Razvoj web aplikacije u React i Node.js okruženjima

Tvrdinić, Karlo

Sveučilište Jurja Dobrile u Puli

Fakultet informatike u Puli

Karlo Tvrdinić

RAZVOJ WEB APLIKACIJE U REACT I NODE.JS OKRUŽENJIMA

Diplomski rad

Pula, svibanj 2020.

Sveučilište Jurja Dobrile u Puli

Fakultet informatike u Puli

Karlo Tvrdinić

RAZVOJ WEB APLIKACIJE U REACT I NODE.JS OKRUŽENJIMA

Diplomski rad

JMBAG: 0303054334, redovni student

Studijski smjer: Informatika

Predmet: Napredni algoritmi i strukture podataka

Znanstveno područje: Društvene znanosti

Znanstveno polje: Informacijske i komunikacijske znanosti

Znanstvena grana: Informacijski sustavi i informatologija

Mentor: izv. prof. dr. sc. Tihomir Orehovački

Pula, svibanj 2020.

IZJAVA O AKADEMSKOJ ČESTITOSTI

Ja, dolje potpisani **Karlo Tvrdinić**, kandidat za **magistra informatike** ovime izjavljujem da je ovaj Diplomski rad rezultat isključivo mojega vlastitog rada, da se temelji na mojim istraživanjima te da se oslanja na objavljenu literaturu kao što to pokazuju korištene bilješke i bibliografija. Izjavljujem da niti jedan dio Završnog rada nije napisan na nedozvoljen način, odnosno da je prepisan iz kojega necitiranog rada, te da ikoji dio rada krši bilo čija autorska prava. Izjavljujem, također, da nijedan dio rada nije iskorišten za koji drugi rad pri bilo kojoj drugoj visokoškolskoj, znanstvenoj ili radnoj ustanovi.

Student

_____

U Puli, 10.05.2020 godine

IZJAVA

o korištenju autorskog djela

Ja, **Karlo Tvrdinić** dajem odobrenje Sveučilištu Jurja Dobrile u Puli, kao nositelju prava iskorištavanja, da moj završni rad pod nazivom **razvoj Web aplikacije u React i Node.js okruženjima** koristi na način da gore navedeno autorsko djelo, kao cjeloviti tekst trajno objavi u javnoj internetskoj bazi Sveučilišne knjižnice Sveučilišta Jurja Dobrile u Puli te kopira u javnu internetsku bazu završnih radova Nacionalne i sveučilišne knjižnice (stavljanje na raspolaganje javnosti), sve u skladu s Zakonom o autorskom pravu i drugim srodnim pravima i dobrom akademskom praksom, a radi promicanja otvorenoga, slobodnoga pristupa znanstvenim informacijama.

Za korištenje autorskog djela na gore navedeni način ne potražujem naknadu.

U Puli, 10.05.2020.

Potpis

_____

RAZVOJ WEB APLIKACIJE U REACT I NODE.JS OKRUŽENJIMA

Karlo Tvrdinić

Sažetak: U svrhu diplomskog rada, implementiran je projekt koristeći Node.js i React razvojna okruženja u kojem je ilustrirana njihova moć. React.js je JavaScript biblioteka koja se koristi za korisničko sučelje, dok se Node.js koristi za programiranje na poslužiteljskoj strani. Kroz projektni zadatak prikazana je i biblioteka Redux koja ima široko korištenje radi lakšeg korištenja podataka. Redux se koristi za lokalno pohranjivanje, dok je baza podataka MongoDB. Na početku diplomskog rada nalazi se teorijski dio, dok se na kraju rada nalazi praktični dio te link repozitorija projektnog zadatka. U projektnom zadatku prikazana je jednostavnost korištenja Reacta i Node.js-a. U prvom je dijelu rada objašnjen React koji je JavaScript biblioteka, dok je u drugome dijelu rada objašnjen Node.js koji je JavaScript razvojno okruženje.

Ključne riječi: Web aplikacija, JavaScript, Node.js, React, SPA, virtualni DOM.

APPLICATION DEVELOPMENT IN REACT AND NODE.JS

Karlo Tvrdinić

Abstract: For the purpose of the thesis, a project was implemented using Node.js and React development environment where their power is illustrated. React.js is a JavaScript library used for front-end, while Node.js is used for back-end. The project also features a widely used Redux library for easy data use. Redux is used for local storage, while the database is MongoDB. In the beginning of the thesis there is a theoretical part, while at the end of the thesis there is a practical part and a link of the project repository. The terms of reference show the ease of use of React and Node.js. The first part of the paper explains the React, a JavaScript library. The second part of the paper deals with Node.js, a JavaScript developing environment.

Keywords: Web application, JavaScript, Node.js, React, SPA, virtual DOM.

Sadržaj

Popis slika

1. Uvod

Sve se više pokazuje interes za razvojna okruženja i biblioteke koje pripadaju JavaScript jeziku. Glavni su razlozi jednostavnost izrade *web*-stranice i sve veća podrška za pakete koji se koriste u JavaScriptu. Objavljuje se sve više knjiga i mrežnih resursa koji pokrivaju osnove Node.js-a i Reacta. Ljudi koji tek uče programirati, s minimalnim trudom mogu napraviti modernu *web*-stranicu pomoću JavaScripta jer JavaScript biblioteke i razvojna okruženja poput Node.js-a i Reacta imaju veliku podršku. U diplomskom se radu većim dijelom programiranje oslanja na osnovne module i pakete.

Node.js idealan je za početnike kao i za programere koji se godinama bave programiranjem. Jednostavan je te ne zahtijeva puno učenja jer sadrži veliku podršku gotovih paketa koji se jednostavno koriste. Kroz paket menadžer (engl. *npm – Node Packet Manager*) moguće je jednostavno instalirati paket koji su napravili drugi programeri te se jednostavno koristi pozivom funkcije ili klase, što pisanje dodatnih kôdova i proučavanje novih informacija svodi na minimum. Stoga je potrebno pokazati kako se Node.js i React koriste u industriji. Osnovni su koncepti obuhvaćeni u diplomskom radu. Node.js razvojno je okruženje u kojem se može koristiti mnogo tehnologija kao što su Express.js, Hapi.js, Mongoose, Amazon Web Servisi te mnoge druge ozbiljne tehnologije koje se koriste za velike projekte.

React je JavaScript biblioteka otvorenog kôda koju je kreirao Facebook te se koristi za izradu *web*-aplikacija. Razvojem *weba* pojavila se potreba za aplikacije s jednom stranicom (engl. SPA - *Single Page Application*) unazad nekoliko godina. Prijašnji razvoji *web*-aplikacija bili su jednostavni. Ako je korisnik promijenio određeni element u stranici, cijela se stranica ponovno učitala te su se tako izvršile promjene koje su se dogodile unutar stranice. Cijela se stranica ne treba učitavati, već je React djelovao na onaj dio koji se promijenio. Problem se nalazi u ponovnom učitavanju stranice jer treba proći određeno vrijeme povratka za kompletan zahtjev koji je klijent poslao poslužitelju i poslužitelj natrag klijentu. Također postoje i druge JavaScript biblioteke koje rješavaju taj problem, a jedna je od njih AJAX koja se koristi za *web*-aplikacije s jednom stranicom, u kojoj se ne mora učitavati cijela stranica, nego samo dijelovi koji se izmjenjuju.

React aplikacija sastoji se od više komponenti, od kojih je svaka odgovorna za stvaranje dijela HTML-a za višekratnu upotrebu. Komponente se mogu ugnijezditi u druge komponente kako bi se omogućilo da se složene aplikacije izrade iz jednostavnih građevnih blokova. React koristi virtualne DOM objekte koje *web*-čitač pruža JavaScriptu. Zbog korištenja virtualnih DOM objekata *web*-stranica se ne mora cijela učitavati pri izmjeni jednog od objekata, nego se određeni dio zamijeni (Young et al., n.d.).

U diplomskom radu napravljen je projekt koji prikazuje prednosti korištenja Reacta i Node.js-a u kombinaciji s MongoDB bazom podataka. Prikazuje se korak po korak jednostavnog koncepta u složenoj aplikaciji. Projekt može poslužiti i za izradu vlastitih projekata, s obzirom na to da sadržava osnovne koncepte korištenja Reacta i Node.js-a. Primjeri koji se nalaze u projektu prikazuju najbolju praksu u industriji programiranja kako bi se izbjegle neželjene greške. Node.js u kombinaciji s Reactom je fokusiran na to da programer u kratkom vremenskom periodu i s malo napora napravi veliki projekt. Paketi koji se nalaze i koriste u projektnom dijelu diplomskog rada možda su s vremenom zastarjeli.

Tema projektnog dijela je slanje pošiljke po korisniku, odnosno kuriru. Kada se korisnik jednom registrira, može odabrati slanje paketa po kuriru ili pružati usluge kao kurir. Jednom kada kurir postavi svoju rutu, ostali korisnici mogu vidjeti njegovu rutu te se pretplatiti na nju i poslati paket uz dogovorenu cijenu s kurirom.

Kroz projekt su se koristile različite tehnologije, a to su:

- Izrada *web*-aplikacije pomoću Express.js-a, MongoDB-a, Reacta
- Korištenje paketa Reacta kao što je Reactstrap
- Manipuliranje podataka u bazi podataka MongoDB
- Korištenje paketa Mongoose
- Kreiranje aplikacijskog programskog sučelja (engl. REST API)
- Korištenje tokena JWT (engl. JSON Web Token) za autentifikaciju

2. REACT

2.1. Uvod

React je biblioteka otvorenog kôda koja se temelji na lakšem dizajniranju sučelja. Odgovoran je samo za prikaz slojeva u aplikaciji. Napravio ga je Facebook te su ga u početku napravili za svoje potrebe kako bi lakše održavali Facebook, no danas se počeo koristiti gotovo za svaku *web*-aplikaciju zbog jednostavnosti korištenja. React koristi virtualni DOM zbog kojeg i ima prednosti nad ostalim bibliotekama i jezicima jer se stranica ne mora ponovno učitavati kada se određeni objekti izmijene, nego React zna koji element treba zamijeniti, dok ostatak stranice ostaje nepromijenjen.

U biblioteci React omogućeno je pisanje komponenti korištenjem specifičnog jezika koji se zove JSX. JSX omogućava pisanje kôda kao što je HTML, no može se kombinirati s JavaScriptom. React će pretvoriti kôd u virtualni DOM te na kraju isporučiti HTML za korisnika.

Naziv biblioteke React na hrvatskom znači reakcija, a tu riječ biblioteka i opravdava. React brzo reagira na promjene stanja u komponentama te ponovno pokazuje komponente unutar HTML DOM-a koristeći virtualni DOM. Virtualni DOM predstavlja memorijski prikaz stvarnog DOM-a. Provodeći većinu obrade unutar virtualnog DOM-a umjesto izravno u DOM pregledniku, React može djelovati brzo te jednostavno dodati, ažurirati i ukloniti komponente od zadnjeg renderiranja koje se dogodilo.

2.2. Budućnost Reacta

Programeri su svjesni da nisu *web*-preglednici jedini na kojima se mogu pokrenuti React aplikacije. React Native koji je objavljen 2015. godine omogućuje da se iskoriste znanja i prednosti React aplikacija u IOS i Android uređajima. Također postoji i React VR koji je razvojno okruženje za izradu interaktivnih aplikacija odnosno VR aplikacija. Biblioteka React kroz pakete omogućuje brzo razvijanje iskustava te prilagođavanje na niz različitih veličina i vrsta zaslona (Banks i Porcello, 2017).

React je i dalje novo područje. Buduće verzije Reacta uključivat će Fiber koji ima funkciju ponovne implementacije React algoritma kojem je fokus povećanje brzine renderiranja. Nema saznanja o tome kako će budućnost Reacta utjecati na programere, ali će zasigurno utjecati na brzinu kojom se aplikacija prikazuje i ažurira. Mnoge će buduće promjene utjecati na određene uređaje.

2.3. Zašto React?

Web-aplikacije sadrže mogućnost programiranja na poslužiteljskoj i klijentskoj strani. Klijentsku stranu predstavlja korisnik koji koristi *web*-aplikaciju te šalje upite poslužiteljskoj strani koja vraća određene podatke. Programiranje na poslužiteljskoj strani sastoji se od poslužitelja, aplikacije i baze podataka. Na poslužitelju programer kôdira skripte koje služe za dohvat podataka iz baze, te nakon toga skripte na poslužiteljskoj strani šalju podatke klijentskoj strani.

Web-aplikacija koja sadrži početnu stranicu te implementaciju za autentifikaciju korisnika u Reactu ne treba sadržavati više stranica. React aplikacije koriste poglede (engl. *Views*) koji se samo izmjenjuju kada korisnik pritisne na drugu stranicu. React kreira aplikacije koje sadrže samo jednu stranicu (engl. *Single Page Application*). Aplikacija s jednom stranicom u sebi ima nekoliko pogleda koji se izmjenjuju kada korisnik želi izmijeniti stranicu, odnosno kada pritisne na navigacijski bar. Iz tog su razloga takve *web*-aplikacije učinkovitije i brže jer korištenjem aplikacija s jednom stranicom, odnosno

Reactom, preglednik ne mora učitavati ponovno cijelu stranicu, već se samo izmijeni pogled te zamijeni sa starim.

Programeri iz Facebooka u početku su izgradili React za vlastite potrebe, a kasnije su ga i javno objavili. Mnogi se pitaju zašto su morali graditi novu biblioteku pod nazivom React kada danas ima mnoštvo drugih biblioteka.

React nije implementiran u Facebook aplikaciji za društvene mreže, već u Facebookovoj organizaciji za oglašavanje. U početku se koristio tipični model-pogled-kontroler (engl. *Model-view-controller*) na klijentskoj strani, koji je imao sve uobičajene dvosmjerne veze i predloške podataka. Pogledi su pretplaćeni i prate promjene na modelima, ali kada bi se dogodila promjena, određeni pogled bi se ažurirao. Problem je u tome što se ažuriranje jednog od elemenata, održava na druge elemente. Takve kaskadne nadogradnje postale su složene.

Tada su se programeri Facebooka zapitali zašto se trebaju baviti time, kada već postoji cijeli kôd za prikazivanje modela u pogledu. Postojalo je puno repliciranja kôda dodavanjem malih isječaka za upravljanje transakcijama. Dosjetili su se da mogu sami koristiti predloške za upravljanje promjenama stanja. To je glavni razlog zašto su počeli razmišljati o izgradnji nečeg deklarativnog, a ne imperativnog (Subramanin, 2017).

Poslužiteljska strana i klijentska strana čine jednu cjelinu, a to je *web*-aplikacija. Klijentska strana preglednika sadrži JavaScript koji se manifestira kroz korisničko sučelje. Također JavaScript pretvara podatke s poslužiteljske strane te ih isporučuje u obliku HTML-a koji se prikazuje krajnjem korisniku.

2.4. React Native

React Native popularno je razvojno okruženje koje omogućuje programerima da izrade robusne mobilne aplikacije koristeći svoje postojeće znanje JavaScripta. Pruža brži razvoj mobilnih aplikacija i učinkovitu razmjenu kôdova na IOS-u, Androidu i Webu bez gubitka kvalitete aplikacije. React Native još je uvijek novi pojam i trenutno je u razvoju. Ako se razvojni tim može nositi s novim tehnologijama i učenju novih stvari za više od jedne platforme, tada je React Native odlična biblioteka za programiranje i korištenje.

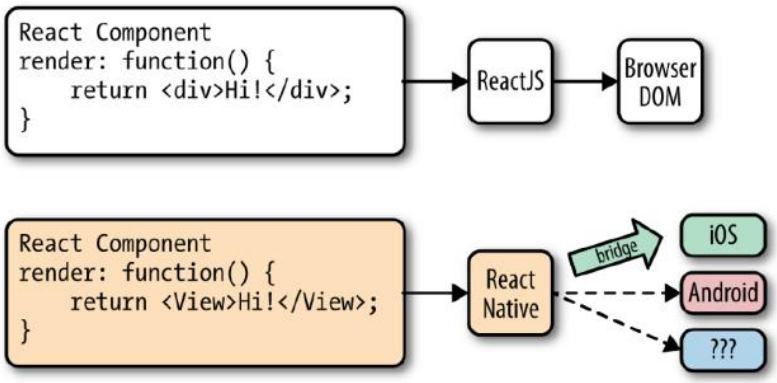
React Native razvojno je okruženje koji se koristi za izradu mobilnih aplikacija na Android i IOS uređajima, dok se React koristi za izradu *web*-aplikacija. Temelji se na React biblioteci koju je kreirao Facebook za izradu korisničkog sučelja. *Web*-programeri sada mogu pisati kôdove za mobilne aplikacije, primjenjujući znanje koje imaju iz Reacta u React Native. Sintaksa je vrlo slična, stoga nije složeno programirati *web* i mobilne aplikacije zajedno. Velika je prednost React Nativea u odnosu na druge jezike što se mogu koristiti istovremeno za dvije platforme kao što su Android i IOS.

React Native, slično kao i React, koristi kombinaciju JavaScripta i XML-a koji je poznat kao JSX jezik. Za izradu Android aplikacija koristi se Java, dok se za izradu IOS aplikacija koristi Objective-C. Aplikacija neće koristiti *web*-preglednike za prikazivanje komponenti, nego će renderirati stvarne UI komponente za mobitel što će korisniku pružiti osjećaj prave mobilne aplikacije. Također moguće je koristiti značajke na mobitelu kao što su kamera, korisnikova lokacija, slike, itd.

2.4.1. Kako funkcionira React Native

Ideja pisanja mobilnih aplikacija u jeziku JavaScript djeluje zbunjujuće. Mnogi se pitaju kako je moguće koristiti React u mobilnom okruženju. Kako bi se razumjele tehničke osnove React Nativea moraju se poznavati značajke Reacta, a to čini Virtualni DOM.

U kontekstu Reacta na *webu*, većina programera smatra virtualni DOM prvenstveno kao optimizaciju performansi. Pomoću korištenja virtualnog DOM-a, aplikacija ima bolje performanse, ali njegov stvarni potencijal leži u snazi apstrakcije. Postavljanje čistog sloja apstrakcije između programerskog kôda i stvarnog prikazivanja, otvara mnoštvo mogućnosti. Funkcioniranje React Nativea prikazano je na slici 1. Umjesto prikazivanja DOM preglednika, React Native poziva Objective-C aplikacijsko programsko sučelje za prikazivanje na IOS komponentama ili Java aplikacijsko programsko sučelja za prikazivanje na Android komponentama. Time se React Native izdvaja od ostalih programskih rješenja za razvoj više platformi koje često završavaju prikazivanjem *web*-baziranih prikaza (Risenman, 2016).



Slika 1. Izvođenje React komponente

2.5. Reactstrap

Reactstrap nudi unaprijed ugrađene Bootstrap verzije 4, odnosno komponente koje omogućuju veliku fleksibilnost. Omogućuje brzu izradu formi koje izgledaju lijepo i moderno te osiguravaju i pružaju intuitivno korisničko iskustvo. Reactstrap omogućava programeru unaprijed pripremljene komponente koje koristi na jednostavan način. Moguće je koristiti razne komponente poput korisničkih ulaza, oblikovanje ikona, fonta teksta, itd.

Korištenje je vrlo jednostavno. Programer mora na početku programskog kôda umetnuti kontejner 'reactstrap', odnosno u datoteku u kojoj će se koristiti. Slika 2 prikazuje umetanje kontejnera te umetanje ostalih elemenata koji su unutar Reactstrapa. Kao što se vidi iz slike 2 umetnuti su elementi koji služe za ljepši prikaz *web*-stranice te ostali elementi poput forme, oznaka, korisničkih ulaza te dugmeta. Kada programer umetne na početku datoteke Reactstrap, jednostavnim tagovima pozove element u programskom kôdu koji je već moderno dizajniran.

```
import {
  Collapse, Navbar,
  NavbarToggler, NavbarBrand,
  Nav, NavItem, NavLink,
  UncontrolledDropdown,
  DropdownToggle, DropdownMenu,
  DropdownItem
} from "reactstrap";
```

Slika 2. Umetanje Reactstrap

U projektnom dijelu Reactstrap se koristio za izradu navigacijskog izbornika, kako bi navigacijski bar lijepo izgledao korisnicima. Slika 3 prikazuje kôd koji se koristi za dizajniranje navigacijskog izbornika.

```
<Navbar color="light" light expand="md">
  <NavbarBrand href="/">SendPacket</NavbarBrand>
  <NavbarToggler onClick={toggle} />
  <Collapse isOpen={isOpen} navbar>
    <Nav className="mr-auto" navbar>
      <NavItem>
        <NavLink href="/about">O nama</NavLink>
      </NavItem>

      <UncontrolledDropdown nav inNavbar>
        {props.isAuthenticated ? (
          <DropdownToggle nav caret>
            Akcije
          </DropdownToggle>
        ) : null}

        <DropdownMenu right>
          <DropdownItem>
            <NavLink href="/offer">Ponudi prijevoz</NavLink>
          </DropdownItem>
          <DropdownItem divider />
          <DropdownItem>
            <NavLink href="/search">Pošalji paket</NavLink>
          </DropdownItem>
        </DropdownMenu>
      </UncontrolledDropdown>
    </Nav>
    {props.isAuthenticated ? userNavigation(props) : guessNavigation()}
  </Collapse>
</Navbar>
```

Slika 3. Korištenje biblioteke Reactstrap

Iz slike 3 se mogu vidjeti tagovi koji se koriste za Reactstrap, a ti tagovi su umetnuti kôdom koji je prikazan na slici 2. Glavni tag za navigacijski izbornik je Navbar, dok svi ostali elementi idu u tijelo Navbara. NavbarBrand tag služi za isticanje logotipa aplikacije, dok NavItem služi za tekstualni prikaz pojedine kartice. Isto tako postoji Dropdown tag koji služi za dodatno otvaranje pojedinih kartica kada korisnik pritisne na određeni gumb, te se na taj način provjerava je li korisnik prijavljen na stranicu i ovisno o prijavi prikazuju se određene kartice koje su zaštićene. Tag NavLink sadrži atribut *href*, koji služi za referenciranje pojedine kartice na određenu stranicu. Ako se ne koristi biblioteka

Reactstrap nego Bootstrap, onda se moraju koristiti klasni atributi te iste tagove stavljati kao atribut klase. Reactstrap je prilagođena verzija Bootstrapa za korištenje u Reactu.

2.6. Virtualni DOM

Tradicionalno, *web*-stranice sastoje se od neovisnih HTML stranica. Kada bi korisnik pregledavao te stranice, preglednik bi zahtijevao različite HTML dokumente te ih učitavao. Biblioteka AJAX je stvorila aplikacije s jednom stranicom (engl. SPA – *Single Page Application*). Budući da preglednici mogu učitati bitove podataka pomoću Reacta, cijela *web*-stranica može nestati i pouzdati se u JavaScript kako bi ažurirao novo korisničko sučelje bez izmjene stranica. U aplikacijama s jednom stranicom, preglednik u početku učitava jedan HTML dokument, a dok korisnik koristi aplikaciju, on zapravo ostaje na istoj stranici, samo se mijenjaju objekti unutar stranice. JavaScript briše i stvara novo korisničko sučelje. Korisnik će imati osjećaj izmjenjivanja stranica, ali je zapravo uvijek na istoj HTML stranici, a za to je zadužen virtualni DOM.

