JavaScript programski okviri za korisnički dio Web aplikacije

Pernjek, Davor

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Davor Pernjek

JavaScript programski okviri za korisnički dio Web aplikacije

ZAVRŠNI RAD

Varaždin, 2021.

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Davor Pernjek

Matični broj: 0016134044

Studij: Primjena informacijske tehnologije u poslovanju

JavaScript programski okviri za korisnički dio Web aplikacije

ZAVRŠNI RAD

Mentor:

Matija Kaniški, mag. inf.

Varaždin, rujan 2021.

Davor Pernjek

Izjava o izvornosti

Izjavljujem da je moj završni/diplomski rad izvorni rezultat mojeg rada te da se u izradi istoga nisam koristio drugim izvorima osim onima koji su u njemu navedeni. Za izradu rada su korištene etički prikladne i prihvatljive metode i tehnike rada.

Autor/Autorica potvrdio/potvrdila prihvaćanjem odredbi u sustavu FOI-radovi

Sažetak

U radu će se objasniti MVC uzorak i bit će prikazan na primjeru. Opisat će se TypeScript i njegova kompatibilnost sa starijim ECMAScript standardima. Nakon toga će se analizirati razlike između poznatih JavaScript okvira. Nadalje, opisat će se načini povezivanja podataka. Također, spomenut će se vrste dokumentno objektnog modela. Objasnit će se kada odabrati koji programski okvir. Detaljnije će se opisati React i kreiranje okruženja za razvoj. Pokazat će se kako koristiti HTML oznake u JavaScript-u sa JSX-om. Opisati će se podjela sučelja na komponente i kreiranje komponenti. Opisat će se kako odrediti vrstu modela podataka (proizvoljni ulazi ili metode stanja) i metode životnog ciklusa. Objasnit će se načini upravljanja događajima, dinamično kreiranje komponenti i kreiranje obrazaca. Prikazat će se testiranje programskog koda u alatima Jest i Jasmine. U praktičnom dijelu izradit će se web aplikacija uz primjenu programskog okvira React.

Ključne riječi: JavaScript, Vue, React, Angular, MongoDB, ExpressJS, programski okviri, korisničko sučelje

Sadržaj

1. Uvod

Tijekom posljednjih godina popularnost web aplikacija postaje sve veća i postupno zamjenjuju stolne (eng. *Desktop*) aplikacije. Web aplikacijama se može pristupiti s gotov bilo kojeg računala koje ima pristup internetu i nije potrebna instalacija zasebnih programa kao što je to u slučaju stolnih aplikacija. Instaliranjem posebnih programa se javlja drugi problem. Sistemski zahtjevi zbog kojih na nekim računalima neće raditi, što nije slučaj kod web aplikacija, to je jedan od razloga za rastuću popularnost web aplikacija.

Sve veća popularnost web aplikacija rezultira i većom potražnjom za programerima web aplikacija koje se dijele na stranu klijenta (eng. *Front-End*) ili programiranje na korisničkom djelu sučelja i stranu poslužitelja (eng. *Back-End*) ili programiranje na strani poslužitelja (eng. *Server*). Programeri na strani korisnika izrađuju dio web aplikacija koji korisnik može vidjeti i s kojim može upravljati. Za razvoj korisničkog dijela se koristi HTML (*HyperText Markup Language*) kojim se izrađuje predložak web aplikacije. CSS (*Cascading Style Sheets*) se koristi za kreiranje stilova i JavaScript za programiranje logike web aplikacije. S druge strane programeri na strani poslužitelja kao i što samo ime govori programiraju web aplikaciju na poslužitelju (eng. *Server*) i najčešće koriste bazu podataka.

Potreba za velikim brojem aplikacija rezultira kreiranjem lakših i jednostavnijih načina za izradu tih aplikacija. Izrađuju se programski okviri koji programerima pružaju „alate“ za razvoj web aplikacija čime se smanjuje vrijeme potrebno za njihov razvoj i koji povećavaju njihovu produktivnost. Danas postoji mnogo programskih okvira te je danas pitanje koje si programeri postavljaju „koji programski okvir odabrati?“ jer svaki od njih ima svoje prednosti i nedostatke. U ovom radu će se usporediti najpopularniji JavaScript programski okviri za korisnički dio sučelja prema sljedećim svojstvima: vrsta dokumentno objektnog modela koji koriste, načinima provjere podataka, kreiranju komponenti i predložaka, način na koji povezuju podatke, performansama, mogućnostima testiranja koda i popularnosti korištenja. U praktičnom dijelu rada će se razviti aplikacija „Kviz“. Za korisnički dio sučelja će se koristiti programski okvir React. Responzivnost sučelja će biti postignuto korištenjem CSS-a i dodatka Flexbox-a. Baza podataka će biti kreirana u MongoDB nerelacijskoj bazi podataka (eng. *NoSql*). ExpressJS okvir će biti korišten za programiranje na strani poslužitelja. Sve će se izvršavati na NodeJS-u. Dijelovi razvijene aplikacije će biti prikazani slikama zaslona i opisima. Na kraju rada će biti donesen zaključak na temelju usporedbi programskih okvira i praktičnog dijela rada.

2. Metode i tehnike rada

Prilikom pisanja teorijskog dijela teorijskog dijela korištene su sljedeće metode: analizu literature, sinteza literature i usporedba literature. Kao literatura su korištene knjige, članci i dokumentacije. Objavljeni radovi su vjerodostojni i prošli su postupak recenzije. Kod referenciranja korišten je Mendeley. Za izradu grafova je korištena web aplikacija draw.io.

Za izradu web aplikacije su korišteni:

- NodeJS poslužiteljsko okruženje otvorenog koda koje omogućava izvođenje JavaScripta na strani poslužitelja.
- NPM (eng. *Node Package Manager*) putem kojeg su preuzeti paketi potrebni za kreiranje predloška React aplikacije.
- HTML (eng. *HyperText Markup Language*) za pisanje strukture sadržaja web stranice.
- CSS (eng. *Cascading Style Sheets*) za definiranje stilova. Uz CSS, u manjem dijelu korišten je i Flexbox dodatak koji olakšava izradu responzivnih web stranica.
- React programski okvir za razvoj korisničkog sučelja web aplikacije.
- MongoDB nerelacijska baza podataka za pohranu podataka.
- ExpressJS programski okvir za komunikaciju između nerelacijske baze i korisničkog dijela aplikacije.
- Insomnia za testiranje aplikacijskog programskog sučelja.
- Visual Studio Code kao razvojno okruženje za pisanje koda.
- GitHub sustav verzioniranja za kreiranje repozitorija i pohranu aplikacije.

Programski kod aplikacije je podijeljen na dva dijela, korisnički dio i dio na strani poslužitelja. Svaki dio ima kreiran svoj repozitorij na GitHub-u:

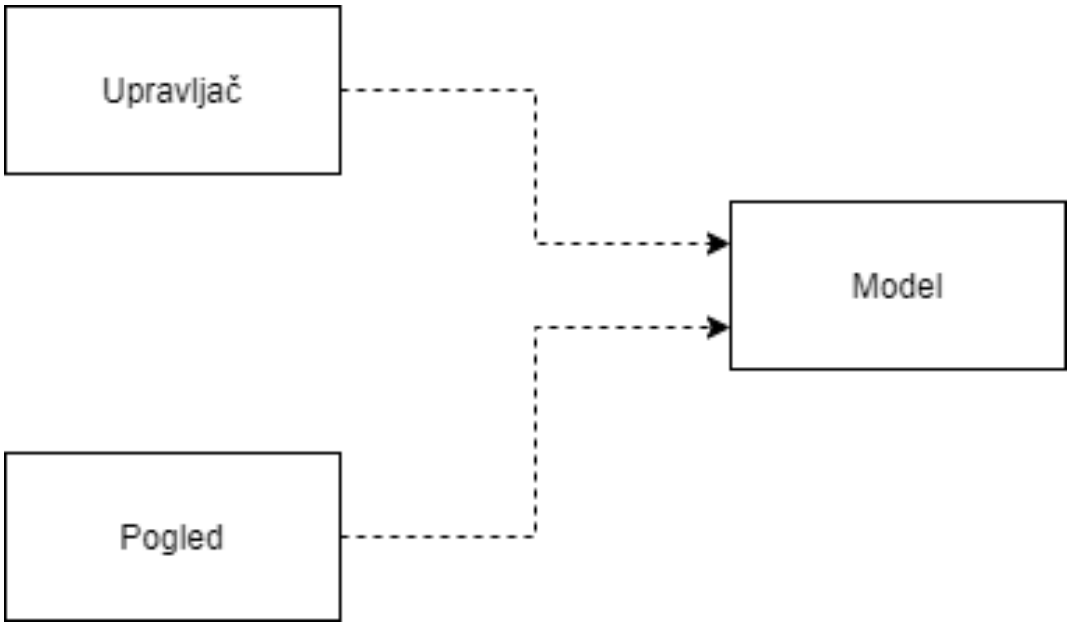
- Strana poslužitelja: https://github.com/PDavor/zavrsni_server
- Strana klijenta: https://github.com/PDavor/zavrsni_frontend

Repozitoriji se mogu preuzeti na računalo klikom na gumb „Clone“ i zatim „Download ZIP“. Ako korisnik ima instaliran GIT na računalo moguće je klonirati repozitorij uz pomoć komande „git clone [poveznica]“.

3. MVC uzorak

MVC (eng. *Model View Controller*) je jedan od najčešćih uzoraka dizajna koji se koristi za razvoj aplikacija. Glavna ideja MVC-a je odvajanje prezentacije, to jest, stvaranje jasne razlike između objekata domene koji modeliraju našu percepciju stvarnog svijeta i objekata kojima je svrha prezentacija elemenata grafičkog sučelja na zaslonu. Objekti domene trebaju biti potpuno samostalni i moraju moći izvršavati svoju zadaću bez referenciranja na prezentacijski dio. Trebaju podržavati više prezentacija, ponekad i istovremeno, to omogućuje korištenje aplikacija kroz različita sučelja (grafičko korisničko sučelje i naredbeni redak). U MVC-u se elementi domene referenciraju kao modeli, a modeli objekata su nevezani o korisničkom sučelju. Prezentacijski dio MVC-a sastoji od dva preostala dijela: pogleda (eng. *View*) i upravljača (eng. *Controller*). Jedno korisničko sučelje se sastoji od mnogo parova pogleda i upravljača. To jest, svaki element na zaslonu čini jedan par kao i što je zaslon također par pogleda i upravljača ("*GUI Architectures"*, bez dat.).

MVC radi na način da upravljač i pogled ne komuniciraju direktno (slika 1), nego putem modela. Upravljač ne upravlja direktno dijelom zaduženim za pogled, nego upravlja modelom. Dio za pogled vidi tu promjenu i sukladno tome reagira.



Slika 1: MVC (Izvor: Fowler, 2010)

MVC uzorak ima više prednosti, uključujući:

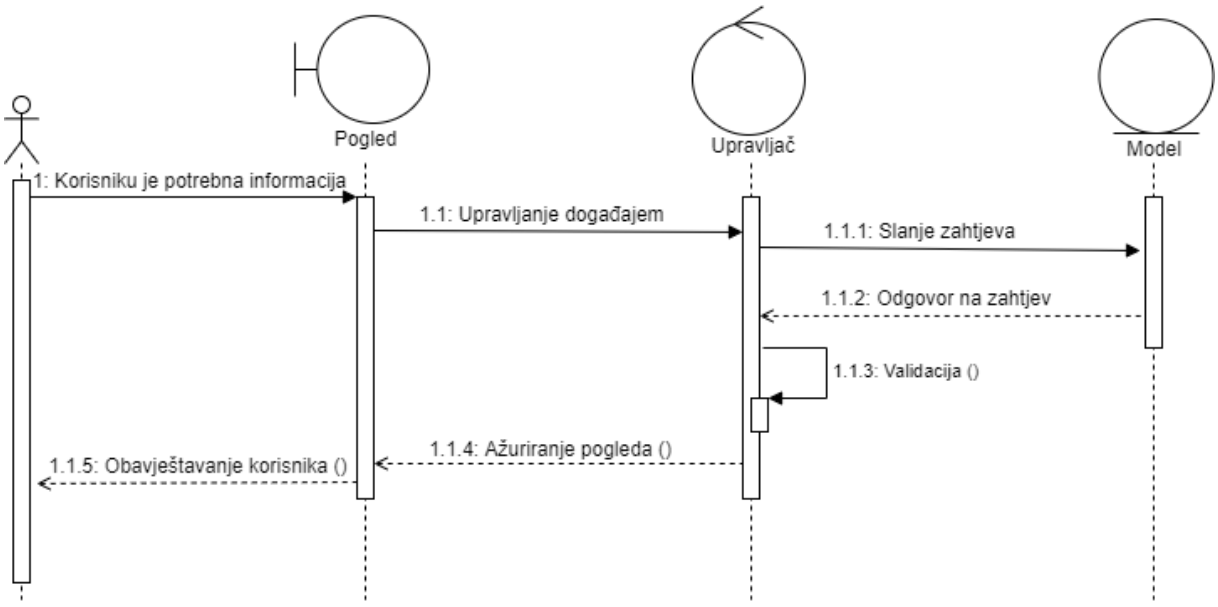
- Brži razvoj - MVC uzorak razdvaja različite komponente aplikacije što omogućuje programerima paralelan rad na različitim komponentama.
- Ponovna iskoristivost - isti (ili sličan) prikaz za jednu aplikaciju može se iskoristiti za drugu aplikaciju s različitim podacima jer pogled upravlja načinom na koji se podaci prikazuju korisniku.
- Poboljšana skalabilnost - u slučaju da aplikacija ima problema s performansama (npr. pristup bazi podataka je spor) može se nadograditi sklopovlje na kojem se nalazi baza podataka bez bilo kakvog utjecaja na druge komponente
- Slabo povezivanje - po dizajnu MVC uzorak ima slabo povezivanje između modela, pogleda ili upravljača.
- Veća proširivost - komponente imaju slabu ovisnost jedna o drugoj. Promjena jedne komponente (radi ispravka greške ili promjene funkcionalnosti) ne utječe na drugu komponentu.

U nastavku će biti opisano kako radi MVC na primjeru kupnje automobila:

1. Korisnik/kupac ulazi u dućan/web dućan (model) i tu ga dočekuje prodavač koji predstavlja upravljač. U web dućanu početna stranica koja predstavlja dio pogleda.
2. Prodavač može na primjer odgovoriti na neka pitanja („Imate li ovaj automobil?“) što bi predstavljalo stranicu s automobilima na web dućanu koja predstavlja dio pogleda.
3. Kupac može prodavača pitati za više informacija o svakom automobilu, te mu prodavač (upravljač) može pružiti podatke iz dućana (modela). Na web dućanu ovaj korak predstavlja svaku pojedinačnu stranicu o automobilima (pogled).
4. Ako se kupac odluči kupiti automobil, prodavač (upravljač) opet uzima podatke iz dućana (model), na primjer cijenu. U web dućanu bi ovo predstavljalo stranicu s „košaricom“.

Kako bi se MVC bolje objasnio izrađen je i dijagram slijeda (slika 2). MVC dijagram slijeda ima objekte sučelja, objekte upravljača i objekte entiteta. Dijagram prikazuje slijed izvršavanja koraka potrebnih za dohvaćanje podataka.

Dijagram slijeda opisuje interakciju između skupa objekata koji su sudjelovali u nekoj radnji. Objekti su poredani kronološkim redoslijedom. Dijagram prikazuje objekte koji sudjeluju u interakciji po svojim "životnim linijama" i poruke koje međusobno šalju (Sequence Diagram Tutorial, bez dat.).



Slika 2: Dijagram slijeda MVC-a (Izvor: How to Model MVC Framework with UML Sequence Diagram?, bez dat.)

Na slici 2 je prikazan slijed događaja u MVC-u. Na primjer korisnik želi vidjeti automobil na web dućanu. Korisnik vidi poveznicu na automobil (pogled). Klikom na poveznicu u pogledu se aktivira upravljač koji šalje zahtjev prema modelu. Model zatim šalje podatke tražene iz zahtjeva prema upravljaču. Upravljač provjerava podatke od modela i zatim ažurira pogled sa podacima koje je primio od modela. Korisnik sad može vidjeti podatke o željenom automobilu, a dio pogleda može i obavijestiti korisnika da su podaci dohvaćeni.

4. TypeScript

TypeScript je proširenje JavaScript-a koje omogućava lakši razvoj velikih JavaScript aplikacija. Svaki JavaScript program je i TypeScript program. TypeScript nudi sustav modula, klase, sučelja i bogatu podršku za pristup tipovima. TypeScript se može naučiti bez prethodnog znanja JavaScript-a. Podržana je JavaScript sintaksa bez većih promjena (Bierman i ostali, 2014).

