RESTful web aplikacija za kupnju avionskih karata

Utmar, Darjan

SVEUČILIŠTE U RIJECI

TEHNIČKI FAKULTET

Prijediplomski sveučilišni studij računarstva

Završni rad

RESTful web aplikacija za kupnju avionskih karata

Rijeka, srpanj 2024.

Darjan Utmar

0069088017

SVEUČILIŠTE U RIJECI

TEHNIČKI FAKULTET

Prijediplomski sveučilišni studij računarstva

Završni rad

RESTful web aplikacija za kupnju avionskih karata

Mentor: Izv. prof. dr. sc. Marko Gulić

Rijeka, srpanj 2024.

Darjan Utmar

0069088017

SVEUČILIŠTE U RIJECI
TEHNIČKI FAKULTET
POVJERENSTVO ZA ZAVRŠNE ISPITE

Rijeka, 21.03.2024.

Zavod: Zavod za računarstvo

Predmet: Razvoj web aplikacija

ZADATAK ZA ZAVRŠNI RAD

Pristupnik: **Darjan Utmar (0069088017)**

Studij: Sveučilišni prijediplomski studij računarstva (1035)

Zadatak: **RESTful web aplikacija za kupnju avionskih karata / RESTful web application for buying airline tickets**

Opis zadatka:

Razviti RESTful web aplikaciju za kupnju avionskih karata. Aplikacija treba imati odvojeni administracijski dio i korisnički dio. Administrator mora imati mogućnosti dodavanja, brisanja i ažuriranja podataka o letovima, pregledu svih kupljenih karata preko aplikacije te pregledu podataka o svim korisnicima. Registrirani korisnik mora imati mogućnost pretraživanja letova s obzirom na odabrane filtere, te mogućnost kupnje karata za određeni let. Kupnja željene karte mora se izvesti uz online plaćanje koristeći proizvoljno odabrani online servis za plaćanje. 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 knjižicu za razvoj korisničkog sučelja uz učinkovito renderiranje aplikacije na uređajima s različitim veličinama zaslona.

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

Zadatak uručen pristupniku: 22.03.2024.

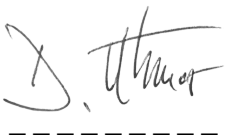
Predsjednik povjerenstva za
završni ispit:
prof. dr. sc. Miroslav Joler

Mentor:
doc. dr. sc. Marko Gulić

Izjava o samostalnoj izradi rada

Izjavljujem da sam samostalno izradio ovaj rad

Rijeka, srpanj 2024.



Darjan Utmar

Zahvala

Zahvaljujem se mentoru, izv. prof. dr. sc. Marku Guliću na strpljenju za svako pitanje i vremenu koje je izdvojio tijekom izrade ovog rada. Zahvaljujem se obitelji koja je bila uz mene tijekom studiranja.

SADRŽAJ

1 UVOD

U ovom radu izrađena je RESTful [1] web aplikacija za kupnju avionskih karata "Flighter" koja služi za pretragu i kupnju avionskih karata na pojedinim rutama. Korisnik može pretraživati i pregledavati letove bez vlastitog računa, ali ako želi kupiti avionsku kartu tada mora kreirati vlastiti račun registrirajući se unosom imena, emaila i lozinke. Također, na stranici predviđenoj za to, mogu se vidjeti sve aviokompanije čiji su letovi registrirani na aplikaciji.

Aplikacija je bazirana na *Representational State Transfer* (skraćeno, REST) [2] arhitekturi koja se oslanja na razmjenu podataka putem standardnih HTTP [3] zahtjeva kao što su GET, POST, PUT i DELETE. Tehnologije koje su korištene za razvoj aplikacije su podijeljene na klijentski i poslužiteljski dio.

Za razvoj klijentskog dijela aplikacije korištena je JavaScript knjižnica otvorenog koda React.js [4]. To je knjižnica koja se temelji na iskoristivosti svojih komponenti. Jednostavna instalacija React projekta stvara sve potrebne datoteke i module za rad.

Poslužiteljski dio je razvijen u Laravelu [5], radnom okviru koji je utemeljen na PHP-u [6]. Laravelov radni okvir je temeljen na MVC (eng. *Model-View-Controller*) [7] arhitekturi. Omogućuje implementaciju jednostavne autorizacije i autentifikacije, koje su neizostavne funkcionalnosti skoro svake web aplikacije. Instalira se putem Composer [8] paketa.

Sustav za upravljanje relacijskim bazama podataka MySQL[9] korišten je u izradi ove aplikacije.. MySQL koristi *Structured Query Language* (skraćeno, SQL) [10] upite za manipuliranjem podacima unutar tablica i baza. Kako bi bilo lakše vidjeti promjene koje se događaju nad bazom korišten je MySQL Workbench [11].

Stiliziranje komponenti i HTML-a kao cjeline izvedeno je pomoću običnog CSS-a za koji nije potrebna dodatna instalacija.

Tehnologije izabrane za razvoj ove web aplikacije izabrane su zbog njihovih karakteristika kod korištenja. Prednosti React knjižnice su deklarativni pristup programiranju i laka integracija s drugim knjižnicama i radnim okvirima. Laravel radni okvir ima svoje prednosti kao što su jednostavna i lako razumljiva sintaksa, velik broj dodatnih paketa koji proširuju radni okvir kao što je Sanctum te objektno-relacijsko preslikavanje (eng.

Object-Relational Mapper, ORM) - Eloquent, koji omogućava jednostavnu interakciju programskog koda s bazom podataka kroz objektno orijentirane modele.

Aplikacija je izrađena s ciljem pružanja jednostavnog i intuitivnog korisničkog sučelja bez previše popratnog sadržaja koji ometa korisnike.

2 TEHNOLOGIJE

2.1 React

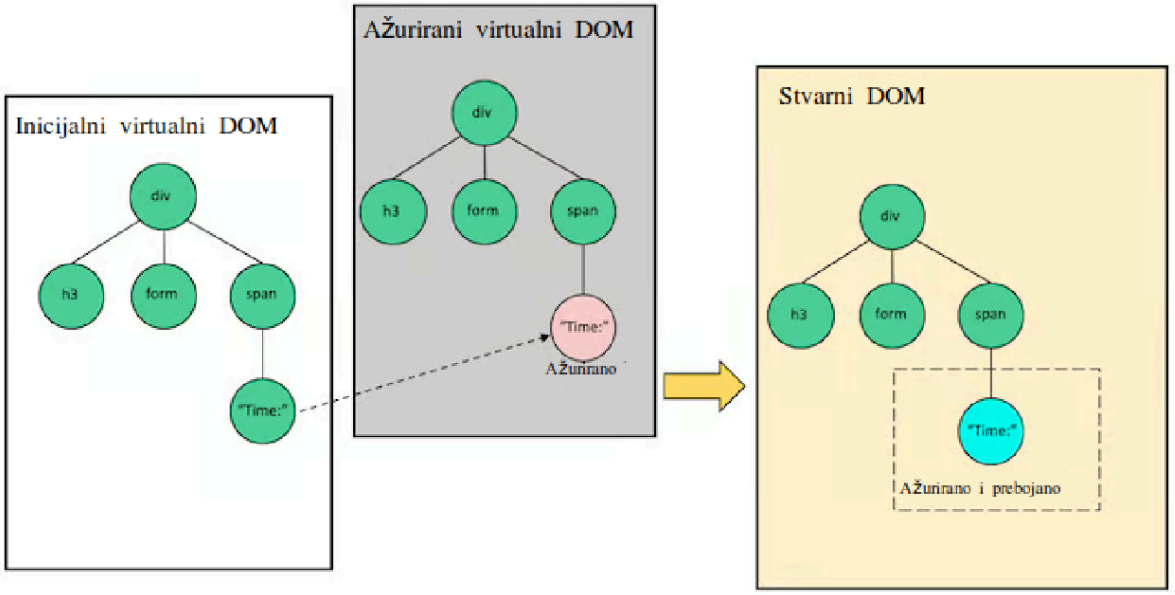
React je knjižnica napravljena za jednostavnu izradu i razvoj interaktivnih korisničkih sučelja koju održava Meta [12]. Njena glavna značajka je sposobnost automatskog renderiranja komponenti prilikom promjene podataka. To znači da nije potrebno ponovno pokrenuti program prilikom svake promjene koda, nego se promjene odmah izmjene. Takav način programiranja omogućuje vrlo visoku učinkovitost i produktivnost jer se ne gubi vrijeme na stalno pokretanje programa.

2.1.1 Deklarativno programiranje

React koristi deklarativni pristup programiranju. To znači da se opisuje kako izgleda korisničko sučelje za svako definirano stanje aplikacije, a React se brine o ažuriranju i renderiranju komponenti kada se stanje promijeni. Takav kod je lakši za čitanje i razumijevanje [13].

2.1.2 Virtualni DOM

DOM je strukturalni model dokumenta (HTML-a ili XML-a) koji prikazuje stranicu unutar preglednika. Koristeći virtualni DOM (eng. *Document Object Model*) kao programski koncept, gdje se "virtualno" korisničko sučelje čuva u memoriji i sinkronizira se s "realnim" DOM-om putem knjižnice kao što je ReactDOM, React osigurava efikasan i brz proces pri renderiranju aplikacija . Taj proces opisan je slikom 2.1 [14]. Takav pristup omogućava deklarativnost: opiše se stanje u kojem korisničko sučelje treba biti te se React potrudi da DOM odgovara tom stanju [15]. Prednosti korištenja virtualnog DOM-a u prilikom programiranja su povećane performanse kroz smanjivanje broja operacija nad stvarnim DOM-om te efikasno upravljanje promjenama. *Diffing* algoritam [16] kojeg sadržava React omogućava efikasno praćenje i primjenu samo potrebnih promjena.



Slika 2.1 - Grafički prikaz funkcije virtualnog DOM-a

2.1.3 React Router

Knjižnica React Router [17] omogućuje navigaciju i putanje unutar React aplikacija. Koristi se za promjenu URL-a preglednika te kako bi korisničko sučelje bilo sinkronizirano s URL-om. Omogućava korisnicima razvoj jednostrukih stranica (eng. *single page applications*, skraćeno, SPA) s kompleksnim navigacijskim strukturama. Slika 2.2 prikazuje isječak koda koji definira putanje kroz aplikaciju. Prije korištenja potrebno je instalirati i uvesti potrebne stavke i *react-router-dom* knjižnice. Unutar komponenti <*Router*> i <*Route*> postavljaju se putanje i komponente koje se trebaju renderirati ukoliko korisnik navigira na URL adresu putanje.

```
<Router>
  <Routes>
      <Route path='/' Component={HomePage} />
      <Route path='/cart' Component={ShoppingCart} />
      <Route path='/about' Component={About} />
      <Route path='/contact' Component={Contact} />
      <Route path='/flights' Component={Flights} />
      <Route path='/login' Component={Login} />
      <Route path='/register' Component={Register} />
      <Route path='/add' Component={AddFlight} />
      <Route path='/update' Component={UpdateFlight} />
      <Route path='/delete' Component={DeleteFlight} />
      <Route path='/tickets' Component={Tickets} />
      <Route path='/checkout' Component={Checkout} />
      <Route path='/users' Component={Users} />
  </Routes>
</Router>
```

Slika 2.2 - Definiranje putanja unutar aplikacije

2.1.4 React Hooks

React kuke (eng. *hooks*) omogućavaju funkcionalnim komponentama mogućnost korištenja stanja i upravljanje promjenama i događajima. One su uvedene u React verziji 16.8. Kuke se mogu pozivati jedino na gornjem nivou komponente te one kao takve ne mogu biti kondicionalne. *React Hooks* eliminiraju potrebu za klasnim komponentama te čine kod jednostavnijim za održavanje.

Najčešća kuka je kuka stanja koja se inicijalizira pozivom *useState()* varijable te se u nju mogu spremati podaci. Ti podaci se nalaze unutar varijable state, a vrijednost se može mijenjati pozivom *setState()* metode. Primjer korištenja kuke stanja prikazan je na Slici 2.3. Da bi koristili *usuState* prvo ga je potrebno uvesti iz React knjižnice. Jedna od prednosti korištenja ove kuke je lokalnost, stanje je lokalno za svaku komponentu pojedinačno te tako olakšava praćenje koda.

