IZRADA WEB APLIKACIJE ZA ORGANIZACIJU IZNAJMLJIVANJA JETSKI PLOVILA

Lučić-Raguž, Marino

SVEUČILIŠTE U SPLITU

SVEUČILIŠNI ODJEL ZA STRUČNE STUDIJE

Stručni prijediplomski studij Računarstvo

MARINO LUČIĆ-RAGUŽ

ZAVRŠNI RAD

IZRADA WEB APLIKACIJE ZA ORGANIZACIJU IZNAJMLJIVANJA JETSKI PLOVILA

Split, rujan 2024.

SVEUČILIŠTE U SPLITU

SVEUČILIŠNI ODJEL ZA STRUČNE STUDIJE

Stručni prijediplomski studij Računarstvo

Predmet: Operativni sustavi

ZAVRŠNI RAD

Kandidat: Marino Lučić-Raguž

Naslov rada: Izrada web aplikacije za organizaciju iznajmljivanja jetski plovila

Mentor: Nikola Grgić, viši predavač

Split, rujan 2024.

Sadržaj

Sažetak

U projektu Izrada web aplikacije za organizaciju iznajmljivanja jetski plovila izgrađen je moderni i funkcionalni alat za upravljanje poslovnim sustavom. Cilj aplikacije je olakšati korisnicima rad pri organizaciji poslovanja. Web aplikacija omogućuje korisnicima brži pristup bitnim podatcima. Prikazuje detaljne informacije o rezervacijama i o jetski plovilima kako bi korisnik mogao brzo donositi važne poslovne odluke. Korisnik može stvarati, prikazivati i uređivati: rezervacije, lokacije, jetski plovila i opcije za najam. Korisniku je omogućeno sortiranje i filtriranje podataka. Ova aplikacija unaprjeđuje poslovanje jer smanjuje mogućnost ljudske pogreške te pojednostavnjuje prikazivanje dostupnih termina najma. Kod izrade web aplikacije korištene tehnologije za razvoj korisničkog sučelja su aplikacijski okviri Next.js i TypeScript, a za oblikovanje korisničkog sučelja se koristio aplikacijski okvir Tailwind CSS uz HTML i CSS. Za razvoj poslužiteljskog okruženja također su se koristili Next.js i TypeScript. Za pohranu podataka koristio se alat za objektno-relacijsko preslikavanje (engl. *Object-relational mapper*) Prisma s pozadinskom bazom podataka koja se temelji na PostgreSQL-u.

Ključne riječi: Jetski**,** Next.js, organizacija poslovanja, Prisma, TypeScript, web aplikacija

Summary

Web application for organization of jet-ski rental business

The project Web application for organization of jet-ski rental business has created a modern and functional tool used for operating a business system. Goal of the application is to ease work of user when it comes to managing business system. This web application allowes users quicker access to neccessary data. It lists details of reservations and jet-skis so user can make business related decisions faster. User can create, list and edit: reservations, locations, jet-skis and rental options. User can sort and filter all the data. This application improves the business because it reduces the human error factor and speeds up the process of finding available slots. For the development of this web application, technologies used for the frontend were framework Next.js and framework Typescript, while for decorating the frontend framework Tailwind CSS was used together with HTML and CSS. Framework Next.js and Typescript were also used for the development of the backend. Object-relational mapper, called Prisma, was used for data storage together with PostgreSQL database.

Keywords: Business organization, Jetski, Next.js, Prisma, Typescript, Web application

1. Uvod

Prva aplikacija za automatizaciju poslovnog sustava je nastala 60-ih godina prošlog stoljeća, kada je J. I. Case, s IBM-om napravio aplikaciju za nadgledanje skladišta i proizvodnje traktora [1]. Zbog široke dostupnosti mreže sve veći broj aplikacija se seli na internet. Pri radu u turističkom sektoru važno je prvi predstaviti ponudu. Zbog brzine poslovanja veća je mogućnost ljudske pogreške. Automatiziranjem poslovnog sustava smanjuje se mogućnost za ljudsku pogrešku.

Web aplikacija za organizaciju iznajmljivanja jetski plovila pridonosi poslovanju tako što smanjuje složenost poslovanja automatiziranjem rasporeda na temelju unesenih podataka. Na temelju unesenih podataka korisnik ima trenutni prikaz što može ponuditi potencijalnome gostu. Korisnik može upravljati jetski plovilima, lokacijama, rezervacijama te opcijama najma u sustavu.

Aplikacija se sastoji od dva glavna dijela: poslužiteljsko i klijentsko sučelje. Klijentsko sučelje i poslužiteljsko sučelje su napravljeni pomoću razvojnih okruženja (engl. *Framework*) Next.js i TypeScript. Kod klijentskog sučelja koristio se aplikacijski okvir Tailwind CSS, HTML (*HyperText Markup Language*) i CSS (*Cascading Style Sheets*). Baza podataka je PostgreSQL. Kod upravljanja bazom podataka koristi se alat ORM (*objektno-relacijsko preslikavanje*) Prisma.

Na samom početku rada bit će objašnjene tehnologije, baza podataka i struktura projekta. Konačno bit će objašnjene funkcionalnosti i izgled aplikacije.

2. Tehnologije

2.1. Next.js

Next.js je razvojni okvir JavaScript, tipa otvorenog kôda utemeljenog na JavaScript biblioteci React. Razvija ga privatna tvrtka Vercel. Njihov cilj je omogućiti programerima stvaranje React aplikacija koje imaju poslužiteljsko učitavanje (engl. *Server-Side Rendering*) i generiranje statičkih web stranica. Next.js je osmišljen kako bi riješio određene probleme koje donosi JavaScript poput loše optimizacije ili kod situacije u kojoj korisnik onemogući JavaScript [2].

SSR (*Server-Side Rendering*) najveća je prednost Next.js u usporedbi s ostalim aplikacijskim okvirima. U području web razvoja, brzina i učinkovitost su ključni. Tu prednost ima SSR koji omogućuje da se web stranice učitaju na poslužitelju prije nego što dođu do korisnika. To ubrzava učitavanje stranice kod korisnika jer internetski preglednik samo prikazuje stranicu. Ovim se izbjegava stvaranje stranice u samom pregledniku. Osim što omogućuje brže učitavanje, ono utječe i na SEO (*Search engine optimization*) koji je iznimno bitan za popularnost stranice tako što je čini privlačnijom internetskim pretraživačima [3].

2.1.1. Prikazivanje web stranice

Prikazivanje web stranice na klijentskom uređaju utječe na korisnikov doživljaj i interakciju s web stranicom. Ako stranica nije dovoljna brza, korisnik će je napustiti. Kod Next.js koristi se princip dohvaćanja podataka na poslužiteljskoj strani što znači da kada klijent zatraži određenu stranicu ona se prvo obradi i organizira na poslužitelju, odnosno posloži u HTML. Zatim se taj gotovi HTML sadržaj šalje klijentu. Poslužitelj također šalje i JavaScript skriptu u kojoj se nalaze upute za dohvaćanje podataka kao i za organizaciju HTML stranice. Zbog toga poslužiteljsko učitavanje dovodi do bržeg prikazivanja web stranice na klijentskoj strani jer klijent ne mora obavljati pozive prema poslužitelju kako bi dobio određene podatke, već primi sve spremno.

2.2. React.js

React je JavaScript biblioteka za izradu korisničkog sučelja. React je nastao 2013. godine u Facebooku. Zasnovan je na ideji komponenti i omogućuje izradu dinamičnog korisničkog sučelja bez potrebe za cjelovitom obnovom stranice što dovodi do veće brzine i performansi, kao i do lakšeg održavanja kôda [4].

Osim korištenja komponenti, React omogućuje korištenje Virtualnog DOM-a (*Document Object Model*) koji omogućuje da se koristi predmemorija strukture podataka u memoriji, a samo se konačne promjene ažuriraju u DOM pregledniku, čime se ubrzava aplikacija. Komponente koje se napišu u Reactu se mogu ponovno koristiti [5].

2.3. React Hooks

React Hooks su snažan dio React biblioteke koje omogućuju da se prate stanja različitih objekata unutar komponenti. Najpoznatije komponente su `useState` i `useEffect`. `UseState` omogućuje da komponentama pridijelimo stanje u obliku atributa ili nekog objekta. `UseEffect` se može postaviti na dva načina. Prvi način postavljanja je da se za vrijeme trajanja sesije podatci konstantno ažuriraju. Drugi način je da se na temelju promjena dohvaćaju ili ažuriraju podatci [6].

