukturu, u smislu obaveznog deklariranja komponenti i drugih modula unutar korijenskog modula, što nije slučaj s njegovim konkurentima. Razvojni tim je objavio mnogo primjera najbolje organizacije velikih projekata koji su temeljeni na iskustvu. Kako bi se smanjile kritike na račun složenosti, podržan je veliki broj naredbi naredbenoga retka koje automatski kreiraju elemente poput usluga i komponenti. Angularovi suparnici ne nameću strogu strukturu deklaracija, već se koriste mehanizmi modula ES6 i jednostavni uvoz, izvoz drugih modula kada je to potrebno.
Zbog manjka funkcionalnosti ReactJS-a, potrebno ga je integrirati s nizom alata kako bi se postigla potpuna podrška razvoju. Postoji veliki broj dostupnih paketa za npm koje je zajednica razvila kako bi se zaobišli nedostatci. Učestalo korišteni paketi su primjerice *react-router* koji oponaša upravljanje rutama višestraničnih aplikacija, kao i niz paketa za validaciju i formatiranje podataka. I u slučaju VueJS-a, često se uvoze vanjski paketi. Nedostatak uvoza vanjskih paketa je što programer treba imati znanje alata za upravljanje paketima te također treba voditi računa o pouzdanosti paketa prilikom odabira, jer je tu riječ o otvorenom kodu čiji autori nisu nužno povezani s timom proizvođača korištenog radnog okvira, što lako dovodi do nekompatabilnosti. Angularov razvojni tim pruža mnogo veći izbor dodatnih funkcionalnosti uvozom modula koji su dostupni unutar okvira, što smanjuje potrebu za uporabom vanjskih paketa. Time se postiže veća pouzdanost te se smanjuje mogućnost pogrešaka i potreba za prilagođavanjem.

6.2. Kvantitativna

Vrlo bitna komponenta aplikacije vezana uz vrijeme pokretanja jest veličina datoteke *bundle*. To je datoteka koju web preglednik preuzima prilikom inicijalnog učitavanja. Kako korisnik ne bi dugo čekao na učitavanje aplikacije, bitno je optimirati njezinu veličinu. Zbog toga je poželjno da radni okviri imaju mogućnost reguliranja njezine veličine.

Angularov radni okvir obavlja prevođenje u trenutku. Datoteke koje programer piše se dostavljaju pregledniku u svome izvornom obliku, zbog čega preglednik svu sintaksu specifičnu za Angular neće razumjeti te ju neće znati prikazati. Kako se to ne bi dogodilo, radni okvir sadrži prevodilac koji prilikom dostavljanja datoteka pregledniku prevodi kod u sintaksu koju web preglednik razumije. Budući da okvir sadrži kod implementacije prevoditelja, veličina datoteke bundle je veća u usporedbi sa veličinom bundle datoteka ostalih okvira te za oglednu aplikaciju iznosi 84.8 kB . Vremenske performanse prilikom učitavanja Angularove aplikacije su zbog navedenoga lošije u usporedbi sa konkurentima. VueJS podrazumijeva *runtime-only* pristup implementaciji. Takvim pristupom nema potrebe za direktnom dostavom prevodioca web pregledniku. Programer implementira komponente u datotekama sa nastavkom .vue, pišući HTML i kod specifičan za radni okvir. Sav sadržaj datoteke se prevodi u optimirani JavaScriptov kod koji se dostavlja web pregledniku. Zbog toga datoteka *bundle* ne sadrži kod implementacije prevodioca kao kod Angulara te iznosi 40.8 kB. ReactJS koristi JSX-sintaksu, koja semantički omogućuje pisanje HTML-a u obliku JavaScripta. Za razliku od VueJS-a, sav kod je optimiran za direktnu dostavu datoteka web pregledniku, jer ne sadrži izvorni HTML i JavaScriptovu sintaksu u istoj datoteci pa nema potrebe za prevođenjem HTML-a u JavaScript. Veličina ReactJS-ove datoteke *bundle* je 45.6 kB.