TypeScript osim osnovnih tipova podataka u JavaScript-u poput: „brojeva“, „stringova“, „boolean tipova“ i „listi“. TypeScript u odnosu na JavaScript ima definirano više tipova podatka koji se koriste u drugim programskim jezicima. Neki od dodanih tipova su „never“ koji predstavlja tipove vrijednosti koje se nikad ne pojavljuju. Na primjer „arrow“ funkcija koja nikad ništa ne vraća. Dodan je i tip „void“ koji predstavlja odsustvo bilo kakvog tipa podatka. Ključna riječ „any“ se koristi kad je potrebno opisati tip podatka koji varijabla sadrži, ali još nije poznat. TypeScript je obavezan za pisanje koda u Angular-u, ali se također može koristiti u React-u i Vue-u ("TypeScript in 5 minutes · TypeScript", bez dat.).

Na kodu ispod je prikazan primjer provjeravanja tipa podatka korištenjem TypeScript sintakse.

```
function izvrsi(): void {
    console.log("Ovdje se izvršava kod");
}
```

U primjeru koda je funkcija „izvrsi“ kojoj je zadano TypeScript-om da ne smije vraćati vrijednost. Ako vraća vrijednost, TypeScript će prikazati grešku.

TypeScript kompajler ima mehanizme poput prevođenja koda (pretvaranje TypeScript-a u JavaScript) i brisanje tipova (brisanje statične anotacije tipova podataka) kako bi se generirao čisti JavaScript kod. Brisanje tipova ne briše samo anotaciju tipova, već i posebne TypeScript osobine poput sučelja. Zbog toga je kod kompatibilan sa većinom preglednika jer se prilagođava ECMAScript standardima. TypeScript primjenjuje ECMAScript 3 specifikacije. Također su podržane i ECMAScript 5 i ECMAScript 6. Zbog toga se TypeScript osobine mogu koristiti za kompiliranje kod bilo kojih dostupnih standarda, ali za neke mogućnosti je potrebno korištenje ECMAScript 5 ili 6 kao standard. TypeScript nije samo kompatibilan s trenutnim načinom pisanja JavaScript koda, već će i vrlo vjerojatno biti kompatibilan s novijim verzijama JavaScript-a. Većina mogućnosti TypeScript-a se bazira na prijedlozima za buduće ECMAScript standarde pa bi TypeScript datoteke u budućnosti mogle postati valjanje JavaScript datoteke (*Remo H. Jansen*, bez dat.).

TypeScript razumije JavaScript programski jezik pa često programer ne treba definirati tipove. Na primjer, kod definiranja varijable s vrijednošću TypeScript će varijabli dodijeliti tip na temelju te vrijednosti. Primjer koda ispod prikazuje varijablu i kako taj kod izgleda kad TypeScript dodijeli tip toj varijabli (*TypeScript in 5 minutes · TypeScript*, bez dat.).

```
let pozdrav = "Pozdrav svijete";
let pozdrav: string;
```

Prva linija koda prikazuje varijablu „pozdrav“ koja ima vrijednost „Pozdrav svijete“. Iz druge linije koda se može vidjeti da TypeScript varijabli dodjeljuje tip „string“ prema vrijednosti koja je definirana.

TypeScript omogućava automatsko definiranje tipova za jednostavnije konstrukte. Kod složenijih konstrukata (npr. objekt) je svakako potrebno „ručno“ definirati tipove. To se postiže korištenjem ključne riječi „interface“. Na primjeru koda ispod je prikaz opisivanja oblika objekta pomoću ključne riječi „interface“.

```
interface Korisnik {
  ime: string;
  id: number;
}
```

Kod prikazuje „sučelje“ naziva „Korisnik“ koji predstavlja objekt koji mora sadržavati dva svojstva. Svojstvo „ime“ koje mora imati vrijednost „string“. Svojstvo „id“ koje mora imati vrijednost „number“.

Nakon što je „sučelje“ kreirano može se definirati da objekt mora imati „oblik“ kako je zadano u tom „sučelju“. Primjer kreiranja novog objekta pomoću „sučelja“ se može vidjeti u kodu ispod.

```
 const korisnik: Korisnik = {
  ime: "Davor",
  id: 0,
};
```

Kod prikazuje kreiranje objekta „korisnik“ koji mora imati „oblik“ kao „sučelje Korisnik“. Objekt ima varijablu „ime“ s vrijednošću „string“ i varijablu „id“ s vrijednošću „number“. Ako se kreira objekt koji ima drugačiji naziv svojstva (npr. „prezime“ umjesto „ime“), TypeScript će javiti grešku da objekt može sadržavati samo svojstva definirana u „sučelju“.

5. JavaScript programski okviri

U ovom poglavlju će se uspoređivati JavaScript programski okviri za razvoj korisničkog dijela sučelja. Okviri koji se uspoređuju su: Angular, React i Vue. Usporedit će se njihova popularnost s različitih izvora. Objasnit će se što su komponente u tim okvirima i koje predloške koriste pojedini okviri. Također će biti opisano kako pojedini okviri upravljaju stanjima aplikacija i na koji način povezuju podatke. Usporedit će se performanse te će se prikazati kakvu ulogu u tome imaju različite vrste modela objekata dokumenta. Još će se spomenuti i TypeScript, Flow i ECMAScript te koji okviri ih koriste. Na kraju će se prikazati korisni alati za testiranje koda u različitim okvirima.

Angular omogućuje pisanje web aplikacija na strani klijenta. Koristi HTML kao jezik predloška i podržava proširivanje sintakse HTML-a za jasno izražavanje komponenti aplikacije (eng. *Application components*). Automatski sinkronizira podatke s korisničkog sučelja (pogleda) i JavaScript objektima (model) putem dvosmjernog povezivanja podataka (eng. *Two-Way Data Binding*). Angular pomaže kod strukturiranja aplikacija i olakšava testiranje. Angular „uči“ preglednik kako raditi ubrizgavanje ovisnosti i inverziju kontrole ("angular/angular: One framework. Mobile & desktop.", bez dat.).

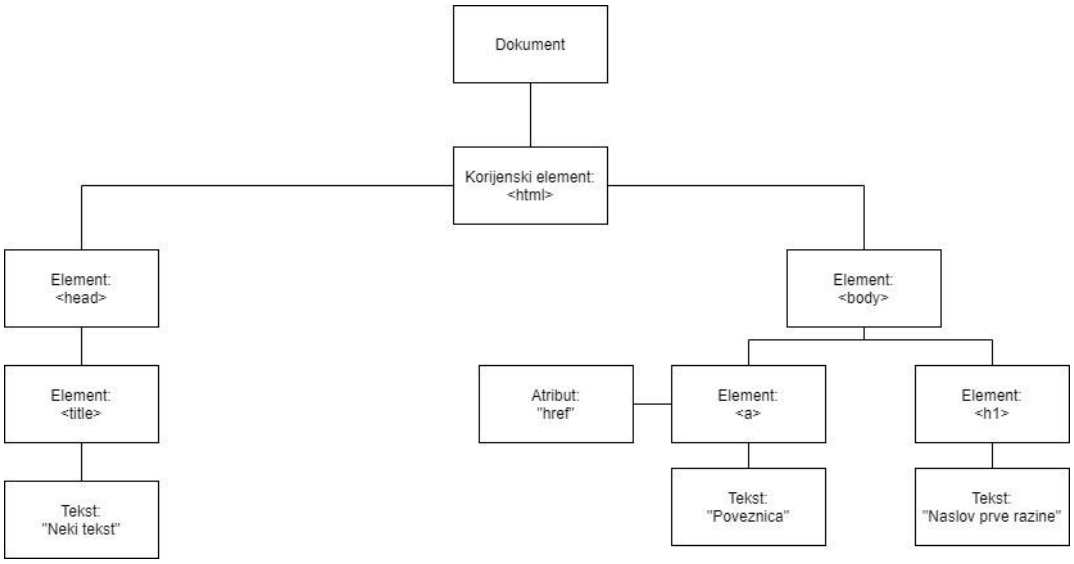
React je JavaScript biblioteka za razvoj korisničkih sučelja. Omogućava jednostavno stvaranje interaktivnih korisničkih sučelja koja ih bez ostatka sučelja prilikom promjene ažuriraju. Koristi deklarativni način programiranje koji čini programski kod predvidljivijim, jednostavnijim za razumijevanje i lakšim za uklanjanje pogrešaka. Baziran je na temelju komponenti koje mogu biti zatvorenog tipa i koje upravljaju vlastitim stanjem i zatim se mogu komponirati kako bi se napravila složena korisnička sučelja. Logika komponenata napisana je u JavaScript-u umjesto predloška. Zbog tog razloga se lako mogu proslijediti bogati podaci (eng. *Rich Data*) kroz aplikaciju i držati se izvan DOM-a ("facebook/react: A declarative, efficient, and flexible JavaScript library for building user interfaces.", bez dat.).

Vue je dinamički JavaScript programski okvir za razvoj korisničkih sučelja. Dizajniran je na način kako bi se postupno mogao usvojiti i jednostavnije razmjenjivati između biblioteke (eng. *Library*) i okvira ovisno o različitim slučajevima uporabe. Sastoji se od pristupačne jezgrene biblioteke (eng. *Core Library*) koja se fokusira samo na sloj prikaza i pomoćnih biblioteka koje pomažu u rješavanju kompleksnosti u velikim SPA (*Single Page Applications*) ("vuejs/vue: Vue.js is a progressive, incrementally-adoptable JavaScript framework for building UI on the web.", bez dat.).

5.1. Vrste dokumentno objektnih modela

Prvo je potrebno definirati što su zapravo vrste dokumentno objektnih modela (eng. *Document Object Model*). Nakon objašnjenja DOM-a, bit će objašnjene različite vrste DOM-a te njihove prednosti i nedostaci.

Dokumentno objektni model (eng. *DOM - Document Object Model*) je sučelje za programiranje aplikacija (eng. *API - Application Programming Interface*) za važeće HTML i dobro oblikovane XML (eng. *Extensible Markup Language*) dokumente (slika 3). Definira logičku strukturu dokumenata i način na koji se dokumentu pristupa i manipulira. U DOM specifikaciji, pojam „dokument" upotrebljava se u širem smislu. XML se sve više koristi kao način predstavljanja različitih informacija koje mogu biti pohranjene u raznim sustavima, a velik dio toga tradicionalno bi se promatralo kao podaci, a ne kao dokumenti. Ipak, XML prikazuje te podatke kao dokumente, a DOM se može koristiti za upravljanje tim podacima. Pomoću dokumentno objektnog modela, programeri mogu izrađivati dokumente, dodavati, mijenjati ili brisati elemente i sadržaj. Sve što se nalazi u HTML ili XML dokumentu može se pristupiti, izmijenili, izbrisati ili dodati pomoću dokumentno objektnog modela (Robie i ostali, bez dat.).



Slika 3: Dokumentno objektni model

Na slici 3 je prikazano stablo dokumentno objektnog modela, na najnižoj (nultoj) razini se nalazi sam dokument. U slučaju web aplikacija je to HTML stranica. Ispod dokumenta se na prvoj razini nalazi korijenski element (eng. *Root element*) koji je kod HTML stranica određen oznakom `<html>`. Taj korijenski element sadrži sve ostale elemente na toj web stranici. Na drugoj razini mogu se vidjeti dva elementa od kojih se taj dokument sastoji (`<head>` i `<body>`).

Ti elementi dalje mogu imati svoje pod elemente. Kao što tijelo dokumenta ima recimo poveznicu ili naslov. Osim samih elemenata, DOM prikazuje i atribute i unutarnji HTML tih elemenata. Na slici 3 se može vidjeti da element za poveznicu `<a>` ima pripadajući atribut (`href`) i unutarnji HTML, to jest tekst „Poveznica“. Ova vrsta DOM-a u današnje vrijeme, s rastućom popularnošću aplikacija na jednoj stranici (eng. *SPA – Single Page Application*), nije više toliko poželjna. Razlog je jer se kod aplikacija na jednoj stranici DOM često modificira. DOM sam po sebi nije brz zbog čega programski okviri koji ga koriste gube na performansama, to jest brzini učitavanja promjena. Da bi se napravila promjena u DOM-u prvo je potrebno pronaći svaki dio DOM-a koji je povezan s događajem koji ga mijenja. Zatim je svaki taj dio potrebno ažurirati, ali samo ako je potrebno. Dokumentno objektni model ima dva glavna nedostatka koja su ispravljena korištenjem virtualnog modela (eng. *Virtual DOM*) ("The difference between Virtual DOM and DOM - React Kung Fu", bez dat.):

1. Teško je upravljati upravljačima događaja (eng. *Event Handlers*) jer ako se izgubi kontekst, potrebno je ići kroz cijeli HTML kako bi se pronašli elementi koji se ažuriraju.
2. Ručno pretraživanje dokumenta nije efikasno.

Virtualni model objekta dokumenta je koncept programiranja kod kojeg se virtualna reprezentacija dokumenta sprema u memoriju i po potrebi se sinkronizira s „pravim“ DOM-om. U React-u je za sinkroniziranje zadužena biblioteka „ReactDOM“. Ona i omogućuje deklarativno programiranje u React-u tako što programer može „reći“ u kojem stanju želi da korisničko sučelje bude. Time se pojednostavljuje programiranje jer se lakše manipulira atributima, upravljanje događajima i ažuriranje DOM-a.

Angular koristi klasičan DOM za ažuriranje stanja korisničkog sučelja. To se u Angular-u postiže „ciklusom pregleda“ (eng. *Digest Cycle*). Angular aktivira ciklus pregleda kada se neka vrijednost u modelu ili pogledu promijeni. Zatim ciklus postavlja promatrače (eng. *Watchers*) koji tada izjednačuju vrijednosti iz modela i pogleda na najnoviju vrijednost. Ciklus se pokreće automatski ako postoji potreba za to i on može samo djelovati unutar konteksta Angular-a. Ciklus se pokreće između 2 i 10 puta nakon što ga Angular aktivira ("What is Digest Cycle in AngularJS & How Does It Work?", bez dat.).

Vue je također u početku (verzija 1.0) koristio „čisti“ DOM. Zbog optimizacije performansi u verziji 2.0 počeo se je koristiti virtualni DOM kao što ga koristi i React, ali uz nekoliko promjena koje ga čine optimiziranijim od React-a. U React-u, kad se stanje komponente promijeni, pokreće se ponovno prikazivanje cijelog podstabla komponente, počevši od te komponente kao korijena (eng. *Root*).

Da bi se izbjeglo nepotrebno vraćanje podređenih komponenti, potrebno je koristiti čiste komponente (eng. *Pure components*) ili implementirati „shouldComponentUpdate“ metodu. Ponekad je potrebno i upotrebljavati i nepromjenjive strukture podataka (eng. *Immutable data structures*) kako bi stanje korisničkog sučelja bilo prilagođenije optimizaciji. Međutim, ponekad se čiste komponente i „shouldComponentUpdate“ metode ne mogu koristiti jer ove metode pretpostavljaju da je ishod ukupnog podstabla određen svojstvima (eng. *Properties*) trenutne komponente. Ako to nije slučaj, takve optimizacije mogu dovesti do nedosljednog stanja DOM-a. U Vue-u se ovisnosti o komponenti automatski prate tijekom njenog prikazivanja. Tako da sustav točno zna koje komponente zapravo trebaju ponovo prikazati kad se stanje promijeni. Za svaku komponentu se može smatrati da ima automatski implementirano „shouldComponentUpdate“ metodu. To uklanja potrebu za čitavom klasom optimizacije performansi od strane programera i pogotovo kod skaliranja se omogućuje bolji fokus na razvoj same aplikacije ("Comparison with Other Frameworks — Vue.js", bez dat.).

5.2. Provjera tipa podataka

JavaScript je programski jezik koji ima slabo povezivanje tipova, što znači da prilikom deklariranja varijable nije potrebno definirati tip podatka koji ta varijabla sadrži. To olakšava programiranje jer nije potrebno definirati tipove podataka u varijabli, nego se oni postave automatski. Na primjer prilikom deklariranja varijable `var n = 0`, n će biti jednak tipu „number“, a `var m = „3“` će biti jednak tipu „string“. Međutim, slaba povezanost nosi sa sobom i posljedice. Prilikom izvođenja programskog koda mogu se javiti greške ili se mogu dobiti pogrešni rezultati ukoliko podaci nisu tipa kojeg je programer zamislio. Zato je u JavaScript uvedeno provjeravanje tipova podataka kako bi se izbjegle takve greške. U nastavku će biti opisano provjeravanje tipova korištenjem JavaScript-a, TypeScript-a i Flow-a, te će biti navedeno koji od alata se preporučuju za Angular, React i Vue.

Postoje dvije vrste provjere tipa podataka, statička i dinamička. Statička provjera se odvija prije izvršavanja koda pa se zbog toga greške mogu primijetiti u ranim fazama prilikom razvoja aplikacija. Dinamička provjera se odvija za vrijeme izvođenja programa. Detaljnije će se obrađivati alati za statičke provjere poput Flow-a i Typescript-a.