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

function Example() {
  // Declare a new state variable, which we'll call "count"
  const [count, setCount] = useState(0);

  return (
    <div>
      <p>You clicked {count} times</p>
      <button onClick={() => setCount(count + 1)}>
        Click me
      </button>
    </div>
  );
}
```

Slika 2.3 - UseState kuka

Druga najčešća kuka se inicijalizira putem poziva *useEffect()*. Ta kuka služi za događaje prilikom promjene stanja kao što su dohvaćanje podataka i manipulacija DOM-om. Ova kuka se poziva nakon svakog renderiranja komponente te je kao takva jedna od neizostavnih funkcionalnosti React radnog okvira. Slika 2.4 se nadovezuje na prijašnju sliku te nakon svakog promjene podataka mijenja naslov dokumenta [18]. Isto kao i kod *useState()* kuke, prvo je potrebno uvesti kuku kako bi se mogla koristiti.

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

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

  // Similar to componentDidMount and componentDidUpdate:
  useEffect(() => {
    // Update the document title using the browser API
    document.title = `You clicked ${count} times`;
  });

  return (
    <div>
      <p>You clicked {count} times</p>
      <button onClick={() => setCount(count + 1)}>
        Click me
      </button>
    </div>
  );
}
```

Slika 2.4 - UseEffect kuka

UseContext kuka je još jedna od često korištenih kuka u Reactu. Ona omogućava način prosljeđivanja podataka bez potrebe za ručnim prosljeđivanjem podataka nižim nivoima komponenti (eng. *prop-drill*). *UseContext* pruža jednostavan način za dijeljenje stanja i funkcionalnosti između komponenti. Idealan je za podatke koji su globalni na razini aplikacije. To su podaci kao što su autentifikacija, jezik aplikacije i slično. Primjer pozivanja *useContext* kuka je na Slici 2.5 [19].

```jsx
import { useContext } from 'react';

function MyComponent() {
  const theme = useContext(ThemeContext);
```

Slika 2.5 - UseContext kuka

2.1.5 Axios

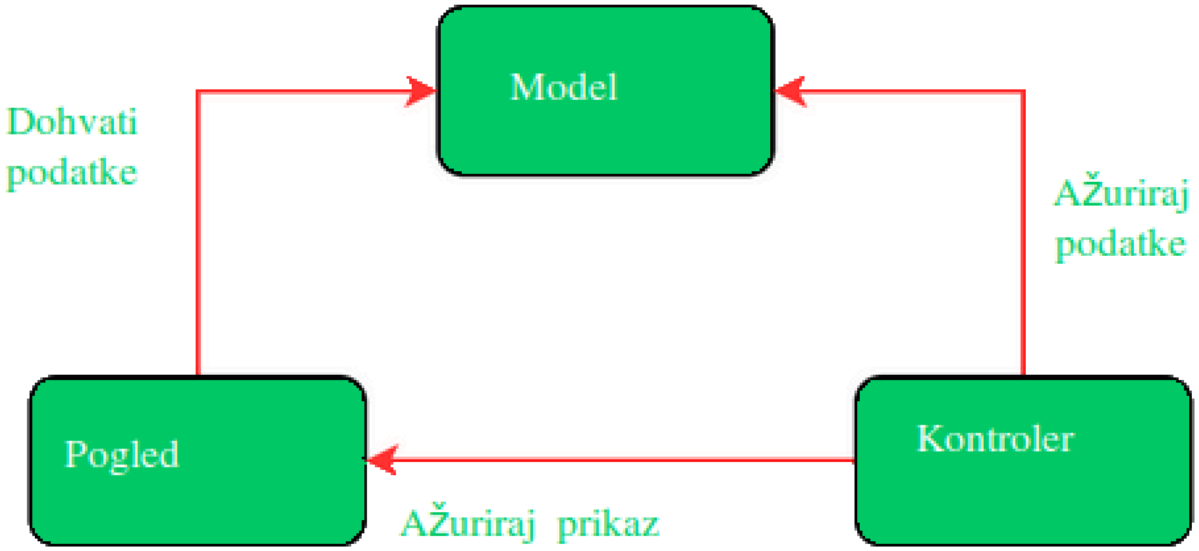
Axios [20] je još jedna popularna JavaScript knjižnica koja se može priključiti Reactu. Ona omogućava slanje HTTP zahtjeva iz preglednika prema krajnjim točkama (eng. *endpoint*) na poslužiteljskoj strani. Glavna značajka Axiosa su njegovi asinkroni zahtjevi koji su temeljeni na obećanjima (eng. *promises*). Kroz ta obećanja se "obećaje" da će se vratit vrijednost u nekom trenutku te da se ostatak koda može izvršiti do kraja bez čekanja. Za njegovu popularnost zaslužna je njegova jednostavna sintaksa koja se temelji na definiranju HTTP zahtjeva, to jest definiranja radi li se o GET, POST, PUT ili DELETE metodi te definiranja putanje prema kojoj se šalje zahtjev. Primjer Axios zahtjeva može se vidjeti na Slici 2.5.

```
await apiClient.get('/api/tickets');
```

Slika 2.5 - Axios zahtjev pomoću definiranog URL-a apiClient

2.2 Laravel

Laravel je popularni, besplatni radni okvir otvorenog koda koji je temeljen na PHP-u, kreiran od strane Taylora Otwella. Napravljen je za razvoj web aplikacija koristeći model-pogled-kontroler (eng. *Model-View-Controller*, skraćeno, MVC) arhitekture te je baziran na Symfony [21] random okviru. Njegova MVC arhitektura razdvaja logiku aplikacije od korisničkog sučelja. Na Slici 2.6 [22] je prikazana MVC arhitektura, gdje pogled predstavlja sadržaj prikazan korisniku, model predstavlja korištene podatke, a kontroler je poveznica između pogleda i modela putem kojeg se izvršava poslovna logika. Zbog toga je kod organiziraniji i lakši za održavanje. Laravel je programerima poznat po elegantnoj i lako čitljivoj sintaksi koja omogućava jednostavno i čisto pisanje koda. Takva sintaksa uvelike povećava produktivnost i smanjuje količinu grešaka.



Slika 2.6 - MVC arhitektura

2.2.1 Eloquent

Radni okvir Laravel dolazi s ugrađenim objektno-relacijskim preslikavanjem (eng. *Object-Relational Mapping*, skraćeno, ORM) [23], koji omogućava interakciju s bazama podatka pomoću objekata umjesto korištenja cijelih SQL naredbi. To čini manipulaciju bazama podataka efikasnim i intuitivnim. Eloquent [24] je paket zaslužan za Laravelov ORM. Sintaksa ORM-a je intuitivna i jednostavna pa se zbog toga koriste naredbe kao što su *get*, *create* i *update*. Na Slici 2.7 je primjer korištenja Eloquenta kako bi se dodao novi podatak u bazu preko modela čije ime odgovara nazivu tablice u bazi podataka.

```
$user = User::create([
    'name' => $fields['name'],
    'email' => $fields['email'],
    'password' => bcrypt($fields['password']),
]);
```

Slika 2.7 - Eloquent metoda za unos u bazu podataka

Migracije unutar Laravela omogućuju verziranje baze podataka, odnosno olakšava stvaranje i mijenjanje baza. Pomoću sjemenki (seeds) brzo i jednostavno popunjavamo baze podataka početnim podacima [25]. Slika 2.8 prikazuje kako smo pomoću *seedera* dodali administratorski račun u aplikaciju. *Seeder* je klasa koja popunjava podacima relacijsku bazu podataka kroz izvršavanje naredbi koje programer napiše.

```php
public function run(): void
{
    $users = [
        [
            'name' => 'admin',
            'email' => 'admin@admin.com',
            'password' => 'admin123',
            'admin_privilege' => 'true',
        ]
    ];

    foreach($users as $key => $value) {
        User::create($value);
    }
}
```

Slika 2.8 - Seeder

Pomoću Laravela vrlo jednostavno se definiraju API (eng. *Application Programming Interface*) [26] rute. Rute se povezuju s kontrolerima unutar kojih se za svaki od HTTP protokola nalazi funkcija koja će se odraditi prilikom poziva određene rute.

2.2.2 Sanctum

Sanctum [27] je paket za jednostavnu autentifikaciju API poziva koji pristižu s klijentskog dijela na poslužiteljski dio aplikacije i zaštitu SPA [28]. Pruža lagan i jednostavan način za upravljanje autentifikacijom korisnika putem tokena. Svaki korisnik može kreirati više API tokena koji onda služe za različite privilegije unutar aplikacije. Prilikom prijave korisnika, ukoliko su podaci za prijavu točni, generira se token koji se pohranjuje u klijentskom dijelu aplikacije. Za pristup zaštićenim resursima, unutar zahtjeva šalje se i pohranjeni token te, ako je ispravan, korisnik može pristupiti zaštićenim resursima. Sanctum automatski upravlja CSRF (eng. *Cross-Site Request Forgery*) zaštitom. Kada korisnik pristupi web aplikaciji, Sanctum postavlja CSRF token koji se koristi za validaciju svih budućih zahtjeva. Također, koristi se *middleware* [29] za zaštitu ruta koje zahtjevaju autentifikaciju koji možemo vidjeti na Slici 2.9. *Middleware* je poveznica između zahtjeva i reakcije na taj zahtjev. Unutar Laravel radnog okvira radi tako da provjerava je li korisnik aplikacije verificiran.

```
//Protected routes
Route::group(['middleware' => ['auth:sanctum']], function() {
    Route::post('/logout', [AuthController::class, 'logout']);
    Route::get('/user', [AuthController::class, 'users']);

});
```

Slika 2.9 - Definicija auth:sanctum middlewarea unutar grupe putanja

2.3 MySQL

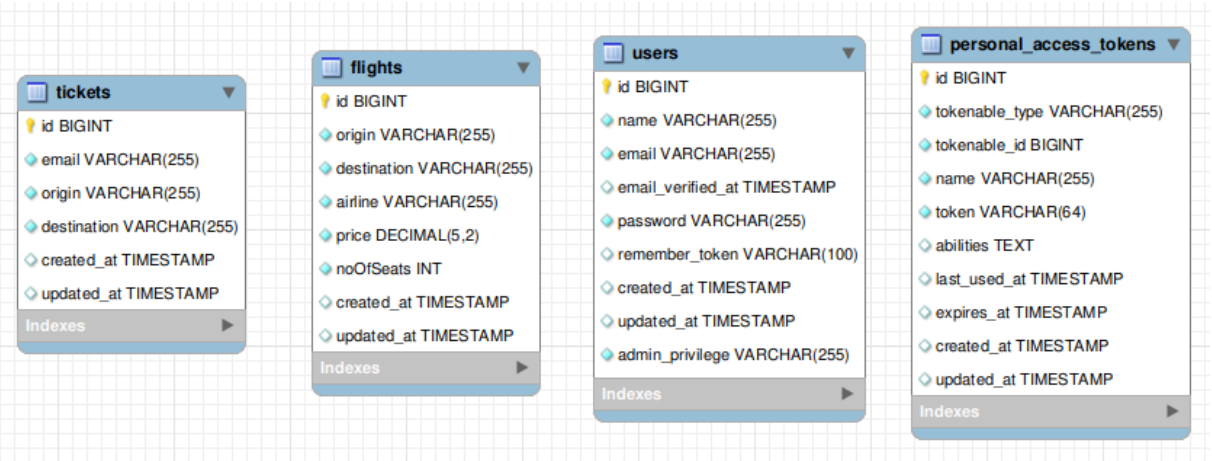
MySQL je jedan od najpopularnijih besplatnih sustava otvorenog koda za upravljanje relacijskim bazama podataka (eng. *Relational Database Management System*, skraćeno, RDBMS) [30] na svijetu. Razvila ga je tvrtka MySQL AB, a kasnije ga je preuzeo Oracle Corporation. MySQL je u programerskim krugovima poznat po svojoj brzini, pouzdanosti i jednostavnosti korištenja.

