RESTful web aplikacija za ticketing sustav

Birkić, Mateo

SVEUČILIŠTE U RIJECI

TEHNIČKI FAKULTET

Prijediplomski sveučilišni studij računarstva

Završni rad

RESTful web aplikacija za ticketing sustav

Rijeka, rujan 2023.

Mateo Birkić

0069090064

SVEUČILIŠTE U RIJECI

TEHNIČKI FAKULTET

Prijediplomski sveučilišni studij računarstva

Završni rad

RESTful web aplikacija za ticketing sustav

Mentor: doc.dr.sc. Marko Gulić

Rijeka, rujan 2023.

Mateo Birkić

SVEUČILIŠTE U RIJECI
TEHNIČKI FAKULTET
POVJERENSTVO ZA ZAVRŠNE ISPITE

Rijeka, 8. ožujka 2023.

Zavod: **Zavod za računarstvo**
Predmet: **Razvoj web aplikacija**
Grana: **2.09.06 programsko inženjerstvo**

ZADATAK ZA ZAVRŠNI RAD

Pristupnik: **Mateo Birkić (0069090064)**
Studij: Sveučilišni prijediplomski studij računarstva

Zadatak: **RESTful web aplikacija za ticketing sustav / RESTful web application for ticketing system**

Opis zadatka:

Razviti RESTful web aplikaciju za ticketing sustav. Aplikacija mora podržavati uslugu kreiranja ticketa, kao i njegovog dodjeljivanja pojedinom korisniku uz ažuriranje, brisanje ili zatvaranje ticketa. Također, potrebno je implementirati filtriranje, sortiranje i pretragu ticketa. Za razvoj poslužiteljskog dijela web aplikacije treba koristiti Laravel radni okvir uz proizvoljno odabran sustav za upravljanje bazama podataka. Također, treba koristiti jedan od Laravel paketa za realizaciju autentifikacije RESTful aplikacije. Za razvoj klijentskog dijela aplikacije treba koristiti React JavaScript knjižicu za razvoj korisničkog sučelja uz učinkovito renderiranje aplikacije na uređajima s različitim veličinama zaslona.

Rad mora biti napisan prema Uputama za pisanje diplomskih / završnih radova koje su objavljene na mrežnim stranicama studija.

Zadatak uručen pristupniku: 26. ožujka 2023.

Mentor:

Doc. dr. sc. Marko Gulić

Predsjednik povjerenstva za završni ispit:

Prof. dr. sc. Miroslav Joler

Izjava o samostalnoj izradi rada

Izjavljujem da sam samostalno izradio ovaj rad.

Rijeka, rujan 2023.

_____

Mateo Birkić

Zahvala

Zahvaljujem se mentoru doc. dr. sc. Marku Guliću na pruženom vremenu i svom strpljenju koje je pokazao prilikom izrade ovoga rada. Zahvaljujem se i svojoj obitelji te djevojci na pruženoj podršci tijekom studiranja.

SADRŽAJ

Popis slika

1 Uvod

U ovom radu predstavljena je RESTful web aplikacija za ticketing sustav Ticket.io koja služi za kreiranje tiketa i njihovo dodjeljivanje ostalim korisnicima kako bi se jednostavnije pratili zadatci koje je potrebno riješiti. Kako bi mogao pristupiti pregledu, kreiranju i obavljanju tiketa, svaki korisnik treba napraviti vlastiti račun unoseći svoje ime te pristupne podatke. Svakom korisniku dostupne su tri liste s tiketima preko kojih lakše može pronaći tiket koji mu je potreban. Tako se nude liste svih tiketa, lista kreiranih tiketa te lista dodijeljenih tiketa. Posljednje dvije liste personalizirane su za svakog korisnika dok je prva lista jednaka za sve registrirane, odnosno prijavljene korisnike. Kako bi se korisniku omogućilo lakše pretraživanje traženih tiketa nudi se mogućnost sortiranja prema datumu izrade i prioritetu, filtriranju prema tagovima ili pretraživanju izraza unutar tiketa. Autor tiketa u mogućnosti je i urediti tiket te ga dodijeliti drugom korisniku. Tiketi se nakon rješavanja mogu zatvoriti te se tada mijenja njihov prikaz unutar liste koji daje jasan indikator da je tiket riješen. Također, nepoželjne tikete korisnik je u mogućnosti obrisati.

Tehnologije korištene za razvoj web aplikacije podijeljene su na dvije kategorije, odnosno na tehnologije za razvoj klijentskog dijela aplikacije i razvoj poslužiteljskog dijela aplikacije. Aplikacija počiva na Representational State Transfer (skraćeno, REST) [1] arhitekturi koja se oslanja na razmjenu podataka putem standardnih HTTP [2] zahtjeva kao što su GET, POST, PUT i DELETE.

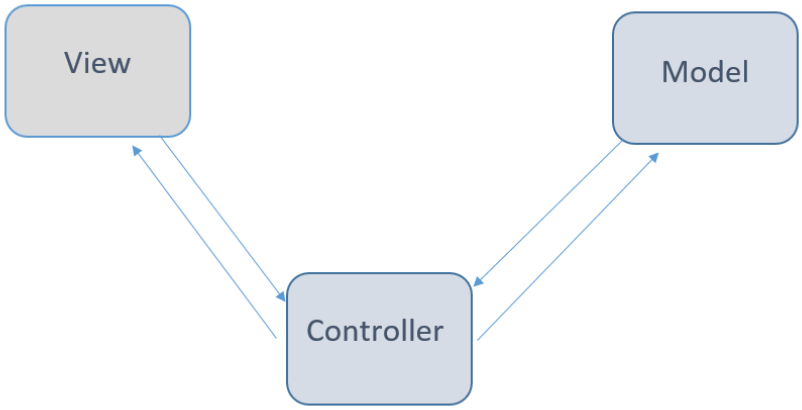
Poslužiteljski dio web aplikacije razvijen je u Laravelu [3], radnom okviru utemeljenom na PHP-u [4]. Laravel je besplatan radni okvir temeljen na MVC (Model-View-Controller) [5] modelu arhitekture. Omogućava jednostavnu implementaciju mnogih neizostavnih funkcionalnosti aplikacije poput autorizacije i autentifikacije. Baza podataka izrađena je u MySQL-u [6]. MySQL je sustav za upravljanje relacijskim bazama podataka, a koristi Structured Query Language (skraćeno SQL) upite za manipuliranje podatcima unutar entiteta odnosno tablica. Za razvoj klijentskog dijela web aplikacije korištena je JavaScript knjižnica otvorenog koda React.js [7] koja se temelji na ponovnoj iskoristivosti svojih komponenti. Za stiliziranje HTML komponenti

korišten je Tailwind CSS [8] radni okvir, koji se nakon instalacije u projekt može koristiti definirajući klase unutar pojedinih komponenti.

2 Opis tehnologija

2.1 Laravel

Laravel je besplatan radni okvir (eng. framework) otvorenog koda (eng. open source) temeljen na PHP-u, a nastao je 2011. godine pod vodstvom Taylor Otwella. Nastao je kao pokušaj da se stvori radni okvir s ugrađenom autentifikacijom kao i autorizacijom što prije nastanka Laravela nije bio slučaj. Temelji se na Model-Pogled-Kontroler (eng. Model-View-Controller, skraćeno MVC) arhitekturi, no to ipak nije bio slučaj u njegovom početku. Iako je Laravel imao podršku za Modele (eng. Models) i Poglede (eng. Views), nedostajala je podrška za Kontrolere (eng. Controllers) zbog čega se u početku nije smatrao MVC radnim okvirom. Tijekom godina to se promijenilo te je sada Laravel jedan od najkorištenijih PHP radnih okvira. Na Slici 2.1 slikovito je prikazana MVC arhitektura, gdje Model predstavlja podatke koji se koriste, Pogled predstavlja sadržaj koji se prikazuje korisniku, a Kontroler predstavlja poveznicu između Modela i Pogleda gdje se izvršava poslovna logika nad podatcima iz Modela koji se zatim prikazuju u Pogledu.



Slika 2.1 Prikaz MVC arhitekture

Korisnik je u direktnoj interakciji s Pogledom, bilo to klikom na neki gumb ili pregledom sadržaja te u tom trenutku Pogled šalje zahtjev za određenim podatcima, a Kontroler vrši logiku i obrađuje taj zahtjev te šalje dalje poziv prema Modelu za dohvat podataka. Model u konačnici šalje upit

prema bazi podataka te nakon što primi tražene podatke, predaje ih Kontroleru koji ih obrađuje i šalje u pogodnom obliku prema Pogledu koji nove podatke prikazuje korisniku [9].

2.1.1 Eloquent

Eloquent je Laravelov ugrađeni Object relational mapping (skraćeno ORM) koji uvelike olakšava interakciju s bazom podataka. Upiti prema bazi šalju se preko objekata i PHP sintakse, a ne preko klasičnih SQL upita kao što je to i inače slučaj. Eloquent automatski postavlja identifikator kao primarni ključ unutar svake tablice u bazi podataka te ga automatski inkrementira. Sintaksa za manipulaciju nad bazom podataka je jednostavna i intuitivna pa se tako koriste metode kao što su *get*, *create* i *update*. Na Slici 2.2 prikazan je primjer korištenja Eloquenta za novi unos u bazu podataka gdje se podatci unose u tablicu koja je jednaka imenu modela nad kojim se poziva metoda osim u slučaju ako korisnik nije specificirao drugačije. [9]

```
Tickets::create($formFields);
```

Slika 2.2 Eloquent metoda za unos u bazu podataka

2.1.2 Middleware

Middleware je još jedna korisna značajka Laravela jer omogućuje pregled i filtriranje HTTP zahtjeva aplikacijskog programskog sučelja (eng. application programming interface, skraćeno API) pristiglih na poslužiteljsku stranu web aplikacije. Uz pomoć *middlewarea*, utvrđuje se zadovoljava li korisnik uvjet kako bi mu se omogućio pristup poslužitelju, primjerice koristeći *middleware* prikazan na Slici 2.3 provjerava se je li korisnik autentificiran te ako nije njegov zahtjev se odbija, no ukoliko ipak je autentificiran tada mu se dopušta daljnji pristup poslužitelju [9].

```
Route::middleware(['auth'])->get('/mail', [TicketsController::class, 'getMail']);
```

Slika 2.3: Definiranje middlewarea unutar putanje

2.1.3 Sanctum

Sanctum je PHP knjižnica (eng. library) koja omogućava izradu autorizacije i autentifikacije web aplikacija. Koristi se zajedno s Laravelom za autentifikaciju API poziva koji pristižu s klijentskog dijela web aplikacija na poslužiteljski dio. Sanctum nudi dvije mogućnosti autentifikacije, a to su preko tokena te preko sjednice (eng. session).

Autentifikacija preko tokena odvija se na način da se prilikom prijave korisnika, upisujući podatke unutar polja za unos podataka (eng. input field) na klijentskom dijelu web aplikacije, podatci šalju na poslužitelj putem API poziva gdje se zatim provjeravaju i ukoliko su podatci validni generira se token. Taj se token zatim šalje prema klijentskom dijelu web aplikacije gdje se i pohranjuje. Korištenjem novonastalog tokena, za pristup zaštićenim resursima korisnik unutar zahtjeva šalje i pohranjeni token te se ukoliko je on ispravan, korisniku dopušta pristup resursima.

Autentifikacija preko sjednice koristi veoma sličan princip kao i kod autentifikacije tokenom, ali se ipak razlikuje u nekoliko ključnih stvari. Primanjem podataka iz obrasca za prijavu sa klijentske strane web aplikacije, generira se i uspostavlja sjednica za korisnika. Sjednica se za razliku od tokena čuva na poslužiteljskoj strani web aplikacije gdje se i vrši autentifikacija korisnika. Prilikom pristupanja zaštićenim resursima provjera se ukoliko je sjednica još uvijek validna te ukoliko je, korisnika se autorizira za pristup resursima. [10]

Sanctum uključuje i zaštitu od *Cross-Site Request Forgery* (skraćeno CSRF) napada koristeći provjeru CSRF tokena pri zaprimanju svakog zahtjeva na poslužitelj, odnosno server. Dodavanjem *auth:sanctum middlewarea* na API pozive prikazanog na Slici 2.4 omogućuje se da samo prijavljeni korisnici s validnom sjednicom ili tokenom mogu pristupiti putanjama. [11]

```
// Store Ticket Data
Route::middleware(['auth:sanctum'])->post('/tickets', [TicketsController::class, 'store']);
```

Slika 2.4: Korištenje auth:sanctum middlewarea

2.2 React

React je besplatna JavaScript knjižnica otvorenog koda za izradu klijentskog dijela web aplikacija. Koristi deklarativan tip programiranja gdje je naglasak na definiraju željenog rezultata (eng.

output). Napisan je u JavaScriptu, a održava ga Meta. Opisuju ga jednostavnost pisanja i čitljivosti koda, brzina izvođenja, ponovna iskoristivost komponenti te modularnost.