Prvo je važno napomenuti da se sa samim JavaScript-om također mogu provjeravati tipovi podataka. Ovakva metoda se može koristiti i kod različitih programskih okvira. U kodu ispod je prikazan primjer korištenja „typeof“ operatora pomoću kojeg se mogu provjeravati tipovi podataka.

```
if (typeof broj == 'number') {
     console.log('Varijabla je tipa broj');
} else {
     console.log('Varijabla nije tipa broj');
}
```

Ovakva sintaksa pomoću „if“ uvjeta provjerava varijablu broj i ako je uvjet zadovoljen, može se nastaviti s izvođenjem. Ukoliko nije onda se može u konzolu preglednika npr. ispisati greška.

React za razliku od Angular-a u svojoj dokumentaciji navodi nekoliko popularnih alata za provjeravanje tipova podataka. Jedan od njih je „PropTypes“ (paket koji se instalira pomoću NPM-a). U kodu ispod je prikazan primjer koda koji provjerava je li svojstvo „ime“ komponente „Pozdrav“ tipa podatka „string“.

```
import PropTypes from 'prop-types';
class Pozdrav extends React.Component {
  render() {
    return (
      <h1>Pozdrav, {this.props.ime}</h1>
    );
  }
}

Pozdrav.propTypes = {
  ime: PropTypes.string
};
```

Osim „PropTypes“, za provjeravanje tipova podataka u dokumentaciji su još navedeni: TypeScript, Flow, Reason i Kotlin. Preporučuje se korištenje „PropTypes“ na manjim projektima, dok se već prije spomenuti TypeScript i Flow preporučuju u radu na većim projektima. Flow je statička provjera tipa JavaScript koda, razvijen je na Facebook-u i često se koristi s React-om. Pošto su već prije prikazani primjeri koda provjere tipova podataka u TypeScript-u i Flow-u, ovdje će se samo prikazati kod iz Flow-a (*Static Type Checking – React*, bez dat.).

```
function kvadrat(n: number): number {
  return n * n;
}
```

Kod iznad prikazuje provjeru tipa podataka koju vraća funkcija „kvadrat“ i provjeru tipa podataka varijable „n“. Može se iz koda uočiti da je sintaksa gotovo identična onoj u TypeScript-u. Jedina razlika je što se ovdje provjerava i varijabla koja se kasnije koristi u funkciji.

Vue osim samog JavaScript-a, Flow-a i TypeScript-a, također ima i ugrađenu provjeru tipova podataka pod nazivom „PropTypes“. Provjera „PropTypes“ se koristi i u React-u. Razlika od React-a je u tome što u Vue-u nije potrebno instalirati dodatne pakete, već je sve dostupno u samom Vue-u ("Props — Vue.js", bez dat.).

Sintaksa se može vidjeti u primjeru koda koji se nalazi ispod:

```
props: {
  naslov: String,
  pregledi: Number,
  objavljeno: Boolean,
  IDkomentara: Array,
  autor: Object
}
```

React i Vue su kao programski okviri za korisnički dio sučelja fleksibilni kod provjere tipova podataka zbog više različitih alata koji se mogu koristiti. Za razliku od Angular-a koji ograničen strogim korištenjem TypeScript-a.

5.3. Komponente

Komponente su važan dio svakog od okvira koji se obrađuju u ovom radu. Komponente omogućuju odvajanje korisničkog sučelja u neovisne dijelove koji se mogu koristiti više puta. Komponente su zapravo JavaScript funkcije koje u ovom slučaju predstavljaju dio „pogleda“ u MVC-u. Također mogu primati vrijednosti kao i funkcije. Na primjer, proizvoljni ulazi (eng. *Props*) u React-u. Kao rezultat vraćaju jedan „izolirani“ dio korisničkog sučelja.

Komponente u Vue i React se razlikuju od onih u Angular-u po načinu pisanja. U Angular-u se komponente sastoje od dvije datoteke. Jedna datoteka sadrži dio koji se odnosi na korisničko sučelje, dok se u drugoj datoteci nalazi logika potrebna za tu komponentu. U Vue i React-u se komponente pišu samo u jednoj datoteci. Jedna funkcija uključuje i dio za korisničko sučelje i dio logike za to sučelje. React koristi „render“ metodu kod svake komponente koja služi za generiranje DOM-a. Nakon što se pozove „render“ metoda na korijenskoj komponenti, ona se rekurzivno poziva nad djecom korijenske komponente i tako se na temelju virtualnog DOM-a (komponenti) kreira stvarni DOM (Aggarwal, 2018).

U Vue se mogu koristiti predlošci koji se stavljaju u dvostruke vitičaste zagrade ({{}}). Na primjer: `<h1>{{nekiElement}}</h1>` ili korištenjem „render“ funkcije:

```
render: function (createElement) {
  return createElement('h1', this.nekiDrugiElement)
}
```

Kao i u React-u, ove metode omogućavaju pretvaranje komponenti u DOM. Ujedno se komponente i DOM ažuriraju tijekom svake promjene u njima pa korisnik dobiva ažurirani pogled s najnovijim vrijednostima. Za razliku od React-a i Vue-a, Angular ne koristi virtualni DOM pa stoga komponente nemaju „render“ funkcije ili metode. U kodu ispod se nalazi primjer kreiranja komponente u Angular-u.

```
@Component({
  selector: 'korisnik',
  inputs: ['ime', 'prezime'],
  template: `
    Ime: {{ ime }}
    Prezime: {{ prezime }}`
})
```

Iz koda se može vidjeti da komponenta ima selektor čija je vrijednost „korisnik“, može primiti dva ulaza: ime i prezime. Komponenta vraća predložak koji ispisuje ime i prezime koji su definirani kod ulaza.

5.4. Predlošci

Angular, React i Vue koriste različite predloške kod izrade aplikacija. Za izradu Angular aplikacija potrebno je naučiti posebnu sintaksu specifičnu samo za Angular. Naime, Angular funkcionira tako da na jedan način povezuje HTML s JavaScript-om, to jest, koristi običan HTML kod i u njega se ugrađuje JavaScript kod. Zbog tog razloga je potrebno naučiti posebnu sintaksu Angular-a koja to i omogućuje. U kodu ispod se mogu vidjeti primjeri sintakse dodavanje JavaScript-a u HTML kod.

```
<input [vrijednost]="ime">
```

Kod iznad dodjeljuje svojstvo „vrijednost“ rezultatu izraza „ime“.

Nadalje kod ispod dodjeljuje klasu „osvijetli“ na div element i „boolean“ vrijednosti izraza „provjera“.

```
<div [class.osvijetli]="provjera">
```

U sljedećem kodu dodjeljuje se stil „width“ u pikselima ovisno o rezultatu izraza „sirina“.

```
<div [style.width.px]="sirina">
```

Nakon toga u kodu ispod poziva se metoda „promijeniTekst“ samo kada se desi događaj klika mišem. Metoda prosljeđuje „event“ objekt.

```
<button (click)="promijeniTekst($event)">
```

Iznad su prikazani samo osnovni primjeri sintakse u Angular-u. Naravno takva sintaksa se koristi uvijek (redovito) pa stoga postoje i mnogi drugi načini za uključivanje JavaScript-a u HTML koji su složeniji.

S druge strane React čini upravo suprotno. Angular uključuje JavaScript u HTML, a React uključuje HTML u JavaScript. To se postiže korištenjem JSX-a (eng. *JavaScript XML*).

JSX je sintaksa koja proširuje ECMAScript. Preporučuje ga se koristiti u React-u i sintaksa je slična XML-u i HTML-u. Sintaksa je namijenjena da ju koriste kompajleri (eng. *Compiler*) poput Babel-a u React-u. Kompajleri transformiraju HTML koji se nalazi u JavaScript kodu u standardne JavaScript objekte (*JSX - What Is a JSX? & Introduction to Advanced*, bez dat.).

U primjeru koji se nalazi ispod je prikazano kako izgleda JSX kod (slika 4). Nakon toga je prikazano kako taj kod izgleda nakon što ga Babel transformira u JavaScript objekt (slika 5).

```
...
<div className={styles.Content}>
  <h3>
    {korisnik && korisnik.ime} {korisnik && korisnik.prezime}
  </h3>
  <h6 className={styles.Cert}>Certifikati</h6>
  <ul className={styles.Ul}>
    {certifikati &&
      certifikati.map((cert) => (
        <li className={styles.Li} key={cert._id}>
          {cert.naziv}
        </li>
      ))}
  </ul>
</div>
...
```

Slika 4: Primjer JSX koda

```
React.createElement(
  "div",
  { className: styles.ProfileCard },
  React.createElement("img", {
    src: "./images/profile.png",
    alt: "Slika profila",
    className: styles.Picture
  }),
  React.createElement(
    "div",
    { className: styles.Content },
    React.createElement(
      "h3",
      null,
      korisnik && korisnik.ime,
      " ",
      korisnik && korisnik.prezime
    ),
    React.createElement(
      "h6",
      { className: styles.Cert },
      "Certifikati"
    ),
    React.createElement(
      "ul",
      { className: styles.Ul },
      certifikati && certifikati.map(function (cert) {
        return React.createElement(
          "li",
          { className: styles.Li, key: cert._id },
          cert.naziv
        );
      })
    )
  )
);
```

Slika 5: Primjer JSX koda - Babel

Kao što se može vidjeti iz primjera (slike 4 i 5) koda iznad JSX je izuzetno koristan za pisanje HTML-a u JavaScript kodu jer olakšava pisanje komponenti. Programski kod je čitljiviji i potrebno je manje linija koda čime se smanjuje mogućnost pogrešaka prilikom pisanja koda. Osim u React-u, JSX se može koristiti i u Vue ukoliko se instalira Babel.

Vue omogućuje stvaranje predložaka na način da se komponente mogu stavljati svaka u svoju posebnu datoteku s ekstenzijom „.vue“. To se naziva komponente s jednom datotekom (eng. *Single-file components*). Na taj način se omogućuje isticanje cjelokupne sintakse datoteke (HTML, CSS i JavaScript). U kodu ispod se nalazi primjer jedne komponente s jednom datotekom koja sadrži cjelokupni kod potreban za tu komponentu.

```
<template>
  <h1>{{ pozdrav }} svijete!</h1>
</template>
```

Kod iznad predstavlja HTML dio komponente. Primjer koda ispod predstavlja JavaScript dio komponente.

```
<script>
   module.exports = {
     data: function() {
       return {
         pozdrav: "Pozdrav"
       };
     }
   };
</script>
```

U kodu iznad se nalazi funkcija koja vraća tekst „Pozdrav“ koji se zatim koristi u HTML dijelu kodu. Kod ispod predstavlja dio komponente s CSS-om.

```
<style scoped>
   h1 {
     text-align: center;
     color: red;
   }
</style>
```

Oblikuje se naslov prve razine s središnjim poravnanjem i crvenom bojom teksta.

5.5. Povezivanje podataka

Kako bi razumjeli razlike u povezivanju podataka između Angular-a, React-a i Vue-a, prvo je potrebno znati što je to zapravo povezivanje podataka. Povezivanje podataka je postupak koji uspostavlja vezu između korisničkog sučelja aplikacije i podataka koje prikazuje. Ako povezivanje ima ispravne postavke, kod promjene njihove vrijednosti, elementi koji su vezani za podatke automatski se mijenjaju. Povezivanje podataka također može značiti da ako se vanjski prikaz podataka u nekom elementu promijeni tada se temeljni podaci mogu automatski ažurirati kako bi se odrazila promjena. Na primjer, ako korisnik uređuje vrijednost u „TextBox“ elementu, osnovna vrijednost podataka automatski se ažurira kako bi odražavala tu promjenu ("Data binding overview - WPF | Microsoft Docs", bez dat.).

Angular podržava dvosmjerno povezivanje podataka (eng. *Two-way data binding*) koje funkcionira na sljedeći način ("Angular - Two-way binding ", bez dat.):

1. Postavlja svojstvo određenog elementa.
2. Osluškuje događaje koji mijenjaju elemente.

U Angular-u se razlikuju dvije vrste povezivanja podataka ("Angular - Introduction to Angular concepts", bez dat.):

1. Povezivanje događaja koje omogućuje aplikaciji da odgovori na unos podataka od strane korisnika tako što podaci koji se nalaze u stanju (eng. *State*) aplikacije mijenjaju u skladu s tim unesenim podacima.
2. Povezivanje entiteta koje omogućuje interpoliranje vrijednosti koje se izračunavaju iz podataka u stanju aplikacije.

React s druge strane podržava jednosmjerno povezivanje podataka (eng. *One-way data binding*) što znači da se podaci mogu kretati samo u jednom smjeru. To se postiže tako što se komponente stavljaju unutar jedne glavne komponente koja sadrži trenutno stanje aplikacije. Podaci o stanju te aplikacije dalje šalju komponentama koje se nalaze niže u hijerarhiji putem koncepta proizvoljni ulaz (eng. *Props*). „Vue“ podržava jednosmjerno povezivanje podatka kao i dvosmjerno povezivanje podataka. Svi proizvoljni ulazi (eng. *Props*) tvore jednosmjernu povezanost između svojstva djece i svojstva roditelja. Kada se roditeljski entitet ažurira, podaci idu od roditelja prema djetetu, ali ne i obrnuto. Komponente djece neće slučajno promijeniti stanje roditelja, što može otežati promišljanje protoka podataka aplikacije. Međutim, također je moguće izričito nametnuti dvosmjerno ili jednokratno povezivanje s modifikatorima tipa „.sync“ i „.once“ ("*Components - vue.js"*, bez dat.).

5.6. Performanse

U ovom poglavlju će se prikazati performanse Angular-a, React-a i Vue-a kako bi se vidjelo koji programski okvir najbrže može obavljati jednostavne operacije. Također će se usporediti brzina tih okvira s Vanilla JavaScript-om (JavaScript bez korisničkog okvira). U ovom poglavlju će se gledati operacije nad tablicama poput dodavanja redaka, brisanja i ažuriranja. Na kraju će se usporediti memorija koja je potrebna okvirima kao i JavaScriptu za čitanje nakon što se web stranica učita i koja je memorija iskorištena nakon dodavanja 1000 redaka u HTML tablicu.

Za testiranje su korištene takozvane „keyed“ verzije, to jest za svaki redak je dodan ključ koji omogućuje programskom okviru da zna nad kojim je to točno retkom provedena operacija. „Keyed“ verzije su upravo korištene jer se koriste i kod razvoja stvarnih aplikacija. Tablica 1 prikazuje rezultate operacija nad tablicama, rezultati su prikazani u milisekundama (ms).

Tablica 1: Usporedba performansi programskih okvira i Vanilla JavaScript-a

NAZIV	Angular	React	Vanilla JavaScript	Vue
Kreiranje redaka	193.1	188.9	138.5	166.7
Zamjena svih redaka	197.4	201.0	148.0	168.5
Djelomično ažuriranje redaka	13.0	16.5	14.1	17.3
Označavanje redaka	3.4	8.8	10.1	9.3
Zamjena redaka	13.4	14.7	11.4	18.3
Brisanje redaka	46.1	47.2	42.8	52.6
Kreiranje mnogo redaka	1946.0	1852.4	1331.1	1587.5
Dodavanje redaka u veliku tablicu	324.6	345.6	295.3	399.5
Čišćenje redaka	379.9	398.4	174.8	254.5
Vrijeme pokretanja	84.3	70.0	40.5	56.6
Usporavanje	1.31	1.30	1.00	1.21

(Izvor: "Interactive Results", bez dat.)

Prvo što se može uočiti iz tablice 1 jest to da Vanilla JavaScript ima najmanje vrijeme potrebno za obavljanje bilo kakve operacije nad tablicama. Razlog tome je to što programski okviri poput Vue, Angular ili React su „nadogradnja“ na JavaScript te dodaju nove mogućnosti koje korisnicima olakšavaju razvoj web aplikacija, ali to ima posljedice. Kao što se iz tablice 1 gore može vidjeti, okviri prave kompromis tako da se za uzvrat za lakše programiranje gubi na brzini izvođenja koda. U nastavku će se zanemarivati rezultati JavaScript-a jer je već objašnjeno zašto je uvijek na prvom mjestu, te će fokus biti samo na programskim okvirima: Angular, React i Vue.

Prvi redak predstavlja vrijeme koje je potrebno za programske okvire da kreiraju 1000 redaka nakon što se web stranica učita. U ovoj kategoriji je Vue najbrži s prosjekom 166 milisekundi. Zatim slijedi React i Angular je na posljednjem mjestu. Razlog ovakvih rezultata je korištenje virtualnog dokumentno objektnog modela. Vue koristi poboljšanu verziju virtualnog DOM-a, React „običnu“ verziju, dok Angular ne koristi virtualni DOM.

Drugi redak predstavlja vrijeme potrebno da se ažurira svih prethodno 1000 kreiranih redaka, ali mjerenje počinje tek nakon što se prvo provede 5 iteracija te akcije. Tu je također Vue najbrži, s prosječnim vremenom od 168 milisekundi, ali ovaj put ga slijedi Angular i React je na zadnjem mjestu. Zašto je ovaj put Angular brži će biti objašnjeno nešto kasnije u ovom poglavlju.