Općenito, veličina *bundle* datoteke proporcionalno raste s količinom koda, zbog čega ona može biti vrlo velika, što uzrokuje vrlo sporo inicijalno pokretanje. Sva tri radna okvira nude mehanizme kao što su *lazy loading* i *treeshaking* koji se nude kao rješenja spore inicijalizacije. Za potrebe implementacije manjih projekata nisu od velikog značaja, zbog čega nisu korištene prilikom implementacije ogledne aplikacije, no njihovom upotrebom se dodatno postiže optimizacija rada i pokretanja aplikacije. Prema mjerenjima alata Chrome Developer, prosječno

vrijeme više nezavisnih pokretanja Angularove aplikacije iznosi 46.6 ms, ReactJS-ove aplikacije 64.8 ms, te VueJS aplikacije 40.3 ms. Usprkos činjenici da Angularov radni okvir zauzima više memorije od ReactJS-ovog, pokretanje aplikacije je značajno brže, što potvrđuje kako razvojni tim Angulara mnogo ulaže u optimizaciju koda s ciljem poboljšanja performansi. Osim inicijalnog učitavanja bitna je i brzina kojom radni okviri šalju zahtjeve poslužiteljima. Tablica 1 prikazuje prosječno vrijeme koje je potrebno kako bi klijentska aplikacija uputila HTTP-zahtjev poslužitelju. Svaka akcija je provedena 15 puta te je prikazana prosječna vrijednost izražena u milisekundama. Na brzinu izvođenja su utjecale karakteristike računala na kojemu se aplikacija izvodila kao i ostatak procesa u okolini. Važan je omjer trajanja i međusobna usporedba, a ne konkretne brojke, jer one variraju ovisno o okolini izvođenja. U tri od pet akcija, Angular je bio najbrži, dok je u preostale dvije akcije najbrži bio VueJS. Iako je sami razvojni okvir Angular najrobusniji i zauzima najviše memorije, pokazuje se kako pruža najbržu komunikaciju s poslužiteljem izmjenom HTTP-zahtjeva i odgovora. Razlog tomu je što razvojni tim ulaže mnogo truda u optimizaciju koda kako bi vremenske performanse bile što bolje.

Tablica 1. Zahtjevi klijenata korištenjem metode HTTP POST, u milisekundama

Akcija/ HTTP zahtjevi	**Angular**	**ReactJS**	**VueJS**
Prijava korisnika	10.35	14.42	13.34
Registracija korisnika	25.78	27.36	11.42
Stvaranje zapisa kilaže	25.34	23.89	21.71
Brisanje zapisa	19.88	26.41	21.45
Zahtjev za zapisima u ograničenom vremenskom periodu	19.23	39.61	23.22

Koncept virtualnog dokumenta, koji podržavaju VueJS i ReactJS, je od velike koristi prilikom izvođenja rada sustava. Iako je po konceptima VueJS vrlo sličan ReactJS-u, s obzirom da zauzima najmanje memorije, on je brži od ReactJS-a. Vrlo je teško uspoređivati druge performanse osim veličine datoteka *bundle* i inicijalnog pokretanja, jer one ovise o velikom broju faktora, prvenstveno o računalima na kojima se izvode procesi.

Tablica 2 prikazuje mjerljive karakteristike oglednih klijentskih aplikacija poput zauzeća memorije, broja datoteka i broja kazala. Datoteke, kazala i memorija vanjskih paketa nisu uzeti u obzir u tabličnom prikazu. Ogledna aplikacija je potvrdila kako radni okvir Angular uistinu zauzima najviše memorije, dok je VueJS vrlo blizu ReactJS, no i dalje zauzima najmanje memorije. Valja primijetiti kako je broj datoteka u Angularovoj aplikaciji znatno veći u odnosu na ReactJS-ove i VueJS-ove aplikacije. To je posljedica organizacije projekta i strukture. Angular odvaja HTML-kod, TypeScript-kod, CSS-kod i TypeScriptov testni kod u zasebne datoteke. Kako ReactJS koristi JSX-sintaksu, kod koji predstavlja pogled je smješten u istoj datoteci kao i kod koji sadrži logiku, zbog čega je aplikacija sastavljena od najmanjega broja datoteka. VueJS-ov tim prema službenoj dokumentaciji preporučuje odvajanje datoteka s obzirom na funkcionalnost, no to se ne nameće kao nužno. Prilikom razvoja aplikacije, većinu sam komponenti pisao prema preporukama, stoga je organizacija datoteka slična Angularu. Manji broj sam implementirao pišući kod koji predstavlja logiku i kod koji predstavlja pogled unutar iste datoteke te je struktura tih datoteka slična ReactJS-u.