2.4. TypeScript

TypeScript je aplikacijski okvir JavaScripta koji dodaje statičku tipizaciju. Razvijen je od strane Microsofta i koristi se za razvoj složenih aplikacija. JavaScript je jezik koji gotovo nema ograničenja, što može biti problematično u kompleksnim sustavima. Na primjer, programer nije siguran koji parametar je koji tip bez da pogleda dokumentaciju što može oduzeti puno vremena. Može se koristiti na klijentskoj i na poslužiteljskoj strani. Sintaksa TypeScripta se snažno oslanja na objektno orijentirano programiranje što je glavna razlika između TypeScripta i JavaScripta. Omogućuje lakše otkrivanje pogrešaka i stvaranje strukture koja se lako može raširiti. TypeScript je potpuno kompatibilan s JavaScript kôdom, što omogućava široku primjenu [7].

2.5. Zod schema

Zod shema (engl. s*chema*) se temelji na aplikacijskom okviru TypeScript. Služi za validaciju i deklariranje tipa podataka. Može se koristiti za validaciju bilo kojeg tipa podatka, od običnog stringa do kompleksnog objekta. Iako se temelji na TypeScriptu, funkcionira i s čistim JavaScriptom. Ne ovisi o ni jednom drugom paketu i zauzima jako malo prostora [8].

2.6. Prisma ORM (engl. *Object-Relational Mapper*)

Prisma je moderni ORM alat otvorenog kôda koji se koristi za rad s bazama podataka u Node.js, Next.js i TypeScript aplikacijama. Sastoji se od tri dijela: Prisma Client, Prisma Migrate i Prisma Studio. Prisma Client generira TypeScript kôd omogućavajući sigurno izvršavanje upita koristeći TypeScript tipove podataka. Prisma Migrate služi kao alat za migraciju te pomaže pri upravljanju shemom baze podataka. Prisma Studio je vizualni alat koji olakšava pregled i uređivanje podataka u bazi podataka, omogućujući izbjegavanje korištenja upita SQL-a. Prisma omogućuje jednostavno definiranje modela podataka, povezivanje s PostgreSQLu bazom te sigurnu i učinkovitu izradu upita prema bazi. Dozvoljava korištenje Typescript tipova za dohvaćanje podataka iz baze što uvelike olakšava posao programera [9].

3. Baza podataka

Za izradu, spremanje i manipuliranje podatcima korišten je ORM alat Prisma. Prisma je moderan alat za upravljanje relacijskim bazama podataka. Prisma omogućuje jednostavno definiranje modela entiteta u programskom kôdu. Koristi vlastiti jezik za modeliranje entiteta. Način definiranja modela je prikazan na ispisu 1. Nakon definiranja modela, Prisma na temelju naredbe u terminalu stvara modele u TypeScript kôdu kako bi osigurali tip podataka. Nakon što su modeli stvoreni i generirani, naredbom *prisma db push* postavljaju se na bazu podataka. Alat Prisma podržava razne baze podataka, ali u ovoj aplikaciji korišten je PostgreSQL. Ona nudi različite alate za vizualizaciju i upravljanje podatcima kroz Prisma Studio, pruža složene mehanizme sigurnosti, kao što su provjera autentičnosti korisnika i enkripcija podataka.

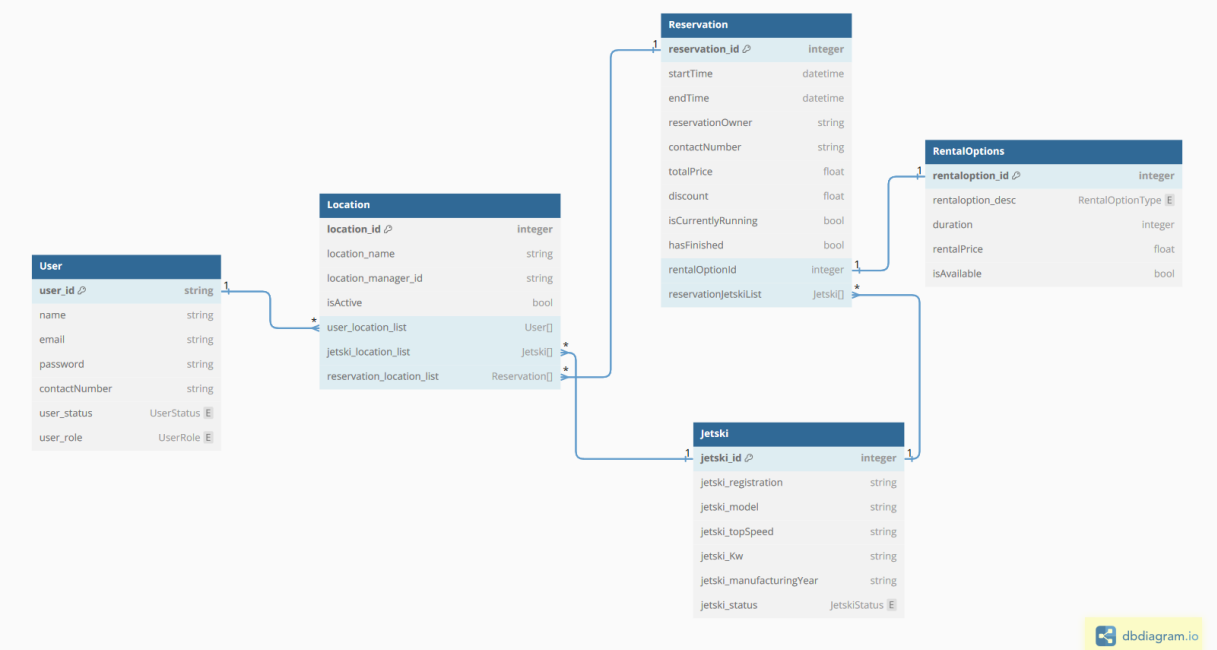
```
model RentalOptions {
       rentaloption_id          Int     @id @default(autoincrement())
       rentaloption_description String
       duration                 Int
       rentalprice              Float
       isAvailable              Boolean @default(true)
}
```

Ispis 1: Prikaz definiranja modela entiteta opcija najma

3.1. Entiteti

Osnovni elementi baze podataka se nazivaju entiteti. Entitet može biti: realan objekt, apstraktan sadržaj, događaj, odnos. Svaki entitet je opisan vlastitim atributima. Svi entiteti mogu imati više atributa, a jedan atribut može vrijediti za više entiteta. Bilo koji od tih atributa ima svoje ime i vrijednost. Entiteti u ovoj aplikaciji predstavljaju ključne podatke za organizaciju poslovanja. Oni pojednostavljuju oblikovanje aplikacije te povećavaju optimiziranost aplikacije pri pregledu podataka.

U aplikaciji se nalazi pet ključnih entiteta. Ključni entiteti su: `User`, `Jetski`, `Location`, `Reservation` i `RentalOptions`. Dijagram koji ilustrira veze entiteta prikazan je na slici 1.



Slika 1: Dijagram E-R

Tablica `User` predstavlja sve registrirane korisnike. Sastoji se od atributa:

- `user_id`: Identifikator koji se generira automatski i služi kao primarni ključ.
- `name`: Ime i prezime korisnika.
- `email`: Elektronička pošta korisnika koja služi za prijavu u aplikaciju. Mora biti jedinstvena i ne smije biti prazna.
- `password`: Hashirana lozinka korisnika. Ne smije biti prazna i mora poštivati zahtjeve sigurnosti.
- `contactNumber`: Telefonski/mobilni kontakt broj korisnika. Mora biti jedinstven jer svaki korisnik ima vlastiti broj.
- `user_status`: Povezuje se s enumom `UserStatus`, određuje radi li korisnik trenutno ili ne radi.
- `user_role`: Povezuje se s enumom `UserRole`, određuje korisnikovu ulogu u aplikaciji i njegovu razinu prava.

Tablica `Location` predstavlja sve lokacije koje se nalaze u aplikaciji. Lokacije se mogu isključiti. To znači da se rezervacije ne mogu ponuditi na toj lokaciji. Isključene lokacije će ostati u bazi podataka. Tablica je povezana s tablicom `User` preko `user_id` formirajući jedan-prema-više (engl. *One-To-Many*) vezu. Sadrži istu vezu prema tablici

Jetski preko jetski_id. Tablica također sadrži userId korisnika koji je upravitelj te lokacije. Ona se sastoji od:

- location_id: Identifikator koji se generira automatski i služi kao primarni ključ.
- location_name: Jedinstveno ime lokacije.
- location_manager_id: Sadrži jedinstveni identifikator korisnika koji je upravitelj te lokacije. Stvara vezu jedan-prema-jedan (eng. *One-to-One)* s tablicom User i user_id.
- isActive: Istina/laž (eng. *Bool*) vrijednost koja označava je li trenutna lokacija u upotrebi. Npr. poslovanje može privremeno izgubiti koncesiju i u tom slučaju poželjno je postaviti lokaciju kao neaktivnu.
- user_location_list: Jedan-prema-više veza s tablicom User i user_id. Predstavlja sve zaposlenike koji se nalaze na toj lokaciji.
- jetski_location_list: Jedan-prema-više veza s tablicom Jetski i jetski_id. Predstavlja sva jetski plovila koja se nalaze na toj lokaciji.
- reservation_location_list: Jedan-prema-više veza s tablicom Reservation i reservation_id. Predstavlja sve rezervacije koje se nalaze na toj lokaciji.