Sljedeći redak predstavlja vrijeme potrebno za djelomično ažuriranje redaka (svaki deseti redak) i također počinje tek nakon 5 iteracija ove operacije. Kod ove operacije dolazi do potpunog preokreta u brzini između programskih okvira. Prvi je Angular s prosječnom brzinom od 13 milisekundi, zatim slijedi React i na zadnjem mjestu je Vue koji je prilikom prijašnjih operacija bio na prvom mjestu.

Četvrti redak predstavlja prosječno vrijeme potrebno za označavanje retka tablice, a mjerenje počinje tek nakon 5 iteracija. Ovdje je najbrži opet Angular s 3.4 milisekunde, React je na drugom mjestu i Vue je zadnji.

Sljedeći (peti) redak predstavlja prosječno vrijeme potrebno da se u tablici s 1000 redaka zamijene pozicije 2 retka, a brojanje počinje nakon 5 iteracija. Angular je ponovno najbrži s 13.4 milisekundi, a slijedi React s 14.7 i Vue s 18.3 milisekundi.

Šesti redak prikazuje prosječno vrijeme potrebno da se iz tablice izbriše jedan redak, uz prethodnih 5 iteracija prije početka mjerenja. Angular je ponovno na prvom mjestu sa samo 46 milisekundi, a zatim slijedi React s 47 milisekundi i na kraju Vue s čak 52 milisekundi.

Nadalje se testira brzina nad operacijama bez prethodnih iteracija. Sedmi redak prikazuje rezultate kreiranja 10000 redaka. Kod ove operacije je Vue na prvom mjestu s prosjekom od 1587 milisekundi. Zatim slijedi React s 1852 milisekunde i na kraju Angular s 1946 milisekundi.

Osmi redak u tablici prikazuje koliko je vremena potrebno za dodavanje 1000 redaka u tablicu koja već ima 10000 redaka. Na prvom mjestu je Angular s 324 milisekunde, zatim slijedi React s 345 milisekunde i Vue s 399 milisekunde.

Deveti redak prikazuje vrijeme potrebno da se očisti tablica ispunjena s 10000 redaka. Na prvom mjestu je Vue s 254 milisekunde. Zatim slijedi Angular s 379 milisekunde i React se nalazi na posljednjem mjestu s 398 milisekunde. U ovom primjeru se može uočiti velika razlika u performansama kod brisanja, brzina Angular-a i React-a se jako razlikuje od Vue-a, to jest puno su sporiji od Vue-a.

Posljednja performansa koja se mjeri nije vezena uz operacije nad tablicama, nego vrijeme potrebno da se programski okvir učita, analizira sintaksu i pokrene. Najbrži je Vue s 56 milisekundi. Zatim React sa 70 milisekundi i na kraju Angular s 84 milisekunde. Kod ovog primjera najveći učinak ima veličina samog programskog okvira. Što programski okvir više zauzima prije samog pokretanja, to mu je potrebno više vremena da učita sve potrebne dijelove za upotrebu.

Posljednji redak u tablici 1 prikazuje koliko su programski okviri u prosjeku sporiji u odnosu na Vanilla JavaScript. Može se vidjeti da je Vue na temelju prikazanih testiranja sporiji za 20% u odnosu na Vanilla JavaScript. Dalje slijedi React koji je sporiji za 30% i na zadnjem mjestu je Angular s 31%.

Drugi dio testiranja performansi programskih okvira se odnosi na korištenje memorije. U tablici 2 se nalaze rezultati testiranja.

Tablica 2: Usporedba korištenja memorije programskih okvira i Vanilla JavaScript-a.

Naziv	Angular	React	Vanilla JavaScript	Vue
Memorija čitanja	4.8	4.5	3.4	3.8
Memorija nakon korištenja	10.9	9.7	4.0	7.5

(Izvor: "Interactive Results", bez dat.)

U tablici 2 se također može vidjeti da je samom Vanilla JavaScript jeziku potrebno najmanje memorije za rad (memorija čitanja). Podaci su iskazani u megabajtima (MB). Može se vidjeti da sljedeći programski okvir koji zauzima najmanje prostora nakon što se sama web stranica učita jest Vue sa 3.8 MB. To je 10% više od Vanilla JavaScript-a. Nakon Vue-a je React sa 4.5 MB potrebnog prostora za rad poslije učitavanja web stranice. To je 30% više od JavaScripta. Na kraju je Angular sa 4.8 MB, to jest 40% više memorije.

Druga kategorija u kojoj su programski okviri testirani je korištenje memorije nakon što je dodano 1000 redaka u tablicu (memorija nakon korištenja). Ovdje se može vidjeti da programski okviri zahtijevaju puno više memorije od Vanilla JavaScript-a koji koristi 4 MB. Zatim slijedi Vue s čak 7.5 MB što je 90% više. Nakon njega React s 9.7 MB, 140% više. Na kraju Angular s 10.9 MB, 170% više od JavaScript-a. U ovoj tablici se može vidjeti da Vue, koji zahtjeva najmanje memorije, je na prvom mjestu.

Tijekom analize rezultata moglo se zaključiti sljedeće. Prvi je taj da je Vue brži kod izvođenja operacija koje mijenjaju cjelokupni dokumentno objektni model (eng. DOM - Document Object Model). Razlog tome je što Vue koristi prilagođeni DOM. Drugi je taj da je Angular najbrži kod operacija ažuriranja, dok je sporiji kod kreiranja novih DOM elemenata. I posljednji je taj da React ni u jednoj operaciji nije najbolji i samo kod brisanja redova je najsporiji. Zaključak na temelju toga je da React nije ni po čemu najbrži, ali je u svemu dovoljno dobar da se može natjecati s ostalim JavaScript programskim okvirima za razvoj korisničkog sučelja web aplikacija.

5.7. Testiranje programskog koda

Pisanje testova olakšava pronalaženje grešaka u kodu prije nego što je prekasno, to jest prije nego se aplikacija već koristi. U suprotnom korisnici mogu namjerno ili nenamjerno uzrokovati greške tako da ne radi kako je zamišljeno. U ovom poglavlju će se opisivati testiranje upotrebom Jest-a i Jasmine alata koje služe za testiranje koda sva tri programska okvira koja su bila spomenuta u ovom radu.

5.7.1. Jest

Jest je JavaScript okvir za testiranje. Svrha mu je da osigura ispravnost bilo kojeg JavaScript koda. Omogućuje pisanje testova s pristupačnim, bogatim API-jem koji brzo daje rezultate ("Jest Delightful JavaScript Testing", bez dat.).

Kako bi se Jest koristio u projektu potrebno je samo izvršiti sljedeću naredbu u komandnoj liniji:

```
npm install --save-dev jest
```

Ovom narednom se pomoću npm-a (eng. *Node Package Manager*) instalira paket Jest. Dodana je i naredba „—save-dev“. Ova naredba instalira paket samo za fazu razvoja aplikacije. Kod testiranja vrijednosti dodaje se ključna riječ „expect“ uz koju se koriste podudaranja (eng. *Matchers*). Pomoću ovih dviju ključnih riječi se mogu testirati očekivane vrijednosti dijelova aplikacije. Na primjeru koda ispod će biti prikazana jednostavna funkcija i kako je testirati pomoću Jest-a korištenjem ključne riječi „expect“ i podudaranja.

```
function suma(a, b) {
  return a + b;
}
module.exports = suma;
```

Kod iznad predstavlja sadržaj datoteke „suma.js“ koja sadrži funkciju „suma“ koja prima dva argumenta (a i b). Funkcija vraća vrijednost zbroja ta dva argumenta. Posljednja linija se koristi za „izvoz“ modula (funkcije) „suma“ kako bi se mogla koristiti u drugim datotekama. To je potrebno za pristupanje funkciji iz datoteke za testiranje. Kako bi se funkcija „suma“ testirala potrebno je kreirati novu datoteku „suma.test.js“ u istom direktoriju kao i „suma.js“.

Kod ispod se nalazi u datoteci „suma.test.js“ i koristi se za testiranje funkcije „suma“ iz datoteke „suma.js“.

```
const suma = require('./suma');
test('zbroj 1 + 2 daje rezultat 3', () => {
  expect(suma(1, 2)).toBe(3);
});
```

Prvom linijom koda se učitavaju moduli iz datoteke „suma“ koja se nalazi u istom direktoriju. U ovom primjeru se uvozi samo funkcija „suma“. Zatim se poziva funkcija „test“ koja prima dva argumenta. Prvi argument je „string“ kojim se opisuje test. Drugi argument je funkcija povratnog poziva kojom se taj test izvršava. Korištena je ključna riječ „expect“ i očekivanje „toBe“. Funkcijom povratnog poziva je definiran test koji očekuje da će vrijednost funkcije „suma“ s argumentima 1 i 2 vratiti vrijednost 3.

Kako bi se testiranje izvršilo potrebno je izvršiti sljedeću naredbu u naredbenom retku:

```
npm run test
```

Zatim Jest ispisuje rezultat testiranja u naredbeni redak:

```
PASS  ./suma.test.js
✓ zbroj 1 + 2 daje rezultat 3 (5ms)
```

Prikazuje se da je test „zbroj 1 + 2 daje rezultat 3“ iz datoteke „suma.test.js“ prošao.

5.7.2. Jasmine

Jasmine je razvojni okvir za testiranje JavaScript koda vođen ponašanjem (eng. Behavior-driven development framework). Ne ovisi o drugim JavaScript okvirima. Ne zahtijeva DOM. Ima čistu sintaksu tako da se testovi mogu brzo pisati ("Jasmine Documentation", bez dat.).

Jasmine se sastoji od „slučaj testiranja“ (eng. *Suites*) koje opisuju testove. Slučajevi testiranja sadrže „funkcije opisivanja“ (eng. *Describe functions*) koje grupiraju povezane „specifikacije“ (eng. *Specs*). Obično svaka testna datoteka ima jednu funkciju opisivanja na najvišoj razini. Svaka od tih funkcija sadrži „string“ i funkciju. „String“ služi za imenovanje skupa specifikacija. Tako se lakše pronalaze specifikacije u velikim „slučajevima testiranja“ ("Your_first_suite", bez dat.).

Specifikacije su definirane pozivanjem globalne funkcije „Jasmine“ koja za opis uzima niz i funkciju. Niz je naslov specifikacije, a funkcija je specifikacija ili test. Specifikacija sadrži jedno ili više očekivanja (eng. *Expect*) koja testiraju stanje koda. Očekivanje u Jasminu je tvrdnja koja je istinita ili lažna. Specifikacija sa svim pravim očekivanjima je prolazna specifikacija (eng. *Specs*). Specifikacija s jednim ili više lažnih očekivanja je neuspješna specifikacija ("Your_first_suite", bez dat.).

Očekivanja su izgrađena funkcijom „expect“. Slijedno je povezana funkcijom podudaranja (eng. *Matcher function*) koja uzima očekivanu vrijednost. Kod ispod prikazuje primjer testiranja koda ("Your_first_suite", bez dat.).

```
describe("Funkcije podudaranja:", function() {
   it("'toBe' funkcija podudaranja se podudara sa ===", function() {
       var a = 12;
       var b = a;

       expect(a).toBe(b);
       expect(a).not.toBe(null);
     });
   it("'toBeDefined' funkcija podudaranja se uspoređuje sa `undefined`",
   function() {
       var a = {
         foo: "foo"
       };

       expect(a.foo).toBeDefined();
       expect(a.bar).not.toBeDefined();
     });
   });
```

Kod iznad prikazuje jedan slučaj testiranja sa dvije specifikacije. Prva specifikacija prikazuje primjer korištenja „toBe“ funkcije podudaranja. Definirane su dvije varijable „a“ i „b“. Prvo očekivanje u specifikaciji očekuje da će varijable „a“ biti jednaka varijabli „b“. Očekivanje je točno jer varijabla „b“ ima vrijednost varijable „a“. Od drugog se očekivanja očekuje da varijable „a“ neće imati vrijednost „null“. To se postiže slijednim povezivanjem funkcija „not“ i „toBe“. Ovo očekivanje je također točno jer varijabla „a“ ima već postavljanu vrijednost.

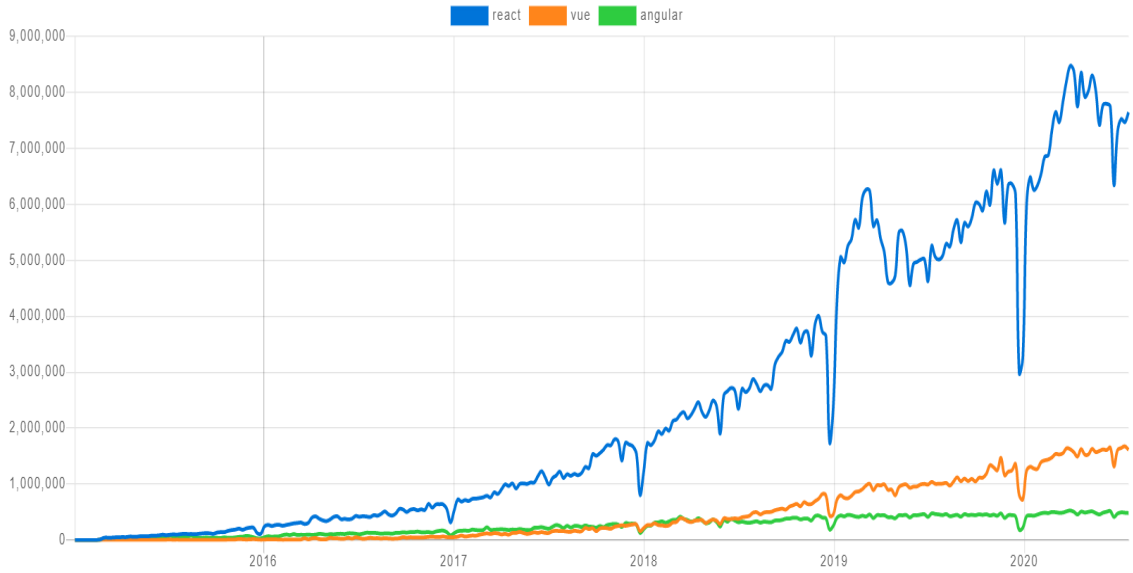
Druga specifikacija prikazuje korištenje „toBeDefined“ funkcije podudaranja. Definiran je objekt „a“ sa svojstvom „foo“ koje ima vrijednost „foo“. Prvo je kreirano očekivanje gdje se provjerava je li svojstvo „foo“ iz objekta „a“ definirano. Ovo očekivanje je točno jer svojstvo ima vrijednost „foo“. Suprotno tome, drugo očekivanje slijednim povezivanjem funkcija provjerava da svojstvo „bar“ objekta „a“ nije definirano. Ovo je očekivanje također točno jer ne postoji svojstvo „bar“ unutar objekta „a“.

5.8. Popularnost

U ovom poglavlju istražit će se popularnost tri JavaScript programska okvira (Angular, React, Vue) za korisnički dio sučelja. Usporedit će se po broju preuzimanja i po broju praćenja na popularnoj platformi verzioniranja (Github).

Prvo će se napraviti usporedba prema broju preuzimanja putem NPM-a (slika 6). NPM (eng. *Node Package Manager*) je inicijalno bio zamišljen kao upravitelj paketa za Node.js, ali je danas najveći i najpopularniji repozitorij koji omogućuje dijeljenje softvera otvorenog koda ili upravljanje verzijama programskog koda ("What is npm", bez dat.).

Upravo zbog toga što je to najveći repozitorij, koristit će se podaci o preuzimanju JavaScript programskih okvira koji se obrađuju o ovom radu (slika 6).