Podaci u MySQL-u su organizirani unutar entiteta, odnosno tablica koje su međusobno povezane. Svaka od tablica sadrži stupce koji predstavljaju tip podataka koji očekujemo. Isto tako sadrži i retke koji predstavljaju konkretne vrijednosti. Putem SQL upita se dohvaća, mijenja te upravlja podacima iz baze. Primjer upita u MySQL bazu kroz Laravel radni okvir može se vidjeti na Slici 2.10.

```
return Flights::where([
    ['origin', $origin],
    ['destination', $destination],
    ])->get();
```

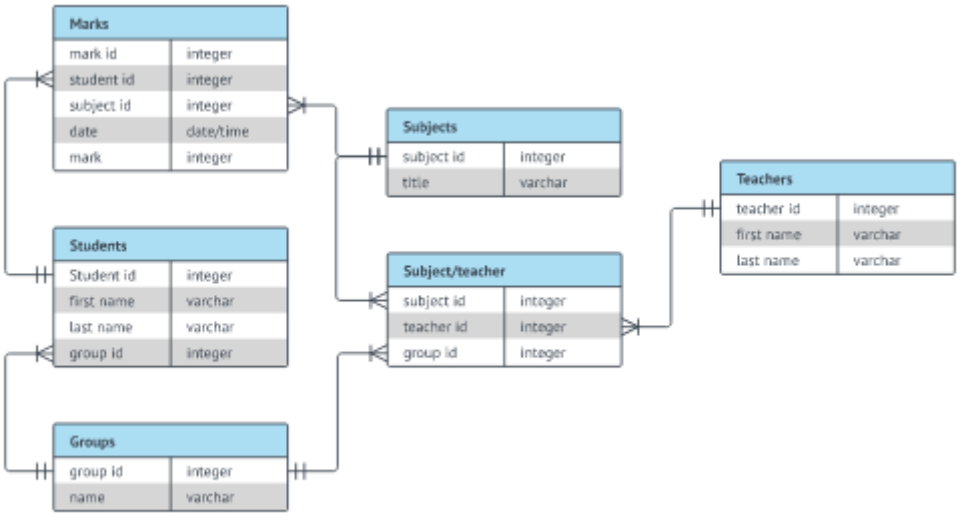
Slika 2.10 - Dohvaćanje podataka iz baze pomoću ORM upita

Glavna značajka MySQL je sposobnost postavljanja relacija između entiteta što omogućuje efikasnost i organiziranost unutar baze. Koristeći se vizualnim modelom, odnosno koristeći *Entity Relationship* (ER) [31] dijagram. Svaka tablica unutar dijagrama sadrži skup atributa koji opisuju taj entitet. Također kroz ER dijagram se mogu vidjeti relacije između pojedinih entiteta. Slika 2.11 prikazuje ER dijagram entiteta korištenih u izradi ove aplikacije.



Slika 2.11 - ER dijagram korišten u izradi ove aplikacije

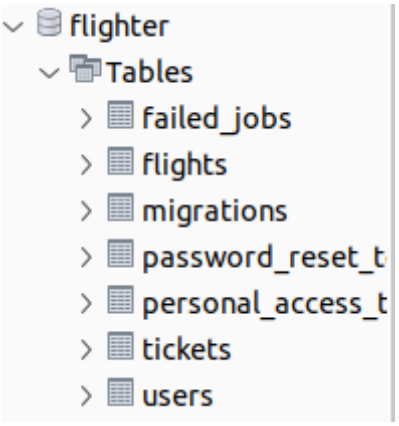
ER dijagram s povezanim entitetima i tablicama može se vidjeti na Slici 2.12 [32]. Isto tako na ovakvom se dijagrami mogu vidjeti i vrste veza s kojima su entiteti međusobno povezani.



Slika 2.12 - Povezani ER dijagram

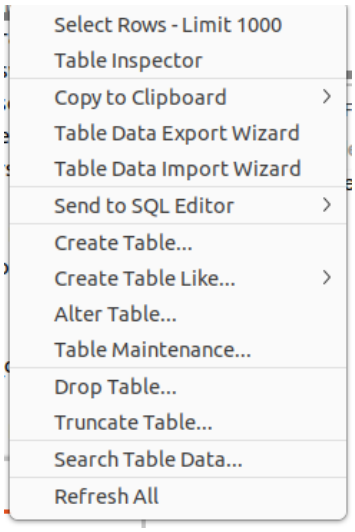
2.3.1 MySQL Workbench

Za lakšu vizualizaciju podataka spremljenih u bazu podataka korišten je MySQL Workbench. MySQL Workbench je vizualni alat razvijen od strane Oraclea za rad s MySQL bazama podatka. Taj alat pruža modeliranje podatka, SQL razvoj i administratorske alate za konfiguraciju poslužitelja. Primjer korisničkog sučelja MySQL Workbencha koji prikazuje sve tablice unutar jedne baze nalazi se na Slici 2.12.



Slika 2.13 - Tablice unutar baze u MySQL Workbenchu

Primjer mogućnosti rada nad podacima unutar tablice je prikazan na Slici 2.13. Podaci se mogu mijenjati unutar MySQL Workbencha te su njihov promjene odmah dostupne u aplikaciji. Isto tako, kroz njegovo sučelje je lako doći do ER dijagrama pripadnog za bazu podataka.

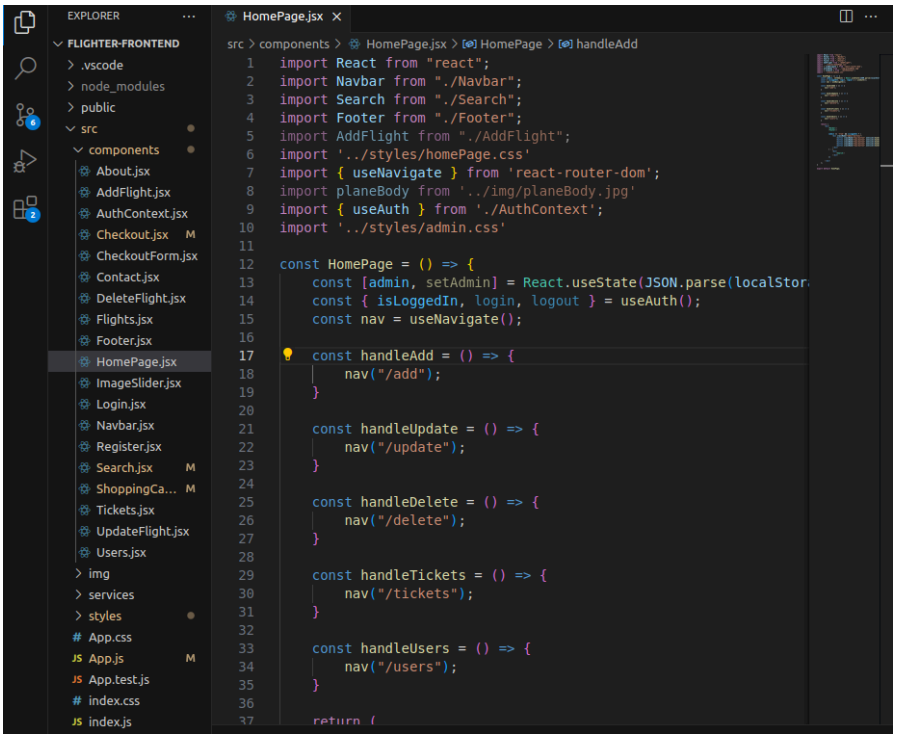


Slika 2.14 - Mogućnosti rada nad tablicom

2.4 Visual Studio Code

Visual Studio Code [33] je besplatan i moćan uređivač izvornog koda razvijen od strane Microsofta 2015.godine. Dostupan je za sve najpopularnije operacijske sustave kao što su Windows, MacOS i Linux. Podržava širok spektar programskih jezika uključujući JavaScript, TypeScript, Python, Java, C++ PHP i Ruby. Nudi značajke kao što su podrška za otklanjanje pogrešaka (eng. *debugging*), isticanje sintakse i ključnih riječi različitim bojama te

ima ugrađenu podršku za *Git* sustav za verziranje. Pored toga ima ugrađen i terminal u kojem se mogu pokretati *Bash* skripte. Visual Studio Code je bogat ekstenzijama gdje njegovi korisnici mogu naći i instalirati različite alate za razvoj kao i teme i prilagodbe. Sučelje *IntelliSense* unutar Visual Studio Codea pruža automatsko dovršavanje koda što ubrzava pisanje koda te smanjuje broj grešaka. Unutar ovog uređivača koda moguće je istovremeno imati otvoren veliki broj datoteka što olakšava snalaženje unutar velikih projekata. Primjer korisničkog sučelja ovog razvojnog okruženja prikazan je na Slici 2.14.



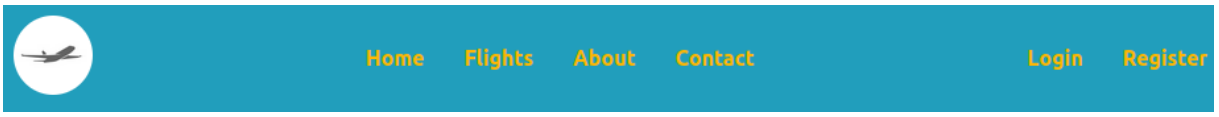
Slika 2.15 - Prikaz Visual Studio Code IDE-a

Anketa koju je proveo StackOverflow 2021.godine [34], otkrilo se kako Visual Studio Code koristi barem 70% ispitanika što ga čini najpopularnijim razvojnim okruženjem.

3 APLIKACIJA ZA KUPNJU AVIONSKIH KARATA

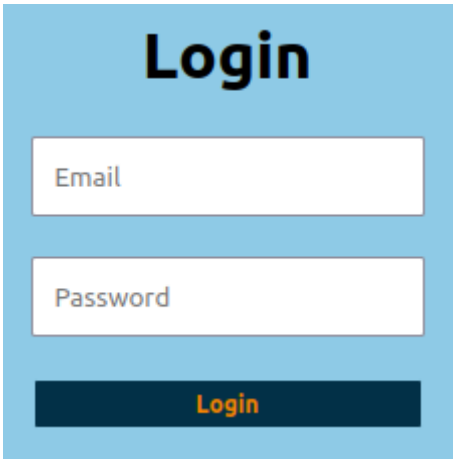
U ovom je poglavlju prikazano korisničko sučelje aplikacije uz objašnjenje funkcionalnosti te uputama o korištenju određenih funkcionalnosti.

Korisniku je omogućen pristup pojedinim funkcionalnostima bez prethodne prijave, a nakon što se prijavi može pristupiti svim funkcionalnostima. Dok korisnik nije prijavljen, na navigacijskoj traci bit će prikazani gumbi *Login* i *Register (Slika 3.1)*



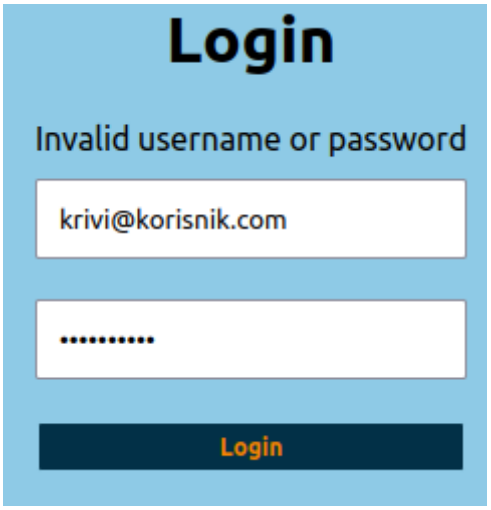
Slika 3.1 - Navigacijska traka za neprijavljenog korisnika

Stranica za prijavu prikazana je Slikom 3.2. Za uspješnu prijavu potrebno je unijeti email i lozinku prethodno registriranog korisnika. Nakon pritiska na *Login* gumb, ukoliko su uneseni podaci ispravni, korisnik je preusmjeren na početnu stranicu aplikacije.