Tablica Jetski predstavlja sva jetski plovila koja se nalaze u aplikaciji, bez obzira na njihovo stanje. Sadrži osnovne podatke o svakom jetskiju. Tablica se sastoji od:

- jetski_id: Jedinstveni identifikator koji se generira automatski i služi kao primarni ključ.
- jetski_registration: Jedinstveni naziv, odnosno registracija jetski plovila koja mora odgovarati određenom formatu registracije.
- jetski_model: Model jetskija, odnosno ime proizvođača i pojedinog modela.
- jetski_topSpeed: Maksimalna brzina u miljama ili kilometrima na sat.
- jetski_kW : Snagu motora u kW.
- jetski_manufacturingYear: Godina proizvodnje.
- jetski_status: Povezuje se s enumom JetskiStatus. Definira trenutno plovno stanje jetski plovila, ali ne i informaciju je li jetski plovilo zauzeto u određenoj rezervaciji.

Tablica `RentalOptions` predstavlja sve opcije najma u poslovanju. Određuje ponudu gostu. Sastoji se od:

- `rentaloption_id:` Predstavlja jedinstveni identifikator koji se generira automatski i služi kao primarni ključ.
- `rentaloption_description:` Povezan s enumom `RentalOptionType`. Opisuje tip najma: `safari` ili `regular`.
- `duration:` Trajanje opcije u minutama.
- `rentalprice:` Cijenu opcije najma.
- `isAvailable:` Dostupnost opcije najma.

Tablica `Reservation` je glavna tablica aplikacije. Sve tablice osim tablice `User` su povezane s njom direktno. Sastoji se od više atributa:

- `reservation_id:` Jedinstveni identifikator koji se generira automatski i služi kao primarni ključ.
- `startTime:` Planirano početno vrijeme rezervacije.
- `endTime:` Planirano vrijeme završetka rezervacije.
- `createdAt:` Vrijeme kada je nastala rezervacija.
- `reservationOwner:` Ime gosta na čije ime rezervacija glasi.
- `contactNumber:` Kontakt broj vlasnika rezervacije.
- `totalPrice:` Ukupna cijena rezervacije.
- `discount:` Popust na konačnu cijenu u postotku.
- `isCurrentlyRunning:` Pri pokretanju rezervacije postavlja se u istinitu vrijednost sve dok se ne označi kraj.
- `hasItFinished:` Ako korisnik označi kraj rezervacije postavlja se u istinitu vrijednost. `isCurrentlyRunning` nije dovoljan kao samostalni atribut jer ne može razlikovati je li rezervacija završila ili nije ni krenula.
- `rentalOption_id:` Jedan-prema-jedan relacija na tablicu `RentalOptions` i `rentaloption_id`. Predstavlja opciju koja je odabrana za ovu rezervaciju.
- `reservation_jetskiList:` Jedan-prema-više veza s tablicom `Jetski` i `jetski_id`. Predstavlja sve jetskije koji se nalaze u toj rezervaciji.

3.2. Enumi

Kako bi preciznije definirali atribute entiteta koriste se enumi. Enum je posebna klasa koja predstavlja grupu nepromjenjivih varijabli. U ovoj aplikaciji postoje četiri tipa enuma: `RentalOptionType`, `JetskiStatus`, `UserStatus`, `UserRole`. U ispisu 2 prikazano je kako definiranje enuma izgleda unutar Prisma sheme.

```
enum RentalOptionType {
  SAFARI
  REGULAR_TOUR
}

enum JetskiStatus {
  AVAILABLE
  NOT_AVAILABLE
  NOT_IN_FLEET
}

enum UserStatus {
  ACTIVE
  INACTIVE
}

enum UserRole {
  ADMIN
  MODERATOR
  USER
  GUEST
}
```

Ispis 2: Prikaz izgleda enuma u Prisma shemi

Enum `RentalOptionType` predstavlja dva moguća tipa opcije koja se nude gostima, a to su safari najam te obični najam. Safari najam je opcija u kojoj su potrebna minimalno dva jetski plovila za rezervaciju. To je zbog zahtjeva da na jednom jetski plovilu bude turistički vodič. Kod običnog najma, minimalan broj jetski plovila je jedan.

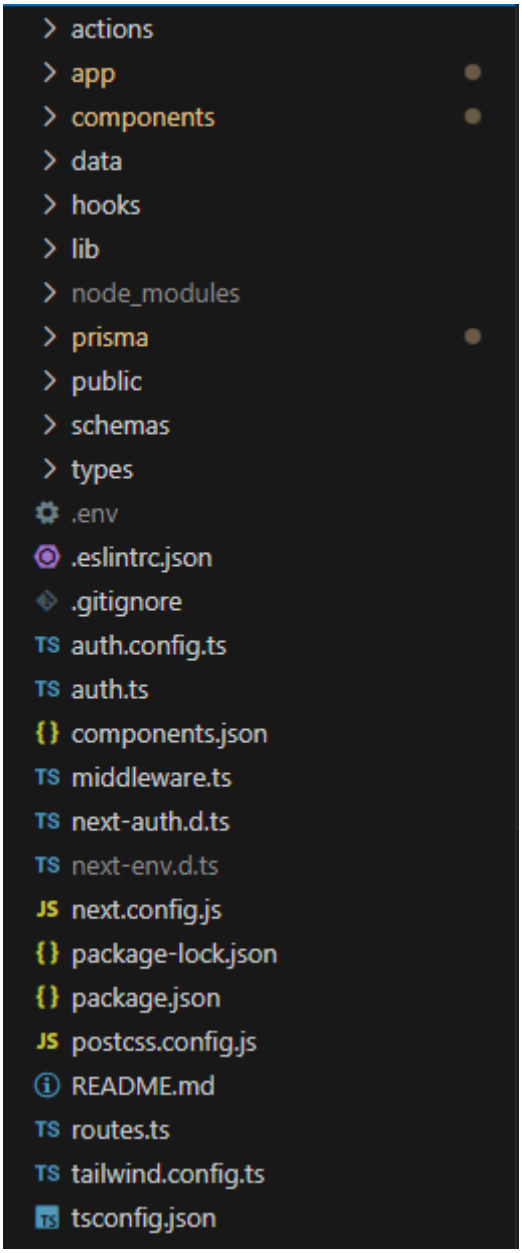
Enum `JetskiStatus` sadrži tri tipa stanja jetski plovila. Ova tri stanja nisu povezana s trenutnom dostupnošću jetski plovila s obzirom na rezervacije. `AVAILABLE` označava jetski plovilo koje je u potpunosti spremno za rezervacije i kao takav dostupan za odabir prilikom stvaranja rezervacije ako je slobodan u tom vremenskom periodu. `NOT_AVAILABLE` status označava da jetski plovilo privremeno nije dostupno. Koristi se u slučaju kada je potrebno održavanje, servis ili ponovno točenje goriva. `NOT_IN_FLEET` status predstavlja da jetski plovilo više nije u floti te da se neće vratiti. To se koristi u slučaju prodaje jetski plovila ili ako dođe do kritičnog kvara zbog kojega se jetski plovilo ne može popraviti.

Enum `UserStatus` sadrži dva tipa podataka koji predstavljaju je li korisnik u radnom odnosu ili ne.

Enum `UserRole` predstavlja korisnikovu ulogu u sustavu. Prilikom kreiranja računa korisniku je početna uloga postavljena na `GUEST`. Administrator aplikacije ima pravo mijenjati korisnikovu ulogu i samim time njegova prava jer `UserRole` ograničava određene funkcionalnosti. Moguće uloge su: `GUEST`, `USER`, `MODERATOR`, `ADMINISTRATOR`.

4. Struktura aplikacije

Svaka web aplikacija sadrži poslužiteljsku i klijentsku stranu. Ta struktura omogućuje razlikovanje između logike koja se izvršava na strani poslužitelja (eng. *Backend*) i logike koja se izvršava na strani klijenta (engl. *Frontend*). Ovaj pristup olakšava održavanje kôda, skaliranje aplikacije i omogućuje bolju korisničku interakciju. Prikaz strukture direktorija projekta prikazan je na slici 2.



Slika 2: Struktura projekta.

4.1. Poslužiteljska strana (engl. *Backend*)

Poslužiteljska strana pokreće web-aplikaciju – korisnik ne vidi niti komunicira s njom, ali ona uvijek radi u pozadini. Pruža funkcionalnost i iskustvo nalik radnoj površini. Poslužitelj osigurava isporuku podataka ili usluga koje zahtjeva klijentska strana. Odnosi se na stvaranje baze podataka, njene integracije te upravljanje kao i na stvaranje i kontroliranje API-ova.