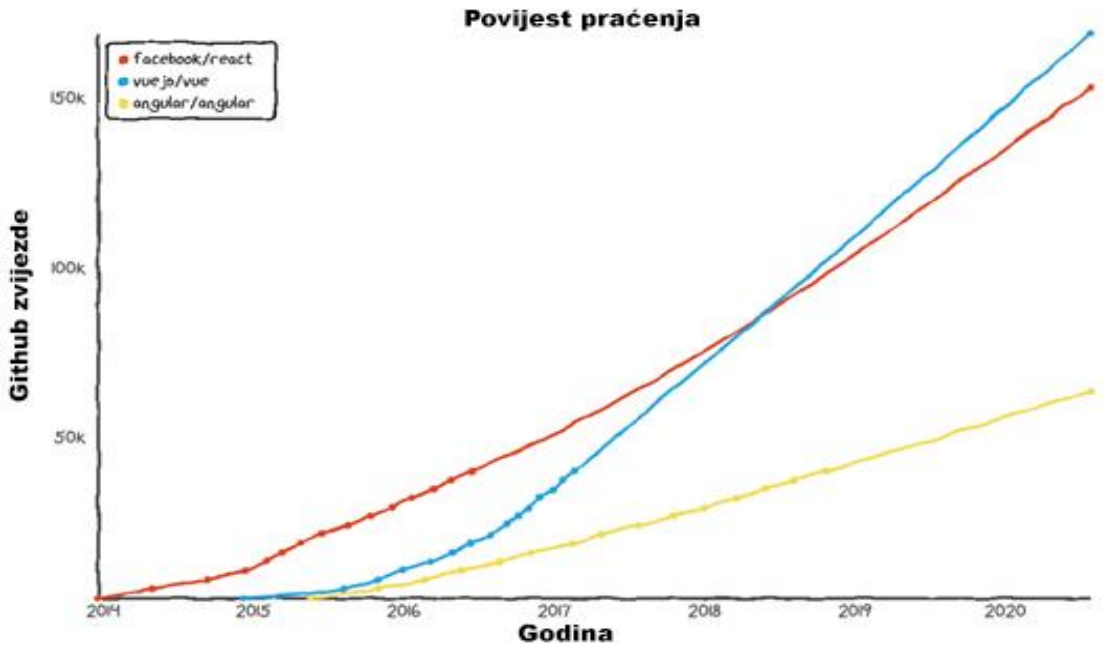
Slika 6: Trendovi preuzimanja s NPM-om (Izvor: angular vs react vs vue | npm trends, bez dat.)

Iz grafa (slika 6) se može vidjeti da su 2016. godine Angular, React i Vue bili gotovo izjednačeni po broju preuzimanja putem NPM-a. Nakon toga React-u raste broj korisnika iz godine u godinu, dok je kod Vue i Angular-a broj preuzimanja gotovo jednak. Sredinom 2018. godine Vue prestiže Angular.

React je u 2020. godini imao najviše oko 8 milijuna i 500 tisuća preuzimanja, dok je Vue imao oko milijun i 600 tisuća preuzimanja, a Angular samo oko pola milijuna. Ovi podaci govore da je React neupitno najpopularniji kod preuzimanja NPM-om.

Ovaj graf prikazuje samo korištenje JavaScript programskih okvira putem NPM-a, ali oni se mogu koristiti i na druge načine. Na primjer, korištenje manje verzije tih okvira koje se mogu samo dodati u projekt HTML oznakom „script“. Zbog toga će biti prikazan i broj praćenja na repozitoriju.

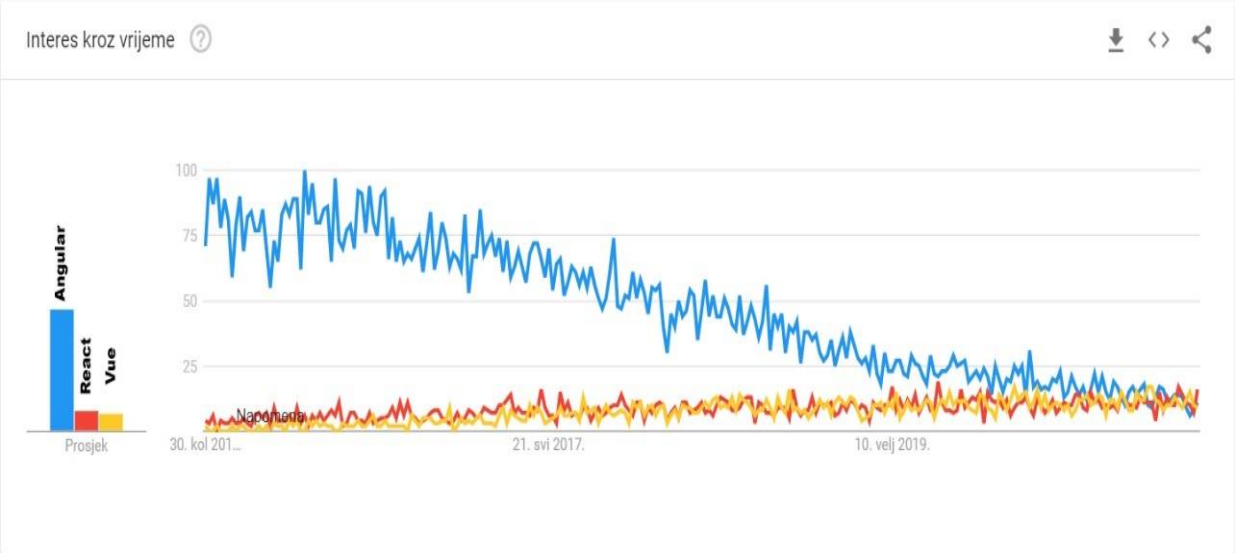
Github sadrži različite projekte, pa tako i projekte koji koriste JavaScript programske okvire koji se ovdje obrađuju. Popularnost tih programskih okvira će se provjeriti brojem zvijezda na Github-u, to jest brojem osoba koja ih prate i podržavaju te projekte (slika 7).



Slika 7: Broj praćenja na Github-u (Izvor: Tim Qian, bez dat.)

Slika 7 prikazuje graf s brojem praćenja React-a, Vue-a i Angular-a tijekom godina. Iz grafa se može vidjeti da je Angular na dnu s brojem praćenja od početka, dok je React bio na prvom mjestu sve do 2018. godine kada ga Vue prestiže. Trenutno u 2021. godini Vue ima oko 170 tisuća pratitelja, React oko 150 tisuća, a Angular oko 60 tisuća pratitelja.

Posljednja kategorija prema kojoj će se uspoređivati popularnost triju programskih okvira je na Google trendovima. To je web mjesto koje analizira popularnost pretraživanja različitih pojmova u različitim regijama svijeta i na različitim jezicima. Na slici 8 se mogu vidjeti rezultati pretraživanja pojmova Angular, React i Vue u cijelom svijetu, u proteklih 5 godina.



Slika 8: Popularnost pojmova na Google trendovima(Izvor: "Angular, React, Vue - Istražite - Google trendovi", bez dat.)

Iz slike 8 se može vidjeti da je Angular najpopularniji pretraživani pojam. Međutim, također se može vidjeti da postoji trend pada popularnosti svake godine pa se u 2020. godini izjednačava s React i Vue. Iz grafa se može vidjeti da je Angular uvjerljivo na prvom mjestu, dok ga slijedi React koji je gotovo izjednačen s Vue.

Zaključak ovog poglavlja je da se za najviše projekata (putem NPM-a) koristi React. Vue ima najviše pratitelja na Github-u te bi i zbog toga u narednih nekoliko godina mogao i prestići React po broju preuzimanja s NPM-om. Temeljem rezultata iz Google trendova se može vidjeti da Angular iz godine u godinu gubi na popularnosti. Iz toga se može zaključiti da programeri sve više posežu za drugim programskim okvirima koji su fleksibilniji kod načina razvoja aplikacija, a da nisu nužno React ili Vue.

6. Kriteriji odabira programskog okvira

Prvi kriterij odabira koji će se obraditi u ovom poglavlju jest prema veličini projekta, odnosno aplikacije koja se razvija. Programski okviri koji zauzimaju više mjesta na tvrdom disku (računalu) sadrže više alata koji su potrebni kod razvoja većih aplikacija ili aplikacija koje se s vremenom skaliraju. Na tablici 3 se mogu vidjeti veličine programskim okvira Angular, React i Vue. Podaci su preuzeti s Github-a gdje se ti okviri i nalaze.

Tablica 3: Veličina programskih okvira Angular, React i Vue

Naziv	Veličina
Angular 2	566K
React 16.2.0	97.5K
Vue 2.4.2	58.8K

(Izvor: Vynogradenko, 2018)

Kao što se može vidjeti iz tablice 3, Angular je „najveći“ programski okvir i sadrži najviše alata za izradu aplikacija. Što znači da je najpogodniji za izradu većih i manjih aplikacija koje se skaliraju. Za razliku od njega su React i Vue manji, brži i pogodniji za izradu manjih aplikacija. Koriste se kad je bitnije da je aplikacija brža.

Iz broja alata ugrađenih u programski okvir se može zaključiti i koliko su oni zapravo fleksibilni kod razvoja aplikacija. Za programere koji žele imati slobodu izbora kod odabira alata se preporučuju React i Vue jer oni nemaju ugrađenu većinu alata. U slučaju da programer preferira striktno definirani način za razvoj aplikacija, onda je Angular pravi izbor.

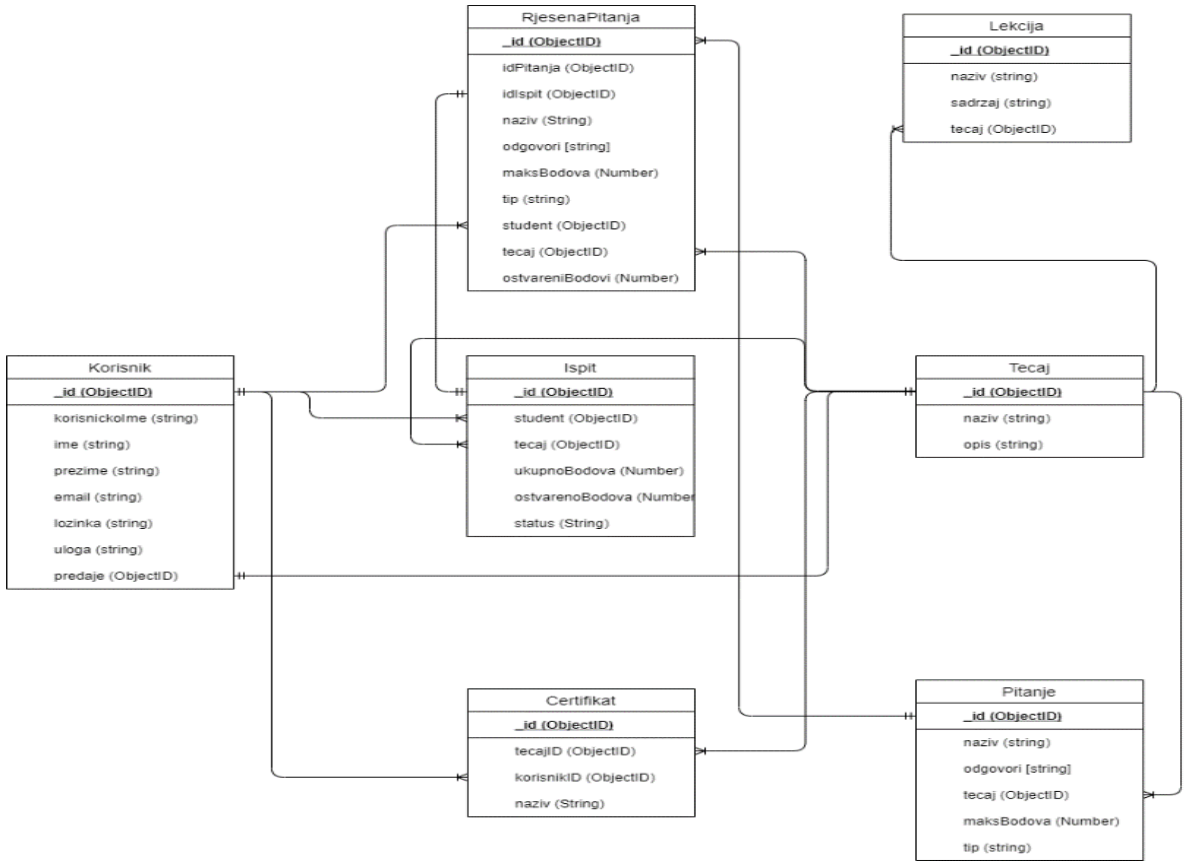
Treći kriterij koji bi se trebao uzeti u obzir prilikom odabira programskog okvira jest količina vremena potrebna za učenje samog okvira. Angular zahtjeva najviše vremena, posebice ako programeri nemaju iskustvo s TypeScript-om koji se uglavnom koristi. Angular također koristi za sebe specifičnu sintaksu koju je potrebno učiti. S druge strane, React koristi JSX koji je lako naučiti jer se bazira na HTML-u koji se ugrađuje u JavaScript kod. Može se reći da je krivulja učenja puno manja u odnosu na Angular, posebice ako programeri imaju iskustva s JavaScript-om i HTML-om. Vue ima sličnu sintaksu kao i Angular, ali nema ni približno veliku krivulju učenja kao Angular. Vue se smatra jednim od najlakših programskih okvira za naučiti, dijelom je tome razlog i dobro napisana dokumentacija koja je detaljna i prikazuju se primjeri u različitim slučajevima korištenja.

7. Izrada aplikacije korištenjem React-a

Kao praktični dio ovog rada je kreirana aplikacija „Kviz“. Za izradu je korišten danas popularan MERN skup alata koji se sastoji od četiri alata za razvoj aplikacija (MongoDB, ExpressJS, ReactJS i NodeJS). NodeJS se koristi za izvršavanje JavaScript na poslužitelju, umjesto u pregledniku kako je slučaj kod samog JavaScript-a. Nadalje, ostali alati će se raspodijeliti prema MVC uzorku. Za model aplikacije je korišten MongoDB, nerelacijska baza podataka u koju se spremaju podaci. Dio pogleda čini sam ReactJS, čija je funkcija prikazivanja podatka iz baze korisniku. Između modela i pogleda se nalazi upravljač pomoću kojeg se obrađuju podaci, upisuju u bazu i dohvaćaju iz baze kako bi se prikazali.

7.1. Model

Kao što je već prije spomenuto za spremanje podataka je kreirana MongoDB baza podataka. Za MongoDB ne treba prethodno definirati strukturu kolekcija. Iako nije potrebno, kreirana je osnovna struktura koju je moguće vidjeti na slici ispod.

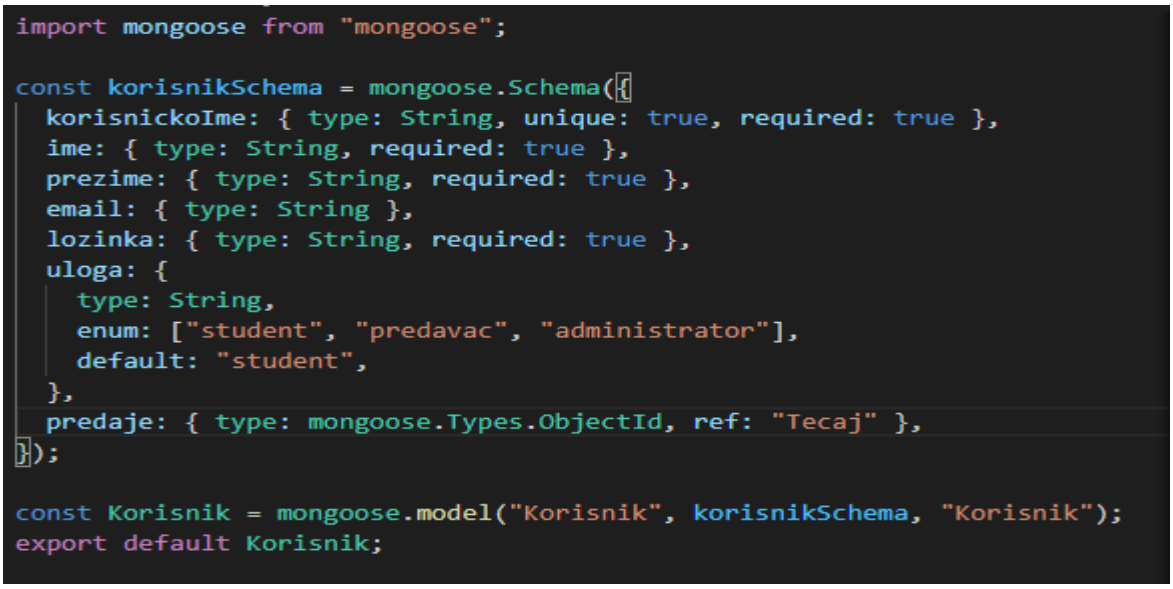


Slika 9: Model podataka

Baza se sastoji od sedam kolekcija:

- Korisnik: sadrži osnovne informacije o svakom korisniku poput: korisničkog imena, imena, prezimena, lozinke, email-a i opcionalno polja predaje (koje se ispuni samo kad je dodan kao predavač). Postoji i polje uloga čija je vrijednost unaprijed postavljena (eng. *Default*) na student, a mijenja se u predavač kad korisnik dobije tu ulogu. Osim te dvije uloge postoji i uloga administrator.
- Tečaj: svaki tečaj ima svoj naziv i kratak opis koji se moraju definirati kod kreiranja istog. Osim toga tečaj sadrži lekcije i pitanja, a također se i iz njega može kreirati ispit. Ti podaci se nalaze u posebnim kolekcijama.
- Lekcija: svaka lekcija ima svoj naziv i sadržaj i pripada jednom tečaju.
- Pitanje: svako pitanje ima svoj naziv, tečaj kojem pripada, tip pitanja koje može biti (jedan točan odgovor, više točnih odgovora, esejsko, dopunjavanje i točno/netočno). Također je svakom pitanju moguće dodijeliti maksimalan broj bodova i ponuđene odgovore ako je potrebno.
- Ispit: svaki „ispit“ koji je zapravo kviz sadrži: studenta koji je pisao, tečaj kojem pripada, ukupno bodova koliko je moguće ostvariti iz svih pitanja, broj ostvarenih bodova i na kraju status koji je odmah postavljen na „zaIspraviti“. Status može poprimiti vrijednosti kasnije „položen“ i „nePoložen“.
- RjesenaPitanja: sadrži sve informacije o pitanju iz kojeg je nastalo, a ima i druge informacije poput: tečaja, ispita i studenta kojem pripada. Osim toga ima i polje u koje se upisuje ostvaren broj bodova.
- Certifikat: kolekcija koja sadrži podatke o tečaju koji je student položio, a kreira se samo ako je status ispita „položen“.