Učinkovito upravljanje DOM promjenama pomoću JavaScripta može postati vrlo komplicirano i dugotrajno. Iz perspektive kôdiranja lakše je očistiti djecu u hijerarhiji određenog elementa i rekonstruirati ih, nego što bi bilo ostaviti te podređene elemente i pokušati ih učinkovito ažurirati. Problem je u tome što možda programer nema znanja o JavaScriptu da bi učinkovito radio s DOM aplikacijskim programskim sučeljem. Rješenje je React i njegov koncept korištenja virtualnog DOM-a. React je biblioteka koja je osmišljena da ažurira DOM preglednik. Više se ne treba baviti složenostima povezanim s izgradnjom aplikacija s jednom stranicom jer React to odrađuje. S Reactom stranica ne komunicira izravno s DOM aplikacijskim programskim sučeljem. Umjesto toga komunicira s virtualnim DOM-om ili nizom uputa koje će React koristiti za izgradnju korisničkog sučelja i interakciju s preglednikom. Virtualni DOM sastoji se od React elemenata koji konceptualno izgledaju slično HTML elementima, ali su zapravo JavaScript objekti. Raditi s JavaScript objektima mnogo je brže nego raditi s DOM aplikacijskim programskim sučeljem. Ako se izvrše izmjene u JavaScript objektu ili virtualnom DOM-u, React čini te

promjene koristeći DOM aplikacijsko programsko sučelje što je efikasnije moguće (Banks i Porcello, 2017).

Virtualni DOM programski je koncept u kojem idealna, odnosno *virtualna* reprezentacija korisničkog sučelja ostaje u memoriji i sinkronizirana je sa *stvarnim* DOM-om u biblioteci poput ReactDOM. Takav pristup omogućuje deklarativnom aplikacijskom programskom sučelju komunikaciju s Reactom tako da navede u kojem stanju želi korisničko sučelje, a virtualni DOM osigurava korisničkom sučelju odgovaranje tom stanju bez izmjene cijele stranice.

```
<section>
  <h1>Naslov</h1>
  <ul >
    <li>Lista 1</li>
    <li>Lista 2</li>
    <li>Lista 3</li>
  </ul>

  <section>
    <h2>Opis</h2>
    <p>Tekst.</p>
  </section>
</section>
```

Slika 4. Prikaz HTML tagova i djece

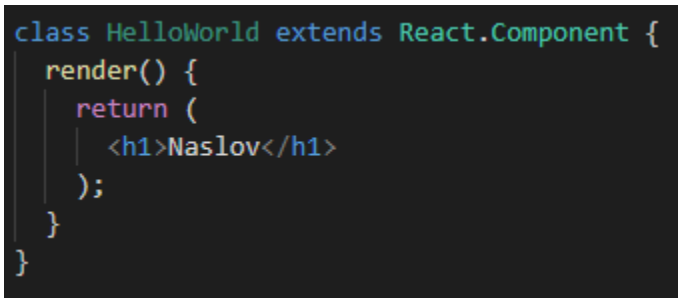
HTML je skup instrukcija koje preglednik slijedi pri konstruiranju modela objekta dokumenta ili DOM-a. Elementi koji čine HTML dokument postaju DOM elementi kada preglednik učita HTML i čini korisničko sučelje. U HTML-u su elementi međusobno povezani u hijerarhiji koja nalikuje hijerarhijskom stablu. Iz slike 4 mogu se vidjeti elementi unutar HTML-a, a virtualni DOM elemente posloži u hijerarhiju. U ovom slučaju može se reći da korijenski element ima troje djece: naslov (h1), popis (li) i odjeljak za opis (p) kojeg virtualni DOM posloži u hijerarhiju.

2.7. JSX

Sve React komponente imaju funkciju renderiranja koja određuje kakve će biti HTML React komponente. JavaScript eXtension ili češće JSX je React ekstenzija koja omogućava pisanje JavaScripa, a izgleda kao HTML. Iako se u prethodnim paradigmama smatralo lošom navikom uključivanje JavaScripta i tagova na istom mjestu, ispostavilo se da kombiniranje JavaScripta i HTML-a u Reactu uvelike pomože.

JavaScript XML (JSX) je XML sintaksa koja konstruira tagove u React komponentama. React djeluje bez JSX-a, ali pomoću JSX-a lakše je čitanje i pisanje komponenti, kao i strukturiranje bilo kojeg drugog HTML elementa.

Slika 5 prikazuje React komponentu koja u sebi sadrži oznaku naslova. JSX omogućuje da taj element izgleda isto kao i HTML.



Slika 5. Primjer JSX-a

Funkcija *render()* u komponenti HelloWorld izgleda kao da vraća HTML, ali ovo je zapravo JSX jezik. JSX se prevodi tijekom JavaScript vremena izvođenja. Izgled komponente nakon prijevoda prikazan je na slici 6. Komponente se koriste za stranice kao što je u projektu početna stranica, a isto tako komponenta može služiti i za određene objekte unutar stranice.

```
class HelloWorld extends React.Component {
  render() {
    return (
      React.createElement(
        'h1',
        'Naslov'
      )
    );
  }
}
```

Slika 6. Prikaz renderirane komponente

Iako JSX izgleda kao HTML, on je zapravo samo brži način za pisanje deklaracije *React.createElement*. Kad se komponenta renderira, ona prikazuje stablo React elemenata, odnosno virtualni prikaz HTML elemenata koje ova komponenta prikazuje kao rezultat. React će tada odrediti koje će se izmjene unijeti u stvarni DOM na temelju prikazivanja trenutnog React elementa.

Iz knjige izdane 2016. godine, čiji su autori Sonpatki i Vipul, spomenut je razgovor Shawa i Mikea. Shawn je imao sjajan prvi dan i tek je počeo sa sljedećim u Adeffic Consultingu, a uz šalicu kave prestravio je Mikea.

"Hej Mike, vidio sam da si koristio JSX za izgradnju prve komponente. Zašto si koristio JSX kada React ima *React.createElement*?"

"Shaw možeš koristiti React bez korištenja JSX, ali JSX olakšava izgradnju komponenti Reacta. Smanjuje količinu kôda potrebnog za pisanje i izgleda kao HTML. Njegova sintaksa je jednostavna i sažeta te vrlo je lako vizualizirati komponente koje se grade (Sonpatki i Vipul, 2016, str. 22)."

2.8. Stanja i svojstva

Većina komponenti može se prilagoditi tek kada su stvorene uz pomoć različitih parametara. Ti se parametri nazivaju u kôdu *props*, što je kratica za svojstva (engl. *Properties*). Naprimjer jedna osnovna komponenta Reacta je slika, a kada se pišu tagovi za sliku može se koristiti atribut *props* za dodavanje izvora slike.

```
export default class Bananas extends Component {
  render() {
    let pic = {
      uri: 'https://upload.wikimedia.org/wikipedia/commons/d/de/Bananavarieties.jpg'
    };
    return (
      <Image source={pic} style={{width: 193, height: 110}}/>
    );
  }
}
```

Slika 7. Prikaz propsa u komponenti

Može se primijetiti iz slike 7 da se unutar zagrada nalazi varijabla *pic*. Unutar vitičastih zagrada može se pisati JavaScript kôd, na slici se vidi da je upotrijebljena varijabla *pic* kao svojstvo.

Glavna funkcija svojstva je da se podaci mogu prenositi iz komponente u komponentu. Programer može kreirati vlastitu komponentu koja također može imati svoja svojstva. Ako se radi o komponenti koja se koristi na različitim lokacijama u aplikaciji, s drugačijim svojstvima, idealno je koristiti svojstva. Unutar komponente može se jednostavnom komandom pozvati trenutno svojstvo koje je predano komponenti pozivom na *this.props* unutar funkcije.

U projektnom se dijelu često koriste svojstva, a jedan primjer svojstva u projektu prikazuje slika 8. Kreirana je komponenta *Autocomplete* koja služi za automatsko popunjavanje gradova unutar polja za unos podataka.

```
<Autocomplete
  parentInfo={this.autoCompleteStartCity}
  placeholder="Početna lokacija"
  places={Places}
  value={this.state.startCity}
/>
```

Slika 8. Slanje svojstva kroz komponentu

Iz slike 8 vidi se da su poslana četiri svojstva. Poslano je svojstvo *places* koje predstavlja JSON objekt svih gradova u Republici Hrvatskoj, te se to svojstvo može koristiti u komponenti *Autocomplete.*

Slika 9 prikazuje korištenje svojstva *places* unutar komponente *Autocomplete*. Bilo gdje u JavaScript kôdu unutar komponente koja je primila svojstvo, moguće je pristupiti vrijednosti svojstvu. Komponenta *Autocomplete* može koristiti gradove u Republici Hrvatskoj jednostavnim pozivom koji je prikazan na slici 9.

```
this.props.places;
```

Slika 9. Korištenje svojstva u komponenti

Stanja (engl. *State*) su drugačija od svojstva. U aplikaciji programer vjerojatno neće postavljati stanja čekajući određeno vrijeme. Obično se stanje postavlja kada primi nove podatke s poslužitelja ili iz korisničkog unosa. Programer može također koristiti Redux ili Mobx za spremanje podataka u stanje. Više o Reduxu u nastavku.

Kako bi se stanje promijenilo, u kôdu se mora navesti funkcija *setState*. Komponenta će se ponovo renderirati svaki put kada se funkcija *setState* pozove. U Reactu postoje dvije vrste kontrole podataka za prikazivanje unutar komponente, to su stanja i svojstva. Svojstvo predstavlja atribut roditelja za komponentu gdje se šalje te se ne može izmjenjivati tijekom životnog ciklusa komponente. Za podatke koji se planiraju mijenjati u komponenti koristi se stanje. Stanje se treba inicijalizirati unutar konstruktora, a zatim se može pozvati *setState* kada se žele podaci promijeniti.

Dobar primjer korištenja stanja je aplikacija koja sadrži tekst koji treperi. U stanju se kreira varijabla koja može primiti dvije vrijednosti, a to su uključeno i isključeno. Kako

bi aplikacija znala u kojem je trenutno stanju tekst, koristi se stanje te se izmjenjuje svake sekunde i primjenjuje određena stanja unutar komponente.

Slika 10 prikazuje kreiranje stanja unutar komponente koji se koristi u projektnom dijelu za nuđenje cijene kuriru. Stanje se kreira unutar konstruktora, tako da pri inicijalizaciji komponente postoji stanje. Unutar konstruktora potrebno je referencirati na klasu s ključnom riječi *this*, zatim navesti stanje i predati JSON objekt stanju. U ovom slučaju je to samo jedan atribut unutar JSON-a.

```
constructor(props) {
  super(props);

  this.state = {
    amount: "",
  };
}
```

Slika 10. Kreiranje stanja

Slika 11 prikazuje polje u kojem korisnik upisuje cijenu koju želi ponuditi kuriru te se može vidjeti da je u atributu *onChange* predana funkcija koja je prikazana na slici 12. Ukoliko korisnik krene upisivati iznos, React odmah reagira na upis i poziva funkciju *changeHandler()* te pomoću funkcije mijenja stanje unutar komponente, tako da stanje *amount* više nije prazno stanje, već upisani iznos.

```
<input
  type="number"
  className="form-control form-input-money"
  name="amount"
  onChange={this.changeHandler}
  placeholder="HRK"
  min={this.props.location.state.amount}
/>
```

Slika 11. Korištenje stanja

Slika 12 prikazuje izmjenu stanja, tako da se poziva funkcija *setState()*. Razlog zašto se mora pozivati funkcija *setState()* je ta da komponenta pozivom funkcije dobiva obavijest da je stanje promijenjeno. Jednom kada se pozove funkcija *setState(),* komponenta se ponovno renderira i mijenja samo stanja koja ne odgovaraju prijašnjim stanjima.

```
changeHandler = (e) => {
  this.setState({ [e.target.name]: e.target.value });
};
```

Slika 12. Izmjena stanja

2.9. Redux

Za bolje razumijevanje Reduxa potrebno je osnovno znanje načina funkcioniranja stanja u Reactu. Stanje služi za držanje podataka koji su potrebni aplikaciji. Korištenje modela stanja funkcionira dobro kada aplikacija ima hijerarhijsko stanje te više ili manje odgovara komponentnoj strukturi. Na ovaj se način stanje širi na više komponenti.

Ponekad komponente, koje su na dnu hijerarhije u aplikaciji, trebaju pristupiti stanju aplikacije, odnosno podacima. Ako programer želi podatke ažurirati u cijeloj aplikaciji u isto vrijeme, teško je tako nešto postići stanjem. Ako se koristi model stanja u Reactu, na kraju će biti gomila velikih komponenata na vrhu stabla koje će slati podatke posrednim komponentama kroz koje prolaze velike količine podataka koje ih ne koriste, samo kako bi ti podaci mogli biti poslani do nekoliko komponenti na dnu hijerarhije ili listova kojima su potrebni ti podaci. Kada se programer nađe u sličnoj situaciji, dobra praksa je koristiti Redux za spremanje i korištenje podataka. Redux se koristi za spremanje i čitanje podataka tako da su podaci na jednoj lokaciji, što znači da svaka komponenta može pristupiti direktno podacima bez da se ti podaci šalju nepotrebno između komponenti.

2.9.1. Primjena Reduxa

U projektnom zadatku koristio se Redux kao sustav za spremanje podataka. Redux se sastoji od akcija i reduktora. Akcije služe za slanje blokova informacija koje se šalju iz aplikacije u skladište (engl. *Store*). Akcije su jedini izvor informacija za skladište. Kako bi se podaci mogli poslati u skladište podataka potrebno je koristiti naredbu *store.dispatch().*

Akcije su obični JavaScript objekti, te moraju imati svojstvo tipa koje označava vrstu akcije koja se izvodi. Tip akcije se obično treba definirati kao konstanta niza karaktera kao na slici 13.

```
export const loadAcceptedSends = () => async (dispatch) => {
  dispatch({
    type: DELIVERIES_LOADING,
  });
  if (localStorage.jwtToken) {
    setAuthToken(localStorage.jwtToken);
  }
  // Headers
  const config = {
    headers: {
      "Content-Type": "application/json",
    },
  };

  try {
    await axios
      .get("http://localhost:5000/api/delivery/getAcceptedSends", config)
      .then((res) => {
        dispatch({
          type: ACCEPTEDSENDS_LOADED,
          payload: res.data,
        });
      });
  } catch (err) {
    console.log(err);

  }
};
```

Slika 13. Prikaz Redux akcija

Nakon što je aplikacija dovoljno velika, potrebno je premjestiti sve konstante u posebni modul kao na slici 14. Navedeni modul se koristi u projektnom dijelu.

```
export const GET_ITEMS = "GET_ITEMS";
export const ADD_ITEM = "ADD_ITEM";
export const DELETE_ITEM = "DELETE_ITEM";
export const ITEMS_LOADING = "ITEMS_LOADING";
export const USER_LOADING = "USER_LOADING";
export const USER_LOADED = "USER_LOADED";
export const USER_NOT_LOADED = "USER_NOT_LOADED";
export const AUTH_ERROR = "AUTH_ERROR";
export const LOGIN_SUCCESS = "LOGIN_SUCCESS";
export const LOGIN_FAIL = "LOGIN_FAIL";
export const LOGOUT_SUCCESS = "LOGOUT_SUCCESS";
export const REGISTER_SUCCESS = "REGISTER_SUCCESS";
export const REGISTER_FAIL = "REGISTER_FAIL";
export const GET_INFO = "GET_INFO";
export const CLEAR_INFO = "CLEAR_INFO";

export const DELIVERIES_LOADING = "DELIVERIES_LOADING";
export const SEARCHDELIVERIES_LOADED = "SEARCHDELIVERIES_LOADED";

export const CURRENTDELIVERIES_LOADED = "CURRENTDELIVERIES_LOADED";
export const HISTORYDELIVERIES_LOADED = "HISTORYDELIVERIES_LOADED";

export const ACCEPTEDSENDS_LOADED = "ACCEPTEDSENDS_LOADED";
export const HISTORYSENDS_LOADED = "  HISTORYSENDS_LOADED";
export const ACCEPTEDDELIVERY_LOADED = "ACCEPTEDDELIVERY_LOADED";
export const REQUESTS_LOADED = "REQUESTS_LOADED";
export const REQUESTDELIVERY_LOADED = "REQUESTDELIVERY_LOADED";
export const REQUESTSEND_LOADED = "REQUESTSEND_LOADED";
export const FINISHEDDELIVERY_LOADED = "FINISHEDDELIVERY_LOADED";
export const FINISHEDSEND_LOADED = "FINISHEDSEND_LOADED";
export const DELIVERIESINFO_LOADED = "DELIVERIESINFO_LOADED";
```

Slika 14. Tipovi akcija

Funkciji *dispatch()* može se pristupiti izravno iz skladišta pomoću komande *store.dispatch(),* ali više se koristi pristup pomoću pomagača poput *react-redux's* funkcije *connect().* Također može se koristiti *bindActionCreators()* da se automatski vežu akcije na funkciju *dispatch().*

Reduktori određuju kako se stanje aplikacije mijenja kao odgovor na radnje poslane u skladište. Akcije opisuju samo ono što se dogodilo, ali ne opisuju kako se stanje aplikacije mijenja, to je posao reduktora.

Za vrijeme kreiranja reduktora važno je pridržavati se nekoliko pravila. Pri prvom pozivanju reduktora, stanje vrijednosti bit će nedefinirano. Redukcija treba obraditi ovaj slučaj davanjem zadanog stanja prije prve radnje. Treba se pogledati prethodno stanje i utvrditi kakvu vrstu posla treba obaviti. Prema pretpostavkama da se trebaju dogoditi stvarne promjene, potrebno je stvoriti novi objekt koji je ažuriran podacima i vratiti ga novom stanju. Ako promjene nisu potrebne, dovoljno je vratiti staro stanje kakvo je i bilo. Slika 15 prikazuje reduktor iz projektnog zadatka. Reduktor u ovom slučaju služi za obavještavanje korisnika, odnosno prikazuju se odgovori s poslužitelja te se obavještava korisnik je li uspješno insertirao element ili ne.

```
const initialState = {
  msg: null,
  status: null,
  id: null
};

export default function(state = initialState, action /*: IAction*/) {
  switch (action.type) {
    case GET_INFO:
      return {
        msg: action.payload.msg,
        status: action.payload.status,
        id: action.payload.id
      };
    case CLEAR_INFO:
      return {
        msg: null,
        status: null,
        id: null
      };
    default:
      return state;
  }
}
```

Slika 15. Prikaz Redux reduktora

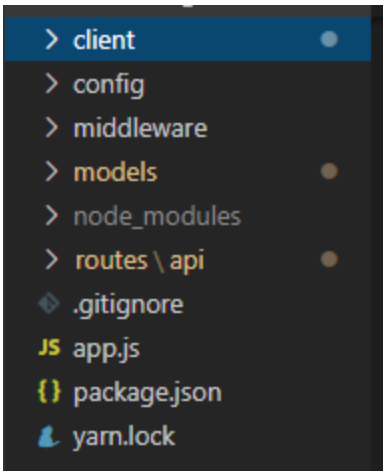
2.10. React i poslužitelj

React je biblioteka koja se koristi za razvoj korisničkog sučelja. Napisana je u JavaScriptu, stoga se može izvoditi na strani klijenta, ali budući da ne ovisi ni o čemu, a može se koristiti samo u *web*-pregledniku, moguće je također renderirati stranice na strani poslužitelja korištenjem Node.js-a. Budući da se React bavi promjenom korisničkog sučelja na temelju promjene podataka, najkorisnije je da se on pokreće na strani klijenta, ali postoje određeni nedostaci koji dolaze kada poslužitelj ne učita dostavljenu stranicu klijentu.

Renderiranje na strani poslužitelja (engl. SSR - *Server side rendering*) popularna je tehnika koja je ista kao i renderiranje na klijentskoj strani unutar *web*-aplikacija s jednom stranicom, samo što se renderira na poslužitelju te zatim šalje potpuno renderiranu stranicu klijentu. Tada klijentska strana može koristiti sve pogodnosti *web*-aplikacija s jednom stranicom i na taj način učitavati stranicu bez da se cijela stranica mora ponovno učitati.

Renderiranje na poslužiteljskoj strani može također pomoći u radu s performansama jer se potpuno učitana aplikacija s prvog poslužitelja šalje na zahtjev klijenta. Programer mora razmotriti hoće li korištenje renderiranja na poslužiteljskoj strani za React aplikaciju zadovoljiti određene kriterije, a trebaju se razmotriti specifičnosti aplikacije, problemi, itd. Nakon toga se treba donijeti odluka o korištenju renderiranja na poslužiteljskoj strani ukoliko pridonosi performansama.

Struktura projekta je prikazana slikom 16. Iz slike se vidi da u hijerarhiji projektnih datoteka postoji mapa pod nazivom *klijent*. Sve datoteke koje se nalaze u mapi *klijent* služe za prikaz i dizajn aplikacije, odnosno unutar mape *klijent* su React skripte. Datoteke izvan mape *klijent* pripadaju poslužitelju. Na poslužiteljskoj strani se nalaze modeli koji služe za stvaranje dokumenata i kolekcija u bazi podataka, te funkcionalnosti koje služe za komunikaciju poslužitelja i klijenta.



Slika 16. Struktura datoteka klijenta i poslužitelja

2.11. Kreiranje React aplikacije

Komanda *create-react-app* je generator React aplikacije koju je kreirao Facebook. Pruža razvojno okruženje konfigurirano za jednostavnu upotrebu uz minimalno postavljanje, uključujući:

- ES6 i JSX;

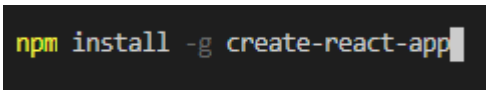
- Poslužitelja s modulom za ponovno punjenje;

- Povezivanje kôdova;

- CSS automatsko podešavanje;

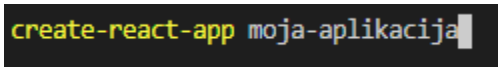
- Gotove skripte s grupiranim JS-om te izvornim mapama.

Kako bi se kreirala React aplikacija, potrebno je koristiti potrebni paket menadžer, au projektnom se dijelu koristi Node paket menadžer. Slika 17 prikazuje kako se pomoću paket menadžera kreira projekt.



Slika 17. Instalacija React aplikacije

Nakon kreiranja projekta potrebno je pokrenuti generator u odabranom direktoriju. Slika 18 prikazuje pokretanje generatora s nazivom *moja-aplikacija*.



Slika 18. Generator aplikacije

Nakon pokrenutog generatora, slika 19 prikazuje React komponentu. U komentarima su objašnjene pojedine linije kôda.

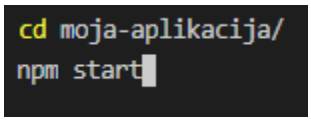
React komponenta može se definirati kao ES6 klasa koja proširuje osnovnu klasu *React.Component*. U svom minimalnom obliku, komponenta mora definirati način prikazivanja koji određuje kako se komponenta renderira DOM-u. Render metoda vraća čvorove Reacta, koji se mogu definirati JSX sintaksom kao HTML oznake. Sljedeća slika pokazuje kako definirati komponentu.

```
import React from 'react';

class Komponenta extends React.Component {
 constructor(props){
  // Budući da se proširuje zadani konstruktor,
  // potrebno je poslati podatke u super klasu React.Component

  super(props);
  // Odvajanje imena od prop svojstva
  let ime = this.props.name.split(" ")[0];

  // U konstruktoru stanje se izmijenjuje ovisno
  // o trenutnom kontekstu.
  this.state = {
    name: ime
  }
 } // Jednostavno korištenje stanja u JSX-u
 render() {
  return <h1>Hello, {this.state.name}!</h1>
 }
}

export default Komponenta
```

Slika 19. React komponenta

Komponenta može primati svojstva. Svojstva su proslijeđena od roditelja komponente kako bi se odredile vrijednosti koje trenutna komponenta ne može sama znati. Svojstvo može sadržavati i funkciju koju komponenta može pozvati nakon što se dogode određeni događaji. Primjerice gumb može primiti funkciju za svojstvo *onClick* i pozvati ga kad god se pritisne. Prilikom pisanja komponente, pristup svojstvu je moguć kroz poziv objekata svojstva same komponente.