React je jedan od najpopularnijih predstavnika za izradu jednostraničnih aplikacija (eng. single-page application, skraćeno SPA). SPA predstavlja implementaciju web aplikacije gdje se korisniku u web pregledniku prikazuje samo jedna stranica, a onda se sadržaj odnosno komponente dinamički mijenjaju kroz promjene stanja (eng. state) te dobivanjem podataka kroz API pozive. Sve to pridonosi bržoj i responzivnijoj interakciji korisnika na web aplikaciji, kao i bržem učitavanju stranice i smanjenom opterećenju servera.

U nastavku su pobliže objašnjene i opisane glavne značajke kao i prednosti Reacta nad ostalim alatima iste namjene [12].

Korisnik na izbor ima mogućnost koristiti jedan od dva tipa komponenata za razvoj svoje web aplikacije, a to su: klasna komponenta (eng. class component) i funkcionalna komponenta (eng. functional component). Svaka komponenta sadrži svoj set varijabli kao i elemenata koje prikazuje. Klasne komponente koriste mnoge metode za upravljanje životnim ciklusom (eng. life cycle) i stanjima čime su u prošlosti po tome bile u prednosti nad funkcionalnim komponentama. To se promijenilo uvođenjem Hook funkcija (eng. Hook function) za postizanje sličnih funkcionalnosti. Time su funkcionalne komponente u potpunosti zamijenile klasne jer su omogućile iste funkcionalnosti, a pritom su funkcionalne komponente čitljive, kraće i optimiziranije. Klasne komponente još su uvijek dostupne za korištenje, no sam daljnji razvoj Reacta usmjeren je ka funkcionalnim komponentama i Hook funkcijama [13].

Koristeći Hook funkcije omogućeno je funkcionalnim komponentama da imaju pristup stanjima te ostalim značajkama koje su prije toga bile dostupne samo za klasne komponente. U nastavku su opisane neke od najčešće korištenih Hook funkcija.

Hook funkcija stanja (eng. State Hook) se inicijalizira pozivom *useState()* varijable stanja u koju se zatim mogu spremati podatci kao što su unosi korisnika unutar forme. Podatci su sadržani unutar varijable state, a vrijednost se mijenja pozivom *setState()* metode. Primjer prikazan na Slici 2.5 prikazuje varijablu stanja count kojoj se vrijednost povećava za 1 svakim pritiskom na gumb koji sadrži funkciju *setCount()* za povećavanje vrijednosti varijable stanja count.

```jsx
import React, { useState } from 'react';

function Example() {
  const [count, setCount] = useState(0);

  return (
    <div>
      <p>Gumb je pritisnut {count} puta</p>
      <button onClick={() => setCount(count + 1)}>
        Pritisni gumb
      </button>
    </div>
  );
}
```

Slika 2.5 UseState kuka

Hook funkcije efekta (eng. Effect Hook) koriste se za sinkroniziranje s vanjskim sustavima kao što su API pozivi, ostalim mrežnim funkcionalnostima te za dohvaćanje elemenata poput animacija. Pozivaju se nakon svakog renderiranja web aplikacije te je primjer useEffect kuke prikazan na Slici 2.6 gdje se prilikom pritiska na gumb povećava vrijednost varijable stanja, koja se zatim uvećana prikazuje korisniku te se zbog novog renderiranja web aplikacije poziva i *useEffect*() kuka (eng. Hook) koja mijenja naslov dokumenta na trenutnu vrijednosti varijable count.

```jsx
import React, { useState, useEffect } from 'react';

function Example() {
  const [count, setCount] = useState(0);

  useEffect(() => {
    document.title = `Gumb je pritisnut ${count} puta`;
  });

  return (
    <div>
      <p>Gumb je pritisnut {count} puta</p>
      <button onClick={() => setCount(count + 1)}>
      Pritisni gumb
      </button>
    </div>
  );
}
```

Slika 2.6 UseEffect kuka

Hook funkcija konteksta (eng. Context Hook) omogućuje primanje podataka od roditeljskih komponenata bez da se podatci šalju kao props.

Ref Hook se koristi za spremanje informacija koje nisu nužne za renderiranje same aplikacije već sadrže podatke kao što su vrijednost čvora u DOM stablu. Za razliku od ostalih Hook funkcija, promjenom stanja u ovoj web aplikacija se neće ponovno renderirati.

React sintaksa pisana je koristeći JavaScript Extension (skraćeno JSX) te omogućuje korištenje JavaScript objekata unutar HTML elemenata. Time je programiranje u Reactu vrlo jednostavno i intuitivno uz osnovno poznavanje HTML-a i JavaScripta, a napisani kod ima visok stupanj čitljivosti i razumljivosti. JSX koristi gotovo identičnu sintaksu kao HTML, no ipak se razlikuje u nekoliko ključnih stvari. Ugnjiježđeni (eng. nested) JSX mora vratiti roditeljski element koji obuhvaća sve ostale ugnjiježđene elemente i svaki element se može napisati kao samozatvarajući (eng. self-closing), primjerice ili <p/>. Također JSX nudi mogućnost jednostavnog referenciranja JavaScript varijabli koristeći vitičaste zagrade kao što je prikazano na Slici 2.7. Web preglednici ne podržavaju JSX stoga React koristi Babel prevoditelj (eng. compiler) kako bi sav kod pretvorio u preglednicima čitljiv oblik odnosno u JavaScript kod.

```
function Example() {

    const name = 'Mateo';

  return (
    <div>
        <h1>Hello, {name}</h1>;
    </div>
  );
}
```

Slika 2.7 JSX sintaksa

Još jedna bitna značajka Reacta je takozvani *Virtual Document Object Model* (skraćeno VDOM). DOM je važan dio web stranice jer ju dijeli na module i na tom principu izvršava kod. Uobičajena praksa je da se cijeli DOM ažurira odjednom, što za utjecaj ima smanjenje brzine web aplikacije, no u Reactu taj problem je otklonjen korištenjem spomenutog VDOM-a jer se on kao identična kopija pravog DOM-a web aplikacije ažurira te se nakon toga uspoređuju Virtualni i stvarni DOM te se ažuriraju samo moduli u kojima se dogodila promjena.

Podatci se kroz komponente prosljeđuju samo u jednom smjeru odnosno od roditeljske komponente do komponenti njegove djece. Dječja komponenta ne može svom roditelju vratiti podatke, no može komunicirati s njim kako promijeniti stanja na temelju primljenih ulaznih varijabli.

React nudi mnoge ekstenzije poput Bootstrapa [14] koje omogućavaju lakše stiliziranje aplikacije uz gotove UI komponente. Također uz React Native [15] omogućena je izrada mobilnih aplikacija za Android te iOS uređaje. Pruža se i *server side rendering* gdje se aplikacija prikazuje na serveru te je time renderiranje aplikacije potpuno uklonjeno iz web preglednika.

Sam React ne nudi vlastito rješenje za putanje i navigaciju kroz aplikaciju odnosno promjenu komponenti koje se renderiraju, ali uz veliku zajednicu programera postoje mnoge *third-party* knjižnice pa tako i za routing postoji knjižnica react-router s jednostavnom implementacijom navigacije kroz aplikaciju. Slika 2.8 prikazuje kodni isječak koji definira putanje kroz web aplikaciju. Kao što je prikazano potrebno je uvesti potrebne stavke iz react-router-dom knjižnica. Uvoze se i sve potrebne komponente: Home, About i Contact te se unutar glavne funkcionalne komponente *App(),* unutar <Router> i <Route> oznaka postavlju putanje i njima pripadajuće komponente koje će se renderirati ukoliko se korisnik navigira na URL adresu iz putanje.

```
import React from 'react';
import { Router, Route } from 'react-router-dom';

import Home from './components/Home';
import About from './components/About';
import Contact from './components/Contact';

function App() {
  return (
    <Router>
        <Route path="/" component={Home} />
        <Route path="/about" component={About} />
        <Route path="/contact" component={Contact} />
    </Router>
  );
}

export default App;
```

Slika 2.8: Navigiranje koristeći knjižnicu

2.3 MySQL

MySQL [6] je besplatan sustav otvorenog koda za upravljanje relacijskim bazama podataka (eng. relational database management system). Razvila ga je Švedska tvrtka MySQL AB, a prva inačica MySQL-a pojavila se 1995. godine. Danas MySQL održava i razvija tvrtka Oracle Corporation .

U relacijskim bazama podataka podatci su organizirani unutar entiteta odnosno tablica, u kojima su entiteti najčešće u nekom odnosu te koegzistiraju. Također svaka tablica sadrži stupce koji predstavljaju tip podataka stoga se još nazivaju i atributi. Tablica sadrži i retke koji predstavljaju konkretne vrijednosti atributa za pojedini unos u tablicu. Zbog svoje visoke popularnosti, danas je MySQL jedan od neizostavnih alata pri izradi aplikacija.
Pisanjem SQL upita korisnik manipulira podatcima iz baze podataka, odnosno pisanjem jednostavne sintakse pohranjuje, upravlja i dohvaća podatke. Na Slici 2.9 vidljiv je upit kojim se iz baze podataka dohvaćaju svi korisnici određenog imena sa pripadajućim rezultatom.

```
SELECT * from users where name='Mateo';
```

Slika 2.9: SQL upit

Koristeći se Laravelovim Eloquent ORM-om prethodni upit za dohvat korisnika određenog imena prikazan je na Slici 2.10.

```
User::where('name', 'Mateo')->get();
```

Slika 2.10 Eloquent ORM upit

Jedan od načina prikazivanja baza podataka je koristeći se vizualnim modelom, odnosno Entity Relationship (skraćeno ER) dijagramom. Entiteti su prikazani u obliku tablica te svaki entitet sadrži skup atributa koji opisuju taj entitet. Svaki atribut može imati svoj tip podataka koji predstavlja, a neki od najčešćih su INT (cjelobrojni tip), CHAR (znakovni tip) i DATE (vremenski tip). ER dijagram prikazuje i odnose između svakog od entiteta gdje oni mogu biti u odnosima jedan na jedan, jedan na više ili više na više. Entiteti su međusobno povezani korištenjem stranih ključeva, a primjer ER dijagram korišten unutar ovog rada prikazan je na Slici 2.11.

Slika 2.11 ER dijagram

2.4 Tailwind

Tailwind CSS [8] je radni okvir otvorenog koda koji korisnicima pruža mogućnost stiliziranja HTML komponenti po vlastitom nahođenju. Razlikuje se od ostalih radnih okvira po tome što nema već gotove komponente koje korisnik samo poziva unutar svoje aplikacije već je potrebno da korisnik sam stilizira HTML komponente kao što su i <button>. Elementi se stiliziraju tako da se unutar samog elementa koriste već gotove klase stiliziranja zajedno sa vrijednostima koje odabire korisnik prema vlastitim željama i potrebama. Tailwind CSS nema ograničenja po pitanju mogućnosti stiliziranja jer je korisniku prepušteno da kombinacijom raznih vrijednosti stvara komponente koje njemu trebaju. Jedina iznimka su vrijednosti koje se međusobno isključuju pa će se u tom slučaju primijeniti prva vrijednost dok će se druga vrijednost zanemariti i neće se odraziti na dizajnu elementa. Na Slici 2.12 vidljiv je primjer stiliziranja jednog elementa gdje su mu postavljene margine na vrijednost 20 piksela, pozadina zelene boje i boja teksta bijele boje.

```
<div className="m-[20px] bg-[#00ff00] text-white">
  Tekst unutar div elementa
</div>
```

Slika 2.12: Korištenje Tailwinda za stiliziranje HTML elemenata

2.5 FontAwesome

FontAwesome [16] je knjižnica fontova i ikona koju je razvio Font Awesome Team 2012. godine. Developerima omogućava jednostavno korištenje vektorskih ikona u projektima. Programerima se na korištenje nudi preko 1000 besplatnih ikona koje je prema vlastitim potrebama moguće stilizirati. Tako je ikonama moguće mijenjati veličinu, boju, rotaciju kao i primijeniti stil kao što je *bold* ili *solid*.