4.1.1. Poslužiteljske akcije (engl. *Server actions*)

Poslužiteljske akcije (engl. *Server actions*) su asinkrone funkcije koje se izvode na poslužitelju, a pozivaju se u klijentskim komponentama za manipuliranje primljenim podatcima iz forme. Definiraju se s Reactovom metodom, tako što se na vrh datoteke postavi: `'use server'`. Poslužiteljske akcije se mogu pozvati iz forme pritiskom na određenu tipku ili koristeći Reactovu metodu `useEffect`. Povećavaju sigurnost aplikacije jer smanjuju rizik od manipulacije podatcima na klijentskoj strani i povećavaju performanse zbog toga što se izvršavaju na poslužitelju te mogu direktno komunicirati s bazom podataka koristeći ORM alate poput Prisme.

Unutar poslužiteljskih akcija se izvodi provjera podataka. Primjer provjere prilikom stvaranja rezervacije imamo na ispisu 3. Nakon što svi uvjeti se ispune i nakon što se potvrdi da su svi podatci ispravni, poslužiteljska akcija poziva određenu ugrađenu funkciju Prisme ORM-a. Izgled pozivanja funkcije Prisme nakon što se svi podatci provjere se nalazi na ispisu 4.

```
const jetskiAvailabilityChecks = reservation_jetski_list.map(async jetski => {
          const reservations = await db.reservation.findMany({
              where: {
                  reservation_jetski_list: {
                      some: {
                          jetski_id: jetski.jetski_id,
                      },
                  },
                  NOT: [
                      { endTime: { lte: startTime } },
                      { startTime: { gte: endTime } },
                  ],
              },
          });
          return reservations;
      });

      const results = await Promise.all(jetskiAvailabilityChecks);
      if (results.some(isAvailable => !isAvailable)) {
          return { error: "One or more jet skis are not available in the chosen
time slot." };}
```

Ispis 3: Provjera dostupnosti jetski plovila u poslužiteljskoj akciji

```
try {

        await db.jetski.create({
            data: {
                jetski_registration,
                jetski_location_id,
                jetski_topSpeed,
                jetski_kW,
                jetski_manufacturingYear,
                jetski_model,
            }
        });

        return { success: "Jetski successfully updated" };
    } catch (error) {
        return { error: "Failed to update jetski", details: error };
    }
```

Ispis 4: Stvaranje jetski plovila u poslužiteljskoj akciji

Prilikom dohvaćanja podataka poslužiteljska akcija je posrednička funkcija. Može prihvatiti argument po kojem će se filtrirati podatci. Na temelju argumenata Prisma stvara upit SQL-a s filterom koji dohvaća podatke iz baze podataka. Time se izbjegava dohvaćanje nepotrebnih podataka. Ispis 5 sadrži prikaz funkcije poslužiteljske akcije za dohvat podataka. U tom ispisu se preko `jetskiId` dohvaća jedno jetski plovilo. Aplikacija može dohvatiti više instanci objekta po određenom parametru što povećava performanse zbog manjeg broja upita prema bazi podataka. Poslužiteljske akcije pozivaju upite (engl. *Query*) koji su definirani kao funkcije unutar direktorija (engl. *Folder*) `Data`.

```
"use server";
import { getJetskiById } from "@/data/jetskiData";

export const getJetski = async (jetskiId: number)=>{

    const jetski = await getJetskiById(jetskiId)

    return jetski
}
```

Ispis 5: Primjer dohvaćanja podataka u *action* dijelu aplikacije.

4.1.2. Data

Unutar direktorija `Data` nalazi se više datoteka sa funkcijama koje pristupaju bazi podataka kroz upite (engl. *Query*) generirane od ORM-a Prisme. Poslužiteljske akcije pozivaju funkcije iz direktorija `Data` kako bi dohvatile podatke. Funkcije definirane u direktorijima su prilagodljive za dohvat bilo kojeg tipa podataka. Pretjerano pristupanje bazi podataka usporava rad poslužiteljskog servisa. Zato je važno precizno definirati funkcije i upite kako bi se izbjeglo bespotrebno i prekomjerno pristupanje bazi podataka.

Sve datoteke su organizirane prema entitetu kojem pristupaju. Sve funkcije su pisane u TypeScriptu kako bi sadržale dodatnu strukturu. Primjer funkcije koja radi upit na bazu se nalazi na ispisu 6. Postoji datoteka za svaki entitet:

- `jetskiData.ts`
- `locationData.ts`
- `rentalOptionData.ts`
- `reservationData.ts`
- `userData.ts.`

```
export const getJetskiById = async (jetski_id:number) => {
    try{
        const jetSki = await db.jetski.findUnique({where:{jetski_id}});
        return jetSki;
    } catch{
        return null;
    }
}
```

Ispis 6: Dohvaćanje objekta jetski plovila po njegovom id-u

4.1.3. Prisma

Prisma je moderni alat za upravljanje relacijskim bazama podataka. Omogućuje jednostavno definiranje modela entiteta koristeći vlastiti jezik za modeliranje entiteta. Modeli se definiraju pomoću shema (engl. *Schema*) u zasebnoj datoteci. Prisma ima ugrađene funkcije za migraciju, postavljanje podataka na bazu i za otvaranje vizualnog alata za upravljanje bazom podataka. Nakon upisivanja naredbe `npx prisma generate` stvaraju se modeli u TypeScript kôdu. Svi modeli i enumi se definiraju u zajedničkoj datoteci `schema.prisma`. Kod postavljanja Prisme odabere se vrsta baze. Za ovu aplikaciju odabrana je PostgreSQL baza. Baza je hostana na poslužitelju Neon.Tech, koji dodjeljuje privatne jedinstvene ključeve koji se postavljaju u `.env` datoteku. Prikaz postavljanje baze prikazan je u ispisu 7.

```
datasource db {
  provider  = "postgresql"
  url       = env("DATABASE_URL")
  directUrl = env("DIRECT_URL")
}
```

Ispis 7: Postavljanje povezivanja baze

Prisma ORM omogućuje deklarativan način opisivanja baze. Kod modela `User`, `user_id` koristi automatski generator identifikatora `@id @default(cuid()).Email` i `contactNumber` sadrže identifikator `@unique` koji osigurava jedinstvenost podataka. `user_status`-a i `user_role` su ograničeni na predefinirane vrijednosti enuma. Tako je osigurano da status i uloga korisnika može biti samo već definirana opcija. U zagradama je definirana vrijednost koja se postavlja ako druga vrijednost nije navedena. Prisma olakšava relacije između modela, gdje je dovoljno navesti `@relation` i ime relacije kako bi povezali dva polja. Primjer definiranja modela `Usera` je na ispisu 8.

```
model User {
  user_id          String     @id @default(cuid())
  name             String?
  email            String?    @unique
  password         String?
  contactNumber    String?    @unique
  user_status      UserStatus @default(ACTIVE)
  user_role        UserRole   @default(GUEST)
  user_location_id Int?
  managed_location Location?  @relation("ManagedLocation")
  user_location    Location?  @relation("UserLocation", fields:
[user_location_id], references: [location_id])
}
```

Ispis 8: Primjer modela `User`

4.2. Klijentska strana (engl. *Frontend*)

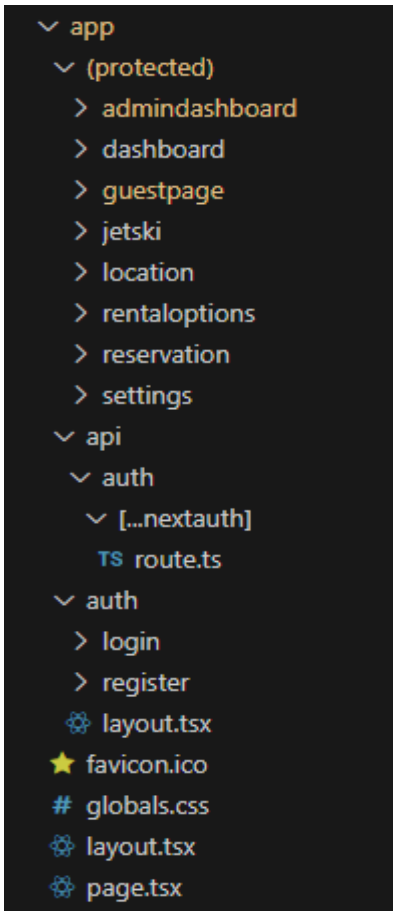
Klijentska strana je nužna u svakoj web aplikaciji jer s njom komunicira korisnik kako bi pristupio željenom sadržaju. Klijentska strana prezentira korisničko sučelje i usredotočuje se na korisničko iskustvo. Važno je da korisnik nema problema s navigacijom kroz aplikaciju. Prikaz mora biti dovoljno jednostavan da se može brzo savladati. U ovom projektu dvije glavne komponente za korisničko sučelje su: `App` i `Components`.