Nakon odrađenih naredbi i kreiranja komponenti potrebno je navesti put prema aplikaciji, u ovom slučaju komandnom linijom ući u mapu *moja-aplikacija* koja je ranije definirana. Kada je naveden put unutar aplikacije, potrebno je samo pokrenuti React. Navođenje puta u mapu i pokretanje React aplikacije prikazuje slika 20.



Slika 20. Pokretanje aplikacije

2.12. Korištenje vlastitih komponenti

Komponente omogućuju da se podijeli korisničko sučelje na neovisne dijelove za višekratnu upotrebu, te za svaki dio odvojeno može razraditi logika. Konceptualne komponente su poput JavaScript funkcija. Prihvaćaju proizvoljne unose poput svojstva te vraćaju React elemente koji opisuju ono što bi se trebalo pojaviti na zaslonu. React može koristiti klase kao svoje komponente te primati određena svojstva, kao što funkcije imaju argumente. Slika 21 prikazuje kreiranu komponentu za prikaz navigacije na gornjoj strani *web*-aplikacije.

```
return (
  <Navbar color="dark" dark expand="sm" className="mb-5">
    <Container>
      <NavbarBrand
        style={{ color: "white", textDecoration: "none" }}
        href="/"
      >
        Raspored
      </NavbarBrand>
      <NavbarToggler onClick={this.toggle} />
      <Collapse isOpen={this.state.isOpen} navbar>
        <NavLink
          style={{ color: "white", textDecoration: "none" }}
          href="/about"
        >
          O nama
        </NavLink>
        <NavLink
          style={{ color: "white", textDecoration: "none" }}
          href="/scheduler"
        >
          Kontrolna ploča
        </NavLink>
        <Nav className="ml-auto" navbar>
          {isAuthenticated ? authLinks : guestLinks}
        </Nav>
      </Collapse>
    </Container>
  </Navbar>
);
```

Slika 21. Kreiranje komponente

Kada je komponenta kreirana ona se može koristiti u drugim komponentama jednostavnim pozivom. Potrebno je na početku skripte umetnuti komponentu koja se želi koristiti s naznačenim putem do komponente. Slika 22 prikazuje umetanje komponente za prikaz navigacije.

```
import Home from "./Components/home";
import About from "./Components/about";
import LogIn from "./Components/logIn";
import SignUp from "./Components/signUp";
import Offer from "./Components/offer";
import Search from "./Components/search";
import UserProfil from "./Components/userProfile";
import NavigationBar from "./Components/navigationBar";
import OfferPrice from "./Components/offerPrice";
```

Slika 22. Umetanje komponente

Iz slike 22 vidi se da je umetnuto više komponenti te je među njima označena komponenta za prikaz navigacije. Jednom kada je komponenta umetnuta, moguće je komponentu koristiti jednostavnim pozivom imena komponente koji se dodijelio u početku, koji je u ovom slučaju *NavigationBar.*

```
<Router>
  <NavigationBar />
  <Route exact path="/" component={Home} />
  <Route path="/about" component={About} />
  <Route path="/logIn" component={LogIn} />
  <Route path="/signUp" component={SignUp} />
  <PrivateRoute path="/search" component={Search} />
  <PrivateRoute path="/offerPrice" component={OfferPrice} />
  <PrivateRoute path="/offer" component={Offer} />
  <PrivateRoute path="/userProfil" component={UserProfil} />
  <PrivateRoute path="/acceptedDelivery" component={AcceptedDelivery} />
  <PrivateRoute path="/currentRoute" component={CurrentRoute} />
  <PrivateRoute path="/acceptedSend" component={AcceptedSend} />
  <PrivateRoute path="/historyDelivery" component={HistoryDelivery} />
  <PrivateRoute path="/historySend" component={HistorySend} />
  <PrivateRoute path="/request" component={Request} />
  <PrivateRoute path="/requestOffer" component={RequestOffer} />
  <PrivateRoute path="/giveRating" component={GiveRating} />
</Router>
```

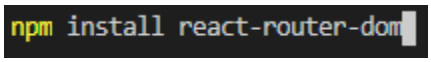
Slika 23. Korištenje komponente

Označen tag na slici 23 prikazuje poziv komponente za prikaz navigacije.

2.13. React usmjerivač

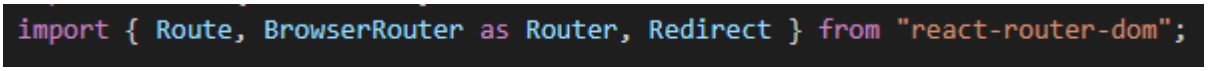
React usmjerivač (engl. *Router*) je biblioteka za usmjeravanje ruta React aplikacije, odnosno usmjerivač koji se koristi za navigaciju korisnika između stranica u aplikaciji.

Usmjerivač u verziji 4 bio je statički, no nakon verzije 4 postaje dinamičan. U aplikacijama za jednu stranicu postoji samo jedna HTML stranica. Ponovno se koristi ista HTML stranica za prikaz različitih komponenti na temelju usmjerivača. U aplikacijama koje koriste više stranica dobit će se posve nova stranica s poslužitelja tijekom navigacije. Kako bi se koristio usmjerivač, potrebno je instalirati paket za usmjerivanje. Slika 24 prikazuje instalaciju paketa.



Slika 24. Instalacija usmjerivača

Kada je instaliran usmjerivač u Reactu, moguće ga je koristiti. Unutar komponente potrebno je umetnuti usmjerivače kako bi se mogli koristiti. Slika 25 prikazuje umetanje usmjerivača.



Slika 25. Umetanje usmjerivača

Nakon što su se umetnuli usmjerivači i komponente koje se koriste u projektnom dijelu, potrebno je i navesti usmjerivače unutar komponente. Komponenta za usmjeravanje sadrži dva svojstva, jedno svojstvo je put, a drugo je komponenta. U put se stavlja potreban *link* u koji se želi usmjeriti korisnika, dok svojstvo komponente prikazuje komponentu koja je unutar svojstva kada korisnik aktivira određeni usmjerivač. Slika 23 prikazuje korištenje usmjerivača.

Postoje stranice koje nisu dostupne korisnicima koji nisu prijavljeni. Da bi se pristupilo određenoj stranici kao što je korisnički profil, potrebna je prijava. Zahvaljujući

privatnim rutama nije moguće pristupiti određenim stranicama bez tokena. Privatne rute je moguće vidjeti na slici 23, dok je njihova implementacija prikazana slikom 26.

```
const PrivateRoute = ({ component: Component, ...rest }) => (
  <Route
    {...rest}
    render={props =>
      localStorage.jwtToken ? (
        <Component {...props} />
      ) : (
        <Redirect exact to={{ pathname: "/" }} />
      )
    }
  />
);
```

Slika 26. Implementacija privatne rute

Privatna se ruta sastoji od funkcije koja provjerava je li korisnik prijavljen. Ako korisnik nije prijavljen onda u lokalnoj pohrani neće biti tokena, što znači da funkcija preusmjerava korisnika na početnu stranicu.

3. NODE.JS

3.1. Uvod

Node.js nastao je kada su ga originalni programeri JavaScripta proširili od nečega što se moglo pokrenuti samo u pregledniku, do nečega što bi se moglo pokrenuti na vlastitim računalima kao samostalni program.

Node.js platforma je na strani poslužitelja izgrađena u JavaScriptu Google Chrome (V8 pokretač). Node.js razvio je Ryan Dahl 2009. godine. Danas se s JavaScriptom može učiniti mnogo više nego prije. JavaScipt je prije služio je za izradu interaktivnih *web*-stranica. JavaScript ima mogućnost raditi druge načine skriptiranja, kao što to može i programski jezik Python.

Node.js platforma je izgrađena na Chromeovom JavaScript vremenu izvođenja radi jednostavne izrade brzih i skalabilnih mrežnih aplikacija. To je jezik koji je vođen *eventima*, asinkronim I/O modelom koji ga čini učinkovitim, savršenim za aplikacije u stvarnom vremenu (engl. *Real time app*) koje intenzivno koriste podatke i koje se pokreću na distribuiranim uređajima.

Node.js jezik je otvorenog kôda, koristi se na više platforma te se može koristiti u razne svrhe, poput skripti na poslužitelju, *web*-stranici itd. Aplikacije Node.js-a napisane su u JavaScriptu i mogu se izvoditi unutar Node.js vremena izvođenja na macOS-u, Microsoft Windowsu i Linuxu.

Node.js također nudi bogatu biblioteku različitih JavaScript modula što u velikoj mjeri pojednostavljuje razvoj *web*-aplikacija koje koriste Node.js. Za korištenje Node.js-a nisu preporučljive aplikacije koje zahtijevaju procesorsku snagu, odnosno CPU.

3.2. Zašto Node.js?

Jedan od razloga zašto izabrati Node.js je sljedeći. U slučaju scenarija u kojem se šalje zahtjev za bazu podataka za određene podatke vezane uz prvog i drugog korisnika, složeno je obrađivati zahtjeve za oba korisnika. Node.js može obrađivati više zahtjeva istovremeno. Odgovor na ovaj zahtjev zahtijeva određeno vrijeme, ali oba se zahtjeva za korisničkim podacima mogu provesti samostalno i u isto vrijeme. Node.js koristi neblokirajući (engl. *Non-blocking*) način, dok u blokirajućem (engl. *blocking*) načinu, zahtjev za podatke drugog korisnika ne pokreće se sve dok se podaci o prvom korisniku ne ispišu na zaslon.

U slučaju *web*-poslužitelja, morale bi se pokrenuti nove dretve za svakog novog korisnika. No JavaScript se koristi jednom dretvom koju može dijeliti s više korisnika. Na taj način, JavaScript nije prikladan za zadaće koje zahtijevaju više dretvi. Tu pomaže neblokirajući dio. Iz tog je razloga Node.js koristan programerima.

3.3. Značajke Node.js-a

Slijedi nekoliko važnih značajki koje Node.js čine prvim izborom softverskih arhitekata.

- Asinkronost – Aplikacijsko programsko sučelje i biblioteke Node.js-a su asinkrone, odnosno neblokirajuće. To znači da poslužitelj temeljen na Node.js-u nikada ne čeka da aplikacijsko programsko sučelje vrati podatke, nego da poslužitelj može nastaviti sa sljedećim zadaćama.
- Brzina - Budući da je izgrađen na pokretaču V8 JavaScript Google Chrome, biblioteka Node.js ima vrlo brzo izvođenje kôda.
- Skalabilna dretva – Node.js koristi model jedne dretve pomoću petlje *evenata*. *Event* mehanizam pomaže poslužitelju da odgovori bez puno blokiranja, odnosno čekanja odgovora, što ga čini vrlo skalabilnim u odnosu na ostale tradicionalne poslužitelje koji koriste limitirane dretve koje obrađuju korisničke zahtjeve. Node.js može pružiti uslugu puno

većem broju zahtjeva od tradicionalnih poslužitelja poput Apache HTTP poslužitelja.

- Nema međuspremnika – Node.js aplikacije nikada ne pohranjuju podatke. Takve aplikacije jednostavno iznose podatke u dijelovima.
- Licenca – Node.js je pod licencom MIT-a.

3.4. Gdje se koristi?

Node.js dokazana je kao savršena biblioteka koja se koristi u sljedećim područjima:

- Aplikacije koje koriste ulaz/izlaz
- Aplikacije za prijenos podataka
- Aplikacije u stvarnom vremenu
- Aplikacije zasnovane na JSON aplikacijskom programskom sučelju
- *Web*-aplikacije koje koriste jednu stranicu

3.5. Razlika između Node.js-a i PHP-a

U ovom poglavlju spomenute su ključne značajke i razlike između Node.js-a i PHP-a, a također je prikazana komparacija, odnosno koja je tehnologija pogodna za koju svrhu.

Razvoj *web*-stranica učestalo se mijenja. Nove tehnologije i alati redovito se pojavljuju te se zbog njih se *web*-programeri moraju suočiti s novim tehnologijama i izabrati pravi izbor. Programer mora odlučiti između dugoročnih i zrelih rješenja te novih nadolazećih.

Programeri na strani poslužitelja su oni koji se obično moraju suočiti s problemom izbora između PHP-a i Node.js-a kada je u pitanju izrada skripti na strani poslužitelja. To je postala uobičajena rutina programera. Ranije se JavaScript (onaj koji je temelj Node.js-a) nije preklapao s PHP-om. JavaScript je bio korišten za izgradnju sučelja na klijentskoj strani, a s druge strane PHP se koristio za razvoj aplikacija na strani poslužitelja. Koristeći danas zajedno JavaScript za klijentsku i poslužiteljsku stranu, mogu se izraditi moderne *web*-stranice.

Zašto više PHP nije najtraženiji jezik kada je u pitanju poslužitelj? Razlog je taj što je JavaScript predstavio potpuno novi Node.js koji je počeo dominirati u sferi na strani poslužitelja te odvratio programere od tradicionalnog PHP-a.

Iako su i Node.js i PHP sposobni upravljati bilo kakvom složenom aplikacijom, izgrađeni su oko različitih arhitektura i koncepata. Odabir između ovih programskih jezika za projekt vezan uz *web*-stranice doista može biti težak.

Ne može se pogriješiti jer obje tehnologije poslužitelja zadovoljavaju osnovne uvjete te su dostupne za upotrebu. Odluka ovisi o tome koji jezik na poslužiteljskoj strani izabrati, Node.js ili PHP, a to ovisi o prirodi projekta. PHP i Node.js su moćni za dinamičke *web*-stranice. Oboje pripadaju istoj kategoriji, no njihova su obilježja različita.

Sličnost između PHP-a i Node.js je tumač (engl. i*nterpreter*). Oboje se mogu pokrenuti u vremenu izvođenja te za to vrijeme koriste tumač. Koriste se za izradu *web*-stranica te se oba pokreću na poslužiteljskoj strani. Jedan od razloga zašto je bolje izabrati Node.js je taj što je opsežniji i brži u usporedbi s PHP-om. Međutim neki programeri više vole PHP jer ga je lakše naučiti, a ima i integriranu bazu podataka (Websoptimization, n.d.).

U nastavku su navedeni razlozi zašto koristiti PHP:

- Centralizirani poslužitelj - U slučaju da se ne planira skalirati aplikacija na više poslužitelja, onda se može koristiti LAMP (Linux, Apache, MySQL i PHP) arhitektura. To bi se moglo promijeniti ovisno o potrebama projekta.
- Prebacivanje u druge jezike - PHP je prijenosni jezik. Jeftini troškovi *web hostinga* i dostupnost poslužitelja za PHP su neusporedivi. PHP se može izvoditi na gotovo bilo kojoj platformi na kojoj je instaliran Apache, IIS i podržani sustav baza podataka, što PHP aplikaciju čini prenosivom i jednostavnom za implementaciju. Sustav upravljanja sadržajem (engl. *CMS - Content Management System*) poput WordPressa, Drupala ili Joomla olakšavaju izradu *web*-stranica i rad na gotovo svim *web* poslužiteljima. WordPress predstavlja 30% interneta.

U nastavku su navedeni razlozi zašto koristiti Node.js:

- MEAN arhitektura (engl. MEAN *stack*) - Node.js je dobar izbor ako projekt uključuje softverske skupove poput MEAN arhitekture (MongoDB, ExpressJs, AngularJs, Node.js), dinamičke aplikacije za jednu stranicu, tehnologije na strani poslužitelja, te tehnologije za sučelja na strani klijenta poput Angular.js-a, Backbone.js-a ili Reacta. Prednost navedenih jezika je što svi koriste isti jezik (Javascript). Također može se koristiti Typescript biblioteka u slučaju da aplikacija koristi Angular kako bi bilo lakše programerima. MEAN arhitektura je također lakša za skaliranje nego LAMP arhitektura na više poslužitelja.
- Aplikacije u realnom vremenu (engl. *Realtime app*) - Node.js je idealan za aplikacije koje zahtijevaju podatke u realnom vremenu. Takve aplikacije nisu dobre kada je u pitanju korištenje Node.js-a za financijske aplikacije jer Javascript nije pouzdan kada je riječ o brojevima, zato što se tipovi varijabli ne odvajaju jasno kao i u ostalim jezicima.
- Brzina - Kao što je već spomenuto, Node.js je brži od PHP-a kada je u pitanju brzina izvršavanja, a ako je brzina ono što je prioritet u aplikaciji,

kao što je *multiplayer* igra temeljna na pretraživaču ili aplikacija za *chat*, Node.js je bolji izbor od PHP-a (Websoptimization, n.d.).

3.6. Sigurnost

Node paket menadžer (npm) zadani je upravitelj paketa za Node.js i jedan je od najvećih otvorenih izvornih paketa ekosustava na svijetu. Ovaj bogati ekosustav paketa otvorenog kôda doveo je do povećanja produktivnosti programera i performansi aplikacija, što dovodi do povećanja kvalitete samih programera i organizacije.

Node.js baza kôdova (engl. *Codebase*) često sadrže stotine ili čak tisuće Node paketa. Programeri možda nisu svjesni izravne i neizravne ovisnosti paketa i sigurnosnih rizika povezanih s njima.

Node paket menadžer počeo se orijentirati na sigurnost 2018. godine kada su objavili paket menadžer *audit*, novu naredbu koja vrši trenutni pregled sigurnosti stabla ovisnosti projekta i izrađuje izvješće o sigurnosti paket menadžer revizije. Izvješće sadrži podatke o sigurnosnim ranjivostima u ovisnostima i daje paket menadžer naredbe i preporuke za daljnje otklanjanje problema. Pitanje je gledaju li tvrtke popis sigurnosnih ranjivosti i upravljaju li ih na odgovarajući način. Današnji sigurniji paket menadžer od Node paket menadžera je Yarn, koji se također koristi u projektnom zadatku.

Mnoge organizacije s kojima radi tim Black Duck Audit Services imaju interne sigurnosne programe i koriste alate za testiranje sigurnosti, kao što su statička i dinamička analiza. Iako su ti alati korisni za identificiranje uobičajenih pogrešaka kôdiranja koji mogu rezultirati problemima u sigurnosti, pokazali su se neučinkovitima u identificiranju ranjivosti koje unose kôd kroz komponente otvorenog kôda. Primjerice 12% baza kôdova koristi razvojno okruženje Node.js u 2018. godini, preko 3% uključuje ranjivost Drown, više od 2% uključuje Freak, a 1,6% *codebasea* čak je sadržavalo i ranjivost Poodlea koja je javno objavljena 2014. (Konsinski i Armstrong, n.d.).

Prvi korak u pružanju sigurnih usluga je korištenje HTTP Secure (HTTPS). Priroda interneta omogućava trećoj strani da presreće pakete koji se prenose između klijenata i

poslužitelja. HTTPS kriptira te pakete što napadaču izuzetno otežava pristup podacima koji se prenose. Nije nemoguće jer ne postoji takva savršena sigurnost. Međutim HTTPS se smatra dovoljno sigurnom za bankarstvo, korporativnu sigurnost i zdravstvo. HTTPS se može smatrati podlogom za osiguravanje *web*-stranice. Ne pruža provjeru autentičnosti, ali postavlja osnovu za provjeru autentičnosti. Naprimjer sustav provjere autentičnosti vjerojatno uključuje slanje lozinke, ako se ta lozinka šalje šifrirano, nijedna sofisticiranost provjere autentičnosti neće osigurati sustav. Sigurnost je jaka kao i najslabija veza, a prva veza u tom lancu je mrežni protokol. HTTPS 207 protokol temelji se na poslužitelju koji ima certifikat javnog ključa, koji se ponekad naziva i SSL certifikat. Ideja iza certifikata je da postoje tijela za certifikate (CA) koja izdaju certifikate. Tijelo za certificiranje pruža dobavljačima pouzdane certifikate dostupnim na preglednicima. Preglednici uključuju ove pouzdane certifikate kada se instalira preglednik i to je ono što uspostavlja lanac povjerenja između CA-a i preglednika. Da bi ovaj lanac radio, poslužitelj mora koristiti certifikat izdan od CA-a (Brown, 2014).

U razvojnom okruženju poput Node.js-a potrebno je da se više vremena posveti Node paket menadžerima koji su kreirali različiti programeri. Treba se znati više o ovisnostima paketa otvorenog kôda u aplikacijama i skrivenim isječcima koji mogu postaviti zakonska ograničenja za te pakete.

3.7. Korištenje terminala

Ako se koristi terminal, uz njega dolazi snaga i produktivnost. Ako programer nema znanja raditi u terminalu onda će programiranje predstavljati složen posao za njega jer je znanje osnovnih komandi terminala neophodno. Ako se koristi OS X ili Linux, postoji više različitih terminala koji imaju svoja sučelja. Najpopularniji je *bash* zato što je sveprisutan, a koristi se na Linuxu i Windowsu.

Ako se upotrebljava Unix operacijski sustav moguće je primijetiti da se 99% vremena koristi *bash* terminal. Korištenje Windowsa je drugačije od Unixa jer Microsoft nikad nije bio osobito zainteresiran za pružanje ugodnog terminalnog iskustva, pa je potrebno malo više proučavati terminal. Git koristi također terminal *Git bash*, koji pruža terminal nalik Unixu (ima samo mali podskup uobičajenih Unix alata odnosno komandi). *Git bash* pruža *bash* terminal i dalje koristi ugrađenu aplikaciju za Windows konzole, što dovodi do frustracije. Čak i jednostavna funkcionalnost poput promjene veličine prozora konzole, odabir teksta, rezanja i lijepljenja zna biti frustrirajuća (Brown, 2014).

Kada se u projektu želi pokrenuti aplikacija, potrebno je prvo pokrenuti poslužitelj te se nakon poslužitelja treba pokrenuti klijent. Slika 27 prikazuje primjer pokretanja projektnog dijela u Visual Studio Code konzoli. Jednom kada se pokrene poslužitelj, moguće je pokrenuti i klijenta jer skripte klijenta dohvaćaju potrebne podatke od poslužitelja.

```
PS C:\Users\Developer\Desktop\SendPacket_NOVO> npm run server
PS C:\Users\Developer\Desktop\SendPacket_NOVO> npm start --prefix client
```

Slika 27. Korištenje konzole za pokretanje aplikacije

3.8. Paketi

Kratica za Node Package Manager je npm. Node paket menadžer je zadani upravitelj paketa za Node.js. Može se koristiti za instaliranje paketa trećih strana i za upravljanje ovisnostima. On je registar (www.npmjs.com) za javno spremište svih modula koje ljudi objavljuju u svrhu dijeljenja.

Iako je Node paket menadžer počeo kao spremište za Node.js module, brzo se transformirao u upravitelja paketa za isporuku drugih modula temeljenih na JavaScriptu, osobito onih koji se mogu koristiti u pregledniku. Razvojno okruženje jQuery daleko je najpopularnija JavaScript knjižnica na klijentskoj strani, a dostupna je kao paket menadžer modul. React je kôd na klijentskoj strani i može ga se izravno uključiti u HTML kao datoteku skripte, a umjesto toga preporučuje se da se React instalira putem paket menadžera. Jednom instalirani paket se treba staviti u zajednički kôd da se može uključiti unutar HTML-a tako da preglednik ima pristup tome kôdu. Za to postoje alati za izradu kao što su *browserfy* ili *webpack* koji mogu sastaviti vlastite module kao i biblioteke trećih strana u paketu koje se mogu uključiti u HTML (Subramanin, 2017).

Node paket menadžer je na vrhu liste modula ili repozitorija za pakete, a ima više od 250 000 paketa. Prije Node paket menadžera je bio najveći je bio Meaven, a danas nema ni upola koliko ima Node paket menadžer. Ovo pokazuje da Node paket menadžer nije samo najveće, već je i najbrže rastuće spremište. Za uspjeh Node.js-a odgovoran je paket menadžer i ekosustav modula.