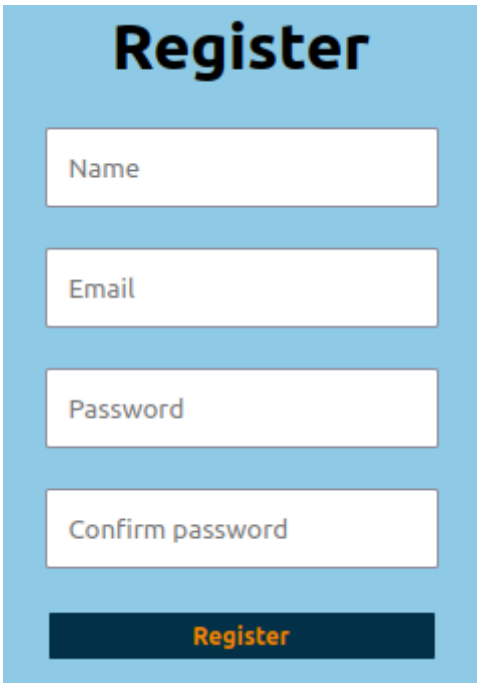
Slika 3.2 - Login forma za prijavu

Ukoliko podaci ne odgovaraju niti jednom registriranom korisniku, ispisuje se greška *Invalid email or password* na ekran, kao što se može vidjeti na Slici 3.3. Greška se ispisuje tek nakon pritiska na gumb *Login.*



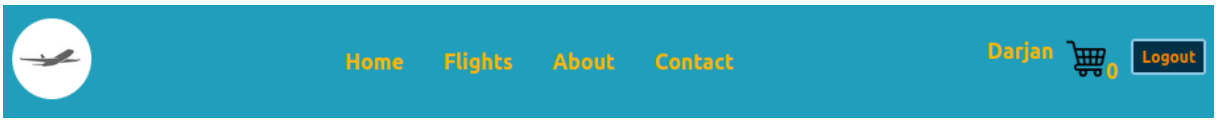
Slika 3.3 - Greška zbog pogrešno upisanih podataka

Ako korisnik prvi put posjećuje aplikaciju, biti će potrebna registracija koja se može izvršiti upisivanjem imena, emaila i potvrđene lozinke unutar forme za registraciju koja je prikazana na Slici 3.4. Lozinka mora sadržavati 8 znakova inače registracija neće biti moguća. Nakon unošenja podataka potrebno je stisnuti na *Register* gumb. Nakon registracije, korisnik je preusmjeren na stranicu s *Login* formom kako bi se mogao prijaviti u aplikaciju te koristiti sve njezine funkcionalnosti.



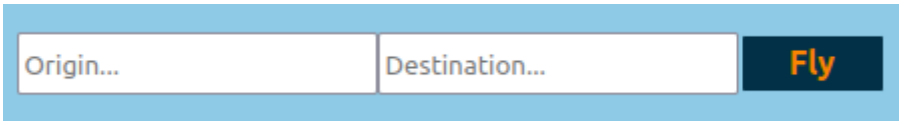
Slika 3.4 - Register forma za prijavu

Nakon uspješne prijave ili registracije korisnik je preusmjeren na početnu stranicu aplikacije gdje može pretraživati željene letove. Na navigacijskoj traci mu se otvara pristup novim gumbima, ikonica košarice uz koju stoji broj letova trenutno u košarici i gumb za odjavu (Slika 3.5).



Slika 3.5 - Navigacijska traka za prijavljenog korisnika

Na početnoj stranici nalaze se polja za upis početne i odredišne destinacije koja možemo vidjeti na Slici 3.6. Algoritam za pretragu je napravljen tako da će prepoznati željene aerodrome i ako se u polje za upis upišu samo početna slova destinacije.

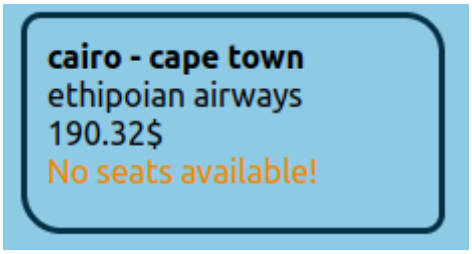


Slika 3.6 - Polja za pretraživanje letova

Nakon upisivanja željenih odredišta te pritiska na gumb *Fly*, prikazati će nam se svi registrirani letovi za tu rutu. Za svaki od letova biti će prikazana njegova ruta, aviokompanija koja upravlja na toj ruti, cijena leta te ukoliko je korisnik registriran i ima dostupnih sjedala, gumb *Add to cart* za dodavanje leta u košaricu (Slika 3.7).

Slika 3.7 - Prikaz pretraženih letova

Ukoliko let nema dostupnih sjedala, umjesto gumba *Add to cart* prikazati će se poruka *No seats available!*. Poruka da nema dostupnih sjedala prikazana je na Slici 3.8.

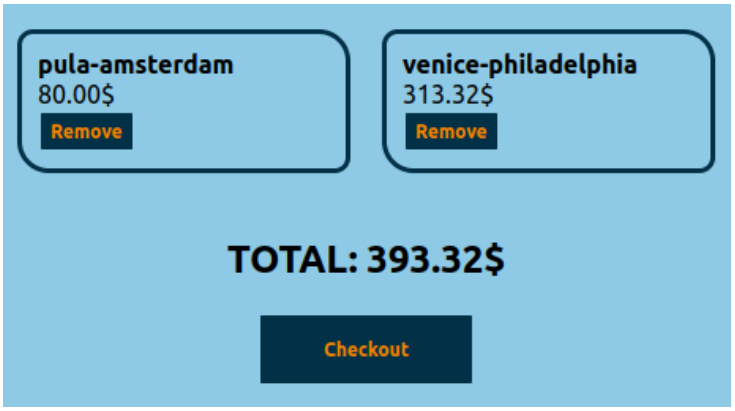


Slika 3.8 - Prikaz poruke ako nema dostupnih sjedala

Pritiskom na gumb *Add to cart* odabrani se let seli u košaricu te se na navigacijskoj traci ažurira broj uz ikonu košarice (Slika 3.9). Broj letova u košarici ažurira se u realnom vremenu tako da korisnik bude svjestan kada je dodao željeni let u košaricu.

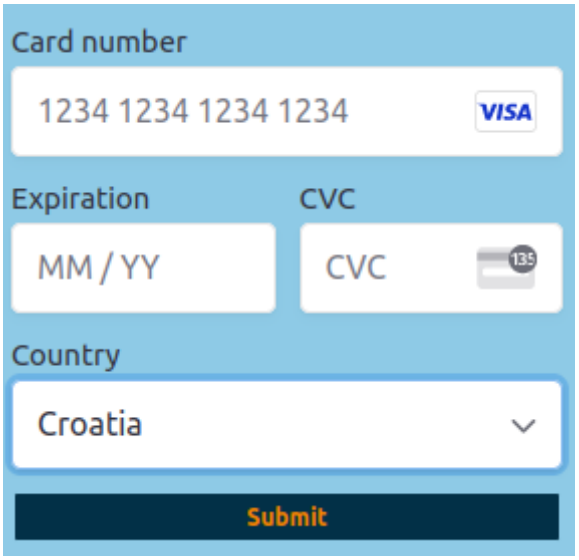
Slika 3.9 - Ažuriran broj letova u košarici

Slika 3.10 predstavlja pogled na košaricu u kojoj se nalaze odabrani letovi, do koje se dođe pritiskom na ikonu košarice. Unutar košarice je prikazana i ukupna cijena svih odabranih letova te gumb *Checkout* koji služi kako bi se otvorila forma za kartično plaćanje.



Slika 3.10 - Prikaz letova u košarici

Pritiskom na gumb *Checkout* prikazuje se forma za upisivanje podataka kartice s kojom se želi platiti, koja se nalazi na Slici 3.11. Država plaćanja se automatski popuni s obzirom na geografsku lokaciju korisnika.

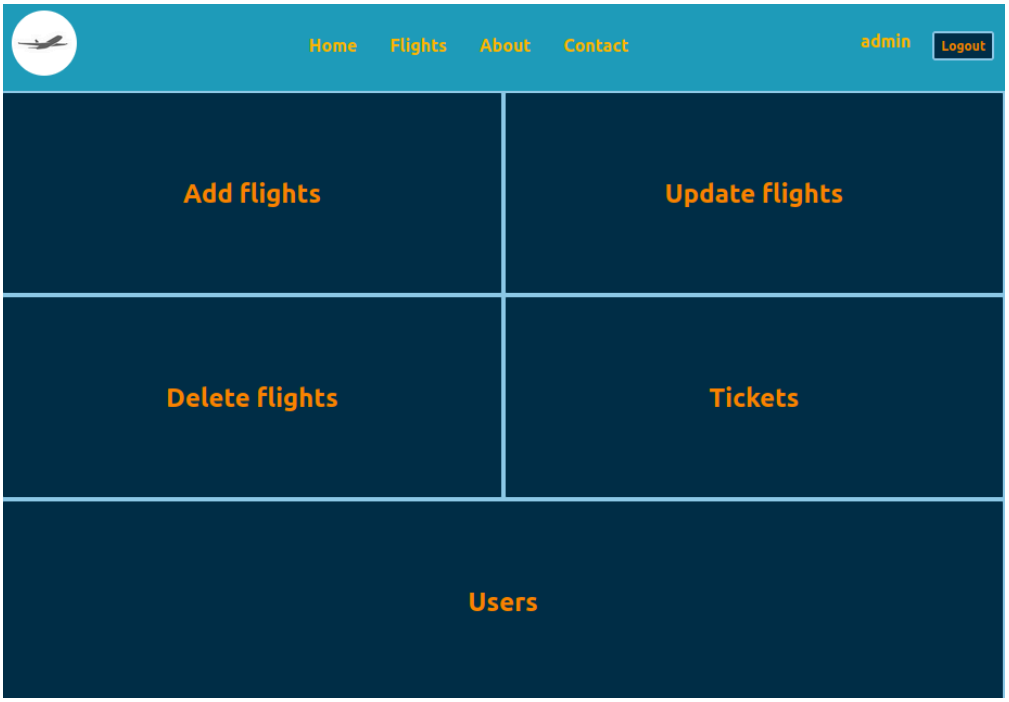


Slika 3.11 - Forma za kartično plaćanje

Nakon pritiska gumba *Submit*, ukoliko su kartični podaci ispravni, aplikacija nas preusmjerava na početnu stranicu aplikacije. Isto tako, kupljene se karte spremaju u bazu podataka po korisniku kako bi administrator aplikacije mogao pristupiti tim podacima.

Ova aplikacija ima predviđenog administratora koji može upravljati funkcijama aplikacije. Administrator može dodati novi let, ažurirati podatke o već dodanom letu, obrisati let, te vidjeti informacije o korisnicima i njihovim kupljenim kartama. Administrator se prijavljuje u aplikaciju posebnim podacima koji su definirani od strane programera.

Slika 3.12 prikazuje početnu stranicu kada se administrator prijavi u aplikaciju pomoću administratorskog emaila i lozinke. Kao što se i može vidjeti, administrator nema mogućnosti pretraživanja letova i dodavanja letova u košaricu te nema prikazanu ikonu za košaricu na navigacijskoj traci.



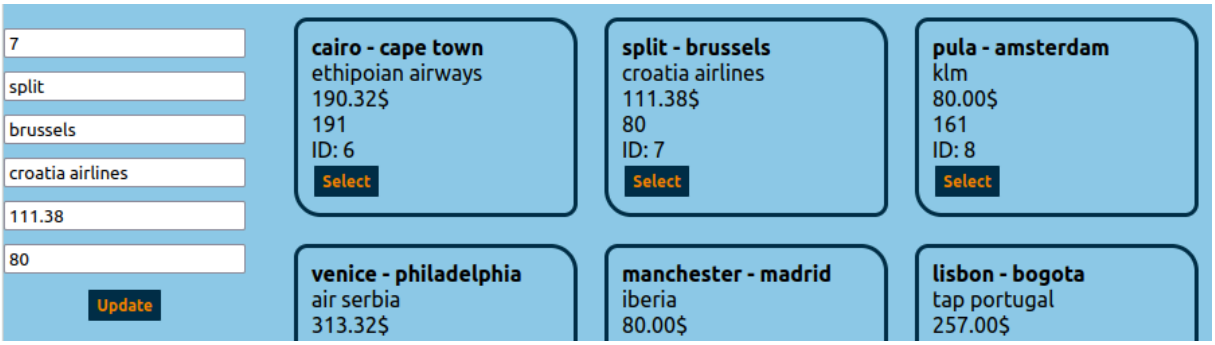
Slika 3.12 - Administratorska početna stranica

Pritiskom na gumb *Add flights* otvara se forma za upisivanje podataka o novom letu koji se želi registrirati na aplikaciju. Potrebno je upisati početnu i željenu destinaciju, aviokompaniju koja će upravljati letom te cijenu i ukupan broj sjedala namijenjenih za prodaju. Sve to može se vidjeti na Slici 3.13. Svi podaci se moraju ispuniti inače dodavanje leta neće biti moguće.