Tablica 2. Karakteristike klijentskih aplikacija

Radni okvir	**Memorija (kB)**	**Broj datoteka**	**Broj kazala**
Angular	520	97	25
ReactJS	494	44	19
VueJS	484	55	20

Tablica 3 pokazuje iste parametre kao i tablica 2, samo je u obzir uzeta memorija, broj datoteka i kazala svih vanjskih paketa koji su instalirani prilikom razvoja. Razvoj ogledne aplikacije je potvrdio teorijsku statistiku i dokazano je kako je VueJS moćan radni okvir sam za sebe, koji uistinu može postići veliki broj funkcionalnosti bez potrebe za korištenjem vanjskih paketa. Angular je najsloženiji radni okvir i nakon instalacije potrebnih paketa projekt zauzima najviše memorije. Također je potvrđeno kako je ReactJS biblioteka koja nudi najuži skup funkcionalnosti te je za razvoj potrebno koristiti veliki broj vanjskih komponenti, zbog čega je zauzeta memorija projekta gotovo jednaka memoriji Angularovog projekta, iako je sama aplikacija bez vanjskih paketa osjetno manja u usporedbi sa

samostalnom Angularovom aplikacijom. Još jedna bitna razlika između Angularove i ReactJS-ove aplikacije je ta da su većina instaliranih paketa u Angularovoj aplikaciji osnovni dio okvira ili su proizvedeni od strane razvojnog tima i uvode se kao opcionalni. S druge strane, većina paketa ReactJS aplikacije su paketi treće strane, odnosno paketi koji nisu direktno razvijeni od strane razvojnog tima ReactJS-ove biblioteke, no kompatabilni su za upotrebu s bibliotekom te ih je nužno uvesti kako bih se ostvarile željene funkcionalnosti.

Tablica 3. Karakteristike klijentskih projekata

Radni okvir	**Memorija (MB)**	**Broj datoteka**	**Broj kazala**
Angular	232	29 822	4 071
ReactJS	226	34 602	4 993
VueJS	124	22 353	3 275

Potrebno je naglasiti kako su sva tri radna okvira vrlo brza i razlike su zapravo nijanse. U usporedbi s ostalim JavaScriptovim radnim okvirima, poput EmberJS-a, KnockoutJS-a i AngularJS-a, ispitani radni okviri su znatno brži te pružaju bolji omjer brzine rada i memorije. Ukoliko se dogodi značajni gubitak performansi, to je vjerojatno posljedica lošeg pristupa razvoja, a ne nedostatak okvira. Kada bih se moralo definirati koji okvir ima najbolje performanse, ReactJS i VueJS su gotov izjednačeni, dok Angular pomalo zaostaje za konkurencijom po pitanju memorije, no po pitanju brzine rada aplikacije je ispred svih trenutno postojećih radnih okvira.

6.3 Osobni dojam

Na temelju svega naučenoga prilikom razvoja i proučavanja smatram kako je najbolji izbor za razvoj ogledne aplikacije bio VueJS. Osim što je najjednostavniji, smatram kako je sintaksa vrlo jasna i intuitivna. Kod koji opisuje komponentu je jasno odvojen s obzirom na funkcionalnost te ga je moguće pisati u istoj datoteci ili radi preglednosti funkcionalno odvojiti. Angular mi se osobno svidio zbog TypeScripta koji pruža mogućnost provjere tipova, implementaciju razreda i upotrebu sučelja. Razvoj u Angularovom radnom okviru pruža najveću

automatizaciju zbog mogućnosti generiranja komponenti i usluga iz naredbenog retka, što mi je znatno smanjilo vrijeme uloženo u pisanje koda. Kada sam naučio glavne koncepte i započeo s procesom razvoja, shvatio sam zašto je dobro koristiti tako snažan radni okvir. Iako je proces učenja Angulara bio znatno dulji u usporedbi s ostalim okvirima, smatram kako se vrijeme uloženo u učenje višestruko isplatilo prilikom implementacije. Zahvaljujući automatiziranosti, potrošio sam mnogo manje vremena na pogreške u kodu u usporedbi s implementacijom aplikacije u ReactJS-u. React smatram najtežim, jer je potrebno brinuti o stanju komponente te mi se nije svidjelo što se ručno moraju osvježavati podaci i prilagođavati pogled kada korisnik okine događaje interakcijom sa komponentom. Smatram kako JSX-sintaksa koja semantički prikazuje kod HTML–a suvišno uvodi nove koncepte i detalje za učenje te se uporabom smanjuje funkcionalna kohezija. Koncept da se odvaja HTML-ov predložak od ostatka koda koju koriste okviri Angular i VueJS mi je osobno preglednija i jednostavnija za upotrebu. Za razliku od ReactJS-a, Angular i VueJS podržavaju dvosmjerno povezivanje i time smanjuju teret na programera što mi je omogućilo da se usredotočim na problematiku zahtjeva i sinkronizaciju komponenti.