4.2.1. Direktorij App

Direktorij `App` služi za organizaciju i strukturiranje te jasno razdvajanje logičkih cjelina korisničke strane. Struktura osigurava da se aplikacija može lakše razvijati. Osigurava lakše održavanje te navigaciju kroz projekt razdvajajući logičke cjeline na više dijelova.

Struktura direktorija `App` unutar ovog projekta sadrži: `api`, `auth`, (`protected`) i zajedničke datoteke poput `page.tsx`, `layout.tsx` i `global.css`. Prikaz strukture `App` direktorija nalazi se na slici 3. Svaki od tih direktorija i svaka od tih datoteka ima svoju ulogu:

- `page.tsx` – početna stranica aplikacija.
- `layout.tsx` – glavni raspored komponenti za cijelu aplikaciju.
- `global.css` – datoteka s globalnim CSS stilovima.
- `api` – direktorij koji sadrži API rute za autentikaciju.
- `auth` – direktorij koji sadrži stranice za login i registraciju, nalaze se izvan (`protected`) direktorija jer neautenticirani korisnici moraju moći imati pristup.
- `(protected)` – direktorij koji sadrži stranice za koje je potrebno biti autenticirani korisnik kako bi mogli vidjeti njihov sadržaj.



Slika 3: Prikaz strukture `App` direktorija

Imena direktorija služe kao putanja. Na dnu stabla direktorija nalazi se `page.tsx` datoteka koja sadrži komponentu stranice kojoj se želi pristupiti. Njoj se pristupa putem dinamičkog url-a koji se generira na temelju imena direktorija u stablu u kojem se nalazi. Tako na primjer `page.tsx` datoteka koja u sebi sadrži funkciju `EditUser` se nalazi na putanji: `app\(protected)\admindashboard\[userId]\edituser.`

Prikaz preusmjeravanja korisnika na temelju pritiska tipke se nalazi na slici 4. Tipka preusmjerava korisnika na putanju `/admindashboard/{userId}/edituser.` Kod generiranja dinamičkih url-ova root folder imena `App` se ignorira. `UserId` je dinamički član url-a koji se dohvaća s pomoću tipke koja je pritisnuta uz određenog korisnika. Ta tipka će poslati `userId` u funkciju kao argument. Kada korisnik pristupi stranici `edituser` podatci o odabranom useru će biti dohvaćeni te će se moći urediti.

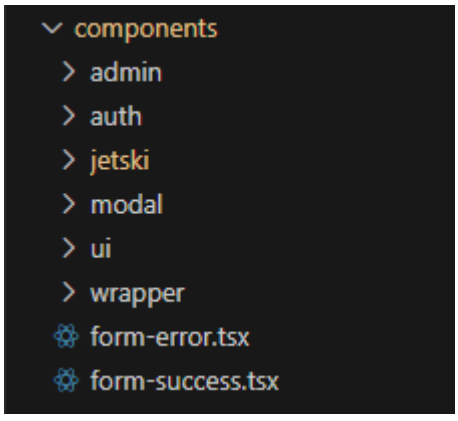
```
const handleEditUser = (userId: string) => {
  router.push(`/admindashboard/${userId}/edituser/`);
};
```

Slika 4: Prikaz navigacijske tipke i njenog url-a

4.2.2. Komponente (engl. *Components*)

Komponente su osnovni gradivni blokovi u aplikacijama koje se temelje na Reactu. Mogu se ponovno koristiti i osiguravaju da se kôd ne duplicira. Mogu sadržavati HTML, CSS i JavaScript te pružaju način za razdvajanje logike i strukture aplikacije na manje, lakše upravljive dijelove. Obilježene su enkapsulacijom, odnosno imaju sve što je potrebno za određeni dio sučelja, uključujući izgled i interaktivnosti. Komponente donose dodatnu sigurnost i izolaciju te sprječavaju da se promjene rade u neželjenim dijelovima aplikacije. Imaju mogućnost i kompozicije, što znači da se mogu ugnijezditi jedna unutar druge kako bi se stvorile složenije strukture. Tako aplikacija može sadržavati samo `layout` u kojem će se mijenjati glavni sadržaj pri čemu će `header` i `footer` ostati ne promijenjeni.

U komponentama se definira osnovni izgled svake stranice. Direktoriji su razdvojeni zbog strukture i lakšeg navigiranja. U `App` dijelu aplikacije se pozivaju komponente. Svaki `page.tsx` poziva komponentu koja je namijenjena za njega. Tako se osigurava sustavnost i lakše održavanje aplikacije zahvaljujući strukturi. Postoje i dijeljene komponente poput `form-error.tsx` i `form-success.tsx` i one koje se nalaze u direktoriju `ui` i direktoriju `wrapper`. Prikaz strukture direktorija `components` prikazan je na slici 5.



Slika 5: Struktura direktorija `components`

4.3. Konfiguriranje

4.3.1. Hooks

Kako bi se osiguralo da samo korisnici s dovoljnim pravima mogu pristupiti sadržaju i komponentama koje su namijenjene za administratora ili upravitelja koriste se kuke (engl. *hooks*). Kuke omogućavaju klijentskoj strani informaciju o korisnikovoj ulozi. Važno je znati korisnikovu ulogu kako bi se onemogućio pristup funkcionalnostima koje su rezervirane za korisnike s većim pravima. Primarno se koristi u komponentama.

4.3.2. Biblioteka (engl. *Library*)

Direktorij koji sadrži pomoćne funkcije. Unutar njega su definirani konfiguracijski podatci o Prismi i bazi podataka. Sadrži funkciju `getUserRole`. Koristi se unutar `layouta` u datotekama direktorija `App` kako bi se odredilo koje stavke i opcije će se prikazati korisniku na temelju njihovih prava.

4.3.3. Middleware

Middleware je softverska komponenta koja se nalazi između klijentske i poslužiteljske strane. Ona obrađuje zahtjeve prije nego što dođu do klijenta. U ovoj aplikaciji middleware se koristi za verifikaciju autentikacije korisnika. Direktoriji unutar `App` direktorija moraju biti pravilno strukturirani kako bi middleware znao gdje djelovati. U ispisu 9 prikazan je način djelovanja middlewarea. Ovisno o prefiksu putanje, kôd odlučuje djelovati ili ne. Sve komponente čiji se direktoriji nalaze unutar `ApiAuth` nema middleware provjere.

```
const isApiAuthRoute = nextUrl.pathname.startsWith(apiAuthPrefix);
const isPublicRoute = publicRoutes.includes(nextUrl.pathname);
const isAuthRoute = authRoutes.includes(nextUrl.pathname);
if (isApiAuthRoute){
        return undefined;
    }

if (isAuthRoute){
      if(isLoggedIn){
           return Response.redirect(new URL(DEFAULT_LOGIN_REDIRECT, nextUrl))
        }
        return undefined;
    }

if (!isLoggedIn && !isPublicRoute){
        return Response.redirect(new URL("/auth/login",nextUrl))}
```

Ispis 9: Prikaz autentikacije za određeni url

4.3.4. .env

Najjednostavnija datoteka konfiguriranja po veličini i količini sadržaja. Sadrži 3 konstante. Prva od njih je `AUTH_SECRET` koji se koristi pri maskiranju zaporke tako pružajući dodatnu sigurnost. Također sadrži `DIRECT_URL` i `DATABASE_URL` koji spajaju Prismu s bazom podataka na poslužiteljskom sustavom `Neon.Tech`. Vrlo je važan za produkciju jer omogućava jednostavnu promjenu baze podataka ako će aplikaciju koristiti više različitih poslovnih sustava. Za svaki odvojeni poslovni sustav se postavljaju različiti `DIRECT_URL` i `DATABASE_URL`.

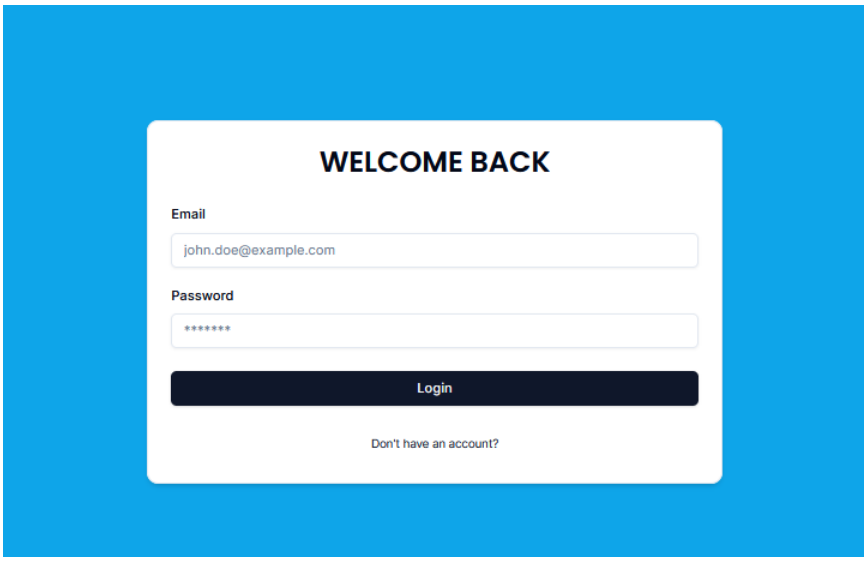
4.3.5. Auth.ts

Auth.ts služi za autentikaciju u aplikaciji koristeći `NextAuth` open-source biblioteku. Sjedinjuje se sa Prismom za upravljanje korisnicima i sesijama. Koristi `callback` funkcije koje primaju drugu funkciju kao argument i potom je izvršavaju. Od iznimne su važnosti u autentikaciji korisnika pri korištenju `JWT (Json Web Token)` i sesija jer dopuštaju fleksibilnost i prilagodljivost u procesu autentikacije. Pomoću tih funkcija u sesiju se mogu priključiti dodatne informacije kao što su korisnikova uloga, njegov status i lokacija na kojoj se nalazi.