Unutar React projekta nude se dvije mogućnoti za korištenjem FontAwesome ikona. Prvi način je globalan te je za njega potrebno unutar glavne roditeljske komponente kao što je App.js uvesti knjižnicu FontAwesomeIcon kao i pripadajuće ikone koje se planiraju koristiti unutar aplikacije. Zatim se unutar komponenata djece uvozi FontAwesomeIcon knjižnica te se dalje unutar komponente mogu pozivati i uređivati ikone.

Drugi način je eksplicitni gdje se unutar komponente koja će koristiti ikone uvozi knjižnica FontAwesomeIcon te se uvoze ikone koje se koriste unutar komponente i u konačnici se pozivaju unutar komponente. Drugi način iako sadržava više koda od prvog ipak je preporučljiviji jer pojedina komponenta uvozi samo one ikone koje planira koristiti što nije slučaj kod prvog načina gdje svaka komponenta uvozi sve ikone definirane u roditeljskoj komponenti. Primjer eksplicitnog korištenja ikona vidljiv je na Slici 2.13.

```
import React from 'react'
import { FontAwesomeIcon } from '@fortawesome/react-fontawesome'
import { faEnvelope } from '@fortawesome/free-solid-svg-icons'

export const Mail = () => (
    <div>
        <FontAwesomeIcon icon={faEnvelope} />
    </div>
)
```

Slika 2.13: Korištenje FontAwesome ikona

2.6 Axios

Axios [17] je JavaScript knjižnica koja se koristi za izradu HTTP zahtjeva prema serverima od strane web aplikacija. Glavna značajka Axiosa su njegovi asinkroni zahtjevi temeljeni na obećanjima (eng. promises). Kroz obećanja Axios „obećaje“ da će vratiti vrijednost u nekom trenutku, te se ostatak koda može nastaviti izvršavati bez čekanja. Ono što Axios čini popularnim i prvim izborom mnogih programera je njegova jednostavna sintaksa koja se sačinjava od definiranja HTTP zahtjeva, odnosno radi li se o GET, POST, PUT, DELETE metodi ili nekoj drugoj te je potrebno definirati putanju na koju se šalje zahtjev prema serveru. Primjer Axios zahtjeva prikazan je na Slici 2.14. Opcionalno moguće je proslijediti i podatke koje Axios automatski pretvara u JSON oblik te ih šalje prema serveru. Axios je i veoma pouzdan i siguran alat jer nudi zaštitu od XSRF napada koristeći CSRF token koji se uključuje u svaki zahtjev poslan na server kako bi se potvrdila vjerodostojnost web aplikacije s koje se šalje zahtjev.

```
await axios.delete("api/tickets/" + id);
```

Slika 2.14: Axios zahtjev

3 FUNKCIONALNOSTI APLIKACIJE

3.1 Navigacijska traka

Navigacijska traka će za korisnika koji posjećuje web aplikaciju, a trenutno nije prijavljen u njoj prikazivati gumbe kao na Slici 3.1 gdje se nude gumbi za navigaciju na pogled za prijavu i pogled za registraciju. Ukoliko korisnik pritisne na logo *Ticket.io* u lijevom kutu, vraća ga se na početni pogled.

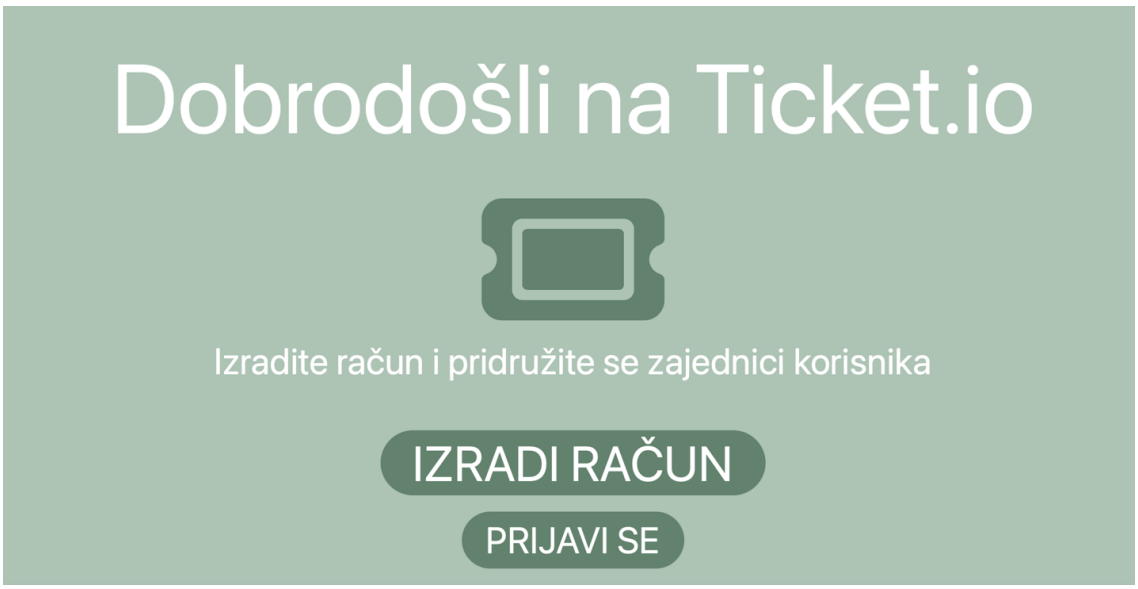
Slika 3.1 Navigacijska traka za neprijavljenog korisnika

Za korisnika koji je u web aplikaciju prijavljen navigacijska traka će ipak izgledati kao na Slici 3.2 gdje se prikazuje ime trenutno prijavljenog korisnika te gumbi koji će korisnika preusmjeriti na pogled s tiketima koje je on kreirao, pogled s tiketima za koje je zadužen te gumb za odjavu iz web aplikacije koja korisnika preusmjerava na početni pogled. Ovoga puta pritiskom na logo *Ticket.io* u lijevom kutu korisnik je preusmjeren na pogled sa listom svih kreiranih tiketa.

Slika 3.2 Navigacijska traka za prijavljenog korisnika

3.2 Početni pogled

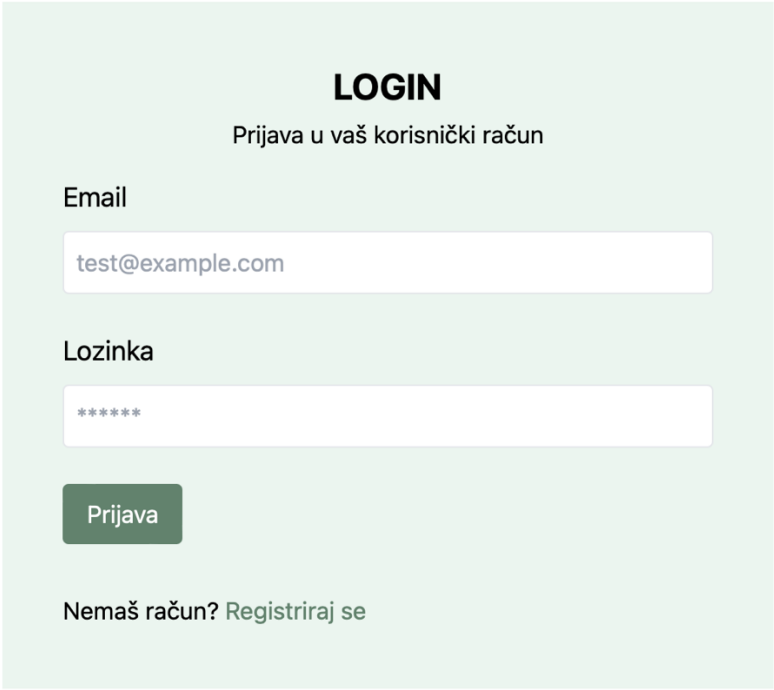
Pri posjetu web aplikacije, ukoliko korisnik nije prijavljen prikazati će mu se početni pogled gdje mu se nude dvije opcije u obliku gumba kao što je vidljivo na Slici 3.3. Pritiskom na gumb *Izradi račun* korisnika se preusmjerava na pogled s formom za registraciju, a pritiskom na gumb *Prijavi se* korisnik će biti preusmjeren na pogled s obrascem za prijavu.



Slika 3.3: Početni pogled

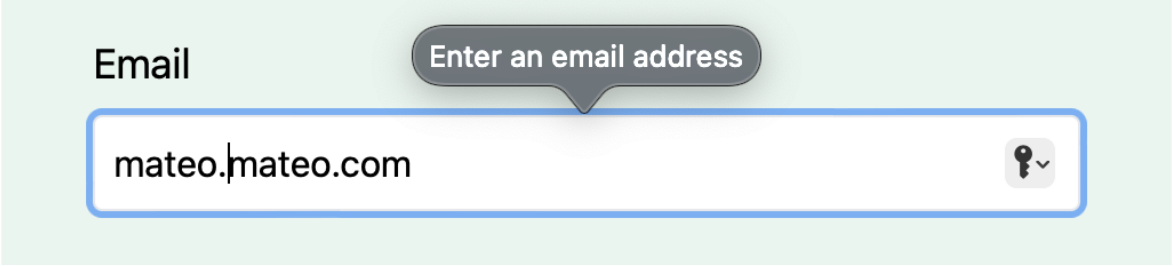
3.3 Prijava

Kao što je prikazano Slikom 3.4 na pogledu za prijavu korisnika prikazana su dva polja za unos podataka. Također vidljiv je i gumb *Prijava* te gumb *Registriraj se* koji korisnik može stisnuti kako bi kreirao račun ukoliko ga već ne posjeduje te će se klikom na njega korisnika preusmjeriti na pogled s obrascem za registraciju.



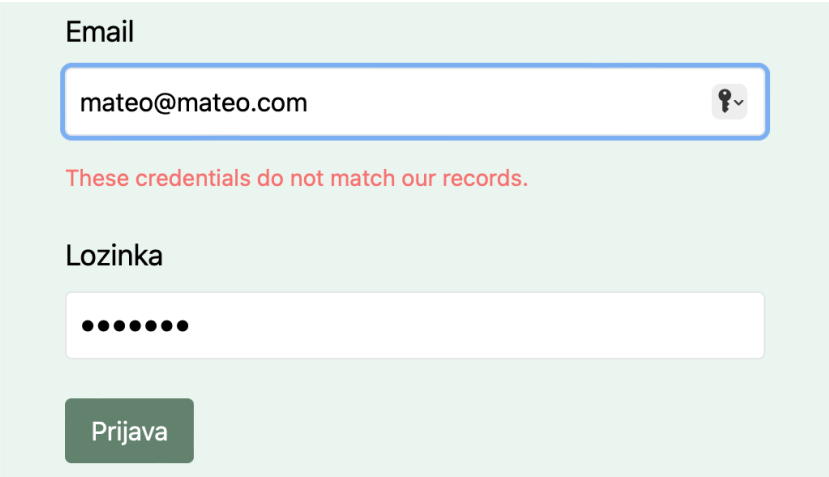
Slika 3.4: Obrazac za prijavu

Unutar polja za unos podataka, od korisnika se očekuje unos validne email adrese, inače će se prikazati greška na Slici 3.5. Također potrebno je unesti i odgovarajuću lozinku koju je odredio prilikom stvaranja računa.