Kao najdraži radni okvir bih se odlučio za VueJS koji me svojom jednostavnom sintaksom podsjeća na klasične skriptne jezike. Također, VueJS ima najmanje specifičnosti i novih koncepata. Dodatno mi se svidjelo što je dokumentacija pisana s naglaskom na primjere, a ne na objašnjenja kao što je u slučaju Angular-a. Prema statistikama je budućnost najviše okrenuta upravo VueJS-u i sve je veći broj aktivnih sustava koji su implementirani njegovom uporabom. Smatram da ukoliko bih nastavio raditi u Angularu, da bi mi postao mnogo draži. Zbog mnoštva novih koncepata potrebno je više prakse s radom okvira no što osobno imam kako bih se stekla brzina razvoja. Prilikom odabira radnog okvira sljedećeg projekta primarno bi na odluku utjecalo vrijeme isporuke i životni vijek projekta. Ukoliko bi bilo riječ o manjem projektu koji zahtjeva brzu implementaciju, odlučio bih se za VueJS, no ako bi projekt bio dugoga vijeka ili bi bile potrebne česte izmijene, odlučio bih se za Angular.

7. Zaključak

Kako se okviri temelje na arhitekturi zasnovanoj na komponentama, primjenom bilo kojega okvira naglasak je na ponovnoj upotrebi koda, uporabi već postojećega, koheziji i smanjenju složenosti funkcionalnih zahtjeva. Zahvaljujući otvorenom kodu radni okviri su vrlo fleksibilni. Implementirani sustav je prenosiv i neovisan o platformi na kojoj se razvija. Timovi proizvođača su vrlo pouzdane i uspješne zajednice koje redovito osvježavaju dokumentaciju i brinu se za kompatabilnost unazad, čineći prijelaz na novije verzije radnih okvira vrlo jednostavnim. Nakon implementirane ogledne aplikacije i svega naučenoga, slažem se s mnogobrojnim pročitanim člancima koji kazuju kako svaki radni okvir ima svoje prednosti te je teško odabrati najbolji. Prilikom odabira radnog okvira najvažniji su faktori složenost projekta, broj članova razvojnog tima, pitanje vremenskih i memorijskih performansi sustava, potencijalne nadogradnje i daljnje održavanje. Ako odaberemo bilo koji od proučenih okvira nećemo pogriješiti, jer se radi o tri trenutačno najučinkovitija i najpopularnija okvira koja zasigurno nude sve što je potrebno za implementaciju.

Glavni razlozi za odabir Angularovog radnog okvira su:

- složenost projekta
- mnogobrojni tim, rad s udaljenih mjesta
- važnost testiranja i provjere tipova – TypeScript
- velik broj gotovih komponenti koje implementiraju Material Design
- dugi životni vijek projekta

Glavni razlozi za odabir ReactJS razvojnog okvira su:

- potreba za velikom fleksibilnošću
- aplikaciju je moguće rastaviti na veliki broj manjih komponenti koje se ponovno upotrebljavaju
- razvojni tim preferira JavaScript nad TypeScriptom
- učestale potrebe za izmjenom oblikovanja

Glavni razlozi za odabir VueJS radnog okvira su:

- brzina razvoja

- neiskusan tim, najmanja potreba za učenjem
- zauzima malo memorije
- jednostavniji projekti
- pozitivna budućnost

Rječnik

1. JSON – *JavaScript Object Notation*, prikaz podataka pomoću JavaScriptove objektne notacije
2. SPA – *Single Page Application*, jednostranične aplikacije
3. CDN – *Content Delivery Network*, naziv za poslužitelja koji dostavlja zahtijevani resurs
4. node.js – Razvojna okolina pisana u JavaScriptu, služi za pisanje koda poslužiteljske strane
5. npm – *Node Package Manager*, poslužitelj paketa npm, koristi se za instalaciju vanjskih paketa
6. MVC – *Model, View, Controller*, arhitektura zasnovana na funkcionalnoj podjeli aplikacije, model predstavlja podatke, pogled je predstavljanje aplikacije korisniku, a upravljač obavlja logički dio i povezuje podatke s pogledom