Node paket menadžer nije jednostavno koristiti za kreiranje i za korištenje modula. Ima jedinstvenu tehniku rješavanja sukoba koja omogućava postojanje više konfliktnih verzija modula paralelno kako bi se zadovoljile ovisnosti. Stoga, u većini slučajeva, paket menadžer upravo djeluje na ispravan način (Subramanin, 2017).

Velika je vjerojatnost da ukoliko programer koristi Node.js skripte, da je upotrijebio jednu od vanjskih funkcionalnosti u obliku modula. U ovom poglavlju objašnjeno je kako ovo sve funkcionira. Pored svih već moćnih i funkcionalnih modula koje Node.js pruža za programere, postoji ogromna zajednica koja i dalje razvija module koji se mogu iskoristiti u programima te je moguće i napisati svoje module.

Pozitivna činjenica u vezi s Node.js-om je ta što se zapravo ne razlikuje između vlastitog modula i modula koji je iz vanjskih spremišta, odnosno od drugih programera. Kad se u Node.js pišu odvojene klase i grupe funkcija, stavljaju se u osnovi uz malo uređivanja i dokumentiranja. Potreban je samo dodatni dio JSON-a, možda redak ili dva kôda da bi vlastiti kôd bio spreman za druge programere. Node.js isporučuje veliki broj ugrađenih modula, koji su svi zapakirani u Node.js te koji se mogu izvršiti u sustavu. Njihov izvor je moguće vidjeti ako se preuzme izvorni kôd s *web*-stranice Node.js-a (Wandschneider, 2013).

Na visokoj razini moduli predstavljaju način grupiranja zajedničkih funkcionalnosti u Node.js-u. Naprimjer, ako imate biblioteku funkcija ili klasa za rad s određenim poslužiteljem baze podataka, bilo bi lakše taj kôd staviti u modul i pakirati za daljnje korištenje. Svaka datoteka u Node.js-u predstavlja modul, no moduli ne moraju biti tako jednostavni. Mogu se pakirati složeni moduli s mnogim datotekama, testovima jedinica, dokumentacijom i drugim datotekama podrške u mape i može ih se konzumirati na isti način kao što bi se koristio jedan modul sa samo jednom JavaScript datotekom.

3.9. Yarn

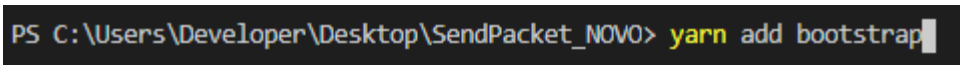
U JavaScript zajednici, inženjeri dijele stotine tisuća dijelova kôda kako bi se izbjeglo prepisivanje osnovnih komponenti, biblioteka ili vlastitih okvira. Svaki dio kôda zauzvrat može ovisiti o ostalim dijelovima kôda, a tim ovisnostima upravljaju paket menadžeri. Najpopularniji paket menadžer JavaScripta je Node paket menadžer (npm) koji pruža pristup više od tristo tisuća paketa u registru. Više od pet milijuna inženjera koristi Node paket menadžer registar, koji bilježi do pet milijardi preuzimanja svakog mjeseca (McKenzie et al., 2016).

Godinama se uspješno koristi Node paket menadžer klijent na Facebooku, ali kako je veličina baze podataka i broj inženjera rastao, postojali su problemi s dosljednošću, sigurnošću i performansama. Nakon pokušaja da se riješe problemi, donijeta je odluka da se izgradi novo rješenje koje će pomoći da se pouzdano upravlja ovisnostima. Proizvod tog dijela naziva se Yarn koji je brži, pouzdaniji i sigurniji u odnosu na Node paket menadžer klijent.

Uz Yarn, inženjeri i dalje imaju pristup Node paket menadžer registru, ali brže mogu instalirati pakete i dosljedno upravljati ovisnostima na svim strojevima ili u sigurnim izvanmrežnim okruženjima. Yarn omogućava inženjerima bržu i samopouzdaniju upotrebu zajedničkog kôda kako bi se mogli usredotočiti na ono što je važno, a to su izgradnja novih proizvoda i značajki.

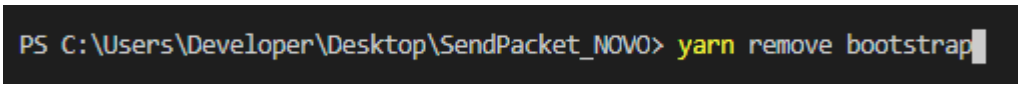
Primarna funkcija svakog upravitelja paketa je instalirati paket, dio kôda koji ima određenu svrhu, iz globalnog registra u inženjersko lokalno okruženje. Svaki paket može, ali i ne mora ovisiti o drugim paketima. Tipični projekt mogao bi sadržavati desetke, stotine, pa čak i tisuće paketa unutar svojeg stabla ovisnosti. Te se ovisnosti prepravljaju i instaliraju na temelju semantičke verzije (Semver). Semver definira shemu inačice koja odražava promjene u svakoj novoj verziji, dodaje novu značajku ili popravlja grešku. Međutim Semver se oslanja na pakete koji ne čine pogreške.

U projektu se često koristio Yarn jer je brži i sigurniji u odnosu na Node paket menadžer. Slika 28 prikazuje primjer instaliranja Yarn paketa Bootstrap.



Slika 28. Instalacija paketa pomoću Yarna

Ukoliko programer ne želi koristiti određeni paket, a slučajno ga je instalirao, slika 29 prikazuje brisanje paketa.



Slika 29. Brisanje paketa pomoću Yarna

3.10. Testiranje

Ako se želi dodati nova valuta u internetsku trgovinu, najprije se mora dodati test kako bi se definirali očekivani izračuni kao što su porez i ukupni iznos. Tada je potrebno napisati kôd kako bi ovaj test prošao. Kako bi testiranje bilo kvalitetno, potrebno je poznavati upravljačke module te testne skripte koje se mogu postaviti u datoteci *package.json*. Jedni su od testnih okvira Mocha i Nodetap.

Jedna od prednosti rada s Node.js-om je ta što je zajednica u ranoj fazi usvojila testiranje, tako da ne postoje nedostaci modula koji bi pomogli u pisanju brzih i čitljivih testova.

Neki se pitaju što je tako dobro u testovima i zašto se pišu rano tijekom razvoja. Testovi su važni za istraživanje ideja prije nego što ih preuzmete. Također prenose namjeru, što znači da pomažu u dokumentiranju i proširivanju ideja u ključnim dijelovima projekta. Testovi također mogu pomoći u smanjenju održavanja projekata tako što omogućuje da provjere jesu li promjene pokvarile postojeće radne značajke. Prvo što je potrebno naučiti su Node.js moduli. Izraz i potvrda očekivanja glavne su svrha testova. Za pisanje testova nije potrebno upotrebljavati *assert,* što je ugrađeni temeljni modul. Slične *asserte* je moguće vidjeti i upotrebljavati i u drugim jezicima (Young i Harter, 2015).

3.11. Korištenje paketa

Moduli se koriste za organiziranje većih programa i distribuciju Node.js projekata, stoga je važno biti upoznat s osnovnim tehnikama potrebnim za njihovu instalaciju i izradu.

Bez obzira na to koristi li se modul koji nudi Node.js ili modul treće strane, podrška za module spremljena je izravno u Node.js i uvijek je dostupna. Glavni problem Node paket menadžera jesu moduli treće strane. Rješenje je instalacija modula putem terminala, a zatim se treba učitati modul. Slika 30 prikazuje primjer instaliranja *express* modula za projekt u diplomskom radu.

```
$ npm search express
express               Sinatra inspired web development framework
$ npm install express
express@x.x.x ./node_modules/express
└── methods@x.x.x
└── (Several more dependencies appear here)

$ node
> var express = require('express');
> typeof express
'function'
```

Slika 30. Instalacija paketa

Paket menadžera naredbenog retka može se koristiti za pretraživanje, instaliranje i upravljanje paketima. *Web*-stranica *https://npmjs.org* pruža još jedno sučelje za pretraživanje modula, a svaki modul ima svoju stranicu koja prikazuje pridružene datoteke koje objašnjavaju kako se koristi određeni paket.

Jednom kada se dozna naziv modula, instalacija je lagana, potrebno je upisati naziv modula i on će se instalirati u *./node_modules* mapi. Slika 30 prikazuje instalaciju modula. Moduli mogu biti i globalno instalirani izvođenjem naredbe *npm install -g module*, a nakon upisane naredbe modul će biti instaliran u globalnoj mapi. To je obično */usr/local/lib/node_modules* na Unix sustave. U Windows sustavima nalazi se gdje i

binarni *node.exe*. Nakon što je modul instaliran, može se učitati jednostavno u aplikaciju, slika 30 prikazuje umetanje modula. Metoda zahtjeva obično vraća objekt ili metodu, ovisno o načinu na koji je modul postavljen.

3.12. Paket Express

Vrijeme izvođenja Node.js-a je okruženje koje se može pokrenuti pomoću JavaScripta. Ručno pisanje *web*-poslužitelja koristeći Node.js nije jednostavno, niti je potrebno. Express je okvir koji pojednostavljuje zadatak pisanja kôda poslužitelja.

Express je fleksibilni okvir *web*-aplikacije Node.js koji pruža robustan skup značajki za razvoj *web* i mobilnih aplikacija. Omogućuje brzi razvoj *web*-aplikacija temeljenih na Node.js-u. Slijedi nekoliko osnovnih značajki Express okvira:

- Omogućuje postavljanje međusoftvera (engl. *Middleware*) kako bi odgovarao na HTTP zahtjeve.
- Definira tablicu ruta koje se koriste za izvođenje različitih radnji na temelju HTTP metoda i URL-a.
- Omogućuje dinamično prikazivanje HTML stranica na temelju slanja argumenata u predloške.

Express.js je okvir koji je na poslužitelju *web*-aplikacije Node.js. Posebno je dizajniran za izradu aplikacija s jednom stranicom, više stranica i hibridnih *web*-aplikacija. Postao je standardno razvojno okruženje na poslužiteljskoj strani za Node.js. Express je dio MEAN arhitekture koji se nalazi na strani poslužitelja.

MEAN je besplatan JavaScript softverski paket za izgradnju dinamičnih *web*-aplikacija koji sadrži sljedeće komponente:

1) MongoDB – Standardna nerelacijska baza podataka

2) Express.js – Zadano razvojno okruženje za *web*-aplikacije

3) Angular.js – JavaScript model-pogled-kontroler razvojno okruženje koje se koristi za *web*-aplikacije

4) Node.js - Okvir koji se koristi za skalabilne aplikacije na strani poslužitelja i umrežavanje

Okvir Express.js-a olakšava razvoj aplikacije koja se može koristiti za obradu više vrsta zahtjeva kao što su GET, PUT, POST i DELETE.

Instaliranje i korištenje Expressa je vrlo jednostavno. Express se instalira putem Node paket menadžera. To se može postići izvršenjem sljedećeg retka u terminalu '*npm install express'*. Gornja naredba zahtijeva od upravitelja paketa Node.js preuzimanje tražene *express* module i u skladu s tim ih instalira (Wandschneider, 2013).

Okvir Expressa omogućuje definiranje ruta, specifikacija što učiniti kada dođe HTTP zahtjev koji odgovara određenom uzorku. Podudarajuća specifikacija temelji se na regularnom izrazu (*regex*) i vrlo je fleksibilna, kao i većina drugih okvira *web*-aplikacija. Dio što Express treba učiniti je samo funkcija koja je dobila analiziran HTTP zahtjev.

Express analizira zahtjev URL-a, zaglavlja i parametara za programera. Na strani poslužitelja, prema očekivanjima, treba imati sve funkcionalnosti koje zahtijevaju *web*-aplikacije. To uključuje postavljanje kôdova za odgovore s poslužitelja, postavljanje kolačića, slanje prilagođenih zaglavlja, itd. Također moguće je pisati Express međusoftver, kojem su prilagođeni dijelovi kôda koji se mogu umetnuti u bilo koji put obrade zahtjeva/odgovora s poslužitelja, te kako bi se postigla zajednička funkcionalnost kao što su prijava, provjera autentičnosti, itd.

Express nema ugrađeni predložak, ali podržava bilo koji pokretački mehanizam po izboru, poput *pug, mustache*, itd. Za *web*-aplikacije s jednom stranicom nije potrebno koristiti mehanizam predloška na strani poslužitelja (Subramanin, 2017, str. 10).

3.12.1. Korištenje Expressa

Odjeljak uključuje opće tehnike izrade *web*-aplikacija. Cilj Expressa je jednostavan za korištenje, što ga čini fleksibilnim, ali ponekad nije lako znati kako ga koristiti na najbolji način. Express pomaže pri problemu kada aplikacija postaje izuzetno velika te kada se želi na bolji način organizirati sve rute. Express ima mogućnost usmjeravanja.

Express je minimalistički okvir u kojem je organiziranje projekta izbor programera, a ako se dobro poznaje Express paket moguće je uz malo napora napraviti dobru aplikaciju. Projekti u početnom razvoju mogu postati složeni za implementiranje, ako se ne obraća pažnja na Express. Tajna uspješnog organiziranja većih projekata podrazumijeva korištenje sustava modula Node. Prvi izazov su rute, ali ta se tehnikamože primijeniti na svim aspektima razvoja pomoću Expressa. Slika 31 prikazuje Express rutu u projektu koju dohvaća korisnik.

```
// @route   GET api/auth/user
// @desc    Get user data
// @access  Private
router.get("/user", auth, (req, res) => {
  User.findById(req.user.id)
    .select("-password -register_date -currentDelivery -negotiations")
    .then((user) => res.json(user));
});
```

Slika 31. Express GET ruta

Usmjeravanje se definira pomoću metoda objekta Express aplikacije koje odgovaraju HTTP metodama. Naprimjer *app.get()* funkcija za obradu GET zahtjeva dok *app.post* za obradu POST zahtjeva. Metoda *app.*METHOD se koristi za sve zahtjeve. Slika 31 prikazuje GET zahtjev u projektu, dok slika 32 prikazuje DELETE zahtjev. Razlika između tipova ruta je u prijenosu podataka, na primjer metodom GET prenesi se trenutni prikaz ciljnog resursa, dok za POST dodatne informacije koje se mogu spremati u tijelo zahtjeva.

```
router.delete("/delSenNeg", auth, (req, res) => {

  // Logika rute

});
```

Slika 32. Express DELETE ruta

3.13. Asinkrono programiranje

Asinkrono programiranje oblik je paralelnog programiranja koji omogućuje da se jedinica rada izvodi odvojeno od primarne dretve (engl. t*hread*). Kada je posao završen, ona obavještava glavnu nit (je li posao dovršen ili još traje). Mnogo je koristi od korištenja asinkronog programiranja, poput poboljšanih performansi aplikacije i poboljšane reakcije aplikacije. Asinkrono programiranje se ne bi trebalo koristiti u svakoj instanci jer postoje neke situacije u kojima bi se trebalo izbjegavati. Asinkrono programiranje zapravo postoji već duže vrijeme, ali posljednjih se godina sve više koristi.

JavaScript je asinkronog karaktera, pa tako i Node.js. Asinkrono programiranje je obrazac dizajna koji osigurava izvršavanje kôda bez zastajkivanja, odnosno u neblokirajućem modu. Neblokirajući kôd ne sprječava izvršenje dijela kôda. Općenito ako se kôd izvršava sinkrono, odnosno jedan za drugim nepotrebno se zaustavlja izvršavanje kôda koji ne ovisi o prethodnom kôdu. Asinkroni način je upravo suprotno, asinkroni kôd se izvršava bez ikakve ovisnosti i bez reda. To poboljšava učinkovitost i propusnost sustava. Asinkrono programiranje se koristi za brže izvršavanje programa, ali dolazi s cijenom.

Prelazak s modela programiranja u kojem se izvršava niz sinkronih ili blokiranih ulazno/izlaznih poziva i čeka se završetak svakog od njih prije nego što prijeđete na sljedeći model u kojem se sve radi asinkrono i čeka se da Node.js najavi kada je zadani zadatak obavljen, zahtijeva malo više razmišljanja i eksperimentiranja. No kada programer jednom svlada paradigmu asinkronog programiranja, više neće razmišljati o starom načinu pisanja *web*-aplikacija (Wandschneider, 2013, str. 61).

U projektnom se dijelu često koristi asinkrono programiranje. Slika 33 prikazuje primjer korištenja asinkronog programiranja. Ako se želi koristiti asinkrono programiranje u pojedinoj funkciji, potrebno je ispred nje napisati ključnu riječ *async*. Kada se napiše ključna riječ *async,* program zna da treba izvršavati kôd asinkrono. Asinkrono programiranje je u većini slučajeva korisno dok se čeka odgovor s poslužitelja jer mnogo podataka i prikaza ovise o rezultatu poslužitelja. Na slici se vide dvije funkcije ispred koji se nalazi ključna riječ *await*, ona služi da bi se Reactu dodatno naznačilo koje se funkcije trebaju asinkrono izvršavati, što znači da se prvo izvršava funkcija *checkOutDateDelivery()* te nakon vraćenog rezultata se izvršava funkcija *getInfoUserProfile()*. Funkcija *checkOutDateDelivery()* provjerava kod korisničkog profila isporuke koje su završile, tek kada provjeri i obriše potrebne isporuke, onda se dohvaćaju dodatne informacije pomoću funkcije *getInfoUserProfile().*

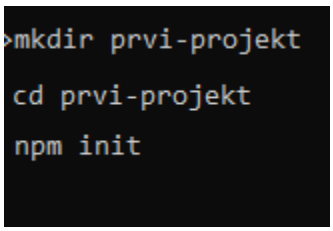
```
async componentDidMount() {
  // Ako korisnik nije logiran, stranica ga vrati na početnu stranicu
  if (this.props.isAuthenticated) {
    this.props.history.push("/");
  }
  await this.props.checkOutDateDelivery();
  await this.props.getInfoUserProfile();
}
```

Slika 33. Asinkrono programiranje

3.14. Kreiranje Node aplikacije

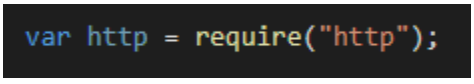
Poglavlje prikazuje kako kreirati običnu aplikaciju u Node.js okruženju. Prije kreiranja aplikacije pomoću Node.js, bitno je obratiti pažnju na komponente koje će biti umetnute. Node.js aplikacija sastoji se od sljedeće tri važne komponente, a to su umetanje modula, kreiranje vlastitog poslužitelja koji će slušati zahtjeve klijenta slične Apache HTTP poslužitelju te čitanje zahtjeva i odgovora klijenta. Aplikacija sadrži poslužitelja kojemu je zadatak da čita HTTP zahtjev klijenta koji može biti preglednik ili konzola te naposljetku vratiti odgovor.

Glavno pitanje vezano za Node.js je kako se stvara projekt. Projekt se može kreirati i pokrenuti zahvaljujući Node paket menadžeru. Iako bi se mogla stvoriti JavaScript datoteka i pokrenuti, jednostavnije je koristiti paket menadžer kako bi se napravio *package.json*. Slika 34 prikazuje kako se kreira folder i inicijalizira projekt u *folderu*.



Slika 34. Kreiranje Node aplikacije

Jednom kada je kreiran projekt, potrebno je kreirati datoteku u kojoj se nalaze kôdovi aplikacije. Kako bi se koristile pogodnosti HTTP-a moraju se na početku aplikacije umetnuti moduli. Slika 35 prikazuje umetanje modula HTTP.



Slika 35. Umetanje http-a

Potrebno je kreirati instancu *http* i pozvati *http.createServer()* metodu kako bi imali instancu poslužitelja. zatim se povezuje na *port* 8081 koristeći metodu slušanja koja je

povezana s instancom poslužitelja. Kao argument šalje se funkcija koja također ima argumente za zahtjev i odgovor. Slika 36 prikazuje kôd koji je dovoljan za stvaranje HTTP poslužitelja koji sluša, odnosno čeka zahtjev preko 8081 *porta*.

```
http.createServer(function (request, response) {
  // Šalje se HTTP zaglavlje
  // HTTP Status: 200 : OK
  // Tip sadržaja: text/plain
  response.writeHead(200, {'Content-Type': 'text/plain'});

  // Šalje se klijentu odgovor "Pozdrav"
  response.end('Pozdrav\n');
}).listen(8081);

// Konzola ispisuje
console.log('Server pokrenut..');
```

Slika 36. Kreiranje Node poslužitelja

Sve što je potrebno u konzoli je napisati komandu koja se nalazi na slici 37.

```
node main.js
```

Slika 37. Pokretanje Node aplikacije

Kada se izvrši komanda, programer je u mogućnosti vidjeti poruku u konzoli *Server pokrenut*, dok će kao odgovor na stranici biti prikazan tekst *Pozdrav*, što znači da je uspješno kreirana Node.js aplikacija te je poslužitelj spreman za primanje upita od klijenta.

4. Korištenje JavaScripta

Od objavljivanja 1995. godine JavaScript je prošao kroz mnoge promjene. U početku je dodavanje interaktivnih elemenata *web*-stranicama bilo mnogo jednostavnije. Tada je postao robusniji s DHTML-om i AJAX-om. Sada je s Node.js-om JavaScript postao jezik koji se koristi za izgradnju *full-stack* aplikacija. Odbor koji je i nadležan za nadmetanje promjena JavaScripta je Europska udruga proizvođača računala (engl. ECMA - *European Computer Manufacturers Association*).

Promjena JavaScripta ima utjecaj na cijelu zajednicu. Potječe od prijedloga koje pišu članovi zajednice. Svatko može podnijeti prijedlog u odboru ECMA-e. Odgovornost odbora ECMA je da upravlja i postavi prioritete prijedlozima koji su utvrđeni kao valjani. Prijedlozi se uzimaju kroz jasno definirane faze, od faze 0 koja predstavlja najnovije prijedloge, do faze 4 koja predstavlja gotove prijedloge.

4.1. Primjena JavaScripta

Node.js, MongoDB, Express i AngularJS temelje se na jeziku JavaScript-a. Lako se može implementirati i ponovo upotrijebiti kôd na svim navedenim razvojnim okruženjima jer se koristi samo jedan jezik. U nastavku je objašnjeno deklariranje i korištenje varijabli u JavaScriptu.

Varijable se koristite u JavaScriptu za privremeno spremanje i pristup podacima iz JavaScript datoteka. Varijable mogu upućivati na jednostavne tipove podataka poput brojeva te nizova znakova ili mogu upućivati na složene vrste podataka, kao što su objekti. React koristi klasu te se u njoj nalazi funkcija za renderiranje stranice.

Za definiranje varijable u JavaScriptu potrebno je koristiti ključnu riječ *var*, a zatim varijabli dodati naziv kao što je prikazano na slici 38.

```
var isAuthenticated;
```

Slika 38. Kreiranje varijable

Vrijednosti se mogu dodijeliti varijabli u istom retku. Naprimjer sljedeći redak kôda stvara varijablu *isAuthenticated* i dodjeljuje joj vrijednost *false*:

```
var isAuthenticated = false;
```

Slika 39. Deklariranje varijable s vrijednosti

Jedan redak na slici 39 funkcionira isto kao sljedeća slika 40 s dva retka.

```
var isAuthenticated;
isAuthenticated = false;
```

Slika 40. Deklariranje i korištenje varijable

Nakon što se deklarira varijabla, može se koristiti ime varijable kako bi se dodijelila vrijednost varijabli te pristup vrijednosti varijable. Naprimjer sljedeći kôd prikazuje provjeru varijable *isAuthenticated*. Ukoliko je vrijednost *true* programski kôd će koristiti funkciju gdje se ispisuje vrijednost korisničkog imena i prikazuje u navigacijski bar, a ako je vrijednost *false* programski kôd će prikazati dva dugmeta za prijavu i registraciju.

```
  </Nav>
  {isAuthenticated ? userNavigation(props) : guessNavigation()}
</Collapse>
```

Slika 41. Deklariranje varijable i dodavanje vrijednosti

Varijabla *isAuthenticated* predstavlja dozvolu za određene prikaze u projektu. Ako je varijabla *true*, to znači da je korisnik prijavljen u aplikaciju. Varijabli se treba dati opisno ime kako bi se kasnije lakše prepoznalo koji se podaci pohranjuju u određenu varijablu. Pored toga imena varijabli razlikuju velika i mala slova, tako da je varijabla *isAuthenticated* različita od *IsAuthenticated.* Varijable su u JavaScriptu od velike važnosti te u njih možemo spremiti vrijednosti, objekte, pa čak i funkcije. U projektnom dijelu, primjer korištenja varijable je spremanje korisničkog imena s poslužitelja te kada

poslužitelj odgovori klijentu, klijent tada taj odgovor spremi u varijablu i prikazuje korisničko ime na vrhu stranice.