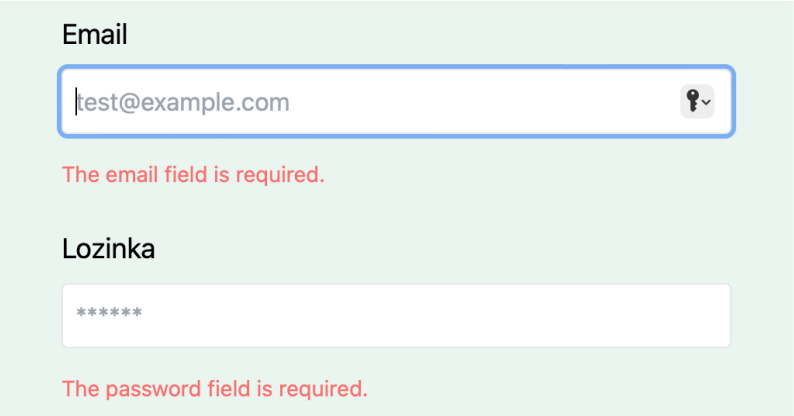
Slika 3.5: Pogrešan format email adrese

Nakon pritiska gumba *Prijava* podatci se validiraju te ukoliko su pristupni podatci točni, korisnika se preusmjerava u dio aplikacije koji nije bio vidljiv neprijavljenim korisnicima, odnosno prikazuje mu se popis svih tiketa. Ipak ukoliko pristupni podatci nisu valjani prikazat će se greška kako korisnik s takvim pristupnim podatcima ne postoji (Slika 3.6).



Slika 3.6: Greška zbog pogrešnih pristupnih podataka

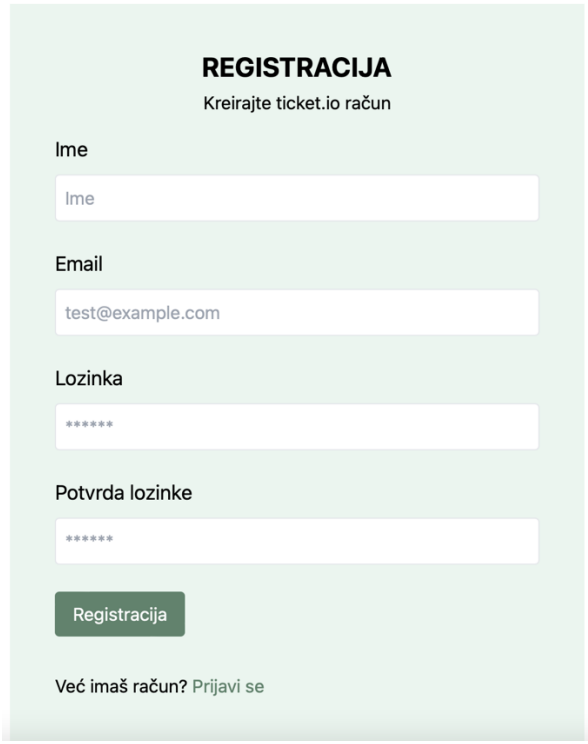
Ukoliko korisnik nije unio pristupne podatke, odnosno email i/ili lozinku prikazat će mu se greška ispod polja kojem nedostaju pristupni podatci (Slika 3.7).



Slika 3.7 Greška zbog nedostataka podataka u poljima za prijavu

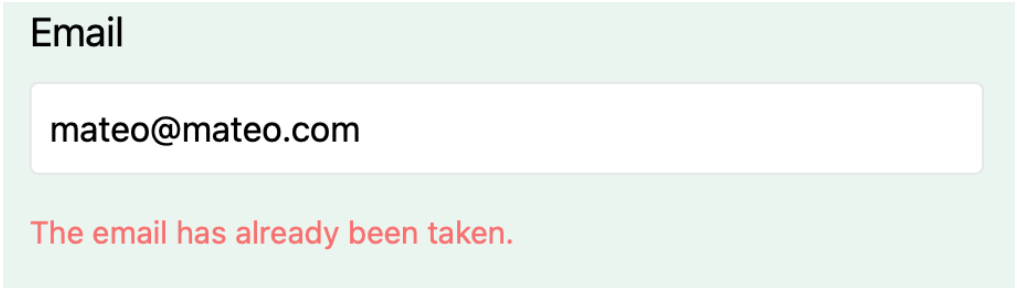
3.4 Registracija

Na pogledu s obrascem za registraciju prikazana su četiri polja za unos podataka, gumb *Registriraj se* te gumb *Prijavi se* kojeg korisnik može stisnuti ukoliko već posjeduje račun za prijavu te ipak želi izvršiti prijavu čime ga se preusmjerava na pogled s obrascem za prijavu. Obrazac s pripadajućim poljima za unos teksta prikazan je na Slici 3.8.



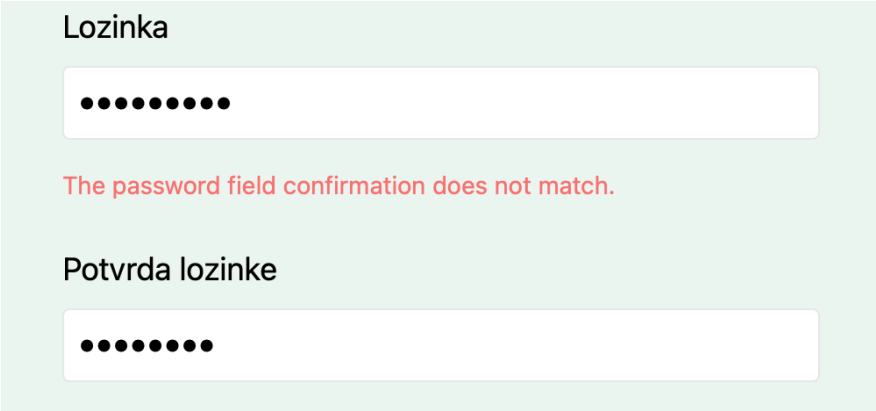
Slika 3.8: Obrazac za registraciju korisnika

Unutar polja *Ime* korisnik upisuje svoje korisničko ime koje će se prikazivati drugim korisnicima unutar tiketa koje taj korisnik kreira ili bude zadužen za njih. U polje *Email* potrebno je unijeti validnu email adresu, koja mora biti unikatna odnosno ne smije ju koristiti niti jedan korisnik koji je registraciju obavio ranije. U slučaju da je email adresa već iskorištena prikazat će se poruka kao na Slici 3.9.



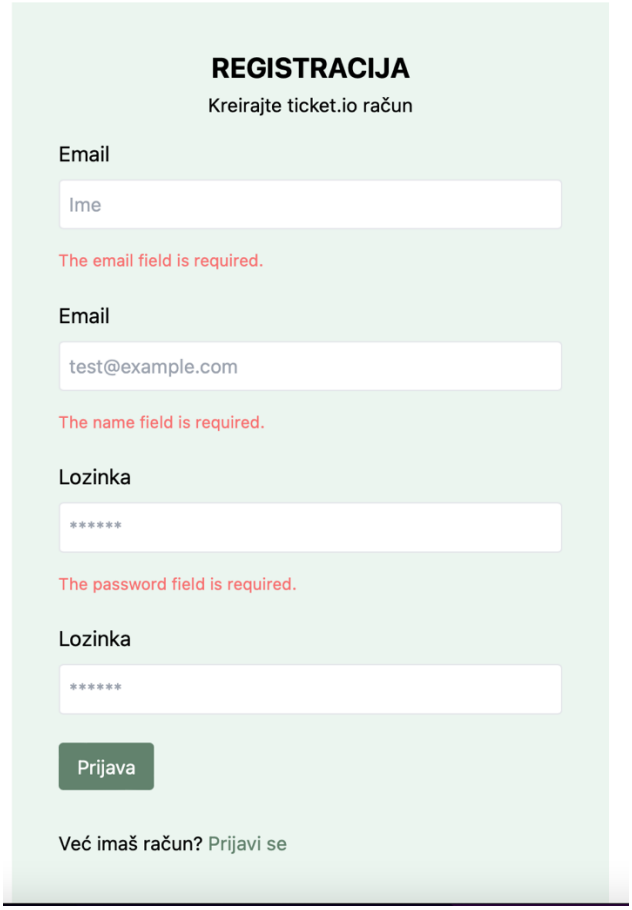
Slika 3.9: Greška zbog email adrese koja je već u upotrebi

Unutar polja *Lozinka* i *Potvrda lozinke* korisnik unosi željenu lozinku koju će koristiti za prijavu u račun, a koja mora sadržavati barem 8 znakova. Ukoliko se unesena lozinka unutar polja *Potvrda lozinke* ne podudara sa lozinkom unutar polja *Lozinka*, ispisat će se prigodna greška (Slika 3.10).



Slika 3.10: Greška zbog nepodudarnosti lozinki

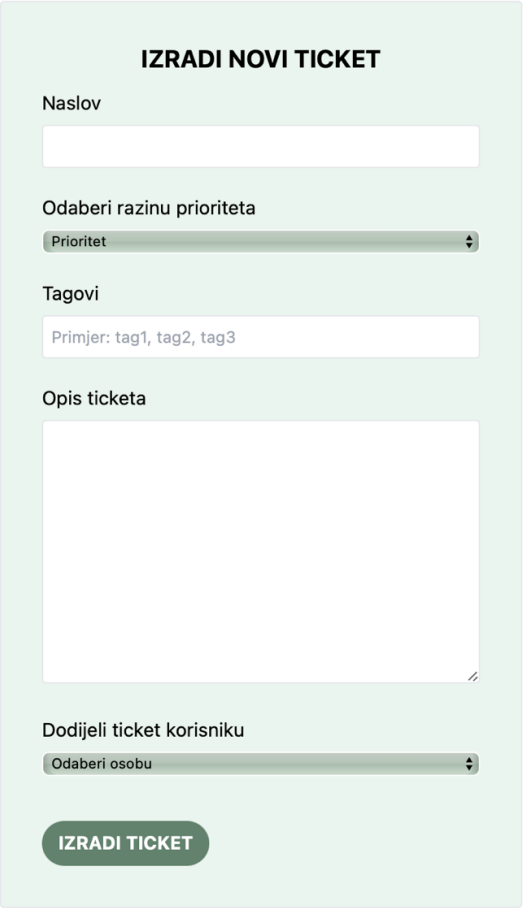
Upisivanjem očekivanih podataka u svako od polja za unos podataka te zatim klikom na gumb *Registriraj se*, validira se upit za registracijom te ukoliko je on uspješan korisnika se preusmjerava na pogled sa prikazom svih kreiranih tiketa. Ukoliko u nekom od polja nedostaju podatci prikazat će se prigodna greška ispod tog polja (Slika 3.11).



Slika 3.11: Greška zbog nedostatka podataka unutar polja

3.5 Kreiranje tiketa

Kao što je prikazano Slikom 3.12 na ekranu za izradu tiketa od korisnika se očekuje ispunjavanje forme upisivanjem teksta u tri polja za unos podataka te odabirom jednog od već pred definiranih odabira iz padajućeg izbornika.

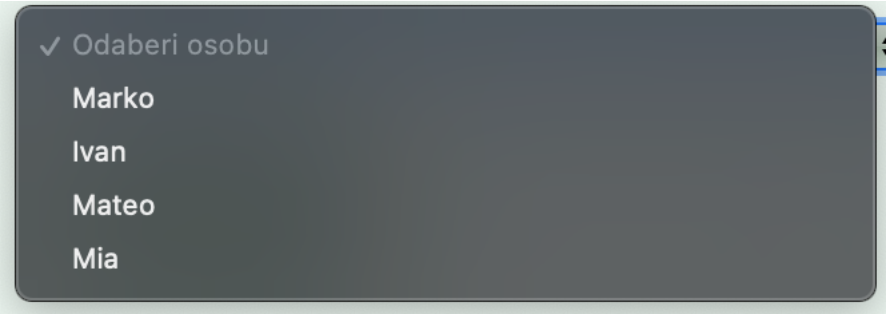


Slika 3.12: Obrazac za izradu tiketa

Unutar polja *Naslov* od korisnika se očekuje da će tiketu dati neki smislen naziv iz kojeg će se već moći razaznati o čemu se radi bez detaljnijeg čitanja opisa tiketa. Kao *Razinu prioriteta* korisniku su dane na izbor 3 vrijednosti te je potrebno odabrati jednu od ponuđenih (Slika 3.13).