LITERATURA

1. Facebook, React: A JavaScript library for building user interfaces , https://reactjs.org/, Pristupljeno: 14. svibnja 2019.
2. Ian Alen, *The Brutal Lifecycle of JavaScript Frameworks*, 11. siječnja 2018. https://stackoverflow.blog/2018/01/11/brutal-lifecycle-javascript-frameworks/, Pristupljeno: 3. svibnja 2019.
3. Google, Angular - One framework: Mobile & desktop, https://angular.io/, Pristupljeno: 14. svibnja 2019
4. Evan You, Vue.js The Progressive JavaScript Framework, https://vuejs.org/, Pristupljeno: 14. svibnja 2019
5. Spec India, *React vs Angular vs Vue.js: A Complete Comparison Guide*, 16. kolovoza 2018. https://medium.com/front-end-weekly/react-vs-angular-vs-vue-js-a-complete-comparison-guide-d16faa185d61/, Pristupljeno: 9.svibnja 2019.
6. Ankit Kumar, *React vs. Angular vs. Vue.js: A Complete Comparison Guide*, 20. kolovoza 2018. https://dzone.com/articles/react-vs-angular-vs-vuejs-a-complete-comparison-gu, Pristupljeno: 9. svibnja 2019.
7. Mosh Hamedani, *React vs. Angular: The Complete Comparison*, 5. studenoga 2018. https://programmingwithmosh.com/react/react-vs-angular/, Pristupljeno: 6. svibnja 2019.
8. Tutorialspoint, *ReactJS tutorial* https://www.tutorialspoint.com/reactjs/, Pristupljeno: 7. ožujka 2019.
9. Tutorialspoint, *VueJS tutorial* https://www.tutorialspoint.com/vuejs/, Pristupljeno: 8. ožujka 2019.
10. Dayana Jabif, *Angular Tutorial: Learn Angular from scratch step by step*, 20. travnja 2019. https://angular-templates.io/tutorials/about/learn-angular-from-scratch-step-by-step, Pristupljeno: 9. travnja 2019.
11. Code academy, *Learn ReactJS*, https://www.codecademy.com/learn/react-101, Pristupljeno: 15. travnja 2019.
12. Vegibit, *Vue.js Tutorial*, https://vegibit.com/vue-js-tutorial/, Pristupljeno: 3. travnja 2019.

Usporedba radnih okvira klijentske strane u razvoju web aplikacija

Sažetak

Cilj ovog završnog rada je proučiti nekoliko trenutno popularnih tehnologija za razvoj klijentske strane web aplikacija. Potrebno je razviti web aplikaciju s jednostavnim funkcionalnostima koje su zajedničke većini programskih sustava, poput registracije, prijave i validacije korisnika te stvaranja, dohvaćanja, uređivanja i brisanja entiteta baze podataka. Implementirati grafičko korisničko sučelje koje je korisniku prigodno i jednostavno za upotrebu. Primjenom biblioteka za oblikovanje, cilj je postići responzivno ponašanje aplikacije prilagodljivo raznim dimenzijama uređaja. Nakon implementacije, potrebno je kvalitativno i kvantitativno usporediti tehnologije. Shvatiti nedostatke i mogućnosti koje pojedina tehnologija pruža te steći iskustvo prilikom odabira tehnologija za rješavanje raznih problema.

Ključne riječi: radni okvir, Angular, ReactJS, VueJS, web komponenta, virtualni dokument, ponovna uporaba, automatiziranost, HTML, sinkronizacija.

Comparison of Frontend Frameworks for Web Application Development

Abstract

The aim of the bachelor thesis is to perform a research about currently most popular technologies used for client side web development. The thesis requires an implementation of common functionalities that are needed for most of web applications, such as: user registration, signing in, validation of user credentials, creating, deleting, fetching and updating entities from a database. Graphical user interface has to be appropriate and user-friendly. Using libraries for design application has to be responsive and adjustable to multiple screen sizes. The thesis also requires qualitatively and quantitative comparation of the technologies. The aim is to realize disadvantages and capabilities of each technology and also to become more experienced in decision making when it comes to choosing technology for solving problems.

Keywords: framework, Angular, ReactJS, VueJS**,** web component, virtual document, reuse, automatization, HTML, synchronization.