Za kreiranje strukture baze podataka je korišten paket „mongoose“. Primjer izvornog koda iz aplikacije se nalazi na slici ispod. Iz slike 10 se može vidjeti i polje „uloga“ koja ima tri moguće vrijednosti. Uz to postoji i jedna zadana uloga (eng. Default) koja se postavlja kada nije ni jedna uloga postavljena u zahtjevu.



Slika 10: Primjer definiranje sheme kolekcije

Slika 10 prikazuje shemu kolekcije „Korisnik“. Može se vidjeti da je za svako polje definiran tip vrijednosti. Polje „korisnickoIme“ je jedinstveno, to je postignuto postavljanjem vrijednosti atributa „unique“ na „true.“ Polja koja su obavezna imaju postavljenu vrijednost atributa „required“ na „true“. Polje „predaje“ ima referencu na kolekciju „Tecaj“ i mora biti tipa „ObjectID“ kao što je i identifikator u toj kolekciji.

7.2. Upravljač

Upravljač se koristi kao premosnica između modela i pogleda. U ovom projektu je za tu svrhu korišten ExpressJS pomoću kojeg se prikazuju podaci iz modela u dijelu pogleda i preko kojeg se tim podacima može manipulirati pozivanjem API zahtjeva u pogledu.

Ovaj dio projekta je strukturiran u više direktorija kako bi pisanje zahtjeva bilo što preglednije i jednostavnije. Postoji posebno direktorij „models“ s modelima. U datoteci „index.js“ se između ostalog nalaze osnovne putanje (eng. *Routes*) za upravljanje pojedinim modelom (slika 11).

Slika 11: Osnovne putanje u datoteci index.js

Svaka od putanja ima dalje svoje putanje koje se nalaze u direktoriju „routes“. One su organizirane u posebnim datotekama radi preglednosti (slika 12).

```
import express from "express";
import {
  korisnikRegistracija,
  korisnikPrijava,
  korisnikPromjeniUlogu,
  korisnikDodajCertifikat,
  korisnikDohvatiStudente,
  korisnikDodajPredaje,
  korisnikDohvatiProfil,
} from "../controllers/korisnik-controllers.js";
const router = express.Router();

//autentifikacija
router.post("/registracija", korisnikRegistracija);
router.post("/prijava/", korisnikPrijava);

//za profil
router.get("/dohvatiProfil/:korisnik", korisnikDohvatiProfil);

// //za administratora
router.patch("/promjeniUlogu", korisnikPromjeniUlogu);
router.patch("/dodajCertifikat", korisnikDodajCertifikat);
router.patch("/dodajPredaje", korisnikDodajPredaje);
router.get("/dohvatiStudente", korisnikDohvatiStudente);

export default router;
```

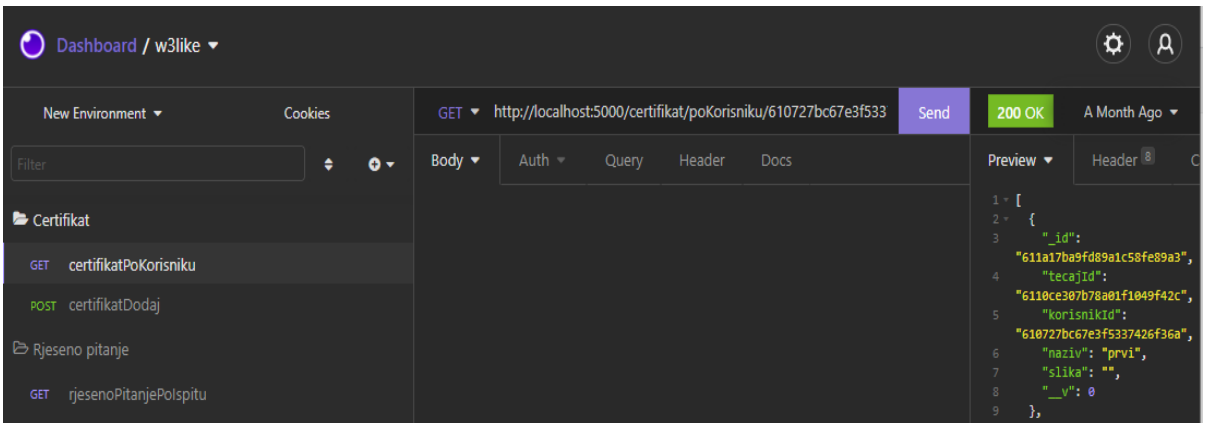
Slika 12: Putanje za korisnika u datoteci korisnik-routes.js

Nadalje, programski kod koji obavlja kontroliranje podacima se nalazi u direktoriju „controllers“ koja također za svaki model sadrži posebnu datoteku. Na sljedećoj slici je prikazan primjer upravljača za prijavljivanje korisnika koji se nalazi u datoteci korisnik-controller.js.

```
export const korisnikPrijava = async (req, res) => {
  const { korisnickoIme, lozinka } = req.body;
  try {
    const korisnik = await Korisnik.findOne({
      korisnickoIme,
      lozinka,
    });
    res.status(200).json(korisnik);
  } catch (error) {
    res.status(409).json({ greška: error.message });
  }
};
```

Slika 13: Primjer upravljača za prijavu korisnika

Tijekom izrade je kreirano ukupno 33 zahtjeva koji su također svi testirani pomoću programa „Insomnia“ kako bi se lakše otkrile greške u kodu.



Slika 14: Sučelje programa Insomnia i primjer testiranja zahtjeva

S lijeve strane se nalaze direktoriji za svaki model unutar kojeg se nalaze testirani zahtjevi. U sredini prozora se nalazi poveznica na API (eng. *Application Programming Interface*). S desne strane je odgovor poslužitelja na taj zahtjev.

Ako je sve u redu sa zahtjevom vraća se status „200 OK“. Ako postoji greška za zahtjevom, vraća se status sa odgovarajućim kodom. Na primjer „404 Not Found“ ako zahtjev nije pronašao odgovor.

7.3. Pogled

React je JavaScript biblioteka za razvoj korisničkog sučelja web aplikacija. Kreiran je 2013. godine od strane Facebook-a i od tada postaje sve popularniji okvir za izradu korisničkog sučelja web aplikacija. React je svoju popularnost stekao jer se je koristio na velikim projektima. Primjer toga su: Facebook, Instagram i ostali. Osim što je moćan okvir za velike projekte, može se i koristiti i za manje projekte, kao što je izrada jednostavne aplikacije koja sprema namirnice za kupnju ("Artemij Fedosejev", bez dat.).

React koristi kombinaciju deklarativnog i imperativnog načina programiranja. Deklarativno programiranje (eng. *Declarative programming*) govori računalu što treba učiniti bez da mu govori kako to učiniti. Takav način programiranja se koristi za lako kreiranje interaktivnih korisničkih sučelja i dizajniranje jednostavnih pogleda za svako stanje u aplikaciji. Pošto su pogledi izrađeni na taj način, programski kod je lako čitati i otklanjati pogreške. S druge strane, imperativni način programiranja (eng. *Imperative programming*) se koristi za API. ("React.js Essentials - Artemij Fedosejev - Knihy Google", bez dat.)("React – A JavaScript library for building user interfaces", bez dat.)

Specifično za React je način izrade korisničkog sučelja. Izrađuje se tako što se kreiraju enkapsulirane komponente koje mogu same upravljati svojim stanjem, a te komponente se zatim sastavljaju u kompleksna korisnička sučelja. Osim komponenti sa stanjima, razlikuju se i jednostavne komponente, to jest, one komponente koje ne sadrže stanja u sebi koja se mogu mijenjati ("React – A JavaScript library for building user interfaces", bez dat.).

7.3.1. Kreiranje razvojnog okruženja

Osim jednostavnog dodavanja React-a u HTML kod, mogu se koristiti i razni alati koji kreiraju predložak za izradu različitih web aplikacija. Razvojni programeri React-a preporučuje alat „Create React App“ za učenje ili kreiranje novih jednostraničnih aplikacija (eng. *Single Page Application*). NextJS za kreiranje web stranica koje se renderiraju na poslužitelju i Gatsby za statične web stranice koje su orijentirane na sadržaj. Osim tih alata postoje i mnogi drugi, od kojih je svaki prikladniji za drugačije vrste web aplikacija ili stranica. Gore navedeni alati su jako korisni, posebno kod većih projekata jer pomažu kod raznolikih zadataka. Neki od njih su skaliranje na veći broj datoteka i komponenti, korištenje biblioteka s npm-a, brzo i lako prepoznavanje učestalih grešaka u kodu, uređivanje CSS-a i JS-a u „živom“ okruženju i optimiziranje aplikacije za produkciju ("Create a New React App – React", bez dat.).

Nadalje će biti opisano kako kreirati razvojno okruženje pomoću „React Create App“ alata. Ovaj alat je kreirao tim iz Facebook-a i sadrži sve potrebno kako za razvoj React web aplikacije. Kreira poslužitelj za razvoj, koristi Webpack za automatsko kompiliranje React, JSX, ES6 i CSS-a. Također koristi i ESLint koji testira kod i pronalazi greške u kodu ("Getting Started with React - An Overview and Walkthrough Tutorial – Tania Rascia", bez dat.)

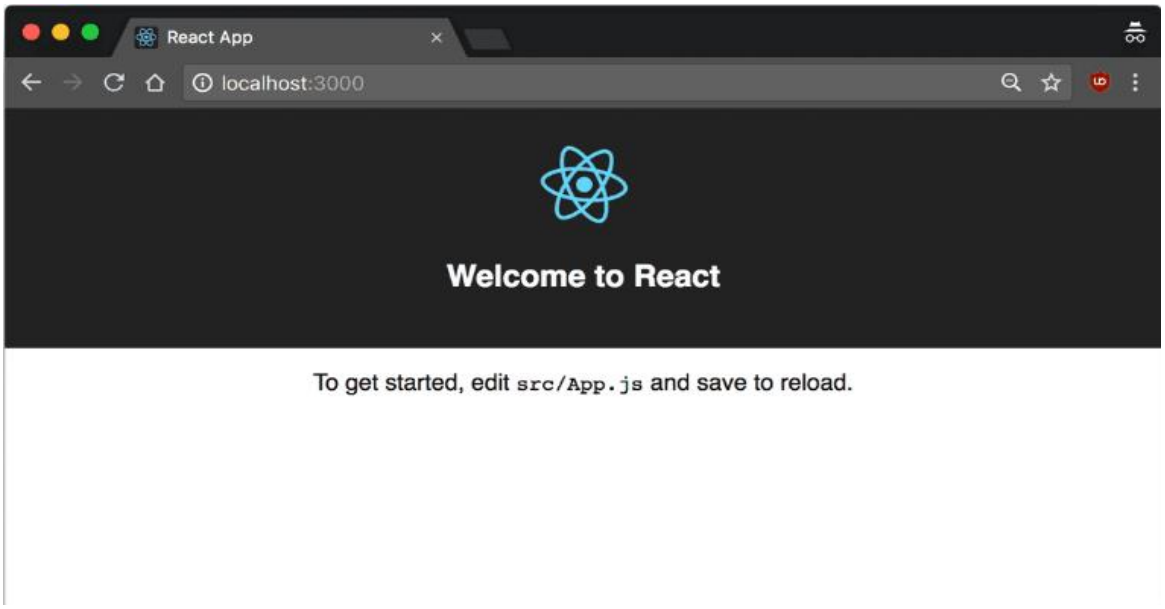
Kako bi se koristio „React Create App“, prvo je potrebno preuzeti Node.js. Nakon što je instaliran, otvorimo Node.js naredbeni redak i unesemo sljedeću naredbu ("Getting Started with React - An Overview and Walkthrough Tutorial – Tania Rascia", bez dat.):

```
npx create-react-app frontend
```

U direktoriju u kojem se nalazimo se kreira novi direktorij „frontend“ u kojoj se nalazi predložak za aplikaciju. Nakon toga se premještamo u tu mapu pomoću „cd“ naredbe i pokrećemo poslužitelj pomoću „npm“ naredbe ("Getting Started with React - An Overview and Walkthrough Tutorial – Tania Rascia", bez dat.):

```
cd frontend
npm start
```

Poslije toga se pokreće lokalni poslužitelj na localhost:3000 i to na kraju izgleda ovako (slika 15):



Slika 15: Početni pogled kod kreirane aplikacije (Izvor: „Getting Started with React - An Overview and Walkthrough Tutorial – Tania Rascia“, bez dat.)

Slika 15 prikazuje kako izgleda početna stranica React aplikacije nakon što se tek kreira pomoću npm-a. Početna stranica prikazuje kratku uputu kako započeti uređivati aplikaciju.

7.3.2. Korišteni paketi

Dio pogleda je napravljen u React-u i korišteni su sljedeću paketi:

- React-router-dom: koristi se kako bi zamijenio tradicionalne poveznice i omogućava prikazivanje više stranica unutar jedne glavne. Zbog ovog se paketa stranice ne osvježavaju klikom na poveznicu, već se učita novi pogled na glavnoj stranici.
- React-markdown: omogućava kreiranje uređivača za tekst koji uz pomoć specijalnih znakova može urediti običan tekst pomoću HTML-a.
- Axios: HTTP klijent koji radi na principu obećanja (eng. *Promise*). Pokušava se poslati zahtjev, ako je sve uredu, izvršava se kod, a ako nije javlja se greška.

7.3.3.Kreiranje komponenti

Konceptualno, komponente su zapravo JavaScript funkcije. Prihvaćaju proizvoljne unose (eng. *Props*) i vraćaju React element koji opisuju ono što bi se trebalo pojaviti na zaslonu. Komponente omogućuju podjelu korisničkog sučelja u neovisne dijelove koji se kasnije opet mogu koristiti.

Kao što je već navedeno, komponente su JavaScript funkcije, pa je zato najjednostavniji način za kreiranje komponente kreiranje JavaScript funkcije (primjer ispod).

```
function Landing(props) {
  return <h1>Pozdrav, {props.ime}</h1>;
}
```

Takva je funkcija valjana React komponenta jer prihvaća pojedinačni argument objekta „props“ (koji označava svojstva) s podacima i vraća element React. Takve komponente nazivamo „funkcionalne komponente“, jer su to doslovno JavaScript funkcije.("*Components and Props – React*", bez dat.)

Osim JavaScript funkcija, komponenta se također može kreirati korištenjem ES6 klasa (primjer ispod).

```
class Landing extends React.Component {
  render() {
    return <h1>Pozdrav, {this.props.ime}</h1>;
  }
}
```

Međutim, danas se najviše koriste „arrow“ funkcije čijim su uvođenjem dodane i „hook“ funkcije koje olakšavaju korištenje React-a, a iste će se primijeniti i prilikom izrade praktičnog rada.

```
const Landing = (props) => {
      return <h1>Pozdrav, {props.ime}</h1>
}
```

U primjeru koda je prikazan dio komponente „Landing“ koja je kreirana korištenjem „arrow“ funkcije. Sintaksa je kraća i lakša za čitanje u odnosu na korištenje klasa.

7.3.4.Koncepti u React-u

U nastavku na primjerima iz rada će se prikazati i objasniti osnovni koncepti koji se koriste za izradu web aplikacija. Na slici ispod je izvorni kod početne stranice u kojem se koriste neki od primijenjenih koncepta.

```
import React, { useState, useEffect } from "react";
import { Card, SecondaryCard } from "./Card/Card";
import axios from "axios";
const Landing = () => {
  const [data, setData] = useState([]);
  useEffect(() => {
    const dohvatiTecajeve = async () => {
      await axios
        .get(`/tecaj/dohvatiSve`)
        .then((res) => {
          setData(res.data);
        })
        .catch((err) => {
          console.log(err);
        });
    };
    dohvatiTecajeve();
  }, []);

  return (
    <div>
      {data.length > 0 &&
        data.map((tecaj, index) =>
          index % 2 === 0 ? (
            <Card
              title={tecaj.naziv}
              description={tecaj.opis}
              key={tecaj._id}
              link={tecaj._id}
            />
          ) : (
            <SecondaryCard
              title={tecaj.naziv}
              description={tecaj.opis}
              key={tecaj._id}
              link={tecaj._id}
            />
          )
        )}
    </div>
  );
};

export default Landing;
```

Slika 16: Izvorni kod početne stranice

Na slici 16 se može vidjeti primjer komponente iz razvijene aplikacije. Na početku su uvezene komponente i biblioteke koje će se koristiti u ovoj komponenti. Definirana je komponenta „Landing“. Zatim je postavljeno stanje (eng. *State*), a nakon toga se dohvaćaju podaci koji se spremaju u to stanje. Pogled za korisnika se nalazi nakon ključne riječi „return“. Posljednja linija izvornog koda „izvozi“ komponentu „Landing“ kako bi se ona mogla uvesti u druge komponente.