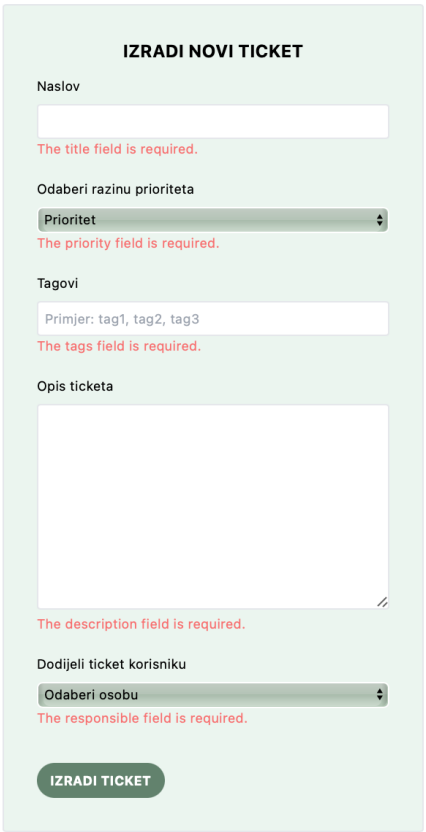
Slika 3.13: Odabir prioriteta iz padajućeg izbornika

Unutar polja *Tagovi* korisnik upisuje kratke oznake koje bi ukratko opisale novonastali tiket. Oznaka odnosno tagova može biti više, ali se moraju pri unosu u polje odvojiti zarezom. U polju *Opis* od korisnika se očekuje da detaljno raspiše problem kao i pristup ka njegovom rješavanju. Zadnji dio koji treba odabrati je *Osoba zadužena za tiket* gdje se korisniku na izbor pružaju svi trenutni registrirani korisnici, a primjer se može vidjeti na Slici 3.14.



Slika 3.14: Padajući izbornik sa popisom svih korisnika

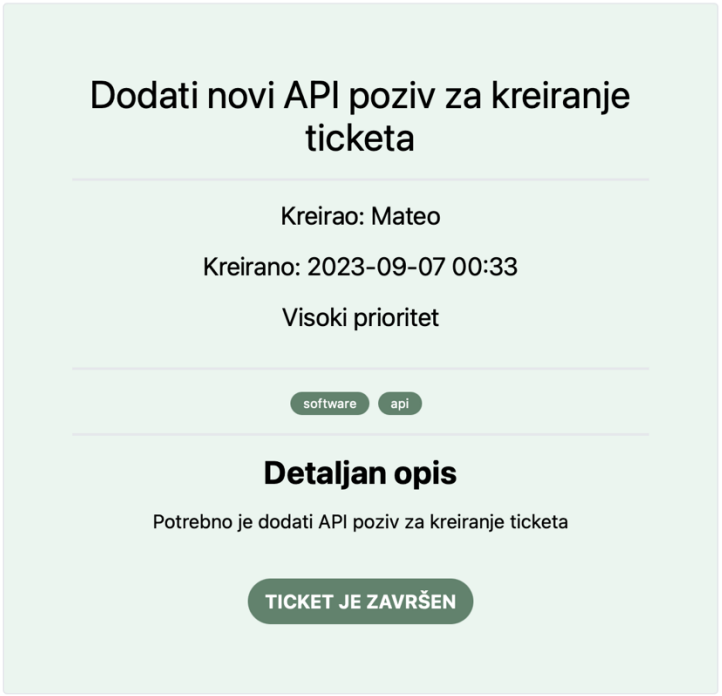
Nakon ispunjenja svih potrebnih vrijednosti, klikom na gumb *Izradi ticket* korisnika se preusmjerava na početni pogled za prijavljenog korisnika, odnosno pogled sa listom svih tiketa. Unutar liste sada mu se prikazuje i novoizrađeni tiket. No ukoliko je neko od polja ostalo neispunjeno prikazat će se poruka kraj svakog elementa kojem nedostaje vrijednost kao što je to prikazano na Slici 3.15.



Slika 3.15: Greška zbog nedostataka podataka pri izradi tiketa

3.6 Pregled tiketa

Nakon pritiska na naslov pojedinog tiketa iz jedne od liste tiketa, korisniku se prikazuje pogled s prikazom detalja tiketa. Tako je moguće vidjeti redom odozgo prema dolje: naslov, kreatora tiketa, datum izrade, prioritet, tagove i opis. Primjer takvog tiketa prikazan je Slici 3.16.



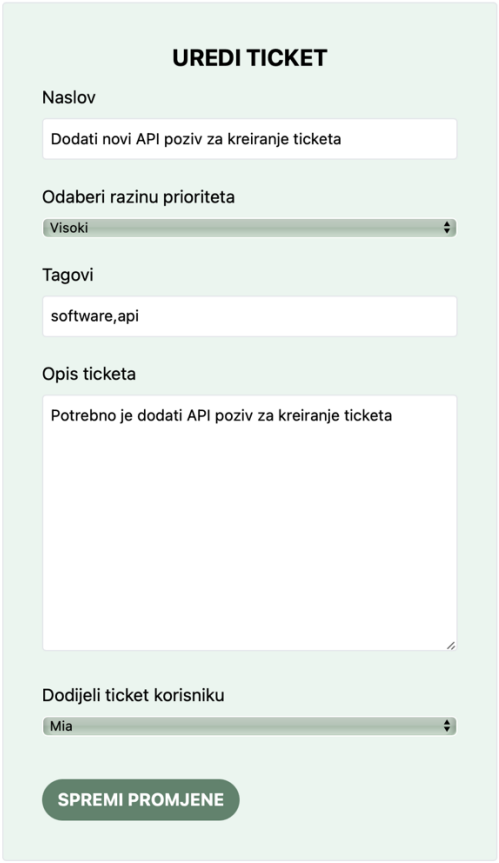
Slika 3.16: Detaljan opis tiketa

Ukoliko je tiket neizvršen, a osoba prijavljena u web aplikaciju je također osoba odgovorna za taj tiket, tada će se na dnu ispod opisa samog tiketa prikazati gumb *ZAVRŠI TICKET* te će pritiskom na taj gumb, tiket biti označen kao izvršen te će svi korisnici koji otvore detalje tog tiketa ispod sekcije za opis vidjeti labelu s natpisom *TICKET JE ZAVRŠEN*. Ukoliko tiket nije izvršen, a osoba prijavljena u web aplikaciju je također i kreator tiketa kojeg pregledava tada joj se u gornjem lijevom kutu prikazuju dodatna dva gumba, a to su gumbi za brisanje i uređivanje tiketa (Slika 3.17). Pritiskom na gumb *Izbriši*, tiket koji se pregledava će se izbrisati za sve korisnike, a pritiskom na gumb *Uredi* korisnika se preusmjerava na pogled s formom za uređivanje tiketa koja je već popunjena s trenutnim podatcima tiketa koji se želi urediti. Također u gornjem u lijevom kutu korisnicima je vidljiv i gumb *NATRAG* koji ih vraća na prethodnu stranicu odnosno stranicu s listom svih tiketa.

Slika 3.17: Gumbi za uređenje i brisanja tiketa te povratak na prethodni pogled

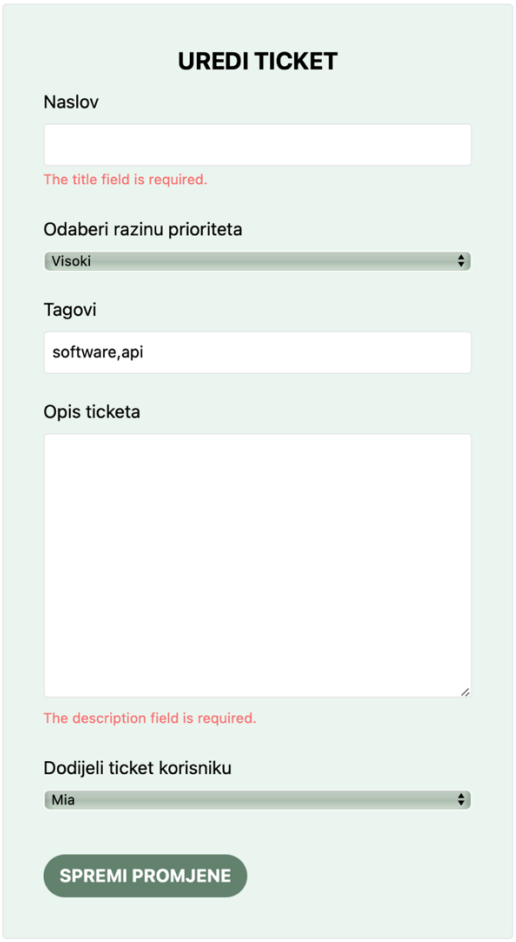
3.7 Uređivanje tiketa

Ukoliko se kreator tiketa odluči pritisnuti na gumb za uređenje tiketa, prikazat će mu se obrazac identičan kao i pri dodavanju novog tiketa, ali će podatci već biti unaprijed popunjeni te je na korisniku samo da promijeni željene podatke. Primjer takvog pogleda s obrascem za uređivanje tiketa prikazan je na Slici 3.18.



Slika 3.18: Obrazac za uređivanje tiketa

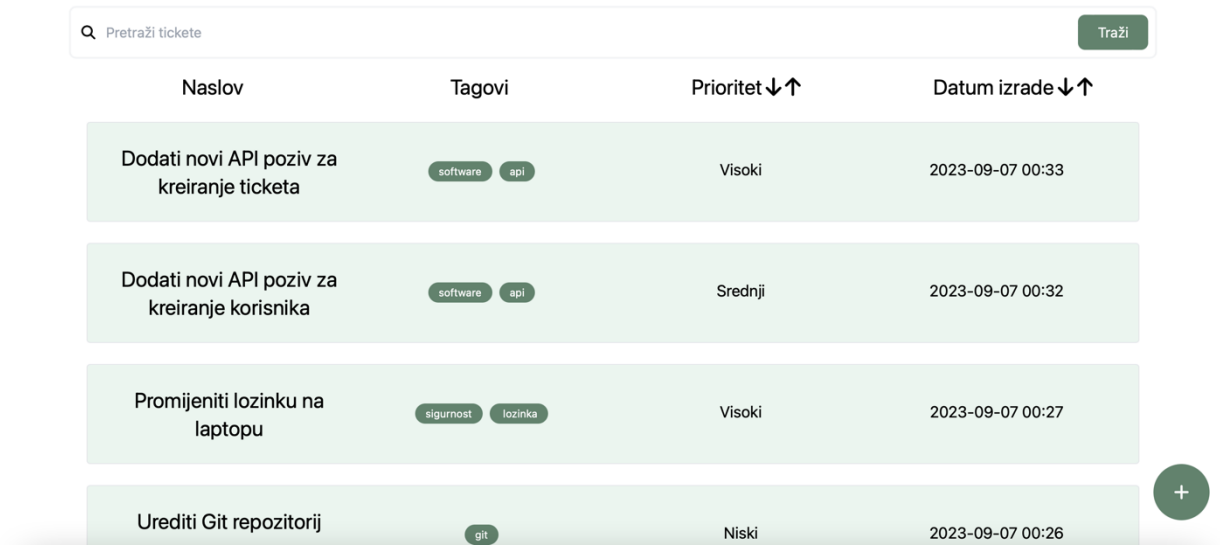
Ukoliko je korisnik napravio sve željene promjene i pritisnuo na gumb *Spremi promjene,* tada se najprije validira jesu li sva polja ostala popunjena te u slučaju da jesu, promjene se spremaju te se korisnika vraća na pogled sa listom svih tiketa. U protivnom se prikazuje greška za polje kojem nedostaju podatci (Slika 3.19).