Slika 3.13 - Forma za dodavanje novog leta

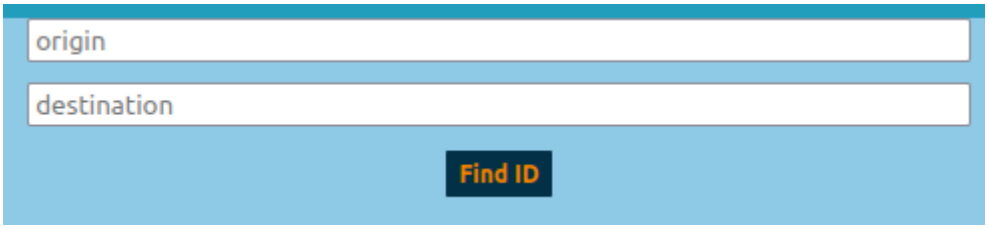
Nakon pritiska gumba *Add,* administrator je preusmjeren natrag na početnu stranicu na kojoj dalje može upravljati letovima.

Slika 3.14 prikazuje formu za ažuriranje postojećeg leta, nakon pritiska na gumb *Update flights* na početnoj stranici. Prikazani su svi registrirani letovi te pritiskom gumba *Select* na željeni let, automatski se popunjava forma s potrebnim podacima. Sada se željeni podatak može promijeniti te se nakon što korisnik pritisne gumb *Update* preusmjerava na početnu stranicu.



Slika 3.14 - Forma za ažuriranje podataka o letu

Gumb *Delete flights* vodi korisnika na formu za pretraživanje letova na ruti koje želi izbrisati. Forma u koju je potrebno upisati početnu i željenu destinaciju prikazana je na Slici 3.15.



Slika 3.15 - Forma za pretraživanje letova za brisanje

Pritiskom gumba *Find ID* prikazuju se letovi te njegovi podaci, kao i gumb *Delete* nakon čijeg se pritiska odabrani let zauvijek briše iz baze podataka te nestaje iz prikaza u realnom vremenu. Slika 3.16 prikazuje letove spremne za brisanje.

Slika 3.17 - Letovi za brisanje

Do još jedne od administratorskih funkcionalnosti se dođe nakon pritiska na gumb *Tickets*. Na Slici 3.18 mogu se vidjeti sve kupljene karte za svakoga korisnika te ruta za koju je avionska karta kupljena.

Email	Origin	Destination
Darjan	new york	los angeles
dutmar@gmail.com	new york	los angeles
korisnik@gmail.com	new york	los angeles
kupac@gmail.com	san francisco	toronto
dutmar@gmail.com	cairo	cape town
korisnik@gmail.com	barcelona	brussels

Slika 3.18 - Prikaz kupljenih avionskih karata

Zadnja od administratorskih funkcionalnosti je prikaz svih registriranih korisnika aplikacije te imaju li pojedini korisnici administratorske privilegije. Tablica korisnika prikazana je na Slici 3.19.

Name	Email	Admin privilege
Darjan	dutmar@gmail.com	false
dado	daky@gmai.com	false
jojo	alan@gmail.com	false
dado	daki@gmail.com	false
dar	dar@gmail.com	false
admin	admin	true
admin	admin@admin.com	true

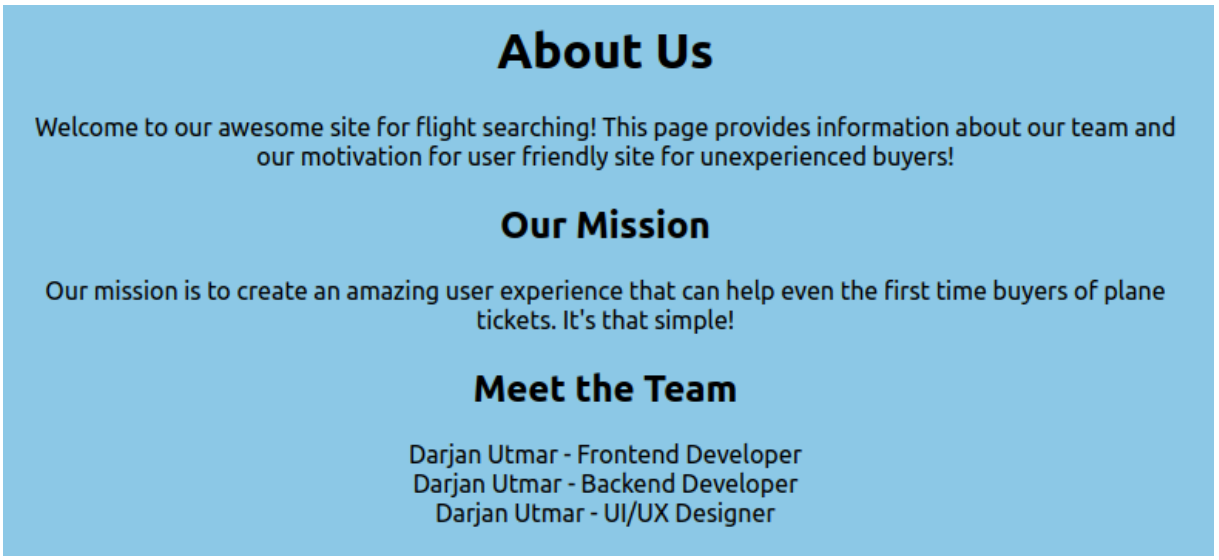
Slika 3.19 - Prikaz svih registriranih korisnika aplikacije

Posljednjih par funkcionalnosti vezane su uz prikaz aviokompanija koje su registrirane na aplikaciji te par informacija o aplikaciji općenito.

Kada korisnik navigira na stranicu *Flights* pritiskom na *Flights* gumb na navigacijskoj traci, otvara mu se popis svih aviokompanija koje su registrirane da upravljaju letovima koji su ponuđeni u aplikaciji. Aviokompanije su prikazane na Slici 3.20.

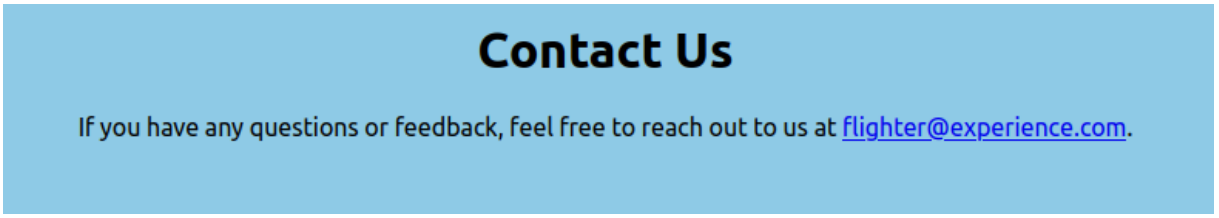
Slika 3.20 - Prikaz registriranih aviokompanija

Stranica *About* je prikazana na Slici 3.21 te opisuje aplikaciju kratko te njezin cilj, također predstavlja dizajnere i programere same aplikacije.



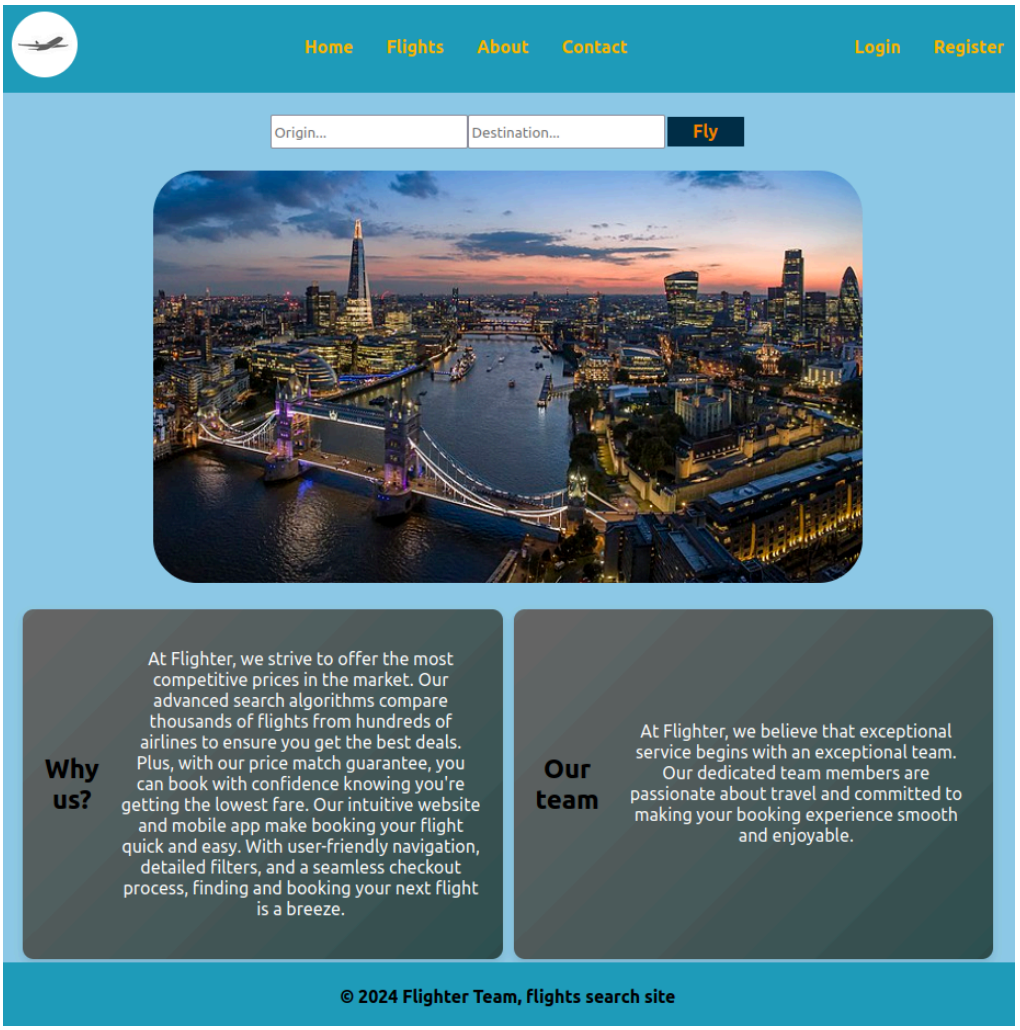
Slika 3.21 - About stranica

Contact stranica, prikazana na Slici 3.22, sadrži email aplikacije na koji se korisnik može javiti ukoliko ima pitanja ili dolazi do nekih nejasnoća.



Slika 3.22 - Contact stranica

Na početnoj stranici cjelokupne aplikacije nalaze se slika destinacija registriranih na samoj stranici. Te slike se mijenjaju u određenom vremenskom intervalu. Ispod spomenutih slika nalaze se dva tekstualna okvira unutar kojih je napisano zašto odabrati baš ovu aplikaciju te se isto tako spominje i tim zaslužan za razvoj same aplikacije (Slika 3.23).



Slika 3.23 - Početna stranica aplikacije

Kada se korisnik registrira putem forme za registraciju njegovi podaci se spreme unutar tablice *users* u bazi podataka. Unutar tablice se spremaju ime i email korisnika kao i kriptirana lozinka te administratorske privilegije kao što se može vidjeti na Slici 3.24. Također može se vidjeti kada se korisnik registrirao te ako je došlo do promjene podataka.