7.3.5. Metode životnog ciklusa

U React-u postoje metode životnog ciklusa (eng. *Lifecycle Methods*) koje govore aplikaciji kada je potrebno izvršiti neki kod, na primjer kad se stranice učitaju. Najviše korištene metode u React-u općenito su:

- componentDidMount: izvršava programski kod nakon što se je komponenta učitala u pogled. Korisna je kod dohvaćanja inicijalnog sadržaja.
- componentWillUnmount: izvršava se neposredno prije nego se komponenta izbriše iz pogleda i koristi se za čišćenje svega kreiranog u „componentDidMount“ metodi.

Uvođenjem „arrow“ funkcija ili „hook-ova“ više nije potrebno pozivati svaku metodu posebno. Sve je ujedinjeno u jednoj „hook“ funkciji koja se naziva „useEffect“. Na slici izvornog koda aplikacije (slika 17) se može vidjeti „useEffect“ funkcija.

```
const Sidebar = (props) => {
  // ...
  useEffect(() => {
    const dohvatiLekcije = async () => {
      await axios
        .get(`/lekcija/${id}`)
        .then((res) => {
          setLekcije(res.data);
        })
        .catch((err) => {
          console.log(err);
        });
    };
    dohvatiLekcije();
  }, [id]);
  // ...
};
export default Sidebar;
```

Slika 17: Izvorni kod "useEffect" funkcije

Na slici 17 se može vidjeti dio koda iz bočnog izbornika. Koristi se „useEffect“ funkcija unutar koje se šalje API zahtjev za dohvaćanje podataka o lekcijama. Funkcija može imati i drugi argument koji mora biti niz. Pomoću tog niza funkcija zna kada se treba ponovno izvršiti. Ako je potrebno funkciju izvršiti samo jednom prilikom učitavanja onda je niz prazan. U ovom slučaju se funkcija izvršava svaki put kad se promjeni „id“, to jest svaki put kad korisnik klikne na novi tečaj funkcija se ponovno izvršava.

7.3.6. Stanje

Stanje (eng. *State*) se u React-u koristi za spremanje dinamičnih podataka, uz pomoć kojih se može upravljati komponentama. Određuje se stanje komponenata. Stanje je zapravo naprednija vrsta varijable koju komponente prate kad se pojavi neka promjena.

U verziji React-a 16.8 je dodan „useState hook“ koji omogućava jednostavno kreiranje i manipuliranje stanjem. Kod kreiranja stanja korištenjem „useState-a“ koristi se destrukturiranje niza (eng. Array destructuring). Kreira se niz koji sadrži dvije varijable. Prva varijabla se odnosi na naziv stanja. Naziv druge varijable se koristi za naziv funkcije koja ažuriranje stanje. Konvencija je da se naziv stanja piše malim početnim slovom, ukoliko sadrži drugu riječ u nazivu koristi se „camel case“ način pisanja. Konvencija kod naziva funkcije za ažuriranje je stavljanje riječi „set“ ispred naziva stanja. Kod kreiranja stanja može se definirati i početno stanje. Primjer korištenja stanja i ažuriranja se nalazi na slici 18.

```
//...
const CreateLecture = () => {
    //...
    const [naziv, setNaziv] = useState("");
    const [lekcije, setLekcije] = useState([]);
    const [lekcija, setLekcija] = useState();
    const [sidebarRefresh, setSidebarRefresh] = useState(1);

    const handleNaziv = (event) => {
      setNaziv(event.target.value);
    };

    const handleLekcija = (event) => {
      setLekcija(event.target.value);
    };
    //...
}

export default CreateLecture;
```

Slika 18: Kreiranje i ažuriranje stanja

Slika 18 prikazuje dio izvornog koda iz komponente za kreiranje lekcije. Ova komponenta koristi četiri stanja (naziv, lekcije, lekcija, sidebarRefresh). Stanje „naziv“ ima stavljenu početnu vrijednost na prazan „string“ jer se očekuje da naziv sadržava niz znakova. Stanje „lekcije“ ima početnu vrijednost praznog niza jer se očekuje da će sadržavati niz lekcija. Stanje „lekcija“ nema definirano početno stanje. Stanje „sidebarRefresh“ ima početnu vrijednost stavljenu na „1“ jer će se kod ažuriranja taj broj povećavati. Iz primjera se mogu vidjeti funkcije „handleNaziv“ i „handleLekcija“, sadržavaju funkcije za ažuriranja stanja „setNaziv“ i „setLekcija“ koje su definirane kod kreiranja stanja.

7.3.7. Dinamično kreiranje komponenti

U React-u se za dinamičko kreiranje komponenti poput lista koristi JavaScript funkcija „map“. Funkcija „map“ poziva „callback“ (funkcija povratnog poziva) za svaki element u ulaznoj listi i kao izlaz vraća novu listu koja je rezultat te „callback“ funkcije.

Ova metoda se često koristi za kreiranje sređenih i nesređenih lista i rezultat koji korisnik vidi je identičan onome u HTML-u. Međutim u programskom kodu postoji značajna razlika. Naime, kod kreiranje lista React zahtjeva dodavanje ključa svakom elementu liste. Ključ je specijalan atribut čija vrijednost mora biti jedinstvena u odnosu na vrijednosti ostalih ključeva stavki. Ključ služi za identificiranje koji elementi liste su se promijenili (dodali ili obrisali). Slika 19 prikazuje primjer dinamično kreiranje popisa ispita.

```
//...
const popisIspita = () => {
    //...
  return (
    <div>
      {ispiti.map((stavka, index) => (
        <FormContainer key={stavka._id}>
          <Link
            to={`/tecaj/${id}/ispitiAdministracija/${stavka._id}`}
            key={stavka._id}
          >
            <Button name={`Ispit za ispraviti broj: ${index + 1}`} />
          </Link>
        </FormContainer>
      ))}
    </div>
  );
  //..
};
export default popisIspita;
```

Slika 19: Dinamično kreiranje komponenti

Slika 19 prikazuje korištenje funkcije „map“ koja prolazi kroz stanje „ispiti“. Funkcija „map“ u ovom slučaju ima dva argumenta (stavka i index). „Stavka“ se odnosi na jednu instancu ispita. „Index“ je ključna riječ i opcionalan je, u ovom slučaju se koristi za postavljanje broja ispita. Funkcija „map“ za svaki ispit kreira komponentu „FormContainer“ i obavezno daje ključ, za ključ se koristi „id“ ispita. Komponenta „FormContainer“ sadrži gumb koji ima naziv „Ispit za ispraviti broj:“ i broj ispita. Gumb ima dinamički kreiranu poveznicu na ispit.

7.3.8. Ugniježđene komponente i proizvoljne vrijednosti

U React-u je praksa kreirati manje komponente koje se koriste više puta kako ne bi trebalo pisati jedan te isti kod više puta. Umjesto toga dovoljno je pozvati tu jednu komponentu. Na slici 20 se mogu vidjeti ugniježđene komponente koje se pozivaju. Slika ispod prikazuje komponentu „Card“, ona sadrži druge ugniježđene komponente poput gumba „Button“.

```
import React from "react";
import styles from "./Card.module.scss";
import Button from "../../../components/Small/Button/Button";
import { Link } from "react-router-dom";
export const Card = (props) => {
  const auth = JSON.parse(sessionStorage.getItem("user"));
  return (
    <div className={styles.Container}>
      <h2 className={styles.h2}>{props.title}</h2>
      <p className={styles.Description}>{props.description}</p>
      <Link to={`/tecaj/${props.link}`}>
        <Button name={`Nauči ${props.title}`} />
      </Link>
      {auth !== null && (
        <Link to={`/tecaj/${props.link}/kviz`}>
          <Button name={`ISPIT ${props.title}`} />
        </Link>
      )}
      {auth !== null && auth.predaje === props.link && (
        <Link to={`/tecaj/${props.link}/ispitiAdministracija`}>
          <Button name={`ISPRAVLJANJE ISPITA`} />
        </Link>
      )}
    </div>
  );
};
```

Slika 20: Izvorni kod ugniježđene komponente „Card“

Ugniježđene komponente često primaju podatke od „roditelja“, odnosno komponenti unutar kojih se nalaze. Da bi se to omogućilo koristi se „props“ ili svojstva. Komponenta mora imati argument „props“, a zatim se kod korištenja koristi „props.NazivSvojstva“. Primjer se nalazi u kodu ispod:

```
<h2 className={styles.h2}>{props.title}</h2>
```

Naslov druge razine ima vrijednost svojstva „title“ koje je definirano kod dinamičnog kreiranja komponente „Card“. Svaka kartica ima svoj poseban naziv.

7.3.9. JSX

JSX je proširenje sintakse za JavaScript i može se koristiti kod izrade React aplikacija. Iako nije obavezno korištenje JSX-a, preporuča se zbog jednostavnijeg i preglednijeg pisanja koda. JSX doprinosi kraćem vremenu potrebnom za pisanje. Iako u ovim primjerima do sad nije spomenut, JSX je korišten, a najviše se može vidjeti u komponenti „Card“ (slika 20). Također se može vidjeti da se JavaScript kod u JSX-u piše unutar vitičastih zagrada. Glavna razlika između JSX-a od HTML-a je u tome što se koristi „camelCase“ način imenovanja. To znači da na primjer HTML atribut za klasu „class“ u JSX-u se piše kao „className“.

7.3.10. Flexbox

Kod izrade web aplikacije je korišten Flexbox. Flexbox je modul CSS-a koji olakšava izradu responzivnih stranica. Na slici ispod se nalazi primjer CSS koda za korištenog za komponentu „Card“.

```
.Container {
  display: flex;
  flex-direction: column;
  width: 100%;
  min-height: 33vh;
  background: $persian-green;
  padding: 20px;
  justify-content: center;
  align-items: center;
  color: $white;
  text-align: center;
  box-sizing: border-box;
}

.h2 {
  font-size: 2rem;
  width: 100%;
}

.Description {
  width: 100%;
  margin-top: 2rem;
  margin-bottom: 2rem;
}
```

Slika 21: Primjer Flexbox-a

Uz sam CSS, Flexbox dodaje nekoliko novih mogućnosti. Kako bi se mogle koristiti u pojedinoj klasi, u ovom slučaju klasi „Container“, prvo je potrebno staviti da se prikazuje kao „display: flex“. Nakon što je to postavljeno mogu se koristiti mogućnosti Flexbox-a. U ovom primjeru je smjer prikaza sadržaja definiran u smjeru stupca. To je postignuto linijom koda „flex-direction:column“. Dalje je sadržaj koji se nalazi unutar te klase poravnat horizontalno i vodoravno u sredinu korištenjem CSS uputa „align-items:center“ i „justify-content: center“.

7.3.11. Obrasci i obrnuti tok podataka

Obrasci (eng. *Form*) u React-u su zapravo proširenje HTML obrazaca, to jest naprednija verzija. Svaki element obrasca ima svoju vrijednost koja je spremljena u stanje i svaki put kad se vrijednost elementa promijeni, mijenja se i njegovo stanje. To se postiže korištenjem osluškivanjem događaja (eng. *Event listener*) i prilikom toga dolazi do obrnutog toka podataka, gdje stanje poprima vrijednost komponente

```
<div>
  <form>
    <textarea value={odgovor} onChange={handleJedanTocan}></textarea>
  </form>
  <Button name="Pošalji" onClick={handlePitanjeOdgovor} />
</div>
```

Slika 22: Primjer obrasca

U primjeru na slici iznad se nalazi element višelinijskog unosa (eng. *Textarea*) čija je vrijednost spremljena u stanje „odgovor“. Prilikom promjene vrijednosti izvršava se funkcija „handleJedanTocan“ koja mijenja odgovor u trenutnu vrijednost.

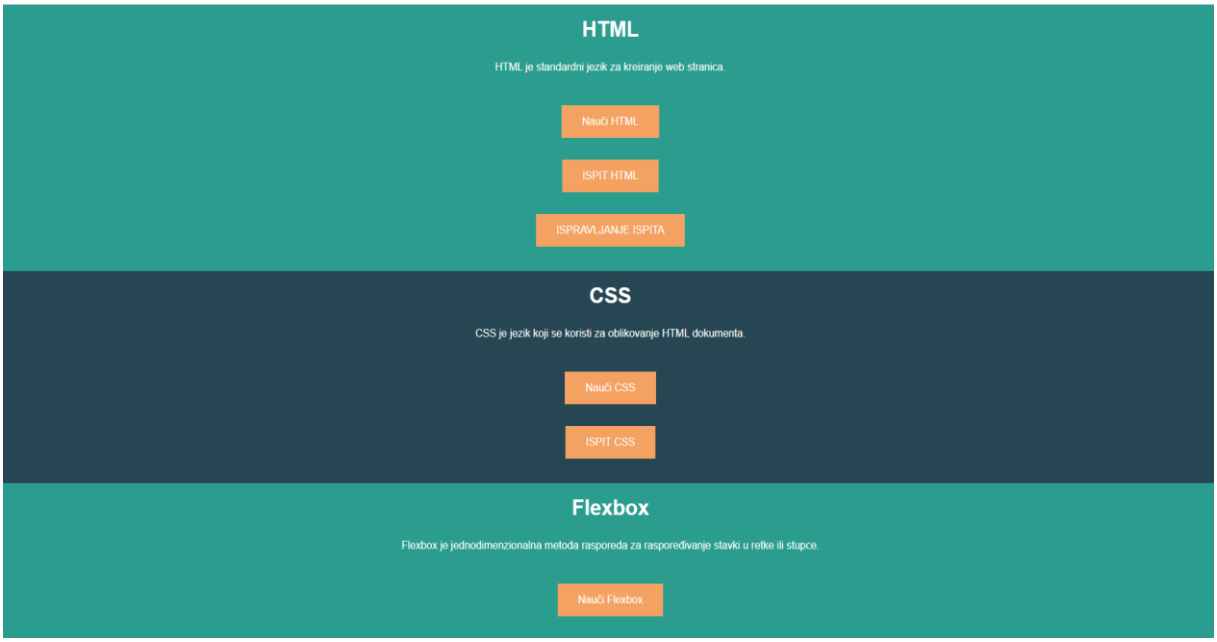
Klikom na gumb „Pošalji“ se izvršava funkcija „handlePitanjeOdgovor“, funkcija sprema trenutno pitanje i odgovor u niz. Zatim dohvaća sljedeće pitanje iz baze. Kad nema više pitanja za dohvatiti šalje niz s pitanjima i odgovorima u bazu kako bi se kasnije mogli ispraviti.

8. Korištenje aplikacije

Aplikacija „Kviz“ omogućava kreiranje tečajeva, lekcija unutar tih tečajeva i pitanja koja se nalaze u kvizu za taj tečaj. Postoje četiri uloge:

- Gost: neregistrirani korisnik koji može vidjeti tečajeve i njihove lekcije
- Student: može sve što i gost, a uz to može pristupiti i kvizu
- Predavač: može sve što i student, a uz to može za svoj tečaj dodavati lekcije i pitanja, ispravljati kvizove
- Administrator: može sve što i student, također može dodavati tečajeve i njihove predavače

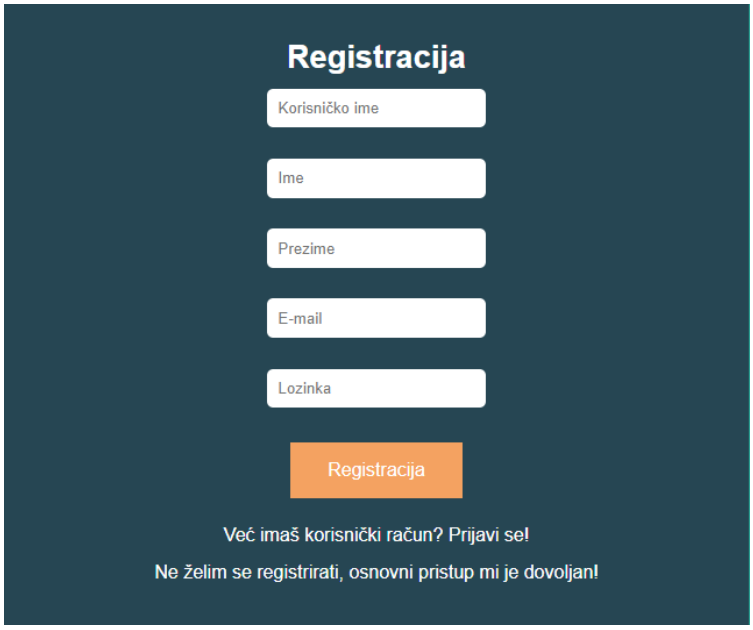
Početna stranica se sastoji od popisa tečajeva. Svaka kartica tečaja se prikazuje drugačije u odnosu na ulogu korisnika. Ako korisnik nije prijavljen prikazuje se samo poveznica na pogled za učenje. Ako je prijavljen postoji i gumb za pisanje „Kviza“. U slučaju da je korisnik predavač, kod njegovog tečaja se prikazuje i gumb za ispravljanje kvizova.