Slika 3.19: Greška zbog nedostatka podataka na pogledu za uređivanje tiketa

3.8 Pregled svih tiketa

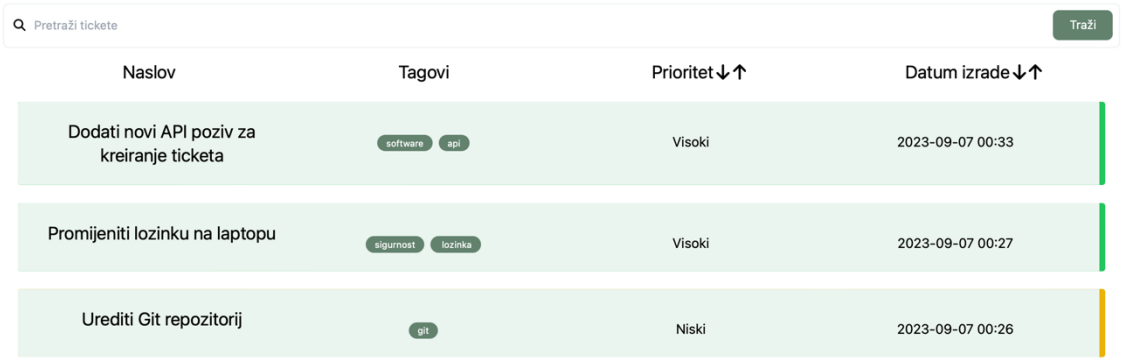
Početni pogled nakon prijave svakog korisnika je onaj s prikazom liste svih izrađenih tiketa (Slika 3.20). Korisniku se za svaki od tiketa prikazuju tek ključni podatci, a to su redom: naslov, tagovi, prioritet te datum izrade. Ukoliko korisnika zanima više detalja o određenom tiketu, pritiskom na naslov otvorit će mu se stranica sa detaljnim prikazom tiketa. Korisnik je u mogućnosti izvršiti filtriranje po tagovima, pretragu, te sortiranje tiketa prema vlastitim željama i potrebama. Moguće je istovremeno primijeniti sortiranje s pretragom/filtriranjem, odnosno međusobno se neće poništavati. Korisniku se u svakom trenutku u donjem desnom kutu prikazuje gumb + s kojim korisnik otvara pogled s obrascem za kreiranje novog tiketa.



Slika 3.20: Pregled svih tiketa

3.9 Pregled kreiranih tiketa

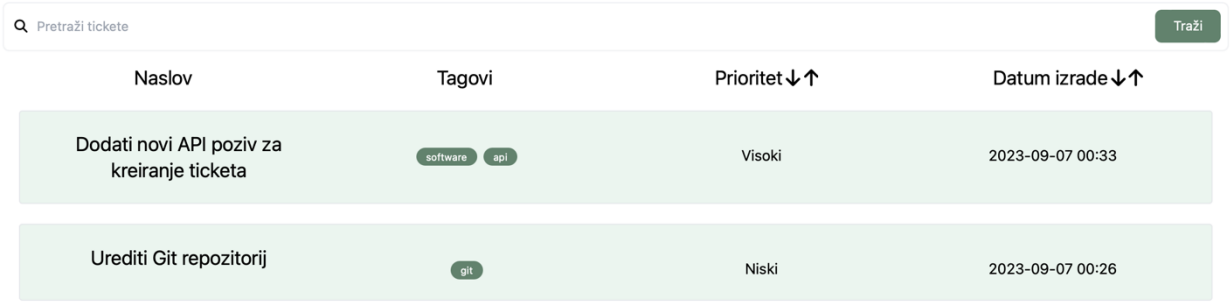
Na pogledu s prikazom kreiranih tiketa korisniku se prikazuju samo oni tiketi koje je on izradio. Dizajn je identičan kao i na pogledu s prikazom svih tiketa te se također omogućuju filtracija, pretraživanje i sortiranje. Ipak ključna je razlika što je korisniku svaki tiket na svojoj desnoj strani obojan s jednom od dvije moguće boje, a to su žuta i zelena. Žuta boja predstavlja da je tiket još u procesu rješavanja dok zelena predstavlja da je korisnik kojemu je tiket dodijeljen, označio tiket kao riješen. Prikaz liste s kreiranim tiketa prikazan je na Slici 3.21.



Slika 3.21: Pregled kreiranih tiketa

3.10 Pregled tiketa određenog korisnika

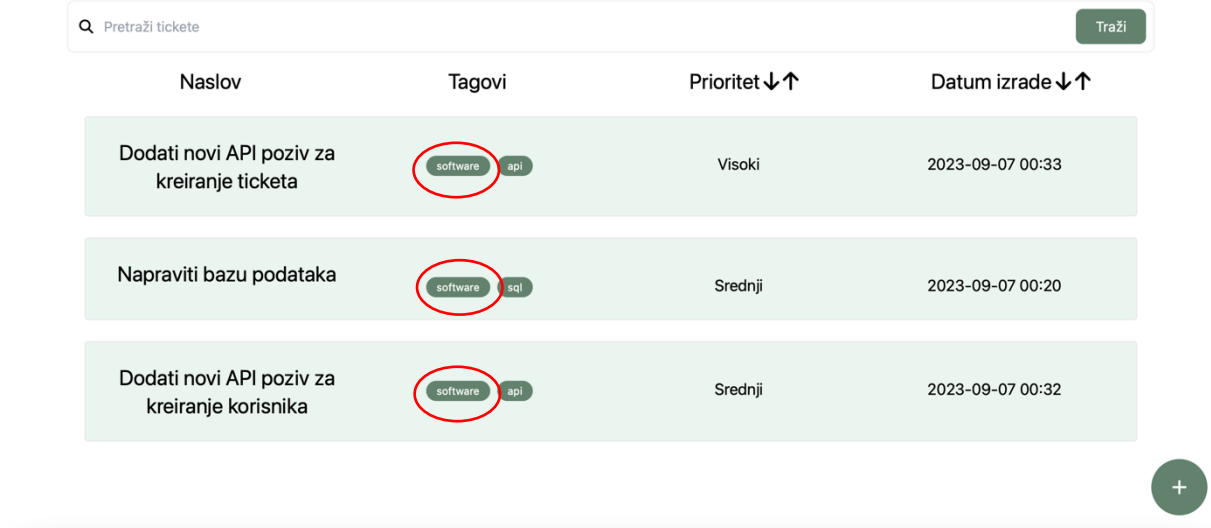
Klikom na gumb *Moji ticketi* na navigacijskoj traci, korisniku se otvara pogled sa listom tiketa koji su njemu dodijeljeni (Slika 3.22). Dizajn je po svemu identičan kao prikazu liste svih kreiranih tiketa, no unutar liste se ne mogu naći tiketi dodijeljeni drugim korisnicima. I na ovom se prikazu također nude opcije filtriranja, pretraživanja i sortiranja.



Slika 3.22: Pogled sa listom tiketa određenog korisnika

3.11 Filtriranje

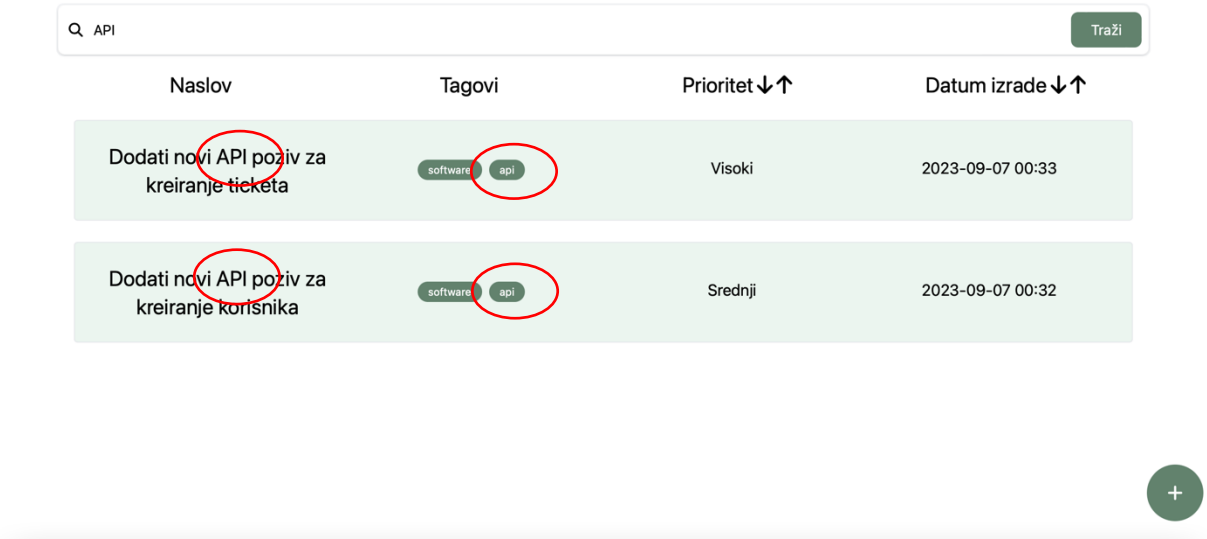
Filtriranje se izvršava pritiskom na jedan od tagova unutar neke od liste tiketa, te se zatim pronalaze svi tiketi koji sadrže traženi tag. Primjer takvog filtriranja je na Slici 3.23 gdje se filtriralo po tagu *software*.



Slika 3.23: Pogled s listom tiketa nakon filtriranja

3.12 Pretraživanje

Upisivanjem izraza unutar tražilice te pritiskom na gumb *Traži* ili pritiskom tipke *Enter* na tipkovnici obavlja se pretraživanje tiketa prema zadanom izrazu. Pronalaze se svi tiketi koji unutar naslova, opisa ili tagova sadrže izraz iz tražilice. Primjer pretraživanja prema ključnoj riječi API prikazan je na Slici 3.24.



Slika 3.24: Pogled s listom tiketa nakon pretraživanja

3.13 Sortiranje

Za opciju sortiranja korisniku su na raspolaganju 4 mogućnosti, a to su: prioritet uzlazno, prioritet silazno, datum izrade uzlazno te datum izrade silazno. Sortiranje se izvršava pritiskom na jednu od strelica pokraj natpisa *Prioritet* ili *Datum izrade* kao što je prikazano na Slici 3.25. Strelica prema dolje predstavlja silazno sortiranje dok strelica prema gore predstavlja uzlazno sortiranje.

Slika 3.25: Gumbi za sortiranje

4 FUNKCIONALNOSTI KROZ KOD

4.1 Kreiranje tiketa

Jedna od glavnih funkcionalnosti aplikacije je kreiranje tiketa. U nastavku je opisan proces kreiranja jednog tiketa kroz programski kod, točnije od pritiska gumba za prikazivanje forme za izradu tiketa, do upisivanja potrebnih podataka u polja za unos podataka na web aplikaciji, preko spremanja tiketa u bazu i u konačnici prikazivanja novonastalog tiketa samom korisniku.

Kako bi se uspješno kreirao tiket, korisnik mora biti prijavljen u aplikaciji te zatim pritisnuti gumb + za dodavanje tiketa koji će prihvatiti event pritiska i pozvati React Hook useState [13] funkciju kao što je vidljivo na Slici 4.1.

```
{user ? (
  <>
    <div className="fixed bottom-8 right-10 opacity-100 h-16 w-16 bg-[#62826d] hover:bg-[#9ab5a3] rounded-full">
      <Link to="/create">
        <button onClick={(e) => setFormValues(InitialForm)} className="fixed opacity-100 bottom-8 right-10 h-16 w-16 text-white">
          <FontAwesomeIcon icon={faPlus} size="lg" />
        </button>
      </Link>
    </div>
  </>
) : (
  <>
  </>
)
}
```

Slika 4.1: Definiranje akcije za klik na gumb

Unutar nje definirana je metoda *setFormValues* kojom se postavljaju vrijednosti za stanje *FormValues*, odnosno parametri koje će se popuniti unutar obrasca se postavljaju na vrijednost „“ kako se ne bi neka zaostala vrijednost proslijedila u formu (Slika 4.2).

Slika 4.2: Vrijednosti postavljene za stanje FormValues

Također pritiskom na gumb + korisniku se prikazuje pogled sa obrascem za kreiranje novog tiketa uz pomoć <Link>, <Route> i <Routes> [18] komponenti koje su uvezene (eng. import) iz react-router knjižnice. <Route> i <Routes> omogućuju definiranje putanja za određenu URL adresu. Kao što je vidljivo na Slici 4.3 unutar <Route> je definirano da se za putanju */create* poziva funkcionalna komponenta *TicketCreate*.