4.2. Funkcionalno programiranje s JavaScriptom

Kada se krene istraživati okruženje Reacta, može se primijetiti da se tema funkcionalnog programiranja pojavljuje sve češće. Funkcionalne tehnike sve se više koriste u JavaScript projektima.

Mnogi su već napisali funkcionalni JavaScript kôd bez razmišljanja o tome. React, Flux i Redux uklapaju se u funkcionalnu JavaScript paradigmu. Razumijevanje osnovnih koncepata funkcionalnog programiranja može podići znanje o strukturi React aplikacija. U projektnom dijelu često se funkcije spremaju u varijable te se varijable jednostavno pozivaju radi kraćeg kôda.

Izumom Lambda-kalkulusa dolazi trend funkcionalnog programiranja 1930-ih. Funkcije se mogu poslati funkcijama kao argumenti ili vratiti iz funkcija kao rezultata. Krajem 1950-ih, John McCarthy uzeo je koncepte izvedene funkcije iz Lambda-kalkulusa i primijenio ga na novi programski jezik pod nazivom Lisp. Lisp je implementirao koncept funkcija i funkcije višeg reda kao članova prve klase. Funkcija se smatra *članom prve klase* kada se može deklarirati kao varijabla i poslati funkcijama kao argument, te se funkcije čak mogu vratiti s funkcija. U JavaScriptu funkcije se koriste slično kao varijable. ES6 dodaje poboljšanja koja mogu poboljšati funkcionalne tehnike programiranja. U JavaScriptu funkcije mogu predstavljati podatke u aplikaciji. Može se primijetiti da funkcije pomoću ključne riječi *var* mogu biti deklarirane na isti način na koji se mogu deklarirati brojevi ili niz znakova. Slika 42 prikazuje kako se funkcija deklarira u JavaScriptu (Banks i Porcello, 2017).

```
const generateToken = (res, data) => {
  const expiration = 3600;
  const token = jwt.sign({ data }, "Tajna" /*, {
    expiresIn: 3600
  }*/);

  return (
    res.cookie("token", token, {
      secure: false, // set to true if your using https
      httpOnly: false // true
    })
  );
};
```

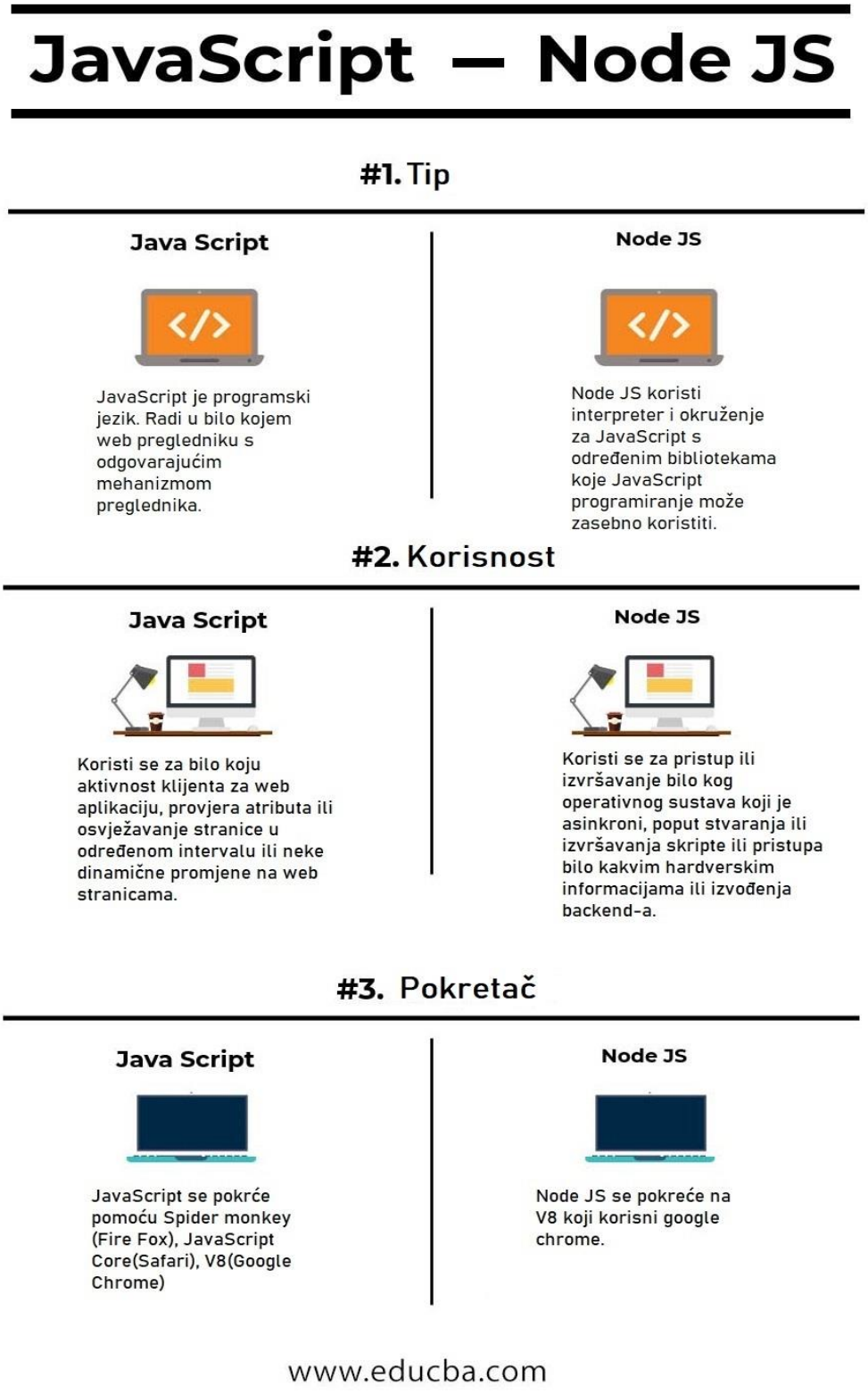
Slika 42. Varijabla kao funkcija

Slika 42 prikazuje spremanje funkcije u varijablu. U varijablu se sprema funkcija za umetanje tokena u kolačić kako bi se mogao token koristiti i provjeravati za autentifikaciju korisnika. Jednostavnim pozivom konstantne varijable *generateToken()* izvrši se kod koji se nalazi u funkciji.

4.3. Razlika između JavaScripta i NodeJS-a

JavaScript je programski jezik koji se pokreće u *web*-preglednicima, dok je Node.js tumač za JavaScript koji sadrži puno potrebnih biblioteka koji olakšavaju programiranje. JavaScript je u osnovi standardni definirajući programski jezik koji može pokrenuti bilo koji preglednik sa zadanim preglednikom. To je vrlo jak jezik koji se obično koristi za *web*-aplikacije na bilo kojoj određenoj poslovnoj logici koja se mora dodati na ekran bez osvježavanja stranica. JavaScript također pomaže u generiranju dinamičnih HTML tablica na temelju poslovnih zahtjeva. JQuery je popularna biblioteka za lakše korištenje JavaScripta izbjegavajući pisanje puno kôdova.

Node.js sadrži puno relativnih biblioteka koje se inače koriste u Javascriptu. To je zapravo vrsta tumača koji može predstavljati JavaScript ili pokrenuti bilo koji JavaScript program. Uglavnom pomaže u izvršavanju nekih neblokirajućih operacija kao posebnih operativnih sustava, poput detalja o certifikatu ili pojedinosti o hardveru. Svi preglednici imaju JavaScript mehanizam koji pomaže u pokretanju JavaScripta u pregledniku. Spider monkey kojeg koristi FireFox preglednik, JavaScript Core koji koristi Safari, V8 koji je koristi Google Chrome neke su od popularnih JavaScript pokretača. Node.js koristi V8 pokretač koji se koristi izravno uz pomoć nekoliko biblioteka za obavljanje operacija ulaza i izlaza te mrežnih operacija. Također pomaže u korištenju JavaScripta izvan preglednika, kao što je stvaranje, pisanje ili izvršavanje skripte, servisa na poslužiteljskoj strani ili pokretanje hardvera. Slika 43 prikazuje glavne razlike između Node.js-a i JavaScripta.



Slika 43. Razlike JavaScripta i Node.js-a (www.educba.com)

5. MERN arhitektura

5.1. Uvod

Prednost MERN arhitekture je u tome što postoji jedan jezik koji se koristi u svim tehnologijama. Koristi JavaScript kôd na klijentskoj strani kao i kôd na poslužiteljskoj strani. Čak i ako postoje skripte baze podataka (MongoDB), pišu se u JavaScriptu. Dakle jedini jezik koji je potrebno znati za MERN arhitekturu je JavaScript.

To se odnosi na sve ostale skupove temeljene na MongoDB-u i Node.js-u, posebno kada je u pitanju MEAN arhitektura. Razlog isticanja MERN-a je taj da jezik predloška nije potreban za generiranje stranica. Također React generira HTML (DOM elemente) koristeći JavaScript. Dakle ne samo da se izbjegava učenje novog jezika, već je moguće dobiti svu snagu JavaScripta. To je suprotno jeziku predloška, koji će imati svoja ograničenja. Potrebno je znanje u HTML-u i CSS-u, no to nisu programski jezici i nema potrebe da se izbjegava učenje HTML-a i CSS-a jer ih je vrlo jednostavno naučiti. Kako nema potrebe za prebacivanjem konteksta tijekom pisanja kôda na klijentskoj i poslužiteljskoj strani, postojanje jednog jezika na više slojeva omogućuje i dijeljenje kôda između njih.

Bilo koja *web*-aplikacija izrađena je korištenjem više tehnologija. LAMP arhitektura je akronim za Linux, Apache, MySQL i PHP, a sve su to komponente otvorenog kôda. Kako je razvoj *weba* sazrijevao i kako su interaktivnosti dolazila do izražaja, *web*-aplikacije s jednom stranicom postale su sve popularnije. *Web*-aplikacija s jednom stranicom je paradigma *web*- aplikacije koja izbjegava osvježavanje *web*-stranice za prikaz novog sadržaja, a umjesto toga koristi slanje poziva poslužitelju za dobivanje podataka ili isječaka, te s dobivenim podacima ažurira dio *web*-stranice. Rezultat izgleda prilično dobro u odnosu na stari način potpunog učitavanja stranice. To je dovelo do porasta razvojnih okruženja za sučelje. Kako je porasla popularnost JavaScript jezicima, NoSQL baza podataka također je počela dobivati na popularnosti.

MEAN arhitektura (MongoDB, Express, AngularJS, Node.js) je jedna od ranih arhitektura otvorenog kôda koji je ojačao pomak prema aplikacijama s jednom stranicom

i prihvaćanju nerelacijskih baza podataka. AngularJS razvojno je okruženje temeljeno na modelu dizajna model-pogled-kontroler, a učinio je MEAN arhitekturu stabilnijom. MongoDB, vrlo popularna nerelacijska baza podataka, korištena je za trajno pohranjivanje podataka. Node.js okruženje, Express, te *web*-poslužitelj izgrađen na Node.js-u, formirali su srednji sloj ili *web*-poslužitelj. MEAN arhitektura se danas široko koristi te je jedan od popularnijih korištenih tehnologija za razvoj skalabilnih *web*-aplikacija.

Razvoj *web*-aplikacija nije ono što je bilo prije nekoliko godina. Danas postoji mnogo izbora i mogućnosti, no mnogi ne znaju koja je prava tehnologija za određenja rješenja. Iz tog razloga za razvoj *web*-aplikacija idealna je tehnologija MERN-a koja sadrži više razvojnih okruženja (Subramanin, 2017).

5.2. NoSQL

U posljednjih 15 godina dogodilo se veliko proširenje *weba*, društvenih mreža, *web*-obrazaca koji se moraju popuniti te veća povezanost s internetom, što znači da se nizovi podataka češće koriste.

Kao rezultat toga, dizajneri sustava više ne mogu po nekoliko godina provesti osmišljavajući sustave za obradu novih podataka. Umjesto toga oni moraju brzo stvoriti sustave koji pohranjuju podatke i informacije koje se lako pretražuju i analiziraju. Sve to znači da je potrebna posebna tehnologija sustava. Dobra je vijest da ogroman niz ovakvih sustava već postoji u obliku nerelacijskih baza podataka, no mnogi ljudi ne razumiju što rade nerelacijske baze podataka te ne znaju zašto i kako ih koristiti (Fowler, 2015).

NoSQL je *nerelacijska baza podataka*, bez obzira na to što ima akronim. Nerelacijska baza podataka nije uobičajena baza podataka koja ima tablice i stupce (koji se nazivaju relacijska baza podataka). Postoje dva atributa nerelacijskih baza podataka koji ga razlikuju od konvencionalnog. Prva je mogućnost horizontalnog skaliranja podataka distribucije opterećenja na više poslužitelja. Nerelacijske baze podataka to čine žrtvujući važan aspekt tradicionalnih baza podataka, a to je snažna konzistentnost. Podaci nisu nužno konzistentni za kratke količine vremena kroz replike. Ustvari, vrlo malo aplikacija zahtijeva internetsko skaliranje, a ovaj aspekt nerelacijskih baza podataka vrlo rijetko dolazi u igru. Drugi važan aspekt je taj da NoSQL baze podataka nisu nužno relacijske baze podataka. Podaci se ne moraju planirati u obliku redaka i stupaca tablica. U MongoDB-u podaci se mogu planirati kao trajni podaci, onako kako ih je moguće vidjeti u aplikacijskom kôdu, tj. kao objekte ili dokumente. Taj korak pomaže programeru da izbjegne sloj prevođenja, pri čemu bi programer morao pretvoriti ili preslikati objekte kojima se kôd bavi u relacijske tablice. Takvi se prijevodi nazivaju slojevima objektnih relacijskih preslikavanja (engl. *Object relational mapping - ORM*) (Subramanin, 2017).

5.2.1. Povijest nerelacijskih baza podataka

Prvo su se pojavile relacijske baze podataka koje pružaju korisnu usporedbu za razumijevanje nerelacijskih baza podataka. Edgar F. Codd kreirao je relacijske baze podataka 1970. godine. Relacijska baza podataka raspoređuje podatke u retke i stupce povezivanjem određenog ključa za svaki red. Gotovo svi sustavi relacijskih baza podataka koriste strukturirani jezik upita (SQL). Oni su tradicionalno kontroliraniji sustavi i imaju ograničenu sposobnost prevođenja složenih podataka poput nestrukturiranih podataka. U skladu s tim, SQL sustavi intenzivno se koriste te su vrlo korisni za održavanje točnih transakcijskih zapisa, kao i za brojne druge slučajeve u organizacijama svih veličina.

Sredinom 1990-ih internet je stekao izuzetnu popularnost. To je dovelo do razvoja nerelacijskih baza podataka, često nazvanih NoSQL. Nerelacijske baze podataka mogu brzo prevesti kompleksne podatke.

Svaki je problem rezultirao vlastitim rješenjem i vlastitom nerelacijskom bazom podataka, zbog čega je nastao veliki broj novih baza podataka. Slično tome, postojeći proizvodi koji pružaju značajke nerelacijskih baza podataka otkrili su i usvojili njihove oznake, što otežava poslove arhitekata, službenika informatičke službe i IT kupaca jer je malo vjerojatno da jedna nerelacijska baza podataka može riješiti sve probleme u određenom poslovnom području (Fowler, 2015, str. 10).

5.2.2. Značajke NoSQL-a

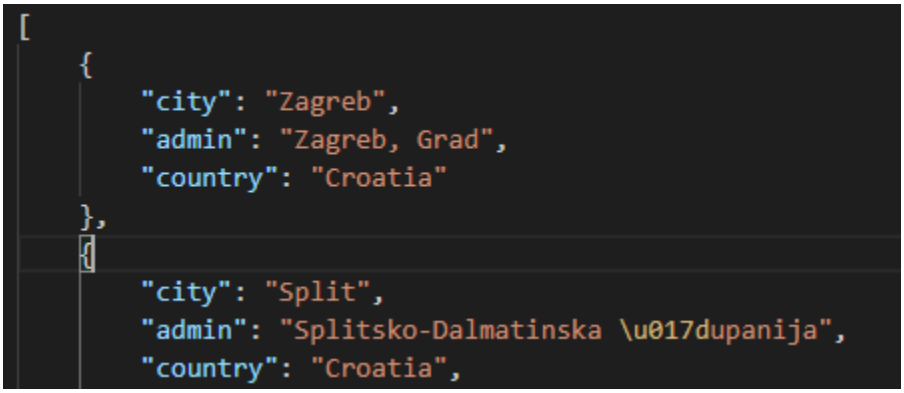
Četiri osnovne značajke nerelacijskih baza podataka prikazane su na sljedećem popisu, a odnose se na većinu nerelacijskih baza podataka. Popis uspoređuje nerelacijsku s tradicionalnom relacijskom bazom podataka:

- Shema - Shema baze podataka je opis svih mogućih podataka i struktura podataka u relacijskoj bazi podataka. Uz nerelacijsku bazu podataka, shema nije potrebna što daje slobodu za pohranu podataka bez izrade unaprijed dizajnirane sheme.

- Nerelacijska baza - Odnosi u bazi podataka uspostavljaju veze između tablica podataka. Naprimjer popis pojedinosti o transakciji može se povezati sa zasebnim popisom podataka o isporuci. Uz nerelacijsku bazu podataka, ove se informacije pohranjuju kao agregat (jedan zapis sa svime o transakciji).
- Hardver - Neke su baze podataka dizajnirane tako da djeluju najbolje sa specijaliziranim hardverom za pohranu i obradu. Uz nerelacijske baze podataka, mogu se koristiti jeftini poslužitelji koji nisu na raspolaganju. Dodavanje većeg broja jeftinih poslužitelja omogućava nerelacijskim bazama podataka obradu većeg broja podataka.
- Visoko distribuirano - distribuirane baze podataka mogu pohraniti i obraditi skup informacija na više uređaja. S nerelacijskom bazom podataka, skupina poslužitelja može se koristiti za držanje jedne velike baze podataka.

5.3. JSON

Mnogo *web*-stranica danas dijeli podatke koristeći JSON uz RSS i to s dobrim razlogom: JSON se može asinkrono učitati mnogo lakše nego što se može XML/RSS. JSON je kratica za JavaScript objektnu notaciju (engl. *JavaScript Object Notation*), a to je način za pohranu podataka u organiziranom poretku. Ukratko pruža čitljivu kolekciju podataka kojima se može pristupiti na jednostavan način.



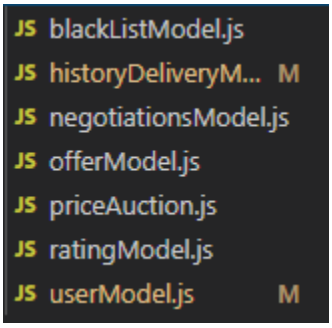
Slika 44. JSON objekt

Slika 44 prikazuje stvaranje JSON objekta koji sadrži bazu podataka hrvatskih gradova. JSON objekt koji sadrži bazu podataka gradova, služi za prikaz preporuka za vrijeme korisnikovog unosa u polja za početnu lokaciju i završnu lokaciju. Sve što se nalazi unutar vitičastih zagrada dio je JSON objekta. Unutar objekta može se deklarirati bilo koja druga varijabla pomoću uparivanja *name: value*, razdvojenog zarezima. Za pristup gradovima pohranjenim u JSON datoteci, može se jednostavno pristupiti umetanjem JSON-a na vrh programskog kôda.

Kada koristite MERN arhitekturu, objektno su prikazani unutar JSON-a. JSON oblik podataka nalazi se u nerelacijskoj bazi podataka, aplikacijskom poslužitelju i klijentu. U većini slučajeva štedi mnogo problema u pogledu transformacija. Nema objektnog relacijskog preslikavanja (ORM), nema potrebe prisiljavanja objekta u redove i stupce. Mapiranje objektnih dokumenata (ODM) kao što je Mangoose, može pomoći u provođenju sheme, ali suština je da se uštedi puno kôda za transformaciju podataka (Subramanin, 2017).

5.4. Kolekcije baze podataka

MongoDB pohranjuje dokumente u kolekcije. Kolekcije su analogne tablicama u relacijskim bazama podataka. Ako kolekcija ne postoji, MongoDB stvara kolekciju kada se prvi put pohranjuju podaci. Kolekcija prema zadanim postavkama ne zahtijeva da njeni dokumenti imaju istu shemu, tj. dokumenti u jednoj kolekciji ne moraju imati isti skup podataka, a vrsta podataka za polje može se razlikovati u različitim dokumentima unutar kolekcije. Da bi se promijenila struktura dokumenata u zbirci, kao što je dodavanje novih polja, uklanjanje postojećih polja ili promjena vrijednosti polja u novi tip, potrebno je ažurirati dokumente u novu strukturu.



Slika 45. Kolekcije projektnog zadatka

Iz slike 45 može se vidjeti šest modela koji se nalaze u projektnom zadatku, odnosno svaki model predstavlja jednu kolekciju. Prva kolekcija je *userModel* koja služi za autentifikaciju korisnika, a u kolekciji se skupljaju različite stvari poput *e-maila*, lozinke, itd. Sljedeća je kolekcija *ratingModel* koja služi za pohranjivanje podataka o ocjenama korisnika, primjerice kada pošiljatelj ocijeni kurira, njegova se ocjena zajedno s korisničkim imenom sprema u model za ocjenjivanje. Sljedeća je kolekcija *priceAuction* koja služi za ponudu cijene. Kada pošiljatelj pošalje ponudu kuriru, kurir dohvaća podatke iz modela *priceAuction*. Sljedeća kolekcija je *offerModel*, koja služi za kurire. Kada kurir ponudi svoju rutu, ona se sprema u model *offerModel* koji isto tako služi i za pretraživanje ruta. Ako pošiljatelj odluči poslati paket po kuriru i kada pošalje svoju ponudu, ta ponuda

se sprema u kolekciju *negotationsModel*. Sljedeći je model *historyDeliveryModel* u koji se spremaju sve usluge koje su izvršene i ocijenjene. Ako pošiljatelj nije zadovoljan uslugom te ako ocijeni kurira niskom ocjenom, moguće je odabrati opciju u kojoj aplikacija više ne spaja pošiljatelja s tim kurirom, a ako se odabere takva opcija onda se njihova korisnička imena spremaju u kolekciju *blackListModel*.

6. PROJEKTNI ZADATAK

6.1. Uvod

U svrhu diplomskog rada napravljena je *web*-aplikacija radi boljeg razumijevanja korištenja razvojnih okruženja kao što su React i Node.js. U projektu je korištena tehnologija MERN arhitekture, što znači da su se koristile različite tehnologije. MERN arhitektura, kao što je ranije spomenuto, koristi nerelacijsku bazu podataka. U projektu je korišteno razvojno okruženje MongoDB za bazu podataka, dok su ostale korištene tehnologije Express paket, React i Node.js. Aplikacija je u stvarnom vremenu.