Slika 23: Početna stranica

Stranica tečaja s lijeve strane ima izbornik s lekcijama tog tečaja, a s desne strane sadržaj trenutno otvorene lekcije. Ako korisnik ima ulogu predavača ima mogućnost uređivanja te lekcije, dodavanje nove lekcije ili pitanja, a ostali korisnici imaju mogućnost samo pregleda.

U web aplikaciji je dodana registracija i prijava korisnika. Kod registracije korisnik upisuje podatke koji su potrebni.

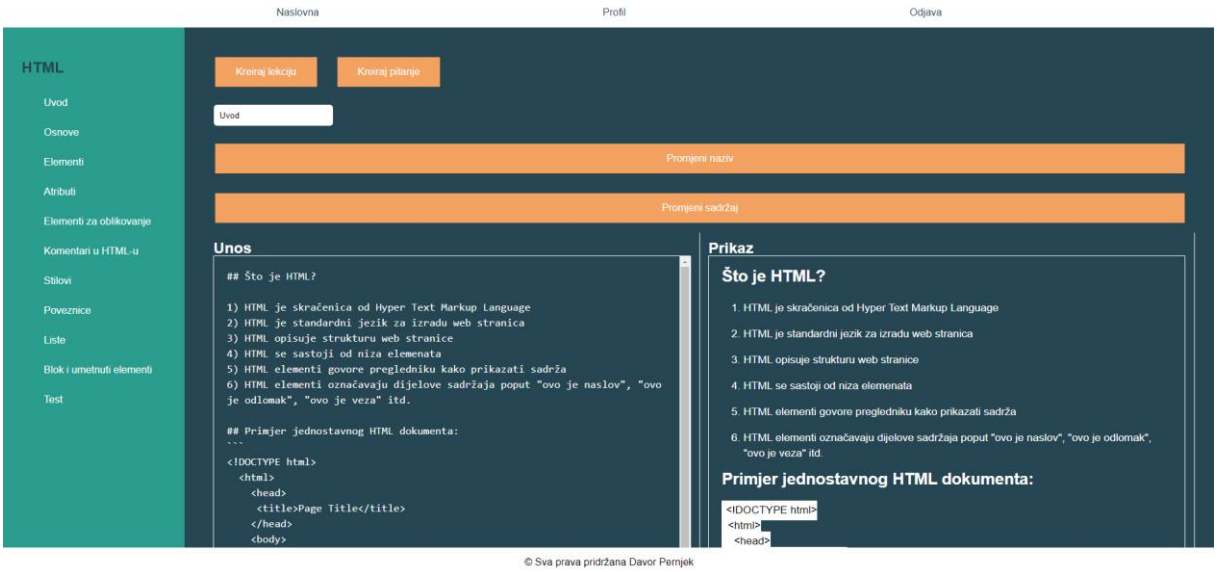


Slika 24: Registracija korisnika

Za registraciju je potrebno unijeti željeno korisničko ime, pravo ime i prezime, email adresu i lozinku (slika 24). Također su dodane poveznice na prijavu ako korisnik već ima kreiran račun i želi se prijaviti te poveznica na naslovnu stranicu ako se korisnik ne želi registrirati.

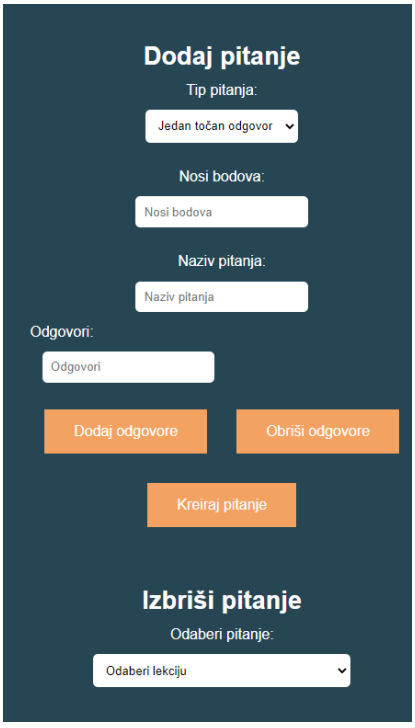
Slika 25: Prijava korisnika

Na slici 25 se može vidjeti da je za prijavu potrebno unijeti samo korisničko ime i lozinku. Također je dodana poveznica na registraciju ukoliko korisnik nema kreiran račun te poveznica na naslovnu stranicu ako se ne želi registrirati.

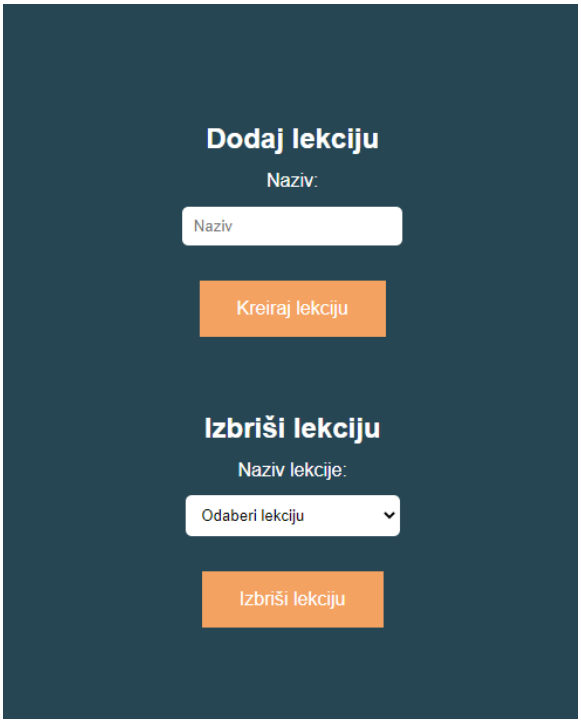


Slika 26: Sučelje uloge predavač

Prilikom dodavanja pitanja predavač odabire vrstu pitanja iz padajućeg izbornika, zatim koliko bodova nosi pitanje, naziv pitanja i opcionalno može ponuditi odgovore ovisno o tipu pitanja. Ispod se nalazi obrazac s padajućim izbornikom za brisanje pitanja. Za dodavanje lekcije se upiše naslov i klikom na gumb se preusmjerava na kreiranu lekciju gdje se zatim upisuje sadržaj. Brisanje lekcije se radi isto kao i kod brisanja pitanja.



Slika 27: Obrasci za dodavanje/brisanje pitanja



Slika 28: Obrasci za dodavanje/brisanje lekcije

Ako predavač na početnoj stranici klikne na ispravljanje ispita, prikažu mu se svi neispravljeni ispiti. Klikom na ispit počinje ispravljanje. Ako je ispit prolazan korisniku se na profilu pod certifikate pokaže naziv tečaja. Korisnik može imati više certifikata istog tečaja. Korisnik dobiva certifikat ako je predavač označio kviz prolaznim.

Slika 29: Ispravljanje kviza

Stranica s profilom korisnika prikazuje njegovo ime i prezime, te certifikate koje je do sada dobio.

Slika 30: Korisnički profil

Administrator može kreirati tečaj, prilikom kreiranja mora odabrati predavača. Predavač može biti samo korisnik koji do sad ima ulogu studenta. Zatim se odabire naziv tečaja i kratak opis koji se prikazuju na početnoj stranici.

Slika 31: Obrazac za dodavanje tečaja

Svaki tečaj ima kviz koji se sastoji od svih kreiranih pitanja za taj tečaj. Pitanje se sastoji od naziva, ponuđenih odgovora (ako tip pitanja to zahtjeva) i gumba za slanje.

Slika 32: Obrazac za predaju pitanja

Svi registrirani korisnici mogu pristupiti kvizu. Na početnoj stranici (slika 23) klikom na gumb „ISPIT“ se otvara kviz. Kviz se sastoji od niza pitanja svih kreiranih pitanja. Pitanja se učitavaju pojedinačno, svako na svojoj stranici. Odgovaranjem na pitanje se učitava sljedeće i nije moguće vratiti se na prethodno. Na posljednjem pitanju korisnik dobiva obavijest da je kviz predan.

9. Zaključak

JavaScript programski okviri olakšavaju programerima izradu web aplikacija, omogućuju brže i jednostavnije korištenje JavaScript-a. Svaki programski okvir ima svoje specifičnosti koje ga razlikuju od drugih. Neki su namijenjeni za veće, aplikacije, neki za manje, dok neki omogućuju fleksibilnost kod odabira pomoćnih alata i imaju već definirane alate koje se moraju koristiti. Svaki programski okvir nije idealan za svaki projekt zbog mogućnosti skaliranja. Zbog tog razloga se danas programeri fokusiraju na izradu okvira koji se mogu skalirati na velike aplikacije, a da pritom zadrže što veću brzinu izrade aplikacije, ali i izvršavanja koda.

Web aplikacije se često izrađuju prema MVC uzorku. Podijeljene su na tri dijela (model, pogled i upravljač) od kojih svaki ima svoju ulogu. Programski kod današnjih web aplikacija se piše pomoću dodatnih biblioteka od kojih je najpopularniji TypeScript. Prilikom izrade praktičnog djela korišten je skup alata MERN. Cijeli skup MERN alata se izvodi na NodeJS okruženju. Tijekom izrade aplikacije bilo je potrebno upoznati se s mnoštvo novih tehnologija koje su danas tražene od programera, ali i s raznim izazovima i problemima tijekom programiranje.

Izrađena je aplikacija koja obavlja operacije kreiranja, čitanja, ažuriranja i brisanja nad podacima u nerelacijskoj bazi podataka MongoDB. Web aplikacija „Kviz“ ima svrhu provjeravanja znanja korisnika. Aplikacija „Kviz“ sadrži tečajeve sa sadržajem za učenje. Korisnici mogu provjeravati svoje znanje o pojedinom tečaju pisanjem kviza. Funkcionalnostima se pristupa pomoću ExpressJS-a. Korisnik ih koristi putem korisničkog sučelja kreiranog u React-u.

React je trenutno najpopularniji JavaScript programski okvir za korisnički dio aplikacije. React-a se preporučuje ne samo zbog potražnje na tržištu rada, već i zbog brze krivulje učenja koncepata u React-u. Korisni koncepti u React-u su slični i u drugim programskim okvirima pa je iz tog razloga lakše naučiti i druge programske okvire. Poznavanje React-a omogućava jednostavniju prilagodbu novim programskim okvirima i zahtjevima poslodavaca.

Popis literature

Aggarwal, S. (2018). Modern Web-Development using ReactJS. U *International Journal of Recent Research Aspects* (Sv. 5).

angular/angular: One framework. Mobile & desktop. (bez dat.). Preuzeto 03. srpanj 2020., od https://github.com/angular/angular

Angular - Introduction to Angular concepts. (bez dat.). Preuzeto 28. srpanj 2020., od https://angular.io/guide/architecture

Angular - Two-way binding [(...)]. (bez dat.). Preuzeto 28. srpanj 2020., od https://angular.io/guide/two-way-binding

Bierman, G., Abadi, M., & Torgersen, M. (2014). Understanding TypeScript. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, *8586 LNCS*, 257–281. Preuzeto 12. rujan 2020., od https://doi.org/10.1007/978-3-662-44202-9_11

Comparison with Other Frameworks — Vue.js. (bez dat.). Preuzeto 30. srpanj 2020., od https://vuejs.org/v2/guide/comparison.html

Components - vue.js. (bez dat.). Preuzeto 28. srpanj 2020., od https://v1.vuejs.org/guide/components.html#Prop-Binding-Types

Components and Props – React. (bez dat.). Preuzeto 12. lipanj 2020., od https://reactjs.org/docs/components-and-props.html

Create a New React App – React. (bez dat.). Preuzeto 06. travanj 2020., od https://reactjs.org/docs/create-a-new-react-app.html

Data binding overview - WPF | Microsoft Docs. (bez dat.). Preuzeto 28. srpanj 2020., od https://docs.microsoft.com/en-us/dotnet/desktop-wpf/data/data-binding-overview?redirectedfrom=MSDN

facebook/react: A declarative, efficient, and flexible JavaScript library for building user interfaces. (bez dat.). Preuzeto 03. srpanj 2020., od https://github.com/facebook/react/

Fowler, M. (2010). *GUI Architectures*. https://martinfowler.com/eaaDev/uiArchs.html#ModelViewController

How to Model MVC Framework with UML Sequence Diagram? (bez dat.). Preuzeto 19. rujan 2021., od https://www.visual-paradigm.com/guide/uml-unified-modeling-language/how-to-model-mvc-with-uml-sequence-diagram/

Jasmine Documentation. (bez dat.). Preuzeto 11. rujan 2021., od https://jasmine.github.io/index.html

Jest · Delightful JavaScript Testing. (bez dat.). Preuzeto 03. rujan 2020., od https://jestjs.io/

JSX - What Is a JSX? & Introduction to Advanced. (bez dat.). Preuzeto 27. srpanj 2020., od https://www.reactenlightenment.com/react-jsx/5.1.html

Krajka, B. (2015). *The difference between Virtual DOM and DOM - React Kung Fu*. Preuzeto 03. srpanj 2020., od https://reactkungfu.com/2015/10/the-difference-between-virtual-dom-and-dom/

Learning TypeScript - Remo H. Jansen - Google Knjige. (bez dat.). Preuzeto 12. rujan 2020., od https://books.google.hr/books?hl=hr&lr=&id=odVOCwAAQBAJ&oi=fnd&pg=PP1&dq=typescript+ecmascript&ots=zqk2YJ3yeZ&sig=JvdJn_SQD-fDLuQ5TlLSiUUlJSA&redir_esc=y#v=onepage&q=ecmascript&f=false

Props — Vue.js. (bez dat.). Preuzeto 31. srpanj 2020., od https://vuejs.org/v2/guide/components-props.html

React.js Essentials - Artemij Fedosejev - Knihy Google. (bez dat.). Preuzeto 06. travanj 2020., od https://books.google.cz/books?hl=cs&lr=&id=Rhl1CgAAQBAJ&oi=fnd&pg=PP1&dq=reactJS&ots=JjutuEyVUI&sig=VWe1EHPUMNIwR1Fj42uU18amBEM&redir_esc=y#v=onepage&q=reactJS&f=false

React – A JavaScript library for building user interfaces. (bez dat.). Preuzeto 06. travanj 2020., od https://reactjs.org/

Robie, J., Research, T., Sutor, R., & Wilson, C. (bez dat.). *Document Object Model (DOM) Level 1 Specification (Second Edition)*. Preuzeto 30. srpanj 2020., od http://www.w3.org/TR/1998/REC-DOM-Level-1-19981001

Sequence Diagram Tutorial. (bez dat.). Preuzeto 19. rujan 2021., od https://online.visual-paradigm.com/diagrams/tutorials/sequence-diagram-tutorial/

Static Type Checking – React. (bez dat.). Preuzeto 14. rujan 2021., od https://reactjs.org/docs/static-type-checking.html#gatsby-focus-wrapper

Tania, R. (2018). *Getting Started with React - An Overview and Walkthrough Tutorial*. Preuzeto 06. travanj 2020., https://www.taniarascia.com/getting-started-with-react/

TypeScript in 5 minutes · TypeScript. (bez dat.). Preuzeto 31. srpanj 2020., od https://www.typescriptlang.org/docs/handbook/typescript-in-5-minutes.html

vuejs/vue: Vue.js is a progressive, incrementally-adoptable JavaScript framework for building UI on the web. (bez dat.). Preuzeto 03. srpanj 2020., od https://github.com/vuejs/vue

Vynogradenko, A. (2018). *Sizes of JS frameworks, just minified + minified and gzipped, (React, Angular 2, Vue, Ember)*. Preuzeto 03. rujan 2020., od https://gist.github.com/Restuta/cda69e50a853aa64912d

What is Digest Cycle in AngularJS & How Does It Work? (bez dat.). Preuzeto 30. srpanj 2020., od https://www.dotsquares.com/news_events/what-is-digest-cycle-in-angular-js-and-how-does-it-make-two-way-data-binding-possible/

What is npm. (bez dat.). Preuzeto 27. srpanj 2020., od https://www.w3schools.com/whatis/whatis_npm.asp

Your_first_suite. (bez dat.). Preuzeto 11. rujan 2021., od https://jasmine.github.io/tutorials/your_first_suite

Popis slika

Popis tablica