5. Funkcionalnosti

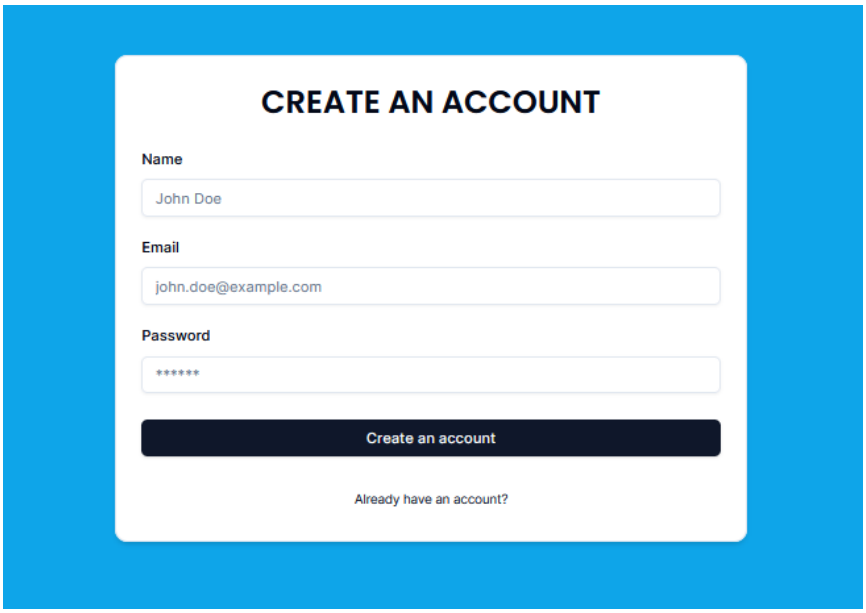
5.1. Autentikacija korisnika

Pri otvaranju web stranice korisniku je ponuđena forma u kojoj se upisuju korisnički podatci. Prijava korisnika sadrži dva tekstualna polja. Prvo tekstualno polje je za unos elektroničke pošte korisnika, dok je drugo tekstualno polje za unos zaporke korisnika. Naposljetku, nalazi se tipka za prijavu koja pokreće provjeru unesenih podataka. Ako su podatci ispravni, korisnik će biti preusmjeren na početnu stranicu. Ako podatci nisu ispravni, korisnik će dobiti upozorenje da su uneseni podatci pogrešni: „*Invalid credentials*“. Važno je ne specificirati koji je podatak pogrešno unesen zbog veće sigurnosti aplikacije. Izgled polja za prijavu je prikazan na slici 6.



Slika 6: Izgled prijavne forme

Ako se radi o korisniku koji prvi put pristupa stranici on neće imati registrirani račun, osim ako administrator nije stvorio račun za njega. U tom slučaju korisnik mora pritisnuti na tekst „*Don't have an account?*“ koji služi kao tipka koja ga preusmjerava na registracijsku formu. Registracijska forma je prikazana na slici 7. Sadrži tri tekstualna polja. Pri stvaranju novog računa sva polja moraju biti ispunjena. Prvo tekstualno polje je ime (engl. *name*) korisnika. Drugo tekstualno polje je elektronička pošta (engl. *email*) korisnika. Ona mora biti jedinstvena za svakog korisnika. Ako se elektronička pošta već koristi, prikazat će se poruka *„Email already in use!“*. Treće tekstualno polje je zaporka. Zaporka mora sadržavati barem osam znakova. Imena korisnika se mogu poklapati.



Slika 7: Izgled registracijske forme

Za provjeru podataka koristi se Zod shema, odnosno Zod objekt. Zod shema služi za stvaranje objekta koji služe za validaciju unesenih podataka. Podatci se prosljeđuju na poslužiteljsku stranu pomoću objekta sheme ako su zadovoljeni uvjeti (npr. lozinka duža od osam znakova). Po potrebi se vrše dodatne provjere te naposljetku podatci se spremaju u bazu podataka. U slučaju prijave, tj. ako se podatci potvrde na klijentskoj strani, podatci se šalju u `NextAuth` funkciju `signIn` koja izvršava autentikaciju. Ako je autentikacija uspješna `NextAuth` dodjeljuje korisniku JWT token koji se koristi za upravljanje sesijom korisnika. Na ispisu 10 se nalazi primjer Zod objekta za validaciju prijave i registracije korisnika.

```
export const LoginSchema = z.object({
    email: z.string().email({
        message: "Email is required"
    }),
    password: z.string().min(1,{message: "Password is required"}),
});

export const RegisterSchema = z.object({
    email: z.string().email({
        message: "Email is required!"
    }),
    password: z.string().min(8,{message: "Minimum length is 8 charachters"}),
    name: z.string().min(1,{
        message: "Name is required!"
    })
});
```

Ispis 10: Prikaz zod objekta za provjeru podataka

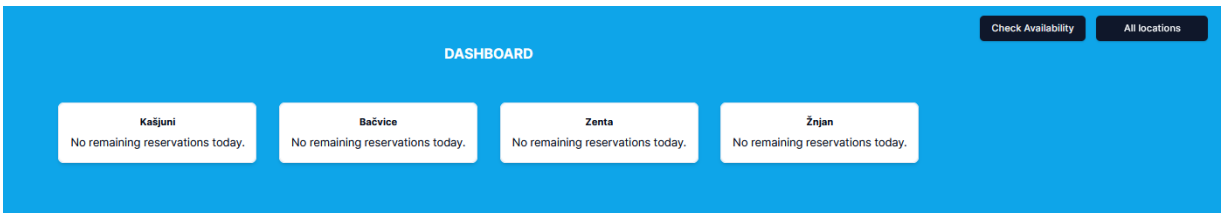
Zod shema ne može provjeriti postoji li određena elektronička pošta u bazi podataka. Provjera jedinstvenosti elektroničke pošte se izvršava u akcijama na poslužiteljskoj strani. Nakon što je sigurno da ta elektronička pošta ne postoji, lozinka se hashira i konačno sprema u bazu podataka. Na ispisu 11 se nalazi prikaz stvaranja novog korisnika.

```
const existingUser = await getUserByEmail(email);

if (existingUser){
        return {error: "Email already in use!"};
};

const hashedPassword = await bcryptjs.hash(password, 10);


await db.user.create({
        data:{
            name,
            email,
            password: hashedPassword,
        }
    })
```

Ispis 11: Prikaz stvaranja novog korisnika

5.2. Početna stranica i navigacijska traka

Početna stranica (engl. *Dashboard*) je prikazana pri prijavljivanju korisnika koji nije gost. Sadrži popis lokacija u sustavu poslovanja te dvije tipke u gornjem desnom kutu. Lokacije su prikazane u bijelim kvadratima s imenom lokacije kao naslovom te listom nadolazećih rezervacija na toj lokaciji. Dvije tipke u gornjem desnom kutu su: `Check Availability` i filter za prikazivanje rezervacija po lokaciji, kao što je prikazano na slici 8.

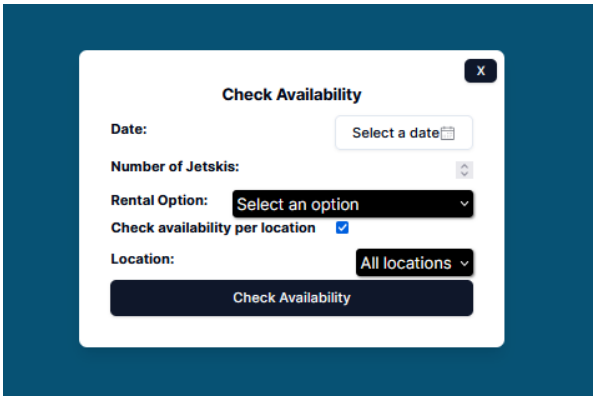


Slika 8: Izgled glavnog dijela početnog sučelja

Pritiskom na tipku `Check Availability` otvara se modalni skočni prozor (engl. *popup*) s kojim korisnik može dobiti preporuku za termin rezervacije. Prikazan je na slici 9. Funkcionira na temelju datuma, broja potrebnih plovila i tražene opcije za iznajmljivanje.