#	id	name	email	email_verified_a	password	remember_toker	created_at	updated_at	admin_privileg
1	1	Darjan	dutmar@gmail.com	NULL	$2y$12$RqLSHpB/MJMCmRMeA…	NULL	2024-02-01 18:06:50	2024-02-01 18:06:50	false
2	2	dado	daky@gmai.com	NULL	$2y$12$wQNvtMUVOv532YVVJFa…	NULL	2024-02-07 18:53:00	2024-02-07 18:53:00	false
3	3	jojo	alan@gmail.com	NULL	$2y$12$htfIEnGJC8OWCaDJKWZ…	NULL	2024-02-07 19:11:52	2024-02-07 19:11:52	false
4	4	dado	daki@gmail.com	NULL	$2y$12$.DKdpB4m0UBnvXHv6W…	NULL	2024-02-12 19:46:24	2024-02-12 19:46:24	false
5	5	dar	dar@gmail.com	NULL	$2y$12$KoS/dF1WLx2JQDyM1YO…	NULL	2024-02-23 12:58:51	2024-02-23 12:58:51	false
6	7	admin	admin	NULL	$2y$12$xNqgfbgcFPenZYwUmjy8…	NULL	2024-04-17 15:28:43	2024-04-17 15:28:43	true
7	9	admin	admin@admin.com	NULL	$2y$12$0PQdt1zM5fT96x4g8OZM…	NULL	2024-04-17 15:50:05	2024-04-17 15:50:05	true

Slika 3.24 - Prikaz registriranih korisnika unutar baze podatka

Letovi registrirani unutar aplikacije su predregistrirani kroz Laravel radni okvir putem *seeder-a* te ih kao takve korisnik ne može dodavati. Jedini način kako se mogu registrirati novi letovi je kroz administratorsko sučelje. Na Slici 3.25 je prikazana tablica *flights* koja sadržava sve registrirane letove. Svaki let sadržava podatke o početnom i odredišnom aerodromu, aviokompaniji koja je odgovorna za taj let te podatke o cijeni i broju trenutno dostupnih sjedala na letu. Broj dostupnih sjedala se smanjuje s obzirom na kupovinu karata.

#	id	origin	destination	airline	price	noOfSeats	created_at	updated_at
1	6	cairo	cape town	ethipoian airways	190.32	0	2024-02-01 18:03:45	2024-06-15 09:44:08
2	7	split	brussels	croatia airlines	89.00	80	2024-02-01 18:03:45	2024-06-17 09:50:06
3	8	pula	amsterdam	klm	80.00	161	2024-02-01 18:03:45	2024-06-13 16:23:40
4	9	venice	philadelphia	air serbia	313.32	221	2024-02-01 18:03:46	2024-05-23 18:26:49
5	10	manchester	madrid	iberia	80.00	161	2024-02-01 18:03:46	2024-02-01 18:03:46
6	11	lisbon	bogota	tap portugal	257.00	197	2024-02-01 18:03:46	2024-02-01 18:07:17
7	12	medellin	miami	united airlines	73.43	148	2024-02-01 18:03:46	2024-02-13 06:51:36

Slika 3.25 - Prikaz registriranih letova unutar baze podataka

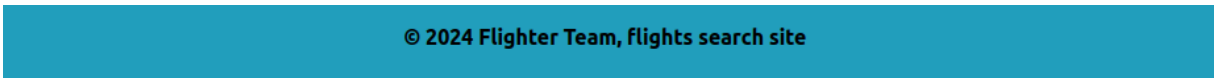
Prilikom kupnje avionskih karata, ukoliko je plaćanje uspješno, sve kupljene karte se spremaju u jednu tablicu unutar baze podataka, pod imenom *tickets*. U toj tablici nalaze se kupljene karte po korisnicima. Tablica sadrži email korisnika koji je kupio kartu te početnu i odredišnu destinaciju. Slika 3.26 sadrži prikaz tablice *tickets*.

#	id	email	origin	destination	created_at	updated_at
1	1	Darjan	new york	los angeles	2024-02…	2024-02-…
2	3	dutmar@gm…	new york	los angeles	2024-02…	2024-02-…
3	4	korisnik@gm…	new york	los angeles	2024-02…	2024-02-…
4	5	kupac@gmai…	san francisco	toronto	2024-02…	2024-02-…
5	6	dutmar@gm…	cairo	cape town	2024-02…	2024-02-…
6	7	korisnik@gm…	barcelona	brussels	2024-02…	2024-02-…

Slika 3.26 - Prikaz kupljenih karata unutar baze podataka

U podnožju ove web aplikacije nalazi se *footer* bez kojeg niti jedna web aplikacija ne bi bila potpuna.

Footer u ovoj aplikaciji služi kako bi se dalo dodatne informacije o imenu tvrtke koja je zaslužna za razvoj aplikacije te godinu kada je aplikacija razvijena. Isto tako pritiskom na *footer* korisnik je preusmjeren na početnu stranicu aplikacije. Izgled samog sučelja prikazan je na Slici 3.27. *Footer* je prikazan na svakoj stranici aplikacije kao i navigacijska traka kako bi korisnik imao sve bitne informacije na pojedinoj stranici.



Slika 3.27 - Izgled footera

4 RAZVOJ I IMPLEMENTACIJA SPECIFIČNIH APLIKACIJSKIH FUNKCIONALNOSTI

U ovom poglavlju biti će opisane funkcionalnosti kartičnog plaćanja putem Stripe-a [35] te pretraga letova i njihovo dodavanje u košaricu kao i administratorska funkcionalnost ažuriranja letova.

4.1 Stripe plaćanje

Početak plaćanja putem Stripea događa se pritiskom gumba *Checkout* unutar košarice s letovima. Prilikom pritiska gumba otvara se komponenta *Checkout* kojoj se šalje ukupna cijena za plaćanje. Cijena je potrebna kako bi putem *PaymentIntent* [36] API poziva na Slici 4.1 dobili *client secret* koji se šalje prema klijentskoj strani umjesto cijelog *PaymentIntent* objekta. *Client secret* kod je potreban kako bi se autentificiralo korisnika i moglo do kraja izvršiti plaćanje.

```
useEffect(() => {
    const fetchData = (async () => {
        const response = await apiClient.post('/api/stripe', {
            amount: props.amount,
        });
        setClientSecret(response.data);
    });

    fetchData();
}, []);
```

Slika 4.1 - API poziv prema Stripe-u kako bi se dobio client secret

Na poslužiteljskoj se strani dohvaća taj poziv te se zbog podrške Stripe-a unutar Laravel radnog okvira, mogu koristit već gotovi *PaymentIntent* objekti. Za konfiguraciju tih objekata unutar Laravela potrebno je definirati *secret* i *public* ključeve unutar *.env* datoteke te ih pročitati. Prihvaćanje API poziva te vraćanje *client secreta* može se vidjeti na Slici 4.2.

```php
public function stripePost(Request $request) {
    try {
        $stripe = new \Stripe\StripeClient(
            env('STRIPE_SECRET')
        );

        $response = $stripe->paymentIntents->create([
            'amount' => $request->amount,
            'currency' => 'eur',
        ]);

        return response() -> json($response->client_secret, 201);
    } catch(Exception $ex) {
        return response() -> json([['response' => 'Error']], 500);
    }
}
```

Slika 4.2 - Dohvaćanje API poziva i vraćanje client secreta

Primjer konfiguracije ključeva unutar *.env* datoteke je prikazan na Slici 4.3. Svaki korisnik prijavljen na Stripe platformu imati će drugačije ključeve. Ključevi se konfiguriraju unutar *.env* datoteke kako bi bili skriveni te kako bi im se moglo pristupiti iz svake druge datoteke.

```
STRIPE_KEY=pk
STRIPE_SECRET=
```

Slika 4.3 - Konfiguracija Stripe ključeva unutar .env datoteke

Kada se dobije *client secret* kod on se dalje šalje *Checkout* komponenti uz dodatne parametre kao što je *stripePromise* koji predstavlja *public* ključ. Slika 4.4 prikazuje *Checkout* komponentu. *Elements* komponente su omotači kako bi se forma za upis kartičnih podataka mogla prikazati korisniku na aplikaciji.

```
const options = {
    clientSecret: clientSecret,
};

return (
    <div>
        {clientSecret ? (
            <Elements stripe={stripePromise} options={options}>
                <CheckoutForm cart={props.cart} />
            </Elements>
        ) : (
            <div></div>
        )}
    </div>
);
```

Slika 4.4 - Checkout komponenta kojoj se šalje client secret

Unutar gotove Stripe komponente *CheckoutForm* nalaze se polja za upisivanje podataka s kartice, broj kartice, datum isteka, kontrolni broj te država u kojoj se korisnik nalazi koja se dohvaća putem geografske lokacije korisnika. Nakon pritiska gumba *Submit* unutar forme, zove se API poziv za potvrdu plaćanja te se putem drugog API poziva kupljene karte zapisuje u za to predviđenu tablicu unutar baze podataka. Slika 4.5 prikazuje API poziv za potvrdu plaćanja.

```
const {error} = await stripe.confirmPayment({
  //`Elements` instance that was used to create
  elements,
  confirmParams: {
    return_url: 'http://localhost:3000/',
  },
  redirect: "if_required",
});

if (error) {
  // This point will only be reached if there i
  // confirming the payment. Show error to your
  // details incomplete)
  setErrorMessage(error.message);
} else {
  // Your customer will be redirected to your `
  // methods like iDEAL, your customer will be
  // site first to authorize the payment, then
  //console.log(props.cart);
  handleBuy();
  nav('/');
```

Slika 4.5 - API poziv za potvrdu plaćanja

Funkcija *handleBuy()* sadrži API poziv koji kupljene karte zapisuje u bazu podataka te ažurira broj dostupnih sjedala leta. Prvo se nađe pojedini let iz proslijeđenog polja letova *props.cart* te se prvo smanjuje broj dostupnih sjedala leta što se može vidjeti na Slici 4.6.

```
props.cart.forEach(flight => {
  apiClient.get('api/flights/'+flight.id)
  .then(response => {
    apiClient.put('/api/flights/'+flight.id, null,  {
      params: {
        noOfSeats: response.data.noOfSeats-1
      }
    })
```

Slika 4.6 - API poziv za smanjenje broja dostupnih sjedala leta

Slika 4.7 prikazuje ažuriranje leta s obzirom na njegov ID. Pomoću ID-a se nađe odabrani let te se kroz *update()* metodu ažuriraju odabrani parametri.

```
public function update(Request $request, string $id)
{
    $flight = Flights::find($id);
    $flight->update($request->all());
    return $flight;
}
```

Slika 4.7 - Ažuriranje parametara leta

Nakon toga se s obzirom na nađeni let šalje API poziv poslužitelju kako bi se kupljena karta zapisala u bazu podataka. Kod API poziva je prikazan na Slici 4.8. Dohvaća se korisnikov email koji je spremljen u *localStorage* te se uz njega još šalju i početna i željena destinacija za zapis u bazu podataka.

```
.then(response => {
  apiClient.post('api/tickets', {
    email: JSON.parse(localStorage.getItem('email')),
    origin: flight.origin,
    destination: flight.destination
  })
```

Slika 4.8 - API poziv za upis kupljene karte u bazu podataka

Kod na Slici 4.9 prikazuje dodavanje kupljenih karata u bazu podataka putem Laravelovog ORM upita. Zapisuju se email i početna i željena destinacija. Prvo se provjerava jesu li u upitu podaci koji su potrebni, ako jesu onda se stvori novi podatak s poslanim atributima te zapiše u bazu podataka.

```php
public function store(Request $request) {
    $fields = $request->validate([
        'email' => 'required',
        'origin' => 'required',
        'destination' => 'required'
    ]);

    $data = Tickets::create([
        'email' => $fields['email'],
        'origin' => $fields['origin'],
        'destination' => $fields['destination']
    ]);

    $response = [
        'data' => $data
    ];

    return response($response, 201);
}
```

Slika 4.9 - ORM upit za upis kupljene karte u bazu podataka

Na Slici 4.10 može se vidjeti prikaz transakcija na Stripe web stranici. Transakcije koje su označene s *Incomplete* su izvršile API poziv za *client secret* kod, ali nisu odradile potvrdu plaćanja upisivanjem validnih podataka s kartice. Uspješno izvršene transakcije označene su sa *Succeeded.* Tablica još prikazuje vrstu korištene kartice, datum kada je transakcija napravljena te *client secret* kod.