Slika 4.3: Navigiranje koristeći knjižnicu i prikazivanje nove komponente

Unutar funkcionalne komponente *TicketCreate* renderira se pogled s poljima i padajućim izbornicima preko kojih korisnik unosi podatke o tiketu. Svakom od elemenata za unos podataka pridruženo je prigodno ime kako bi se nakon slanja podataka na poslužiteljsku stranu uspješno proslijedili podatci. Također vrijednost elementa je postavljena na trenutnu vrijednost unutar

formValuesa koja za sve elemente zbog prethodno spomenute inicijalizacije u početku iznosi „“ (Slika 4.4).

```
<div class="mb-6">
    <label htmlFor="title" className="inline-block text-lg mb-2">Naslov</label>
    <input name="title" value={formValues["title"]} onChange={onChange} class="border border-gray-200 rounded p-2 w-full" />
    {errors.title && (<span className='text-red-400 text-sm mt-2 pt-2'>{errors.title[0]}</span>)}

</div>
```

Slika 4.4: Polje za unos naslova tiketa

Upisivanjem podataka u polje poziva se funkcija *OnChange,* kao što je prikazano na Slici 4.5, koja će unutar formValuesa uz ime elementa, primjerice *title* pridružiti i vrijednost unutar polja koja se zatim i prikazuje unutar elementa jer sada *value* poziva vrijednost koja je pridružena uz njegovo ime primjerice title: „Naslov“.

```
const onChange = (e) => {
    const { name, value } = e.target;
    setFormValues({ ...formValues, [name]: value });
}
```

Slika 4.5: Funkcija za unos spremanje podataka iz polja

Uzmimo za primjer da su upisane sve vrijednosti te u konačnici formValues izgleda kao na Slici 4.6.

```
{
    title: "Napraviti završni rad",
    priority: "Visoki",
    tags: "fakultet",
    description: "Potrebno je napraviti web aplikaciju i napisati rad.",
    responsible: "4"
}
```

Slika 4.6: FormValues sa vrijednostima nakon ispunjavanja obrasca

Nakon upisivanje svih vrijednosti korisnik je pritisnuo na gumb *Izradi ticket* koji se nalazi unutar <form> elementa te se njegovim pritiskom poziva asinkrona funkcija *storeTicket()* koja kao argument prihvaća događaj *e* što je i prikazano na Slici 4.7.

```
const storeTicket = async (e) => {
    e.preventDefault();
    formValues.user_id = user.id
    try {
        await axios.post("api/tickets", formValues);
        setFormValues(InitialForm);
        navigate("/")
    } catch (e) {
        if (e.response.status === 422) {
            setErrors(e.response.data.errors)
        }
    }
}
```

Slika 4.7: Metoda za spremanje tiketa

Za početak koristi se metoda *preventDefault()* [19] nad događajem kako bi se spriječilo osvježavanje preglednika nakon pritiska na gumb za izradu tiketa. U formValues se pod imenom user_id dodaje i identifikator trenutno prijavljenog korisnika. Unutar *try* bloka se uz korištenje Axiosa šalje *POST* zahtjev na putanju http://localhost/api/tickets te se uz to HTTP zahtjevu pridodaju i vrijednosti iz formValuesa prikazane na Slici 4.8. Dio putanje http://localhost/ dobiven je iz modula definiranog u zasebnom Axios dokumentu koji se zatim uvozi u trenutnu funkcionalnu komponentu. Podatci se automatski pretvaraju u JSON oblik te se kao *Content*-Type postavlja *application/json* kako bi čitanje tih podataka na poslužiteljskoj strani prošlo što jednostavnije. Ukoliko poslužiteljska strana web aplikacije vrati HTTP status jednak 422, koji označava da se podatci ne mogu procesuirati, spremit će se greške koje će se zatim i prikazati ispod polja za unos podataka.

```
{
    title: "Napraviti završni rad",
    priority: "Visoki",
    tags: "fakultet",
    description: "Potrebno je napraviti web aplikaciju i napisati rad.",
    responsible: "4",
    user_id: "4"
}
```

Slika 4.8: FormValues sa konačnim vrijednostima

Zahtjev stiže do poslužiteljskog dijela web aplikacije podignute na adresi http://localhost:8000 napisane u Laravelu. Unutar dokumenta api.php nalaze se sve putanje za koje će poslužitelj obraditi zahtjev te poslati odgovor natrag klijentskoj strani web aplikacije. Tako će u ovom slučaju zahtjev stići na putanju te se pokreće izvršavanje definirane funkcije, u ovom slučaju *store* koja se nalazi u klasi kontrolera pod nazivom *TicketsController.* Važno je napomenuti kako sve putanje unutar api.php dokumenta dobivaju prefix /api te se zbog toga poslani zahtjev mapira na onaj prikazan na Slici 4.9.

```
// Store Ticket Data
Route::post('/tickets', [TicketsController::class, 'store']);
```

Slika 4.9: Putanja koja je odgovorna za spremanje tiketa

Unutar funkcije *store(Request $request)* definira je varijabla *formFields* u koju se spremaju podatci poslani sa klijentske strane, a koji su sadržani unutar varijable *request.* No prije nego se podatci spreme u varijablu vrši se provjera ukoliko su zaprimljeni svi potrebni podatci za unos u bazu podataka. Kao što je prikazano na Slici 4.10 za neke od podataka je nužno da im vrijednost nije *null* kako bi se uspješno napravio novi unos u bazu.

```
public function store(Request $request)
{
    $formFields = $request->validate([
        'title' => 'required',
        'priority' => 'required',
        'responsible' => 'required',
        'tags' => 'required',
        'description' => 'required',
        'user_id' => 'required'
    ]);

    $formFields['date'] = Carbon::now()->toDateTimeString('minute');

    $formFields['done'] = 0;


    Tickets::create($formFields);

    return response()->json("Ticket created");
}
```

Slika 4.10: Validacija podataka prije unosa u bazu

Također varijabli *formFields* se pridružuje i vrijednost naziva *date* koja je jednaka vrijednosti trenutnog datuma zajedno sa trenutnim vremenom, primjerice *2023-07-07 15:07.* Sprema se i vrijednost za parametar *done* za koji se sprema vrijednost 0 što će u aplikaciji označavati da tiket nije riješen te će se tek pri promjeni vrijednosti u 1, tiket smatrati riješenim. Nakon dodavanja dvije nove vrijednosti, formFields se prosljeđuje u model *Tickets* pozivom Eloquent metode *Tickets::create($formFields).* Pozivom te metode zapravo se u bazu podataka šalje INSERT upit kojim se onda podatci popunjavaju u novi redak tablice *tickets* kao što je vidljivo na Slici 4.11.

id	user_id	title	priority	responsible	done	tags	description	date
12	4 →	Napraviti završni rad	Visoki	4	0	fakultet	Potrebno je napraviti web aplikaciju i napisati rad.	2023-09-07 21:52

Slika 4.11: Prikaz novog unosa u bazi podataka

Nakon uspješnog dodavanja u bazu podataka funkcija *store()* iz kontrolera vraća odgovor natrag na klijentski dio web aplikacije zajedno sa porukom i HTTP statusom 200 koji označava da je sve prošlo uspješno.

4.2 Prikaz detalja jednog tiketa

Za prikazivanje detalja tiketa prvi korak je pritisnuti na naslov nekog od tiketa sa liste. To je omogućeno jer je svaki naslov omeđen komponentom <Link> koja vodi korisnika na URL s putanjom /ticket/{ticket.id} gdje ticket.id predstavlja jedinstveni identifikator tiketa (Slika 4.12).

```
<h3 className="text-2xl align-middle">
  <Link to={`/ticket/${ticket.id}`}>
    {ticket.title}
  </Link>
</h3>
```

Slika 4.12: Navigacija pritiskom na naslov tiketa

Korisniku se prikazuje komponenta s detaljima tiketa jer je unutar App.js odnosno roditeljske komponente u kojoj se nalazi lista svih tiketa definirana putanja sa elementom <Route> kao što je prikazano na Slici 4.13.

```
<Route path='/ticket/:id' element={<TicketDetails />} />
```

Slika 4.13: Putanja na koju se navigira korisnika nakon pritiska na naslov

Kako bi se prikazao odabrani tiket, unutar funkcionalne komponente *TicketDetails,* koristi se *Hook* funkcija *useEffect* koja poziva *GetTicket(id)* metodu za dohvaćanje tiketa s traženim identifikatorom te *GetUsers()* za dohvaćanje svih kreiranih korisnika. Identifikator koji se prosljeđuje u *getTicket(id)* metodu dobivem jer korištenjem Hook funkcije *useParams*() koja je dio react-router knjižnice za navigaciju. *useParams()* će iz URL-a izvući proslijeđeni identifikator jer je unutar <Route> komponente bilo definirano da nakon ticket/ dijela dolazi *:id* odnosno identifikator te će preuzeti vrijednost koja se nalazi na tom mjestu i spremiti ju u varijablu id koja se onda dalje koristi (Slika 4.14).

```
let { id } = useParams();

useEffect(() => {
    getTicket(id);
    getUsers();
}, []);

return (
```

Slika 4.14: Definiranje useEffect metode

Metode *getTicket(id)* i *getUsers()* definirane su unutar funkcionalne komponente *AuthProvider* iz koje se koristeći *useContext*() omogućava ostalim komponentama u ovom slučaju komponenti TicketDetails pristup metodama i stanjima roditeljskih komponenata neovisno o dubini u stablu komponenti. Slika 4.15 prikazuje metode koje se dohvaćaju iz *AuthProvidera.*

```
const { ticket, getTicket, users, getUsers, user, deleteTicket, CompleteTicket } = useAuthContext();
```

Slika 4.15: Dohvaćene metode i stanja

Unutar *getTicket* metode, proslijeđeni identifikator se ubacuje unutar putanje za POST HTTP zahtjev. Bazni URL definiran je unutar modula definiranog u zasebnom Axios dokumentu koji se uvozi u trenutnu komponentu koja koristi *Axios* poziv (Slika 4.16). Unutar Axios modula

postavljen je i parametar *withCredentials* na true kako bi se pri axios pozivima koristili i kredencijali poput kolačića (eng. cookies).

```
import axios from "axios";

export default axios.create({
    baseURL: "http://localhost:8000",
    withCredentials: true
});
```

Slika 4.16: Definiranje Axios bazne putanje

U konačnici URL adresa za dohvaćanje tiketa kreiranog u prethodnom poglavlju glasi http:localhost:8000/api/ticket/12. Na tu se adresu šalje zahtjev za dohvatom podataka koji će se zatim spremiti u varijablu *response* (Slika 4.17).

```
const getTicket = async (id) => {
    const response = await axios.get("api/ticket/" + id);
    const apiTicket = response.data;
    setTicket(response.data);
    setFormValues({
        title: apiTicket.title,
        priority: apiTicket.priority,
        tags: apiTicket.tags,
        responsible: apiTicket.responsible,
        description: apiTicket.description
    });
}
```

Slika 4.17: Metoda za dohvaćanje tiketa

Putanja dolazi do poslužiteljske strane gdje je definirana putanja sa Slike 4.18. Vrijednost 12 koja predstavlja identifikator traženog tiketa spremit će se u parametar *tickets* te će se pozvati funkcija *show()* definirana unutar klase kontrolera *TicketsController.*

```
// Single Ticket
Route::get('/ticket/{tickets}', [TicketsController::class, 'show']);
```

Slika 4.18: Putanja za dohvat tiketa

Funkcija *show* u varijablu *ticket* sprema rezultat upita prema *TicketsModel* klasi koristeći *Eloquent* metodu Tickets::where koja će pronaći unose u tablici koji odgovaraju upitu na Slici 4.19. Upit će dohvatiti prvi redak unutar tablice *tickets* gdje je identifikator odnosno *id* jednak identifikatoru spremljenom u varijabli *tickets* koji je dohvaćen iz URL-a poslanog sa klijentskog dijela aplikacije, odnosno pronaći će se tiket kojemu je vrijednost identifikatora jednaka 12.

```php
// Get Single Ticket Data
public function show(Tickets $tickets)
{

    $ticket = Tickets::where('id', $tickets->id)->first();

    return response()->json($ticket, 200);

}
```

Slika 4.19: Dohvaćanje tiketa prema zadanom upitu

Nakon upita prema bazi podataka u varijablu ticket se sprema redak iz tablice *tickets* prikazanoj na Slici 4.20. Podatci se u varijablu spremaju u JSON obliku koji je pogodan za obradu na klijentskoj strani web aplikacije. Podatci se u obliku odgovora šalju natrag na klijentski dio web aplikacije zajedno sa HTTP statusom 200 koji označava da je sve prošlo u redu, odnosno u ovom slučaju da je tiket uspješno dohvaćen.