Korisnik može ograničiti da se plovila, koja se uzimaju u obzir, nalaze na istoj lokaciji, ali početno pravilo je da preporuke za rezervaciju nisu ograničene lokacijom plovila.

Provjera dostupnosti se izvršava tako da se dohvate sve rezervacije odabranog dana. Tada se generiraju mogući periodi na temelju trajanja izabrane opcije. Zatim, ovisno o tome je li odabrana pojedina lokacija, dohvaćaju se dostupna plovila za pojedinu lokaciju ili ako lokacija nije odabrana, dohvaćaju se dostupna plovila svih lokacija. Potom se za svaki period uzimaju rezervacije koje se preklapaju s tim periodom. Ako postoje takve rezervacije, uzima se broj plovila koji se nalazi u tim rezervacijama. Kada je broj plovila manji ili jednak zatraženom broju prilikom stvaranja upita, taj period se smatra dostupnim i dodaje se u listu dostupnih perioda. Na ispisu 12 prikazana je provjera preklapajućih rezervacija.



Slika 9: Prikaz modalnog popup-a za provjeru dostupnosti

```
while (currentSlotIndex < slots.length) {
    const slot = slots[currentSlotIndex];
    const slotEndWithBuffer = new Date(slot.end.getTime() + BUFFER_MINUTES * 60 *
1000);

    const overlappingReservations = reservations.filter((reservation) => {
      const reservationStartTime = new Date(reservation.startTime);
      const reservationEndTime = new Date(reservation.endTime);
      const reservationEndWithBuffer = new Date(reservationEndTime.getTime() +
BUFFER_MINUTES * 60 * 1000);
      return (
        (reservationStartTime < slotEndWithBuffer && reservationEndWithBuffer >
slot.start)
      );
```

Ispis 12: Provjera preklapajućih rezervacija

Tipka za filtriranje rezervacija po lokaciji služi korisniku da na jednostavan način dobije informacije o rezervacijama pojedine lokacije. U izborniku se nalazi popis svih lokacija kao i opcija bez filtriranja, ako korisnik ne želi prikaz za pojedinu lokaciju.

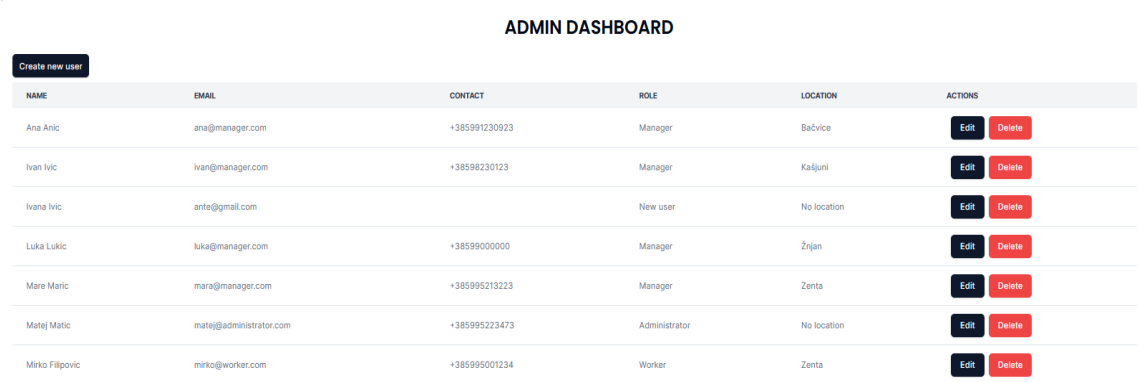
Na temelju razine korisničkih prava, odnosno korisnikove uloge, prikazat će se različiti elementi navigacijskog sučelja. Svi elementi su zapravo tipke koje preusmjeravaju korisnika na stranicu koju je odabrao. Na svim stranicama se nalazi dodatna provjera koja

će preusmjeriti korisnika na početnu stranicu ako korisnik nema dovoljna prava za pristupiti stranici.

5.3. Korisničke uloge i upravljanje korisnicima

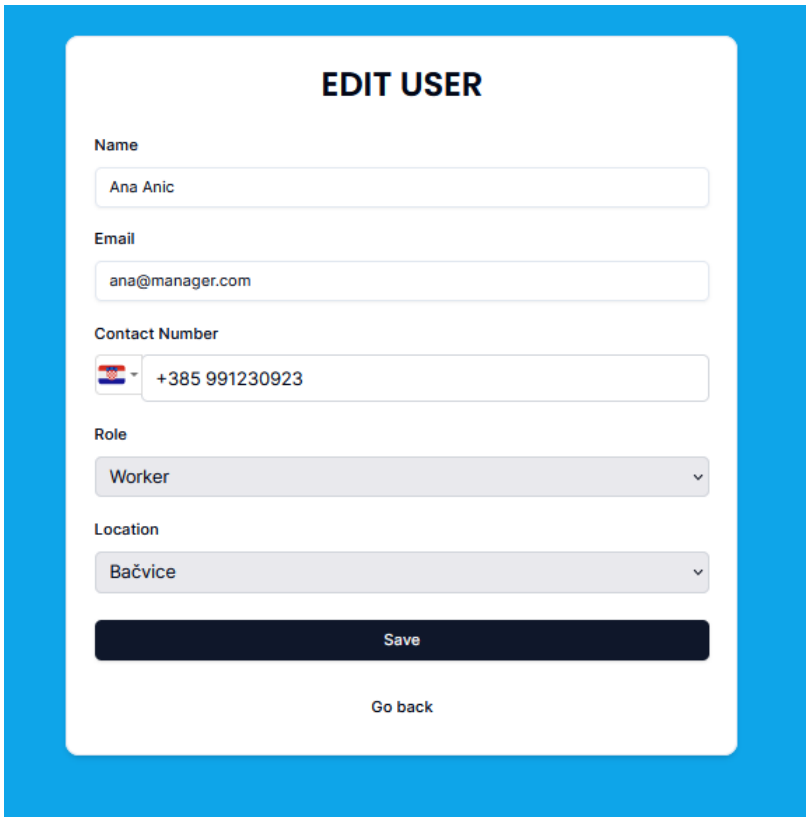
Aplikacija sadrži četiri razine korisničkih prava, tj. korisničke uloge. To su redom: gost, radnik, upravitelj i administrator. Korisnik pri stvaranju računa ima početnu ulogu gosta (`GUEST`). Korisnički račun će biti prikazan na stranici `Admin dashboard` nakon što je stvoren. To je nadzorna ploča administratora. Na ploči je prikazan popis svih korisničkih računa s popratnim podatcima o korisnicima. Uz imena se nalaze dvije tipke: za uređivanje korisnika i za brisanje korisnika. U gornjem lijevom kutu nalazi se tipka za stvaranje korisničkog računa ako administrator želi sam stvoriti račun za drugog korisnika. Ako se korisnik samostalno registrira neće imati postavljenu lokaciju niti ulogu. Ipak, ako administrator stvori novog korisnika, morat će postaviti ulogu i lokaciju korisnika.

Na sučelju su ispisani podatci: ime, email, kontakt (koji nije obavezan), uloga, lokacija. Izgled nadzornog sučelja prikazan je na slici 10. Administrator može poredati korisnike po njihovom imenu, emailu, po njihovoj ulozi te po lokaciji korisnika. Uz svakog korisnika nalaze se dvije tipke. Prva tipka služi za uređivanje korisnika. Druga tipka služi za trajno brisanje korisnika.



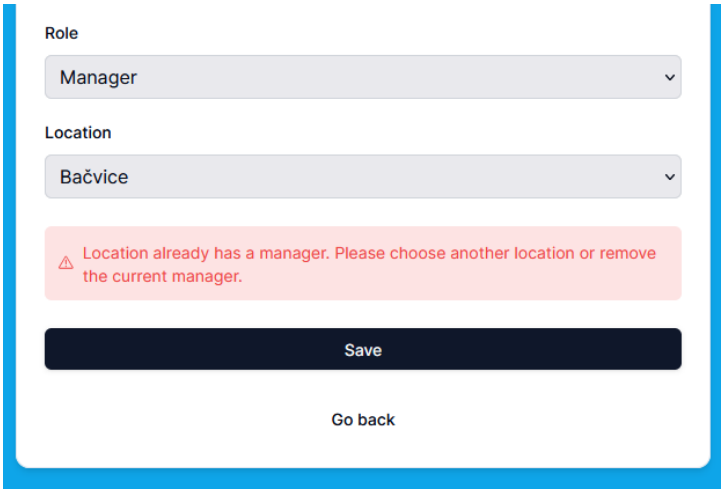
Slika 10: Prikaz nadzornog sučelja administratora

Izgled stranice za uređivanje korisnika prikazan je na slici 11. Administrator prilikom uređivanja podataka o korisniku može izmijeniti njegovo ime, email, kontakt broj te ulogu i lokaciju. Korisnički email i kontakt moraju biti jedinstveni. Ako novi podatci već postoje u bazi podataka kod drugog korisnika, spremanje uređenih podataka neće biti moguće.