Aplikacija koristi većinom gotove pakete što je prednost Reacta i Nodea. Jednostavnom komandnom linijom u terminalu uključeni su određeni paketi, poput paketa Express koji vodi računa o poslužitelju. Aplikacija također ima ugrađenu autentifikaciju, a korisnik se mora registrirati kako bi mogao koristiti aplikaciju. Kada se korisnik registrira, poslužitelj provjerava postoji li već takav korisnik u bazi, a ako postoji korisnik, aplikacija korisnika traži unos novog računa.

Za više detalja projekt se može pronaći na linku: https://drive.google.com/open?id=1U2VmuJ_3GlZ5BntpOzaz9YJOtFbRtlJN

6.2. Problem koji aplikacija rješava

Slanje kurira zahtijeva mnogo vremena te je potrebno najmanje četiri do pet dana za dostavu paketa. Mnogo ljudi putuje različitim lokacijama na kojima bi mogli prenositi različite predmete, poput paketa ili pisma. Aplikacija pomaže spojiti kurire i pošiljatelje kako bi stupili u kontakt te kako bi se dogovorili oko slanja pošiljke. Aplikacija rješava problem ljudi koji putuju, tako da oni mogu dodatno zaraditi ili si pokriti troškove putovanja. Isto tako rješava problem pošiljatelja, tako da ne moraju zvati kurirsku službu i odlaziti u njihove prostorije kako bi ostavili paket, a u ovom je slučaju moguć kurirov dolazak na lokaciju pošiljatelja i preuzimanje paketa. Kurir može prenositi paket na razne načine, bilo da putuje motociklom, autobusom ili avionom. Naprimjer ako kurir putuje iz Zagreba za

Rijeku i prolazi kroz Delnice, kurir može preuzeti paket iz Zagreba te ga dostaviti u Delnice, Rijeku ili neku lokaciju koja se nalazi na putu do Rijeke. Određivanje cijene paketa odvija se tako da pošiljatelj ponudi cijenu za prijevoz paketa, a kurir može prihvatiti, odbiti ili ponuditi svoju cijenu, sve dok se kurir i pošiljatelj ne dogovore za cijenu. Jednom kada se dogovore za cijenu, pošiljatelj može vidjeti kurirov broj, kao što kurir može vidjeti pošiljateljov te se dalje mogu dogovarati oko preuzimanja pošiljke.

6.3. Analiza konkurentnog tržišta

Za aplikaciju koja je kreirana u svrhu diplomskog rada postoji konkurentno tržište. Jedna konkurentna aplikacija je BlaBlaCar koja funkcionira na sličan način. Aplikacija BlaBlaCar spaja vozače i putnike, uz nadoplatu putnika, dok aplikacija u diplomskom radu spaja putnike i pošiljatelje. Aplikacija diplomskog rada ima prednost u odnosu na BlaBlaCar jer u slučaju da vozač putuje s punim automobilom ljudi i dalje može prevoziti pošiljke i dodatno zaraditi, dok putem aplikacije BlaBlaCar ne može dodatno zaraditi. Isto tako kurir ne mora nužno biti vozač, on može putovati vlakom ili autobusom te dodatno zaraditi, što nije moguće kod BlaBlaCar aplikacije.

6.4. Dodatne funkcionalnosti u odnosu na konkurentno tržište

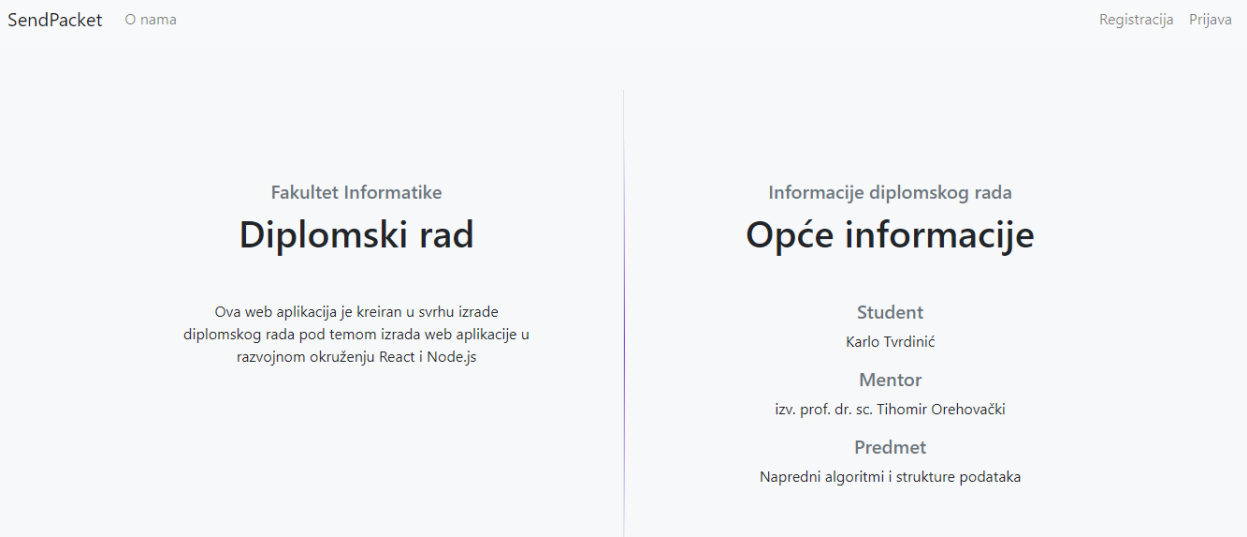
Aplikacija diplomskog rada ima dodatne funkcionalnosti koje druge aplikacije poput BlaBlaCara nemaju. Jedna od funkcionalnosti koja ističe aplikaciju u odnosu na druge je mogućnost dogovora cijene, dok u drugim aplikacijama vozač postavlja svoju cijenu koju ostali putnici moraju prihvatiti. U aplikaciji diplomskog rada pošiljatelj paketa prvo ponudi svoju cijenu te time obavještava kurira o svojoj ponudi, a kurir ima mogućnost prihvatiti ili ponuditi svoju cijenu. Mogu se dogovarati sve dok dogovor nije postignut. Isto tako aplikacija nudi dodatnu opciju u kojoj se može putovati i drugim prijevoznim sredstvima poput zrakoplova, vlaka, itd. U slučaju BlaBlaCara moguće je samo putovati vlastitim automobilom.

6.5. Funkcionalnosti projekta

Za dizajn aplikacije korišten je paket Bootstrap. Slika 46 prikazuje izgled početne stranice.

Slika 46. Početna stranica

Kao što se iz slike 46 vidi, stranica sadrži izbornik na gornjoj traci. Izbornik sadrži mogućnost odabira nekoliko kartica, a prva je kartica logotip koja dovodi korisnika na početnu stranicu, dok je druga mogućnost *O nama* i prikazana je slikom 47. Na desnoj strani trake nalaze se *Prijava* i *Registracija*. Ako korisnik nije registriran i prijavljen, nije u mogućnosti koristiti pogodnosti aplikacije. Ako korisnik upiše u traku za adresu put prema stranicama koje su zaštićene u aplikaciji, a nije prijavljen, aplikacija automatski korisnika odvodi na stranicu za prijavu.

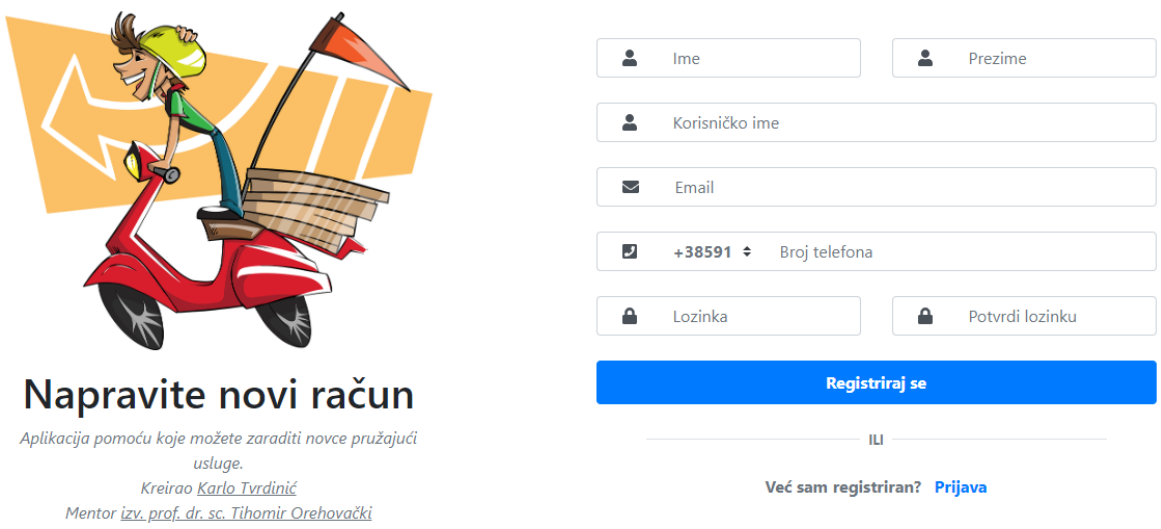


Slika 47. Kartica 'O nama' u aplikaciji

Kada korisnik pritisne na karticu *O nama* onda ga *web*-aplikacija preusmjeri na stranicu koja je prikazana na slici 47. Na stranici *O nama* napisane su osnove informacije vezane uz diplomski rad.

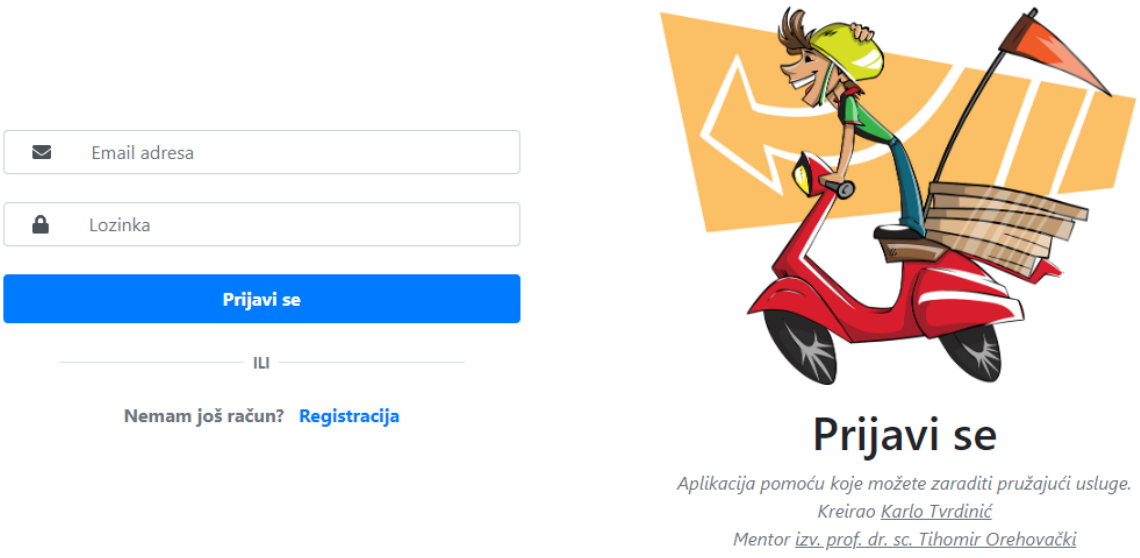
6.5.1. Autentifikacija

U aplikaciji je implementirana autentifikacija pomoću Node.js-a i MongoDB-a. Prikaz u aplikaciji omogućuje razvojno okruženje Reacta, te je zahvaljujući paketu Bootstrap prikaz moderno dizajniran. Prikaz registracijske forme moguće je vidjeti na slici 48. Kada korisnik ispuni formu te pritisne na gumb *Registriraj se*, daljnje akcije preuzima Node.js koji verificira podatke poslužiteljskoj strani. Jednom kada podaci stignu na poslužitelja i Node.js ga preuzme, provjerava se korisnik u bazi podataka. Node.js komunicira s MongoDB-om koji javlja povratno Node.js-u postoji li korisnik ili ne. Ako korisnik postoji, stranica obavještava korisnika o već postojećem računu te se proces ponavlja s drugim unosima. Jednom kada se korisnik registrirao, moguće se prijaviti te se nakon prijave mogu koristiti pogodnosti *web*-aplikacije. Prijavom u stranicu korisnik može koristiti pogodnosti aplikacije kao što je slanje paketa, a u suprotnom neće moći koristiti aplikaciju.



Slika 48. Forma za registraciju

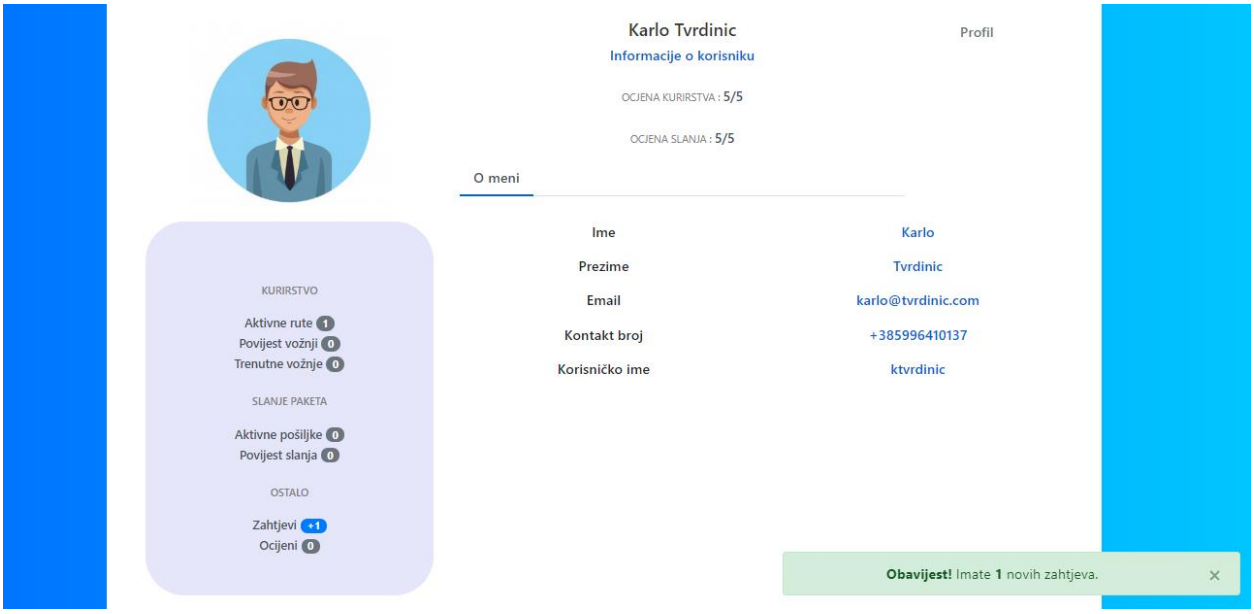
Ako korisnik nije ispunio sva polja, aplikacija odmah upozorava korisnika, kako za prijavu tako i za registraciju. Validacija za provjeru polja vrši se na klijentskoj strani, odnosno u Reactu, dok se postojanost korisnika obrađuje na poslužiteljskoj strani. Kada se korisnik registrira, moguća je prijava koja je prikazana slikom 49. Validacija se kod prijave vrši na klijentskoj i poslužiteljskoj strani.



Slika 49. Prikaz prijave

6.5.2. Korisnički profil

Većina ozbiljnih aplikacija koriste korisničke profile kako bi korisnik imao sve informacije na jednom mjestu. Tako je i u diplomskom radu implementiran korisnički profil u kojem se nalaze sve informacije. Slika 50 prikazuje kako je dizajniran korisnički profil.



Slika 50. Korisnički profil

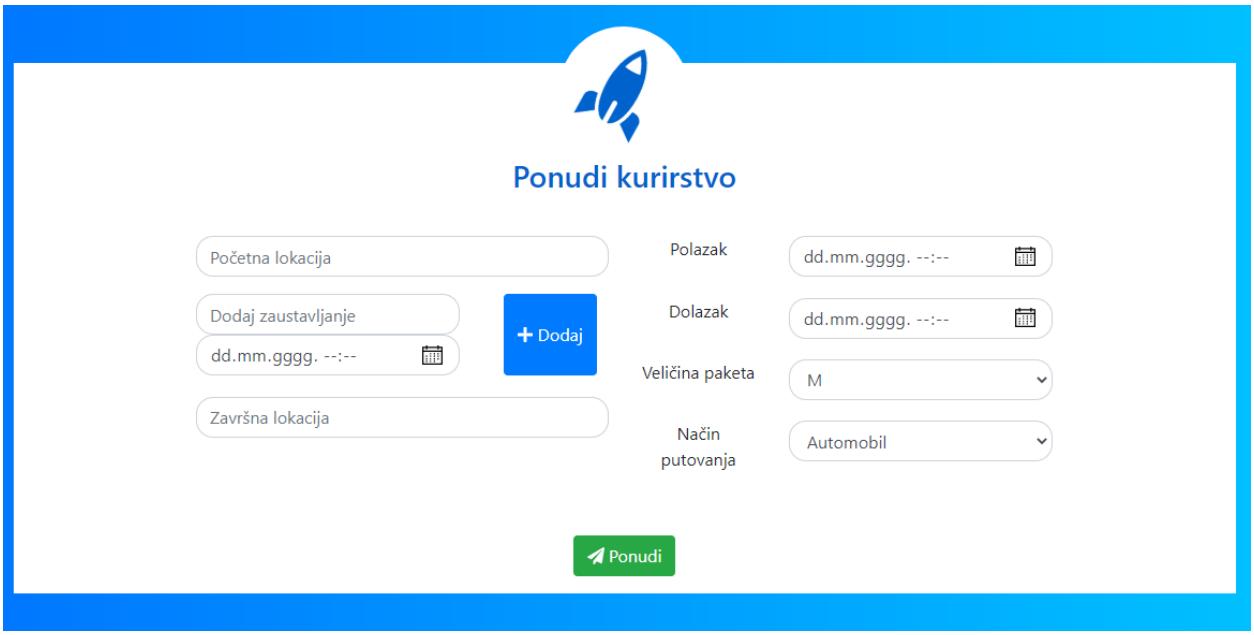
Na korisničkom profilu nalaze se informacije o korisniku, a s lijeve je strane izbornik koji prikazuje aktivne rute te zahtjeve koje korisnik dobiva za dostavljanje paketa. Svaki korisnik može pružati usluge kao kurir te slati paket po kuriru. Na početku korisničkog profila može se vidjeti ocjena kao kurira, te ocjena kao pošiljatelja. Korisniku je važno brinuti o ocjeni jer će to moći vidjeti svi budući korisnici koji žele koristiti usluge. Jednom kada pošiljatelj ocjeni kurira, kuriru će biti vidljivo na vrhu korisničkog profila trenutna ocjena.

Izbornik s lijeve strane ima tri grupe. Prva je grupa namijenjena kuririma, druga grupa pošiljateljima, a treća je grupa namijenjena za oba korisnika. Jednom kada korisnik postavi svoju rutu, pod karticom *Aktivne rute* moći će vidjeti svoju vožnju koju je moguće i otkazati.

Kada kurir postavi vožnju, ta je vožnja vidljiva svim korisnicima. Jednom kada jedan od korisnika pošalje zahtjev za slanjem pošiljke, kuriru dolazi zahtjev i obavijest na korisnički profil.

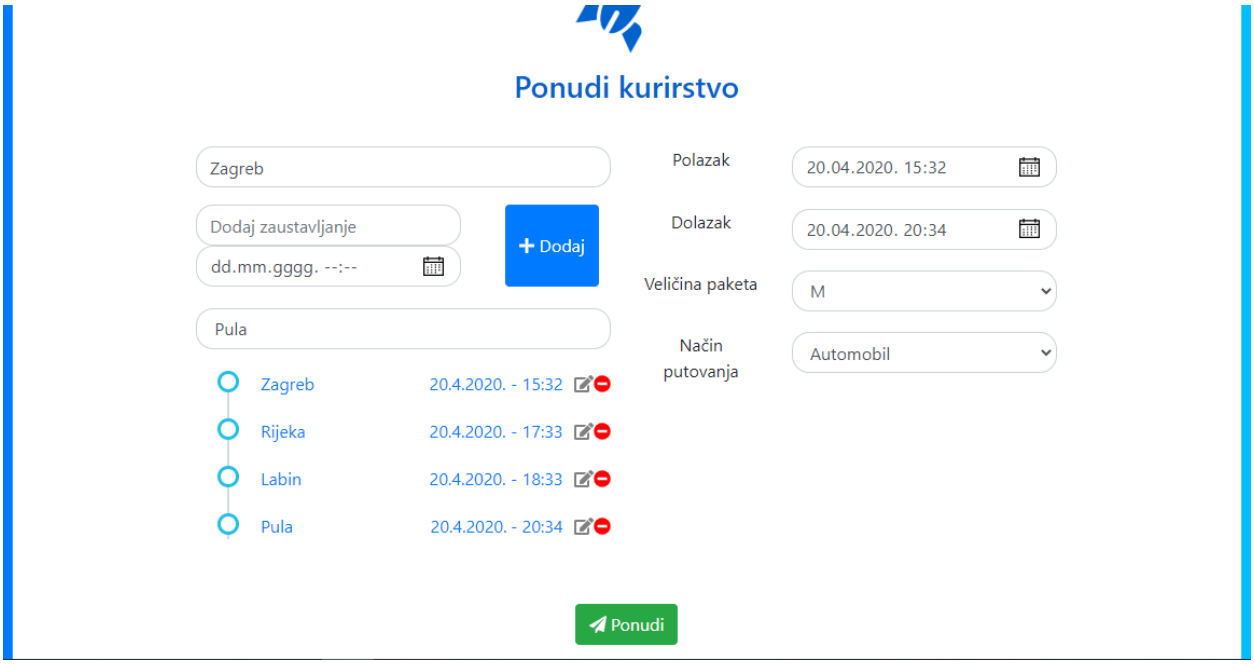
6.5.3. Kreiranje putovanja

Kada se korisnik prijavi u aplikaciju, moguće je postaviti svoje putovanje. Potrebno je popuniti sva polja jer u suprotnom aplikacija neće poslati zahtjev poslužitelju. Forma za kreiranje putovanja nalazi se na slici 51. Prvo polje koje se mora ispuniti je *Početna lokacija*, naprimjer ako se putuje od Zagreba do Pule, potrebno je u prvom polju upisati Zagreb. Drugo polje za ispunjavanje je *Polazak,* odnosno vrijeme polaska iz Zagreba, a implementirana je i provjera datuma. Ako korisnik postavi datum koji je prošao, aplikacija ga upozorava. U sljedeće polje trebati dodati zaustavljanje i vrijeme zaustavljanja na pojedinim lokacijama. Sljedeće je polje *Završna lokacija*, odnosno Pula. Kada se upiše završna lokacija potrebno je postaviti vrijeme dolaska u završnu lokaciju. Također je potrebno unijeti veličinu paketa koju korisnik može preuzeti, te naposljetku način putovanja.