Amount		Payment method	Description	Customer	Date
€393.32 EUR	Succeeded ✓	VISA •••• 4242	pi_3PRGN7ICSrlzlIyR0gaAo9Z7		Jun 13, 4:23 PM
€232.38 EUR	Succeeded ✓	VISA •••• 4242	pi_3PJsupICSrlzlIyR1rqSP9M7		May 24, 7:52 AM
€547.32 EUR	Succeeded ✓	VISA •••• 4242	pi_3PJgLWICSrlzlIyR1E5EaJIw		May 23, 6:26 PM
€311.32 EUR	Succeeded ✓	VISA •••• 4242	pi_3PEuwMICSrlzlIyR0dPOY2YF		May 10, 3:01 PM
€568.32 EUR	Incomplete	—	pi_3PEu8oICSrlzlIyR1arS0S1D		May 10, 2:09 PM
€764.75 EUR	Incomplete	—	pi_3PDZNwICSrlzlIyR0tEZBWiV		May 6, 9:47 PM
€658.70 EUR	Incomplete	—	pi_3PDZMyICSrlzlIyR1qX9zBJn		May 6, 9:46 PM

Slika 4.10 - Transakcije na Stripe platformi

4.2 Pretraživanje letova i dodavanje u košaricu

Glavna funkcionalnost aplikacije je pretraživanje letova. U nastavku je detaljno opisan način pretraživanja kroz programski kod.

Kako bi korisnik uopće mogao dodati let u košaricu, mora biti prijavljen unutar aplikacije. Nakon što se korisnik prijavi unutar aplikacije može unutar polja za unos podataka početne i željene destinacije upisati željene destinacije te stisnuti gumb *Fly*. Pritiskom na gumb prikazuju se letovi dostupni za tu rutu, a ako letovi nisu dostupni prikazuje se određena poruka. Upisivanjem podataka u polja postavlja se stanje pomoću *setState* funkcije kao što je vidljivo na slici 4.11.

```
<form onSubmit={handleSubmit}>
    <input
        type="text"
        placeholder="Origin..."
        value={origin}
        onChange={e => setOrigin(e.target.value)}
    />

    <input
        type="text"
        placeholder="Destination..."
        value={destination}
        onChange={e => setDestination(e.target.value)}
    />
  <button className="buy-button press-search" type="submit">Fly</button>
</form>
```

Slika 4.11 - Forma za pretragu po željenim destinacijama

Pomoću Axios poziva na Slici 4.12 poziva se API poziv koji je napisan u Laravel random okviru. Kao parametre unutar API poziva šalje se početni i odredišni aerodrom, a kao odgovor se dobiju svi letovi za tu rutu.

```
const handleSubmit = (e) => {
  setSubmitted(true);
  e.preventDefault();
  apiClient.get('api/search', {
    params: {
      origin: origin,
      destination: destination
    }
  })
  .then(response => {
    setFlights(response.data)
  })
  .catch(error => console.error(error));
}
```

Slika 4.12 - Pretraga po željenim destinacijama

Kao odgovor dobije se polje letova koji se također postavljaju pomoću *setState* funkcije. Isto tako prikazuje se i gumb *Add to cart* ako je korisnik prijavljen i ako ima dostupnih sjedala na tom letu. To se može vidjeti na Slici 4.13.

```
const flightsList = flights.map((flight) =>
    <div key={flight.id} className="flight-in-list">
      <div className="flight-route">
        {flight.origin} - {flight.destination}
      </div>
      <div>
        {flight.airline}
      </div>
      <div>
        {flight.price}$
      </div>
      {isLoggedIn && flight.noOfSeats > 0 ? (
        <button className="buy-button"
        onClick={() => handleButtonClick(flight)} >
          Add to cart</button>
      ): (
        <div>
          {isLoggedIn && flight.noOfSeats === 0 ? (
            <div className="error-div">
              No seats available!
            </div>
          ) : (
            <div></div>
          )}
        </div>
      )}
    </div>
);
```

Slika 4.13 - Prikaz letova kao odgovor API poziva

Logika dohvaćanja podataka iz baze prilikom API poziva na URL *api/search* može se vidjeti na Slici 4.14. Prihvaća se čak i djelomično potpuno ime destinacija zbog znakova *%* prije i nakon poslanog podatka. ORM upit dohvaća poslane podatke te traži odgovara li neki od podataka unutar tablice. Oba uvjeta, početne i odredišni aerodrom, moraju biti zadovoljeni kako bi korisnik dobio odgovarajući podatak.

```php
public function search(Request $request) {
    $origin = $request->input('origin');
    $destination = $request->input('destination');

    return (Flights::when($origin, function ($query) use ($origin) {
            return $query->where('origin', 'like', "%$origin%");
        })
        ->when($destination, function ($query) use ($destination) {
            return $query->where('destination', 'like', "%$destination%");
        })
        ->get());
}
```

Slika 4.14 - Logika dohvaćanja letova za rutu u Laravelu

Odabrana ruta je registrirana unutar *api.php* datoteke (Slika 4.15) u Laravelu te ta ruta radi putem GET metode za dohvaćanje podataka.

```php
Route::get('/search', [FlightsController::class, 'search']);
```

Slika 4.15 - Registracija API rute

Na Slici 4.16 nalazi se metoda koja dodaje letove u košaricu pritiskom na gumb *Add to cart*. Šalje se odabrani let u metodu te se pomoću *setState()* metode postavlja stanje koje se kasnije sprema u *localStorage*. Letovi se iz *localStorage-a* kasnije dohvaćaju u komponenti *ShoppingCart* kako bi se mogli prikazati u košarici.

```javascript
const handleButtonClick = (flightInCart) => {
  setFlightsInCart(current => [...current, flightInCart]);
}
```

Slika 4.16 - Metoda za dodavanje letova u košaricu

Pritisak na gumb za dodavanje letova u košaricu radi na principu iterativnog prolaska kroz mapu letova te se pri pritisku gumba na određenom letu šalje baš taj let u metodu

handleButtonClick() koja kao argument prima let kao objekt. Prikaz toga može se vidjeti na Slici 4.17.

```
const flightsList = flights.map((flight) =>
    <div key={flight.id} className="flight-in-list">
      <div className="flight-route">
        {flight.origin} - {flight.destination}
      </div>
      <div>
        {flight.airline}
      </div>
      <div>
        {flight.price}$
      </div>
      {isLoggedIn && flight.noOfSeats > 0 ? (
        <button className="buy-button"
        onClick={() => handleButtonClick(flight)} >
          Add to cart</button>
```

Slika 4.17 - Prikaz Add to cart gumba te prosljeđivanje odabranog leta u metodu handleButtonClick()

Spremanje letova u *localStorage* odvija se unutar *useEffect()* kuke. Prvo se provjerava ako unutar polja u koje se spremaju letovi za košaricu ima letova. Ako ima odabranih letova, onda se putem *localStorage.setItem()* metode spremaju podaci u memoriju. Na Slici 4.18 je prikazano spremanje u *localStorage.* Varijabla *flightsInCart* unutar uglatih zagrada služi kao polje ili niz ovisnosti čija promjena pokreće *useEffect()* kuku.

```
useEffect(() => {
  if(flightsInCart.length > 0) {
    localStorage.setItem('cart', JSON.stringify(flightsInCart));
  }
}, [flightsInCart]);
```

Slika 4.18 - Spremanje u localStorage

Kod dohvaćanja podataka iz *localStorage* koristimo *localStorage.getItem()* metodu. Dohvaćeni podaci spremaju se u varijablu koju možete vidjeti na Slici 4.19. Potrebno je parsirati dohvaćene JSON podatke u obično polje elemenata kako bi se mogle izvršavati manipulacije nad podacima. Kada dođe do promjene unutar tog polja elemenata *useEffect* kuka će se ponovo izvršiti.

```
var cart = JSON.parse(localStorage.getItem('cart'));
```

Slika 4.19 - Dohvaćanje podataka iz localStorage-a

Na isti način kao što su prikazani letovi kod pretraživanja, putem *map()* metode za polja podataka, tako se prikazuju i letove u košarici. Iterira se kroz polje te se za svaki let prikazuju njegovi bitni podaci kao na Slici 4.20.

```
if(cart) {
  cartList = cart.map((flight) =>
    <div className='flight-in-list' key={flight.id}>
      <div className='flight-route'>
      {flight.origin}-{flight.destination}
      </div>
      <div>{flight.price}$</div>
      <button className="buy-button"
      onClick={() => handleRemove(flight)}>Remove</button>
    </div>
  );
}
```

Slika 4.20 - Prikaz letova u košarici

4.3 Ažuriranje letova - administratorska funkcionalnost

Kako bi pritisak gumba *Update flights* radio potrebno je registrirati rutu i komponentu koja će se renderirati nakon navigiranja na odabranu rutu u URL-u. Registracija rute je prikazana na Slici 4.21.

```
<Route path='/update' Component={UpdateFlight} />
```

Slika 4.21 - Registracija rute URL-a

Slika 4.22 prikazuje kondicionalno renderiranje komponenti s obzirom na uvjet je li korisnik administrator ili nije. Provjerava se uvjet ako je korisnik administrator te ako se korisnik prijavio unutar aplikacije. Ako je korisnik administrator, prikazat će se gumbi za administratorske funkcionalnosti. Ukoliko korisnik nije administrator, prikazuje mu se komponenta za pretragu letova.

```
{admin == 'true' && isLoggedIn ? (
    <div className="admin-buttons">
        <button className="buy-button" onClick={handleAdd}>Add flights</button>
        <button className="buy-button" onClick={handleUpdate}>Update flights</button>
        <button className="buy-button" onClick={handleDelete}>Delete flights</button>
        <button className="buy-button" onClick={handleTickets}>Tickets</button>
        <button className="buy-button" onClick={handleUsers}>Users</button>
    </div>
) : (
    <div>
        <Search/>
    </div>
)}
```

Slika 4.22 - Kondicionalno renderiranje prikaza gumba za administratorske funkcionalnosti

Funkcionalnost opisana u ovom poglavlju je ažuriranje podataka o letu te se tako pritiskom na gumb *Update flights* poziva metoda *handleUpdate()* prikazana na Slici 4.23. Pozivom na metodu *handleUpdate()* navigira se korisnika na URL */update* koji smo registrirali kao rutu koja prikazuje komponentu *UpdateFlight.*

```
const handleUpdate = () => {
    nav("/update");
}
```

Slika 4.23 - Metoda handleUpdate()

Komponenta *UpdateFlight* zaslužna je za prikaz svih letova spremnih za ažuriranje kao i polja za unos podataka u kojem se mijenjaju podaci o odabranom letu. Slika 4.24 prikazuje dohvaćanje svih letova čije podatke je moguće promijeniti. *UseEffect()* kuka omogućuje da se kod prvotnog renderiranja komponente poziva API poziv koji dohvaća sve letove iz baze podataka. Kao parametri se šalju prazni znakovni nizovi za početnu i odredišnu destinaciju jer je potrebno dohvatiti sve letove. Pomoću metode *setState()* iz *useState()* kuke, postavljaju se svi dohvaćeni letovi kako bi im se kasnije moglo pristupiti.

```
useEffect(() => {
    const handleDisplayFlights = async () => {
        try {
            const response = await apiClient.get('/api/search', {
                params: {
                    origin: '',
                    destination: ''
                }
            });
            setFlights(response.data)
        } catch (error) {
            console.error('Failed', error.response.data);
        }
    }

    handleDisplayFlights();
}, []);
```

Slika 4.24 - Dohvaćanje letova spremnih za ažuriranje

Kako bi se prikazali dohvaćeni letovi iz polja letove potrebno je pomoću *.map()* metode iterirati kroz svaki od letova te prikazati njegove podatke kako bi administrator znao što može i želi promijeniti. Pomoću te metode stvara se nova varijabla u koju se sprema prikaz letova u HTML obliku. Prikazuju se podaci o letu kao i do sada te dodatni gumb *Select* kojim se odabire let za ažuriranje kao što se može vidjeti na Slici 4.25.

```
const flightsList = flights.map((flight) =>
    <div key={flight.id} className="flight-in-list">
        <div className="flight-route">
            {flight.origin} - {flight.destination}
        </div>
        <div>
            {flight.airline}
        </div>
        <div>
            {flight.price}$
        </div>
        <div>
            {flight.noOfSeats}
        </div>
        <div>
            ID: {flight.id}
        </div>
        <button className="buy-button" onClick={() => handleSelect(flight)}>Select</button>
    </div>
)
```