id	user_id	title	priority	responsible	done	tags	description	date
12	4	Napraviti završni rad	Visoki	4	0	fakultet	Potrebno je napraviti web aplikaciju i napisati rad.	2023-09-07 21:52

Slika 4.20: Vrijednost dohvaćena iz baze podataka

Na klijentskoj se strani web aplikacije nakon uspješnog dohvaćenja podataka, vrijednosti spremaju u Hook funkciju *useState* za pohranu stanja. Istim principom se podatci o dohvaćenom tiketu spremaju pozivajući metodu *SetTicket* koja u *ticket* sprema podatke o tiketu koji se tada mogu dohvatiti pristupajući stanjima preko *ticket* elementa. Tek tada komponenta *TicketDetails* može koristiti vrijednosti o tiketu kako bi ih prikazala korisniku, a to čini pozivajući vrijednosti zapisane unutar elementa *ticket*, primjerice *ticket.title*. Služeći se Tailwind knjižnicom definira se stil uređenja naslova koji je, kao što je prikazano na Slici 4.21, centriran unutar *div* komponente te je uvećan sa marginama na dnu.

```
<div class="flex flex-col items-center justify-center text-center">

    <h3 class="text-4xl mb-1">
        {ticket.title}
    </h3>
```

Slika 4.21 Stiliziranje naslova

Kako bi komponenta dohvatila sve potrebne podatke preko *getUsers()* metode prikazanoj na Slici 4.22 šalje *GET* zahtjev služeći se Axios modulom na URL adresu http://localhost:8000/api/user gdje je bazni URL jednak http://localhost:8000/, a koji je dohvaćen iz Axios modula definiranog unutar vanjskog dokumenta.

```
const getUsers = async () => {
    const apiUsers = await axios.get("api/allusers");
    setUsers(apiUsers.data);
}
```

Slika 4.22: Metoda za dohvaćanje svih korisnika

Na poslužiteljskoj strani se zatim pronalazi putanja koja odgovara poslanoj putanji. Nakon pronalaska putanje kao na Slici 4.23 poziva se metoda *users* iz *UsersController* klase.

```
//Get all users
Route::get('/allusers', [UsersController::class, 'users']);
```

Slika 4.23: Putanja koja odgovara na zahtjev za dohvatom svih korisnika

Kao što je prikazano na Slici 4.24 metoda *users()* jednostavno poziva sve podatke iz *users* tablice unutar baze podataka, sprema ih u pogodan oblik, odnosno JSON format i u konačnici šalje odgovor klijentskoj strani web aplikacija unutar kojeg se nalaze podatci sa svim korisnicima web aplikacije.

```php
public function users()
{

    return response()->json(User::all(), 200);
}
```

Slika 4.24: Metoda za dohvat svih korisnika iz baze podataka

Podatci koje će baza podataka vratiti kontroleru prikazani su na Slici 4.25, a uključuju jedinstveni identifikator, ime, email te *hashiranu* lozinku.

id	name	email	password
2	Marko	marko@marko.com	$2y$10$qBiNEOZ80tmxHpDkPybe..jzmXIK5Cv93OmZ341LSxtsKG...
3	Ivan	ivan@ivan.com	$2y$10$BXZ44p4Y8n2q5ZJRSM1sZ.pSvby9zuyn6iqv2mXyZ/gHbK...
4	Mateo	mateo@mateo.com	$2y$10$iv8kwQaUQm58rlfAh1/ujurf6frjnCfRgR1uhuGCK1rZaBnFW...
5	Mia	mia@mia.com	$2y$10$xB1zenyl1kBpkxlC4gAaY.5Jc6Gx4DeP20BEYMbUwEbsUr.4...

Slika 4.25: Podatci o svim dohvaćenim korisnicama unutar baze podataka

Nakon što se podatci vrate na klijentsku stranu web aplikacije u obliku odgovora spremaju se metodom *setUsers()* u varijablu *users* koja je dio *useState Hook* funkcije. U konačnici komponenta *TicketDetails* se renderira te koristi dohvaćene korisnike kako bi se izvršila provjera, prikazana na Slici 4.26, o imenu osobe koja je izradila tiketa, odnosno kako bi se ime moglo prikazati. Provjera se vrši tako da se za identifikator svakog od dohvaćenih korisnika provjerava odgovara li vrijednosti *user_id* spremljenog u element *ticket* koji predstavlja vrijednost jedinstvenog ključa osobe koja je stvorila tiket.

```jsx
{users.map((userfind) => {
    if (ticket.user_id == userfind.id) {
        return (
            <div>Kreirao: {userfind.name}</div>
        )
    }

})}
```

Slika 4.26: Provjera o vlasniku tiketa

5 ZAKLJUČAK

Aplikacija Ticket.io služi za prikazivanje zadataka koje je potrebno riješiti u obliku tiketa. Tiketi se dodjeljuju određenom korisniku koji onda mora riješiti tiket. Aplikacija omogućuje jednostavno kreiranje tiketa i pregled nad njima što omogućuje timovima ljudi bolji pregled te transparentnost o trenutnim problemima koje je potrebno razriješiti. Svaki se tiket prikazuje tek uz nekoliko osnovnih podataka koje autor tiketa mora ispuniti, a što za rezultat ima da je svaki tiket jednostavno pregledati te se u svega nekoliko minuta već može započeti s njegovim rješavanjem.

Tehnologije korištene pri izradi ovoga rada (Laravel, React, MySQL, Tailwind, Axios) omogućuju jednostavnu i brzu implementaciju svih potrebnih funkcionalnosti. Tehnologije su jednostavne za korištenje te je potrebno poznavati tek osnove razvoja web aplikacija i otvara se mnoštvo mogućnosti koje je moguće ostvariti sa alatima opisanih u ovome radu.

Tako se u Laravelu pri implementaciji API putanja omogućava i korištenje middlewarea koji sprječava pristup resursima neautoriziranim korisnicima. Korištenjem Laravela nudi se i jednostavan, no ipak siguran sustava za autentifikaciju te autorizaciju koristeći Sanctum.

Jednostavnost pristupanja i manipuliranja podataka u bazi podataka koristeći MySQL pokazuje zašto je jedan od najkorištenijih alata za upravljanje relacijskim bazama podataka.

React omogućava izradu SPA web aplikacija izrađujući komponente te ih pozivajući i mijenjajući po potrebi. Svaka komponenta je ponovno upotrebljiva te je cijeli sustav intuitivan, a kod vrlo jednostavan za čitanje.

U kombinaciji sa Tailwindom omogućuje se brza izrada komponenti željenog izgleda bez potrebe za pisanjem CSS koda.

Mjesta za napredak ipak ima te bi se tako moglo omogućiti dodavanje timova koji bi bili u mogućnosti kreirati privatne tikete koji ne bi bili vidljivi ostalim timovima. Također nedostaje i

mogućnost oporavka lozinke u slučaju njenog zaboravljanja jer se u trenutnoj verziji aplikacije zaboravljanjem lozinke bespovratno gubi pristup računu.

Iako je aplikaciju moguće poboljšati, ipak u trenutnoj verziji je upotrebljiva te bi se mogla distribuirati timovima kako bi im se olakšao pregled nad svim zadatcima koje je potrebno rješavati.

BIBLIOGRAFIJA

[1] »What is REST,« [Mrežno]. Available: https://restfulapi.net. [Pokušaj pristupa 29. kolovoz 2023.].

[2] »HTTP Methods,« [Mrežno]. Available: https://doc.oroinc.com/api/http-methods/. [Pokušaj pristupa 1. rujan 2023.].

[3] »Laravel,« [Mrežno]. Available: https://laravel.com. [Pokušaj pristupa 28. kolovoz 2023.].

[4] »PHP,« [Mrežno]. Available: https://www.php.net. [Pokušaj pristupa 31. kolovoz 2023.].

[5] »MVC Arhitecture,« [Mrežno]. Available: https://www.tutorialspoint.com/mvc_framework/mvc_framework_introduction.htm. [Pokušaj pristupa 1. rujan 2023.].

[6] »MySQL,« [Mrežno]. Available: https://www.mysql.com. [Pokušaj pristupa 2. rujan 2023.].

[7] »React,« [Mrežno]. Available: https://react.dev. [Pokušaj pristupa 28. kolovoz 2023.].

[8] »Tailwind,« [Mrežno]. Available: https://tailwindcss.com. [Pokušaj pristupa 3. rujan 2023.].

[9] »What are core concepts of Laravel,« [Mrežno]. Available: https://www.resourcifi.com/blog/what-are-the-core-concepts-of-laravel-framework/. [Pokušaj pristupa 2. rujan 2023.].

[10] »How does sanctum work,« [Mrežno]. Available: https://dev.to/nicolus/laravel-sanctum-explained-spa-authentication-45g1. [Pokušaj pristupa 4. rujan 2023.].

[11] »CSRF,« [Mrežno]. Available: https://portswigger.net/web-security/csrf. [Pokušaj pristupa 4. rujan 2023.].

[12] »10 Basic Concepts About React You Should Learn,« [Mrežno]. Available: https://www.letsreact.org/10-basic-concepts-about-react-you-should-learn/. [Pokušaj pristupa 31. kolovoz 2023.].

[13] »React Hooks,« [Mrežno]. Available: https://legacy.reactjs.org/docs/hooks-overview.html. [Pokušaj pristupa 3. rujan 2023.].

[14] »Bootstrap,« [Mrežno]. Available: https://getbootstrap.com. [Pokušaj pristupa 5. rujan 2023.].

[15] »React Native,« [Mrežno]. Available: https://reactnative.dev. [Pokušaj pristupa 6. rujan 2023.].

[16] »FontAwesome,« [Mrežno]. Available: https://fontawesome.com. [Pokušaj pristupa 5. rujan 2023.].

[17] »Axios,« [Mrežno]. Available: https://axios-http.com/docs/intro. [Pokušaj pristupa 28. kolovoz 2023.].

[18] »Mastering React Navigation: A Comprehensive Guide To Routing In React JS,« [Mrežno]. Available: https://marketsplash.com/tutorials/react-js/navigation-in-react-js/. [Pokušaj pristupa 2. rujan 2023.].

[19] »Event: preventDefault,« [Mrežno]. Available: https://developer.mozilla.org/en-US/docs/Web/API/Event/preventDefault. [Pokušaj pristupa 4. rujan 2023.].

Sažetak

U ovom radu predstavljena je web aplikacija za ticketing sustav Ticket.io koja služi za kreiranje i uređivanje tiketa te dodjeljivanje ostalim korisnicima aplikacije. Aplikacija je temeljena na REST arhitekturi, a glavna tehnologija korištena za izradu poslužiteljskoj dijela ovog rada je Laravel dok je za klijentsku stranu korišten React.js. Od ostalih tehnologija korišten je MySQL za upravljanje relacijskom bazom podataka, Tailwind i FontAwesome za stiliziranje komponenti, Axios za slanje asinkronih zahtjeva te Sanctum za autentifikaciju i autorizaciju. Korištenjem aplikacije moguće je kreirati tiket, urediti ga i time dodijeliti novoj osobi ili ga potpuno izbrisati. Nakon izvršenja tiket je moguće zatvoriti i tako ga označiti kao riješenim. Također prilikom prikazivanja liste s tiketima nudi se opcija pretraživanja, sortiranja i filtriranja.

Ključne riječi: web aplikacija, tiketi, Laravel, React, REST

Abstract

This paper presents a Ticket.io web application for a ticketing system which is used for creating and editing tickets and assigning them to other users of the application. The application is based on the REST architecture, with Laravel being the primary technology used for the server-side implementation, while React.js is used for the client-side. Other technologies used to make this apllication include MySQL for managing the relational database, Tailwind and FontAwesome for styling components, Axios for handling asynchronous requests, and Sanctum for authentication and authorization. Using the application, users can create tickets, edit them, assign them to new individuals, or delete them entirely. After the users finishes the task described in ticket it can be closed, marking ticket as resolved. Additionally, when displaying a list of tickets, options for searching, sorting, and filtering are provided.

Keywords: web application, tickets, Laravel, React, REST