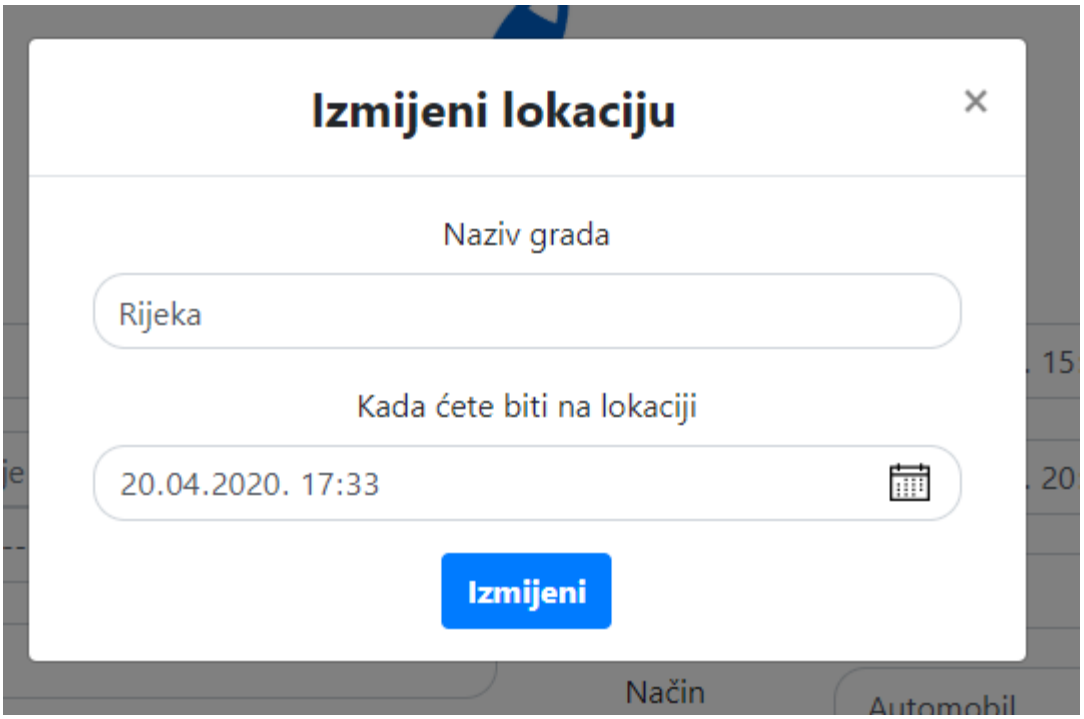
Slika 51. Forma za kreiranje putovanja

Slika 52 prikazuje kreiranje putovanja s ispunjenom formom za insertiranje.



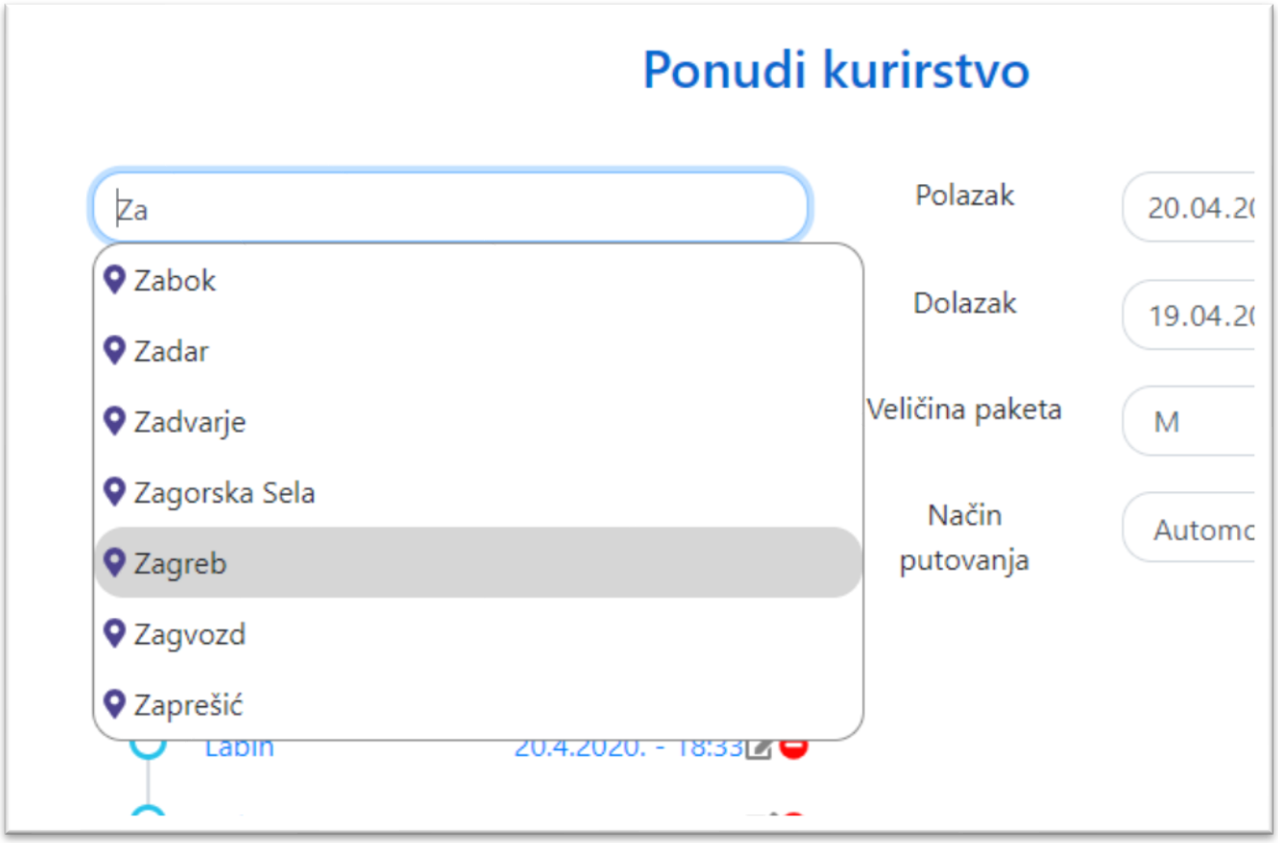
Slika 52. Ispunjena forma za kreiranje putovanja

Iz slike 52 na dnu s lijeve strane mogu se vidjeti četiri točke, a te točke predstavljaju gradove kroz koje dostavljač prolazi. Ako kurir nije zadovoljan pojedinim unosom, implementirana je mogućnost brisanja pojedinog grada te isto tako ispravak pojedinog grada. Ako kurir obriše početni grad Zagreb, implementirana je funkcionalnost tako da se postavlja sljedeći grad kao početni grad, u ovom slučaju početni grad je Rijeka. Isto tako vrijedi i za završnu lokaciju. Ako kurir obriše završnu lokaciju, onda prethodna lokacija postaje završna.



Slika 53. Modalni prozor izmjene lokacije

Slika 53 prikazuje modalni prozor izmjenjivanja pojedine lokacije. Kada kurir nije zadovoljan sa svojim unosom, jednostavnim klikom ga može prepraviti. Kada pritisne na gumb za ispravak lokacije, dobije mogućnost ispravka naziva grada i vremena dolaska na lokaciju.

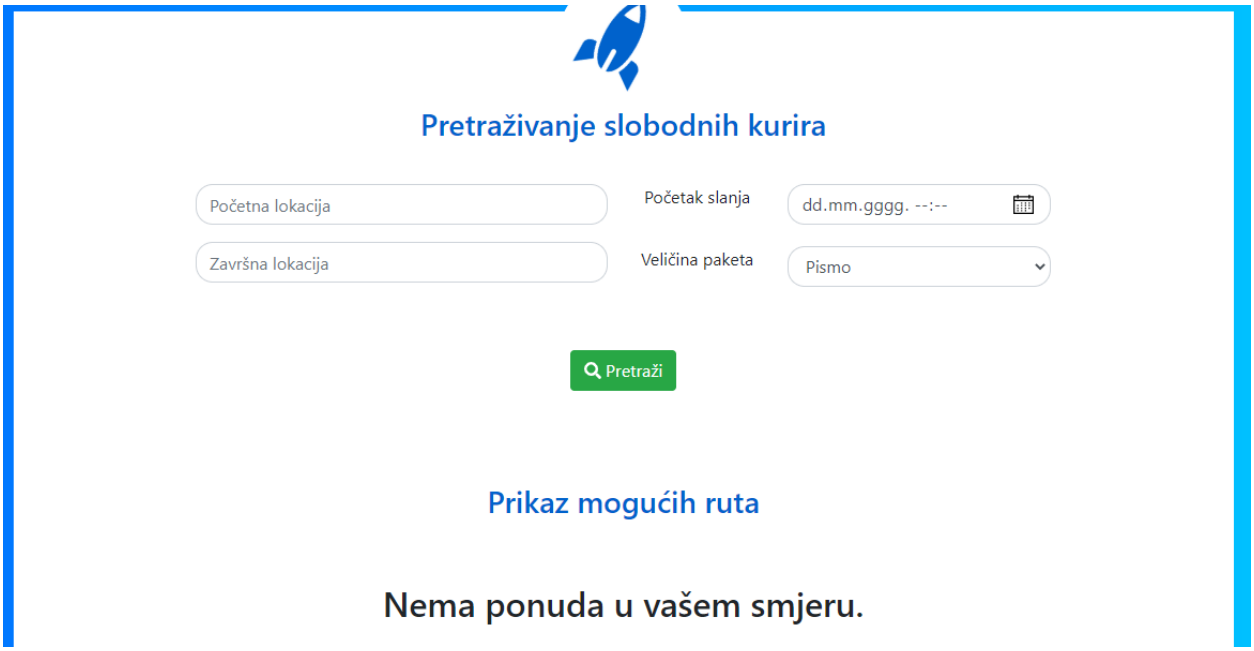


Slika 54. Samoispunjavanje lokacije

Aplikacija također ima implementirano samoispunjavanje forme gradova, što znači da aplikacija ima bazu podataka svih hrvatskih gradova. Kada korisnik krene upisivati naziv grada, svako slovo šalje zahtjev bazi podataka u kojoj dohvaća podatke o nazivima gradova. Jednom kada kurir kreira svoje putovanje, ostalim korisnicima je moguće poslati paket po navedenom putovanju. Samoispunjavanje forme gradova moguće je vidjeti na slici 54.

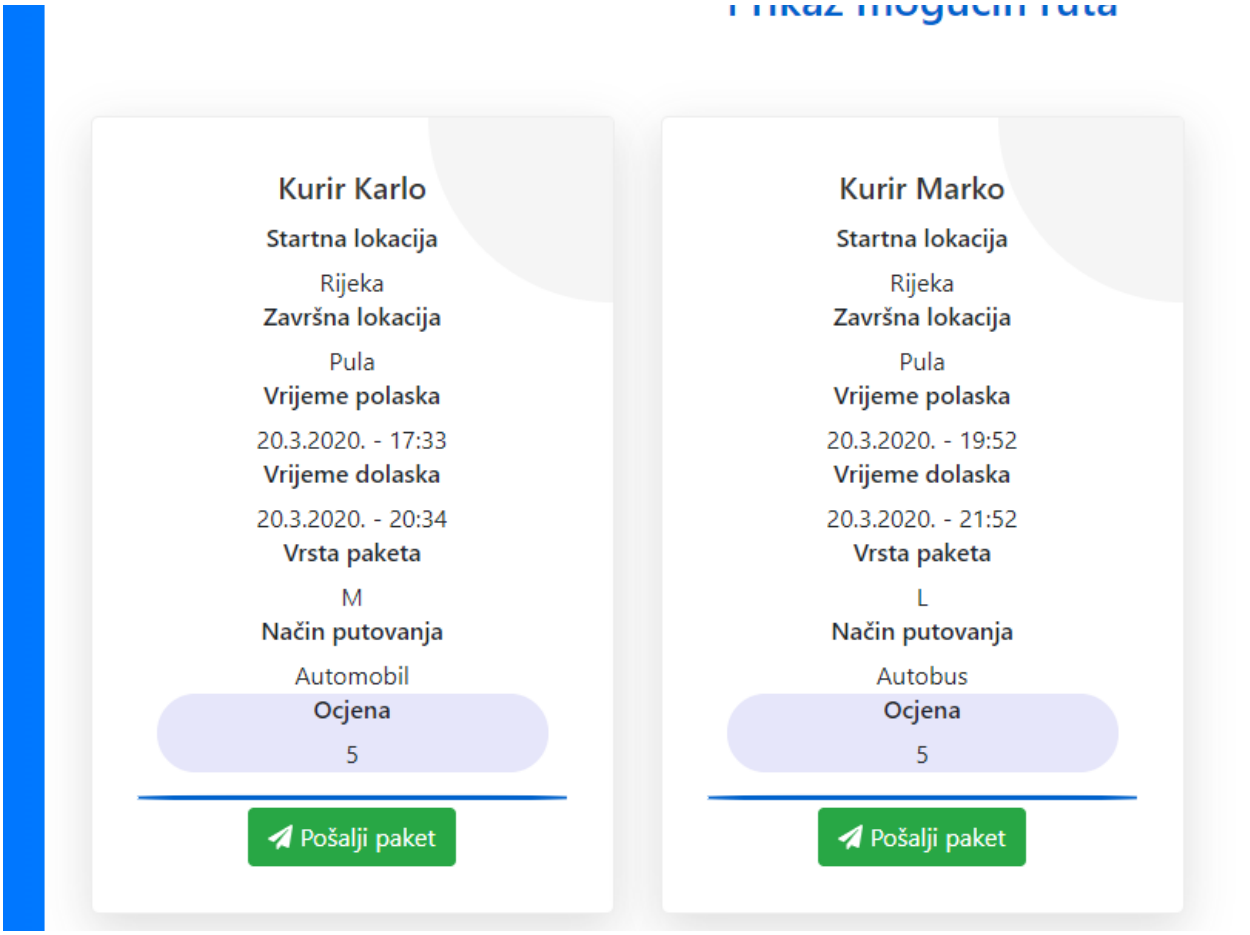
6.5.4. Pretraživanje putovanja

Kada se korisnik prijavi, moguće je pretraživanje dostupnih kurira u navedenom smjeru. Slika 55 prikazuje dizajn forme za pretraživanje kurira.



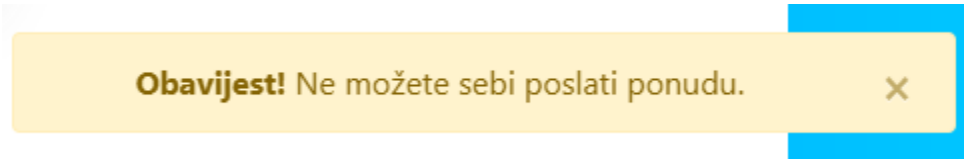
Slika 55. Forma za pretraživanje kurira

Slika 55 ima četiri polja za unos podataka. Prvo je polje unos početne lokacije, dok drugo polje čini vrijeme kretanja iz početne lokacije. Slijede polja pod nazivom *Završna lokacija* i *Veličina paketa*. Ako je kurir naveo veličinu paketa M, onda je moguće dobiti kurira samo ako korisnik šalje paket veličine M ili manji paket, no nije moguće dobiti kurira ako korisnik navede paket veći od veličine M.



Slika 56. Prikaz mogućih ruta

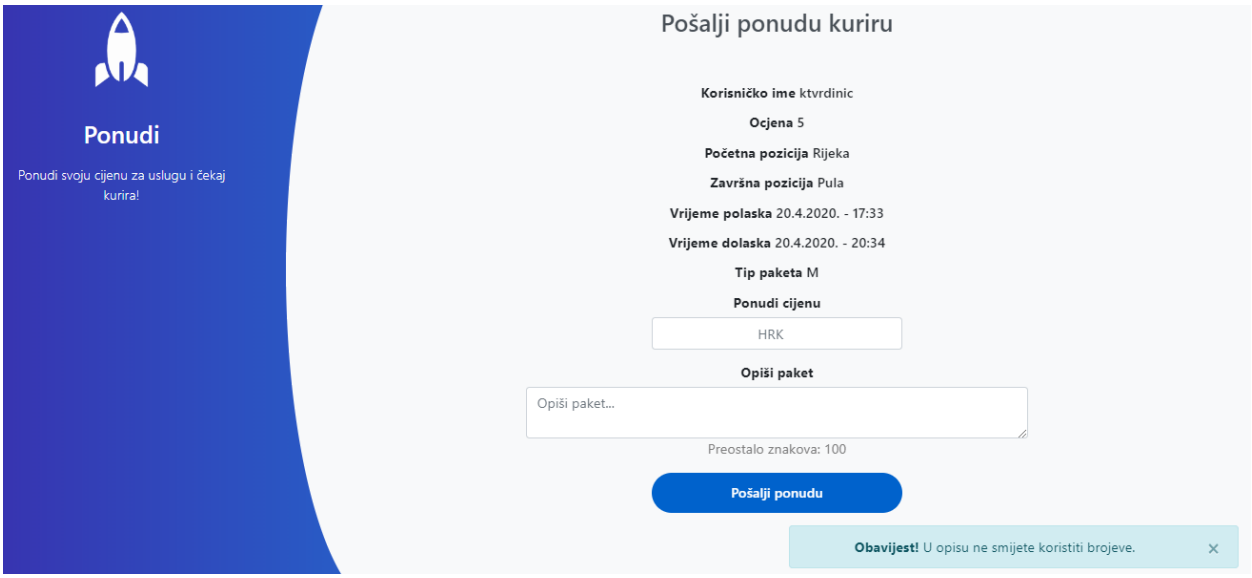
Kada se ispuni forma, aplikacija prikaže sve kurire koji idu u navedenom smjeru. U ovom slučaju forma je ispunjena tako da je početni grad Rijeka, a završni grad Pula. Rezultat je vidljiv na slici 56. U slučaju da je korisnik ponudio rutu, onda on ne može odabrati samog sebe za slanje paketa. Ako korisnik pokuša poslati paket, a da je on istu tu rutu ponudio, aplikacija ga upozori. Upozorenje je prikazano slikom 57.



Slika 57. Upozorenje vlastite rute

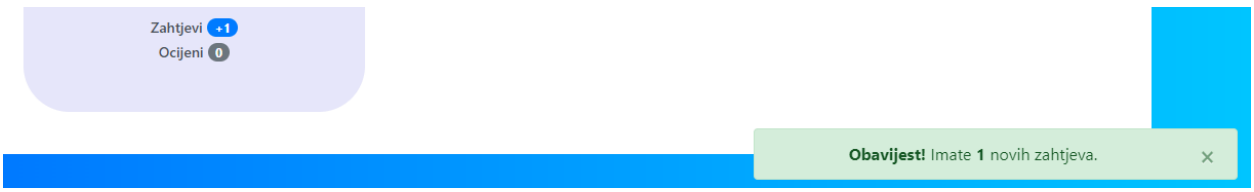
6.5.5. Ponuda cijene

Jednom kada se odabere ruta kojom se šalje paket, aplikacija korisnika navede na stranicu za ponudu cijene koja je prikazana na slici 58. Stranica sadrži osnovne informacije o putovanju i kuriru. Kada korisnik ponudi cijenu može i opisati svoj paket kako bi kurir dobio više informacija. U opisu je zabranjeno koristiti brojeve, kako korisnici ne bi poslali svoj kontakt broj u opisu. Razlog tomu je što korisnici međusobno dobiju kontakt brojeve tek kada jedan od korisnika prihvati ponudu. Ako korisnik pokuša poslati svoj kontakt broj, aplikacija ga upozori. Također opis je limitiran na sto znakova te je ispod opisa prikazan brojač znakova.



Slika 58. Stranica za ponudu cijene

Kada pošiljatelj pritisne na gumb za slanje ponude, kurir koji je ponudio rutu, dobije obavijest za novim zahtjevom koji je prikazan slikom 59.



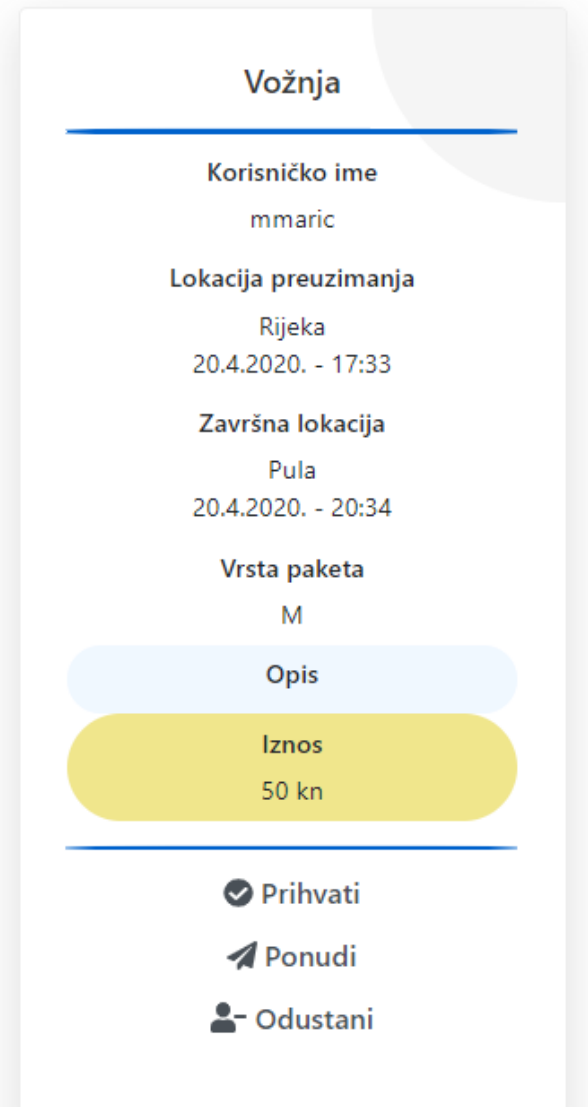
Slika 59. Obavijest za novim zahtjevom

6.5.6. Zahtjevi

Kada kurir dobije novi zahtjev za slanje pošiljke, moguće je provjeriti taj zahtjev pod karticom *Zahtjevi* koja je prikazana slikom 60. Kartica *Zahtjevi* nalazi se na korisničkom profilu.

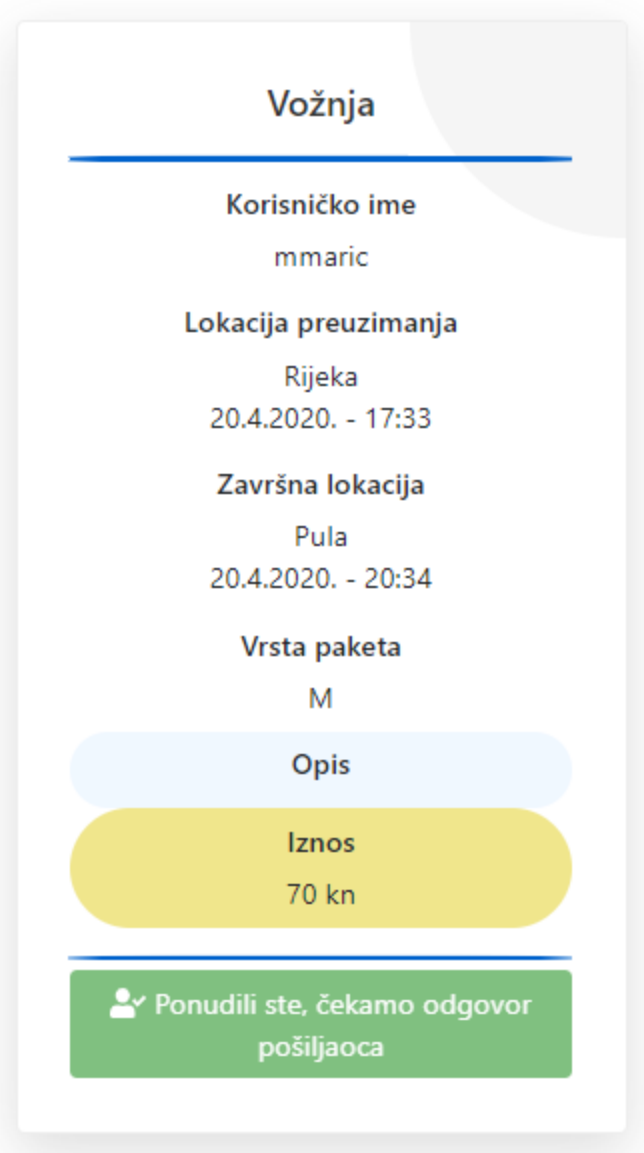
Slika 60. Prikaz novog zahtjeva

Kada kurir pritisne na karticu *Zahtjevi* moguće je vidjeti sve zahtjeve koji su upućeni prema prijavljenom kuriru. Slika 61 prikazuje zahtjev koji je kurir dobio. Kuriru su na kartici vidljive osnovne informacije o pošiljatelju, njegova ocjena te opis paketa, ukoliko on postoji. Ponuđena cijena je istaknuta drugom bojom kako bi je kurir lakše uočio. Moguće je odabrati tri opcije. Ako je kurir zadovoljan ponudom može odabrati opciju *Prihvati*, a ako kurir nije zadovoljan ponudom te smatra da njegova usluga vrijedi više ili možda manje, onda kurir može ponuditi svoju cijenu. Ako pak kurir nije zadovoljan i ne želi pružiti uslugu navedenom korisniku, moguće je odustati.