Slika 4.25 - Prikaz dohvaćenih letova pomoću .map() metode

Pritiskom na gumb *Select* poziva se metoda *handleSelect()* koja kao argument prima odabrani let putem *setState()* metode postavlja trenutno stanje varijabli unutar polja za unos podataka temeljeno na podacima odabranog leta. *HandleSelect()* metoda se može vidjeti na Slici 4.26.

```
const handleSelect = (flight) => {
    setId(flight.id);
    setOrigin(flight.origin);
    setDestination(flight.destination);
    setAirline(flight.airline);
    setPrice(flight.price);
    setSeats(flight.noOfSeats);
}
```

Slika 4.26 - HandleSelect() metoda

Programski kod za prikaz podataka od odabranog leta u polja za upis podataka može se vidjeti na Slici 4.27. Pomoću *value* atributa postavlja se vrijednost unutar polja na vrijednost koja je putem *handleSelect()* metode odabrana. Također, atribut *onChange* služi kod promjene podataka kako bi se na svaku promjenu ažuriralo stanje u koje je spremljena vrijednost podataka o letu.

```
<div className="addFlight-box-update">
    <input placeholder="flight id" value={id}
    onChange={(e) => setId(e.target.value)}></input>
    <input placeholder="origin" value={origin}
    onChange={(e) => setOrigin(e.target.value)}></input>
    <input placeholder="destination" value={destination}
    onChange={(e) => setDestination(e.target.value)}></input>
    <input placeholder="airline" value={airline}
    onChange={(e) => setAirline(e.target.value)}></input>
    <input placeholder="price" value={price}
    onChange={(e) => setPrice(e.target.value)}></input>
    <input placeholder="seats" value={seats}
    onChange={(e) => setSeats(e.target.value)}></input>
    <button className="buy-button" onClick={handleUpdate}>Update</button>
</div>
```

Slika 4.27 - Polja za unos podataka

Nakon što je administrator promijenio vrijednost podataka o letu potrebno je taj let ažurirati unutar baze podataka. Kako bi to bilo moguće mora se pozvati API ruta koja ažurira podatke unutar baze. API poziv je prikazan na Slici 4.28. Nakon pritiska na gumb *Update* poziva se metoda *handleUpdate()*. Na URL rute mora se dodati i ID leta kako bi

poslužiteljsko sučelje znalo o kojem se letu radi kako bi ga moglo ažurirati. Kao *header-i* unutar poziva, šalju se svi podaci o letu te se nakon uspješnog ažuriranja administrator, pomoću metode *nav(),* preusmjeruje natrag na početnu stranicu aplikacije. Ukoliko API poziv nije uspješan, podaci se ne zapisuju u bazu te se u konzoli unutar preglednika može vidjeti o kojoj greški je riječ i zašto je došlo do baš te greške.

```
const handleUpdate = async () => {
    try {
        const response = await apiClient.put('/api/update/'+id, {
            origin: origin,
            destination: destination,
            airline: airline,
            price: price,
            noOfSeats: seats
        });
        nav('/')
    } catch (error) {
        console.error('Failed', error.response.data);
    }
}
```

Slika 4.28 - API poziv za ažuriranje podataka u bazi

Na poslužiteljskom dijelu aplikacije putem ORM upita dohvaća se let po njegovom ID-ju. Kada se let dohvati onda se može ažurirati uz pomoć *update()* metode unutar Laravel radnog okvira. Kao argumenti metode prosljeđuju se svi podaci koje je administrator promijenio. Cijela metoda za ažuriranje je prikazana na Slici 4.29.

```
public function update(Request $request, string $id)
{
    $flight = Flights::find($id);
    $flight->update($request->all());
    return $flight;
}
```

Slika 4.29 - Ažuriranja podataka leta na poslužiteljskom dijelu aplikacije

Kroz *FlightsController* kontroler dohvaća se metoda *update* prikazana na Slici 4.29 te se poziva nad letom s odabranim ID-jem.

Registracija rute zaslužne za ažuriranje podataka dostupna je na Slici 4.30. Registrirana ruta je PUT HTTP metoda. Unutar URL-a rute šalje se ID leta kako bi se moglo ažurirati samo taj let.

```
Route::put('/update/{id}', [FlightsController::class, 'update']);
```

Slika 4.30 - Registracija rute za ažuriranje podataka

5 ZAKLJUČAK

Aplikacija Flighter je *RESTful* aplikacija čija namjena je prikazivanje i kupnja avionskih karata na pojedinim rutama. Korisnik bez vlastitog računa može pretraživati letove po rutama, ali ako želi kupiti avionsku kartu, potrebno se registrirati u aplikaciju. Također, na stranici predviđenoj za to, moguće je vidjeti sve registrirane aviokompanije. Aplikacija je osmišljena s ciljem da bude jednostavna za korištenje novim korisnicima. Primarni način plaćanja unutar aplikacije je putem kreditnih i debitnih kartica.

Tehnologije koje su korištene prilikom izrade ove aplikacije, React, Laravel, MySQL, izabrane su zbog svojih visokih sposobnosti. Tehnologije su jednostavne za korištenje te omogućuju brzu i jednostavnu implementaciju svih potrebnih funkcionalnosti kroz iskoristivost komponenti. Unutar Laravel radnog okvira nalazi se *middleware* paket koji pri implementaciji API ruta sprječava pristup resursima neautoriziranim korisnicima te je kao takav, uz pomoć Sanctum paketa za autorizaciju, odlično rješenje za sve aplikacije koje koriste neku vrstu autorizacije i autentifikacije. React omogućava SPA aplikacijama brži razvoj pomoću iskoristivih komponenti. Svaka komponenta se može opet upotrijebiti te je cijela aplikacija na kraju intuitivna te je kod jednostavan za čitanje i shvaćanje. Zbog jednostavnosti pristupanja i upravljanja podacima, jasno je zašto je MySQL sustav jedan od najkorišteniji alata za manipulaciju relacijskim bazama podataka.

Kako bi aplikacija bila u potpunosti spremna za tržište, trebalo bi poraditi na dodatnim funkcionalnostima koje trenutno nedostaju. To bi bila mogućnost oporavka lozinke u slučaju njenog zaboravljanja jer se trenutno gubi pristup račun ako korisnik zaboravi podatke za prijavu. Isto tako potrebno je napraviti filtriranje letova po pojedinim datumima kako bi sve bilo spremno za korisnike iz realnog svijeta. Trenutna verzija aplikacije može poslužiti kao platforma za učenje informatički nepismenih korisnika u interakciji s malo kompliciranijim aplikacijama.

LITERATURA

[1] "What does RESTful web applications mean", s interneta, https://softwareengineering.stackexchange.com/questions/153581/what-does-restful-web-applications-mean, (8.lipanj 2024)

[2] "What is RESTful API", s interneta, https://aws.amazon.com/what-is/restful-api/, (8.lipanj 2024)

[3] "What is HTTP", s interneta, https://www.cloudflare.com/learning/ddos/glossary/hypertext-transfer-protocol-http/, (8.lipanj 2024)

[4] "React", s interneta, https://react.dev/, (8.lipanj 2024)

[5] "Laravel", s interneta, https://laravel.com/, (8.lipanj 2024)

[6] "PHP", s interneta, https://www.php.net/, (8.lipanj 2024)

[7] "MVC framework", s interneta, https://www.tutorialspoint.com/mvc_framework/mvc_framework_introduction.htm, (8.lipanj 2024)

[8] "Composer", s interneta, https://getcomposer.org/, (8.lipanj 2024)

[9] "MySQL", s interneta, https://www.mysql.com/, (8.lipanj 2024)

[10] "What is SQL", s interneta, https://aws.amazon.com/what-is/sql/, (8.lipanj 2024)

[11] "MySQL Workbench", s interneta, https://www.mysql.com/products/workbench/, (8.lipanj 2024)

[12] "Meta Open Source", s interneta, https://opensource.fb.com/, (04.srpanj)

[13] "What is Declarative programming in React", s interneta, https://www.educative.io/answers/what-is-declarative-programming-in-react, (10.lipanj 2024)

[14] "What is Virtual DOM in React", s interneta, https://ibaslogic.com/virtual-dom-react/, (04.srpanj 2024)

[15] “Virtual DOM and internals - React”, s interneta, https://legacy.reactjs.org/docs/faq-internals.html, (10.lipanj 2024)

[16] “The Unseen Power of the React Reconciliation Algorithm”, s interneta, https://www.dhiwise.com/post/a-deep-dive-into-react-reconciliation-algorithm, (10.lipanj 2024)

[17] “Feature overview | React Router”, s interneta, https://reactrouter.com/en/main/start/overview, (10.lipanj 2024)

[18] “Introducing hooks”, s interneta, https://legacy.reactjs.org/docs/hooks-intro.html, (10.lipanj 2024)

[19] “The React useContext Hook”, s interneta, https://www.telerik.com/blogs/react-usecontext-hook, (18.lipanj 2024)

[20] “Getting started | Axios”, s interneta, https://axios-http.com/docs/intro, (12.lipanj 2024)

[21] “Symfony”, s interneta, https://symfony.com/, (12.lipanj 2024)

[22] “MVC Architecture Pattern”, s interneta, https://www.geeksforgeeks.org/mvc-model-view-controller-architecture-pattern-in-android-with-example/, (04.srpanj 2024)

[23] “What is an ORM”, s interneta, https://www.freecodecamp.org/news/what-is-an-orm-the-meaning-of-object-relational-mapping-database-tools/, (12.lipanj 2024)

[24] “Eloquent”, s interneta, https://laravel.com/docs/11.x/eloquent, (12.lipanj 2024)

[25] “Migration & seeding”, s interneta, https://laravel.com/docs/4.2/migrations, (12.lipanj 2024)

[26] “What is and API”, s interneta, https://aws.amazon.com/what-is/api/, (14.lipanj 2024)

[27] “Laravel Sanctum”, s interneta, https://laravel.com/docs/11.x/sanctum, (14.lipanj 2024)

[28] “SPA (Single page applications)”, s interneta, https://developer.mozilla.org/en-US/docs/Glossary/SPA, (14.lipanj 2024)

[29] “Example of Laravel Sanctum with API Tokens”, s interneta, https://www.youtube.com/watch?v=Ql5z9TjXWLY, (14.lipanj 2024)

[30] “What is a RDBMS”, s interneta, https://www.techtarget.com/searchdatamanagement/definition/RDBMS-relational-database-management-system, (18.lipanj 2024)

[31] “ER diagram”, s interneta, https://www.lucidchart.com/pages/er-diagrams, (18.lipanj 2024)

[32] “ER Diagram (ERD)”, s interneta, https://www.lucidchart.com/pages/er-diagrams, (04.srpanj 2024)

[33] “Visual Studio Code”, s interneta, https://code.visualstudio.com/, (18.lipanj 2024)

[34] “Stack Overflow Developer Survey”, s interneta, https://survey.stackoverflow.co/2021, (04.srpanj 2024)

[35] “Stripe”, s interneta, https://stripe.com/, (14.lipanj 2024)

[36] “The Payment Intents API”, s interneta, https://docs.stripe.com/payments/payment-intents, (14.lipanj 2024)

SAŽETAK

U ovom radu predstavljena je web aplikacija za pretragu letova i kupnju avionskih karata za odabrane letove. Aplikacija je temeljena na REST arhitekturi te se aplikacija sastoji od odvojene klijentske i poslužiteljske strane. Za klijentsku stranu aplikacije korišten je radni okvir React.js dok je za poslužiteljsku stranu korišten Laravel, radni okvir temeljen na PHP-u. Druge korištene tehnologije su MySQL za upravljanje bazama podataka, React knjižnica Axios za slanje asinkronih zahtjeva prema poslužitelju te Sanctum paket unutar Laravela za autentifikaciju i autorizaciju. Aplikacija ima jednostavne funkcionalnosti i intuitivno korisničko sučelje kako bi bila jednostavna za korištenje svim korisnicima.

Ključne riječi: Web, aplikacija, API, Laravel, React, REST, letovi, MySQL, avionske karte

ABSTRACT

This paper presents a web application for flight search and ticket purchase for selected flights. The application is based on REST architecture and consists of separate client and server sides. The React.js framework is used for the client side of the application, while Laravel, a PHP-based framework, is used for the server side. Other technologies used include MySQL for database management, the React library Axios for sending asynchronous requests to the server, and the Sanctum package within Laravel for authentication and authorization. The application features simple functionalities and an intuitive user interface to ensure ease of use for all users.

Keywords: Web, application, API, Laravel, React, REST, flights, MySQL, airline tickets