Slika 61. Prikaz opcije zahtjeva

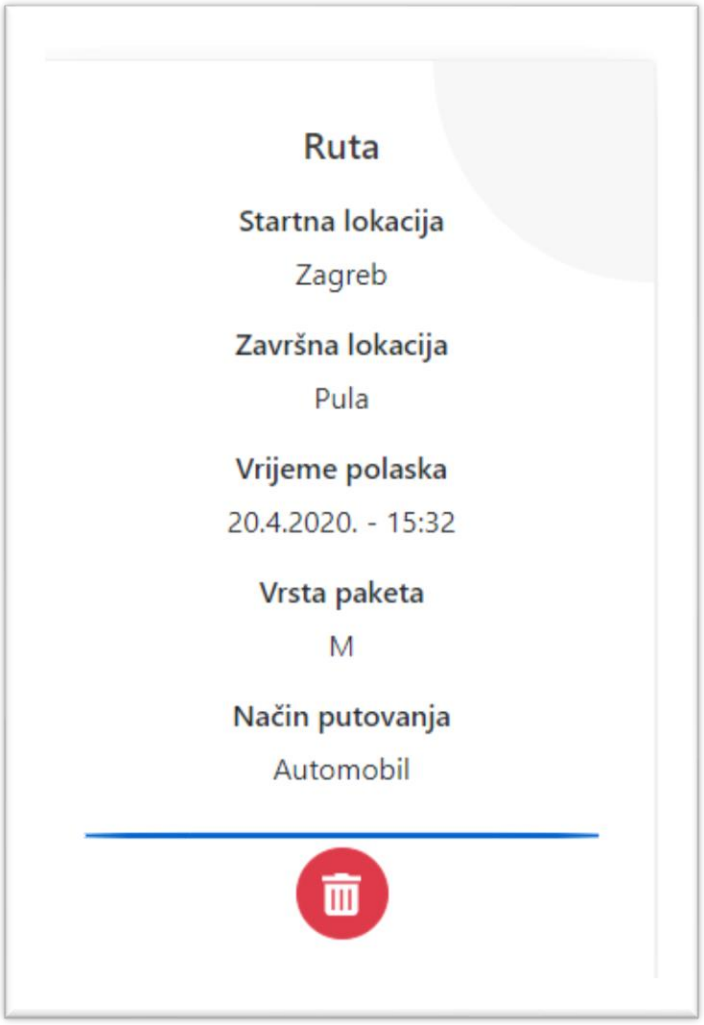
Slika 62 prikazuje karticu u koju je kurir ponudio svoju cijenu. Na dnu kartice nalazi se obavijest za kurira ili pošiljatelja, ovisno o tome tko je trenutno ponudio svoju cijenu.



Slika 62. Prikaz ponuđenog zahtjeva

6.5.7. Aktivna ruta

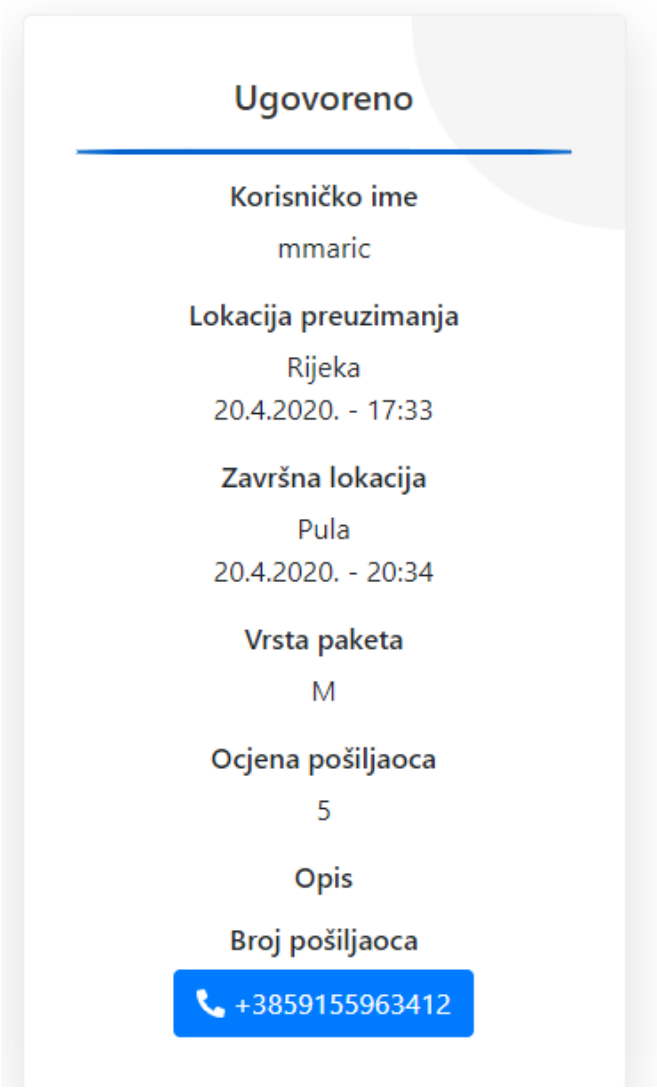
Kada kurir ponudi svoju rutu, on može provjeriti rute te ih naknadno obrisati ako je došlo do promjena. Slika 63 prikazuje rutu koju je kurir naveo, a ta je ruta od Zagreba do Pule. Na dnu kartice nalazi se dugme za brisanje ruta, no ako se navedena ruta obriše, svi korisnici koju šalju paket po toj ruti ostat će bez kurira. Do stranice s aktivnim rutama potrebno je doći putem korisničkog profila.



Slika 63. Prikaz rute

6.5.8. Prihvaćen zahtjev

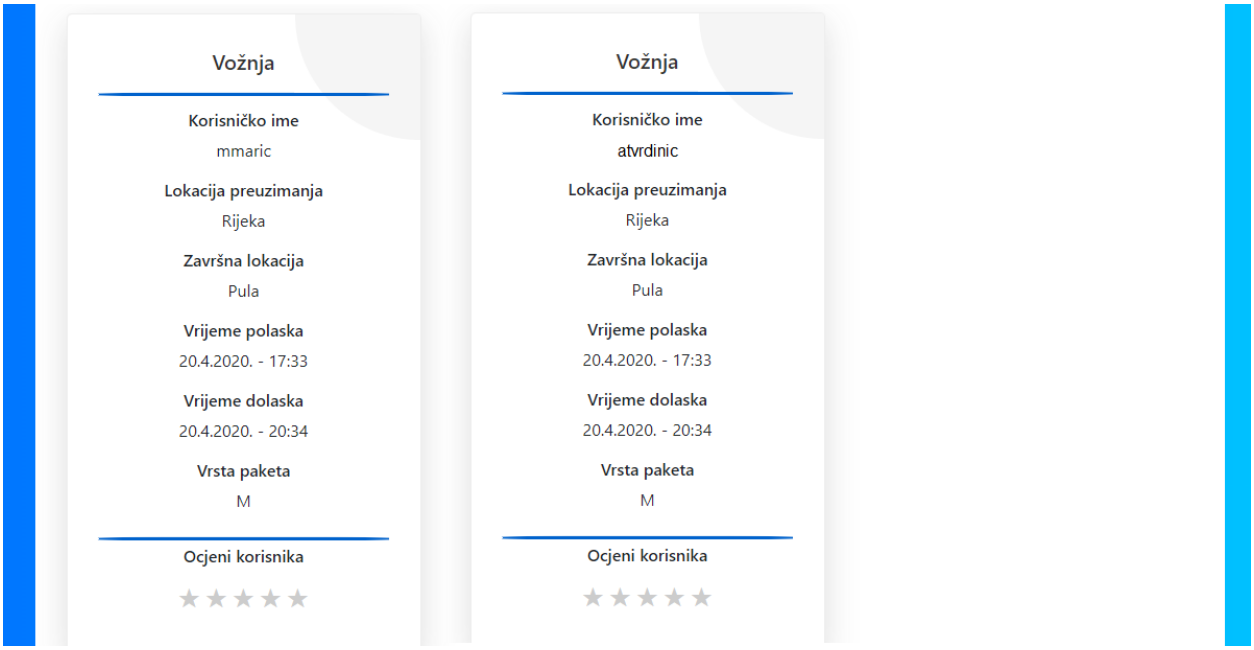
Kada kurir ili pošiljatelj prihvate zahtjev, onda će kurir moći vidjeti kontaktni broj pošiljatelja, kao i pošiljatelj kurirov. Slika 64 prikazuje kako izgleda kartica prihvaćenog zahtjeva. Do prikaza prihvaćenih zahtjeva dolazi se putem korisničkog profila.



Slika 64. Prihvaćen zahtjev

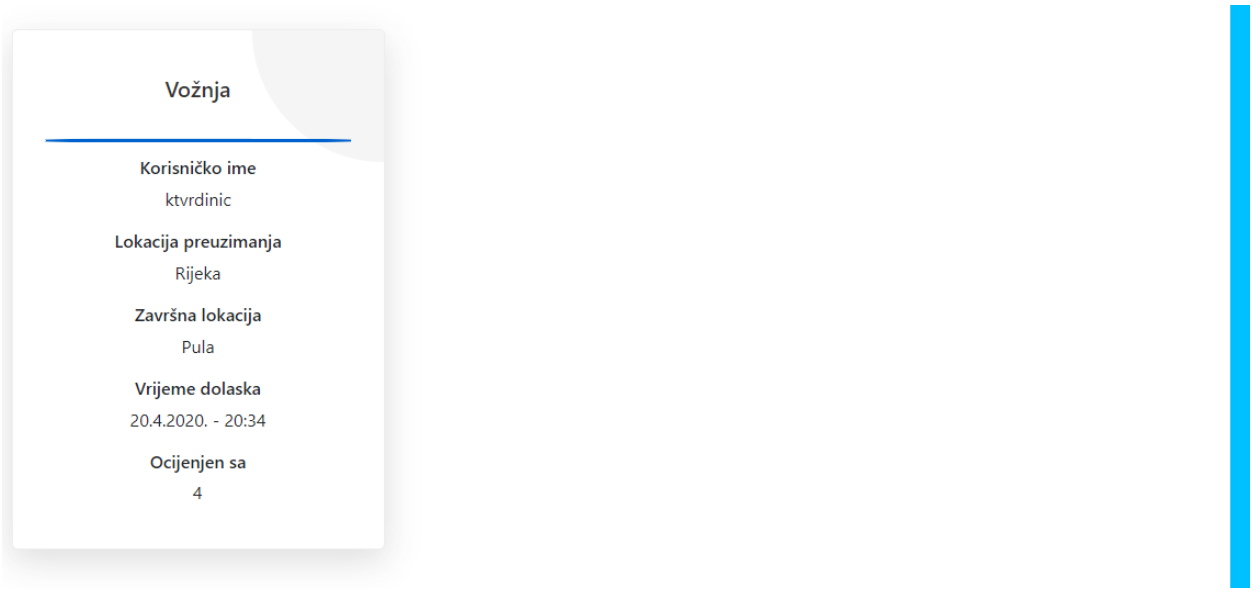
6.5.9. Ocjenjivanje

Kada prođe vrijeme dolaska, aplikacija automatski prebacuje pružanje usluge korisniku za ocjenjivanje. Korisnik će dobivati obavijest o ocjenjivanju usluge sve dok se isto ne izvrši. Ako korisnik nije zadovoljan uslugom te ju ocijeni ocjenom dva ili jedan, implementirana je mogućnost u kojoj korisnik odabire želi li ponovno koristiti tu uslugu od te osobe. Slika 65 prikazuje stranicu za ocjenjivanje korisnika.



Slika 65. Ocjenjivanje korisnika

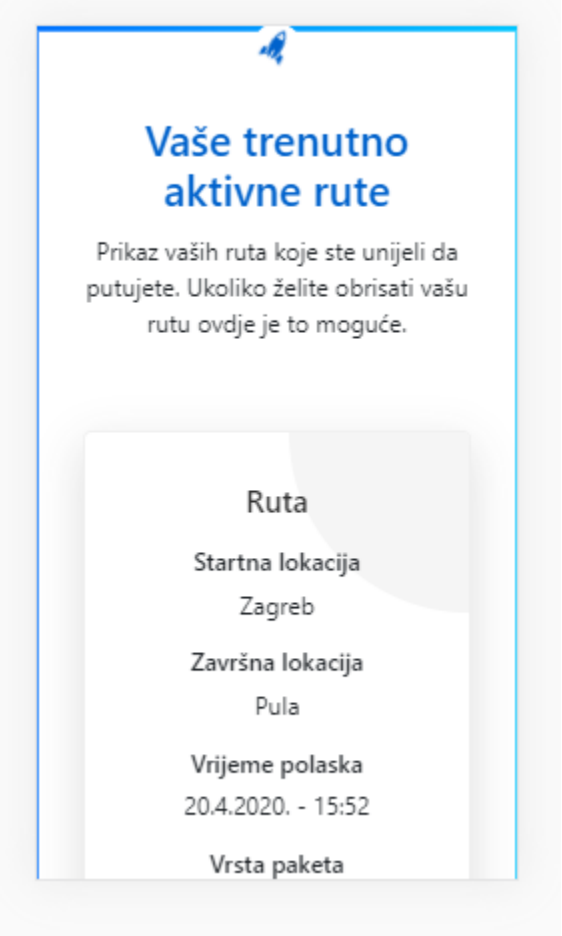
Kada korisnik ocijeni uslugu, ta se usluga sprema u kurirovu povijest vožnji te u pošiljateljevu povijest pošiljki. Prikaz povijesti vožnji moguće je vidjeti na slici 66.



Slika 66. Ocijenjen korisnik

6.5.10. Responzivna aplikacija

Gotovo svaki novi klijent ovih dana želi mobilnu verziju svoje *web*-stranice. Iz tog je razloga aplikacija napravljena responzivno, tako da se može prilagoditi svim rezolucijama zaslona. Responzivni *web*-dizajn pristup je koji poručuje da dizajn i razvoj trebaju odgovarati ponašanju i okruženju korisnika na temelju veličine zaslona, platforme i orijentacije. Prikaz responzivne stranice s rutama može se vidjeti na slici 67.



Slika 67. Prikaz responzivne stranice

7. Zaključak

U svrhu diplomskog rada napravljen je projektni zadatak kako bi se čitatelju pokazala moć kombiniranja Node.js-a i Reacta. Njihovim je kombiniranjem jednostavno izrađivati skalabilne *web*-aplikacije. Programer samo treba postaviti određenu arhitekturu kao što je baza podataka, a u ovom slučaju to je MongoDB. Prednost Node.js-a u odnosu na druge jezike također je spomenuto u diplomskom radu, no jedan je od glavnih razloga taj što se Node.js koristi za skalabilne *web*-aplikacije, no ni on se ne mora nužno koristiti za *web*-aplikacije, već može imati i druge funkcije koje ga čine omiljenim. Za razliku od Node.js-a, PHP se koristi samo za komunikaciju i slanje podataka između klijenta i poslužitelja. Isto tako React je vrlo jednostavno razvojno okruženje za korištenje te da bi se kreirao određeni mehanizam, vrlo je vjerojatno da već postoji u paketima, a jednom ga se linijom kôda može uključiti i koristiti.

U diplomskom radu postoji kratki pregled JavaScripta te njegove osnovne sintakse, kako bi čitatelj dobio osnovno znanje o JavaScriptu. S osnovnim znanjem može se lakše sagledati način na koji Node.js tako učinkovito koristi jezik za stvaranje moćnih i brzih aplikacija. Većina programera treba imati instalirani Node.js jer se on danas sve više koristi, ne samo za izradu *web*-stranica, već ima i druge funkcije.

Mnogima prijelaz sa sinkronih ili blokirajućih ulaza i izlaza na asinkroni način programiranja predstavlja problem. Kod sinkronog načina treba se čekati da se svaka od dretvi dovrši prije nego što se prijeđe na sljedeći zadatak, dok kod asinkronog programiranja sve ide brže jer se jedna dretva može koristiti za više zadataka istovremeno. Asinkrono programiranje zahtijeva više mentalnog razmišljanja, odnosno razmišljanja o strukturi asinkronog kôda. Jednom kada se usvoji asinkrono programiranje, teško se vratiti na stari način sinkronog programiranja.

Diplomski rad pokriva veliku količinu novog područja i materijala. Korištenje novog područja ima mnogo prednosti u odnosu na stari način programiranja. Progameru je omogućeno više vremena kako bi se usredotočio na poslužitelja i ključne koncepte koji su potrebni za ugodno iskustvo uz Node.js. Također može provjeriti je li aplikacijsko programsko sučelje poslužitelja dobro organizirano i učinkovito.

U radu su predstavljeni moduli Node.js-a. Prikazano je na koji način su napisane, kako ih Node.js pronalazi za uključivanje i kako se pomoću Node paket menadžera može pronaći i instalirati modul. Moguće je napisati vlastite složene module s datotekama *package.json* i povezati ih preko vlastitih projekata ili objaviti za upotrebu drugih putem Node paket menadžera.

Predstavljeno je razvojno okruženje Express koje se koristi unutar Node.js-a, a također je spomenut i međusoftver. Moguće je vidjeti kako Node.js nije samo dobar za pisanje kôda *web*-aplikacija, također je moćan i zabavan kada je u pitanju pisanje komandnih linija za sinkrone aplikacije. Da bi se dovršio postupak programiranja JavaScripta u Node.js-u, vrijeme je da se skrene pažnja na testiranje skripti i aplikacija. Testiranje aplikacija i skripti Node.js-a jednostavno je i brzo s mnogim raspoloživim okvirima za testiranje koji su dostupni putem Node paket menadžera.

Jedan od najvećih poteza je taj što se React razlikuje od ostalih biblioteka jer koristi potpuno novi jezik pod nazivom JSX kojim bi definirao kako će izgledati vizualni prikazi. To je moguće vidjeti pri definiranju oznaka za naslov h1 unutar metode Render.

Utjecaj JSX-a nadilazi način na koji se definiraju elementi korisničkog sučelja. Također mijenja način izgradnje aplikacije u cjelini. Budući da preglednik ne može razumjeti JSX u njegovom izvornom prikazu, potrebno je konvertirati JSX u JavaScript. Jedan je od pristupa izgradnja aplikacije za generiranje prevedenog JavaScript izlaza koji odgovara JSX izvoru. Drugi pristup (onaj koji se ovdje koristi) je korištenje Babelove biblioteke za prevođenje JSX-a u JavaScript na samom pregledniku. Svakako bi programer trebao proučiti novu sintaksu JSX-a jer je jednostavna i korisna.

Zadatak ovog diplomskog rada bio je razvoj *web*-aplikacije za pružanje usluge i slanje paketa uz pomoć Reacta i Node.js-a razvojnog okruženja, a detaljno je opisan postupak kreiranja svih dijelova poslužitelja. Prikazana je snaga Reacta te jednostavno korištenje njegovih komandi. Ako se koristi HTML za izradu iste *web*-stranice, programer bi trebao potrošiti više vremena te vjerojatno ne bi dobio bolji rezultat nego što ga može postići u Reactu.

Ako programer treba brzo i moćno rješenje za izradu skalabilnih aplikacija u realnom vremenu, najbolji načini za to su korištenje Node.js-a koji se koristi na poslužiteljskoj strani te korištenje Reacta koji se koristi na klijentskoj strani.

8. Literatura

1. Banks A. & Porcello E. (2017), *Learning React – Functional Web Development with React and Redux*, <raspoloživo na: https://books.google.hr/books?id=ycTADgAAQBAJ&printsec=frontcover&dq=react&hl=hr&sa=X&ved=0ahUKEwiq2c6WqJzmAhVMtIsKHZb_Cz4Q6AEIJzAA#v=onepage&q&f=true>, [pristupljeno: 20.12.2019]

2. Brown E. (2014), *Web Development with Node & Express,* <raspoloživo na: http://www.vanmeegern.de/fileadmin/user_upload/PDF/Web_Development_with_Node_Express.pdf>, [pristupljeno: 11.01.2020]

3. Fowler A. (2015), NoSQL For Dummies, <raspoloživo na: http://index-of.es/Miscellanous/LIVRES/Wiley.NoSQL.Mar.2015.ISBN.1118905741.pdf>, [pristupljeno: 17.01.2020]

4. Risenman B. (2016), *Learning React Native – Building native mobile apps with JavaScript*, <raspoloživo na: https://books.google.hr/books?id=274fCwAAQBAJ&printsec=frontcover&dq=react&hl=hr&sa=X&ved=0ahUKEwiq2c6WqJzmAhVMtIsKHZb_Cz4Q6AEIMjAB#v=onepage&q&f=true>, [pristupljeno: 15.01.2020]

5. Sonpatki P & Vipul A M (2016), *ReactJS by Ecample – Building Modern Web Applications with React*, <raspoloživo na: https://books.google.hr/books?id=Ht3JDAAAQBAJ&pg=PA6&dq=react+js&hl=hr&sa=X&ved=0ahUKEwiDj5y7qJzmAhUp_SoKHVb_BnkQ6AEITDAE#v=onepage&q&f=true>, [pristupljeno: 05.01.2020]

6. Subramanin V. (2017), *Pro MERN Stack – Full stack web App Development with Mongo, Express, React, and Node*, <raspoloživo na: https://github.com/mgelster/React/blob/master/Vasan%20Subramanian-Pro%20MERN%20Stack_%20Full%20Stack%20Web%20App%20Development%20with%20Mongo%2C%20Express%2C%20React%2C%20and%20Node-Apress%20(2017).pdf>, [pristupljeno: 12.01.2020]

7. Wandschneider M. (2013), *Learning Node.js – A Headns-On Guide to Building Web Applications in JavaScript*, <raspoloživo na: http://index-of.co.uk/Cloud%20Technology/Learning%20Node.js%20A%20Hands-on%20Guide%20to%20Building.pdf>, [pristupljeno: 12.12.2019]

8. Young A. & Harter M. (2015), *Node.js in Practice – Includes 115 techniques*, <raspoloživo na: https://docs.google.com/viewerng/viewer?url=https://www.programmer-

books.com/wp-content/uploads/2018/09/Node.js-in-Practice.pdf>, [pristupljeno: 10.12.2019]

9. Young A. et al. (n.d.), *React JS Notes for Professionals,* <raspoloživo na: file:///C:/Users/Developer/Downloads/ReactJSNotesForProfessionals.pdf>, [pristupljeno: 05.01.2020]

10. Websoptimization (n.d.), *What is the difference between php and node js* [Web izvor], <raspoloživo na: https://www.websoptimization.com/blog/what-is-the-difference-between-php-and-node-js/>, [pristupljeno 10.001.2020]

11. Konsinski R. & Armstrong G. (n.d.), *The license and Security Risks of Using Node.js* [Web izvor], <raspoloživo na: https://dzone.com/articles/the-license-and-security-risks-of-using-nodejs>, [pristupljeno: 14.01.2020]

12. McKenzie S. et al. (2016), *Yarn: A new package manager for JavaScript* [Web izvor], <raspoloživo na: https://engineering.fb.com/web/yarn-a-new-package-manager-for-javascript/>, [pristupljeno: 15.01.2020]