Slika 11: Prikaz forme za uređivanje korisnika

Jedna lokacija može imati samo jednog upravitelja. Ako administrator želi postaviti određenog korisnika kao upravitelja lokacije koja već ima upravitelja bit će spriječen. Prikazat će se upozorenje da ta lokacija već sadrži upravitelja. Na slici 12 je prikazano upozorenje ako administrator pokuša postaviti upravitelja na lokaciju koja već ima upravitelja.



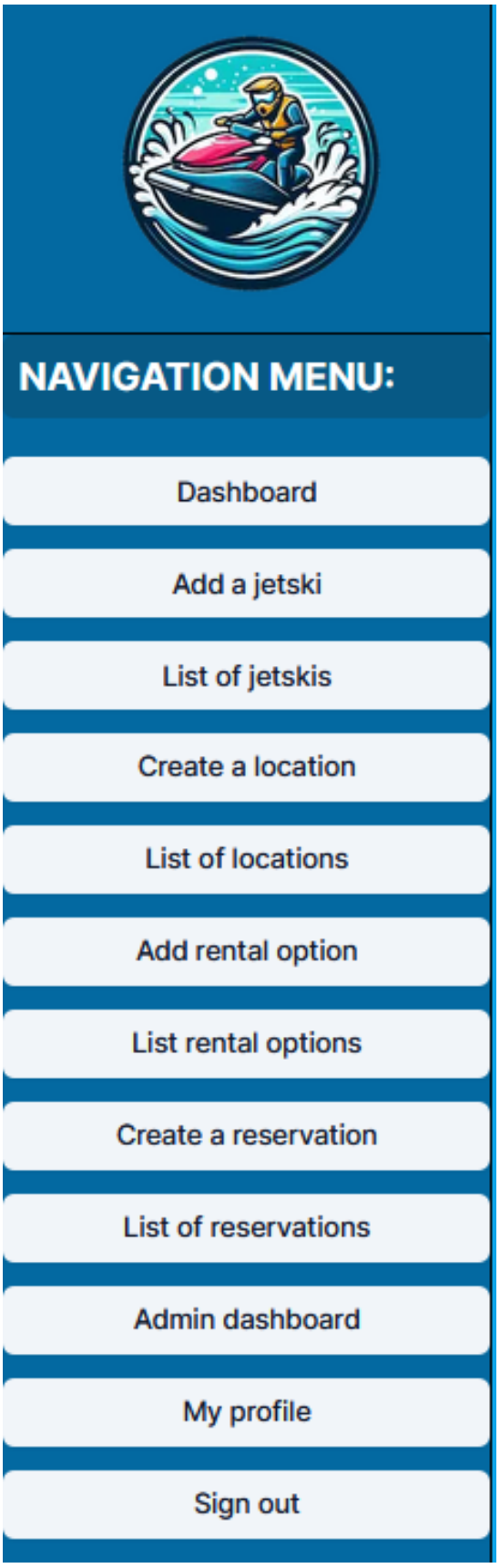
Slika 12: Prikaz upozorenja pri postavljanju upravitelja lokacije

Korisnik koji ima ulogu upravitelja se ne može izbrisati jer bi lokacija ostala bez upravitelja. Potrebno je prvo ukloniti korisnikovu ulogu kako bi se trajno izbrisao taj korisnik. Prikaz provjere nalazi se u ispisu 13. Korisnik s ulogom administratora se ne može izbrisati kroz nadzorno sučelje administratora.

```
try {
       if (user.user_role === "ADMIN") {
           return { error: "Cannot delete administrator!" };
        }
       const isManager = await db.location.findFirst({ where: { manager_id:
user_id}
                });
       if (isManager) {
           return { error: "Cannot delete user. They are a manager of a
location!" };}
       await db.user.delete({where: {user_id}});

       return { success: "User deleted successfully!" };
```

Ispis 13: Provjere prilikom brisanja korisnika

Različite uloge imaju različita prava. Početna korisnička uloga, gost, ne može pristupiti ni jednoj stranici. Korisnik koji ima ulogu gosta mora pričekati da mu administrator promijeni ulogu. Kada mu administrator promijeni ulogu on ima pristup ostalom sadržaju aplikacije, ovisno o njegovoj novoj ulozi. Uloga radnika ima osnovna prava. On ima pristup tipkama na početnoj stranici za provjeru dostupnosti rezervacija te ima prava vidjeti listu plovila, popis lokacija te popis rezervacija za taj dan. Upravitelji imaju veća prava. Oni mogu dodavati plovila na svojoj lokaciji i stvarati rezervacije za svoju lokaciju. Konačno, administratori imaju apsolutna prava. Na slici 13 prikazana je navigacijska traka administratora.

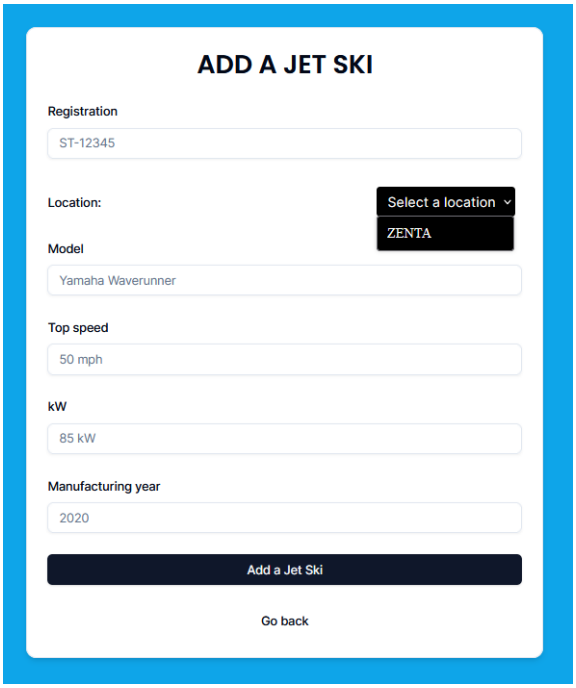


Slika 13: Navigacijska traka administratora

5.4. Dodavanje plovila, lokacija i opcija najmova

5.4.1. Dodavanje plovila

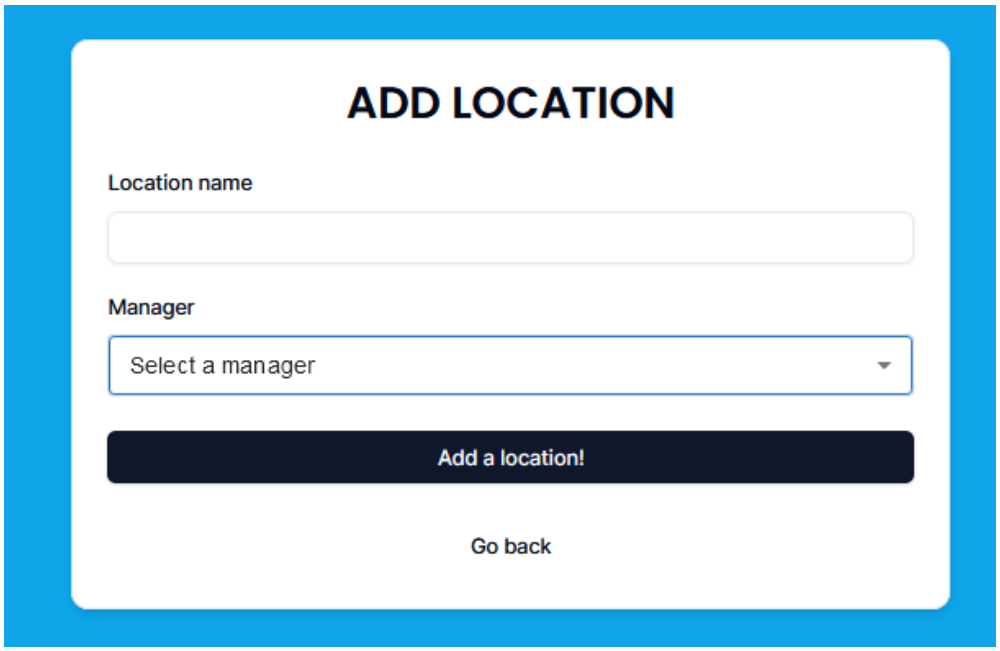
Samo korisnici s dovoljnom razinom prava mogu stvoriti plovila. To su korisnici koji imaju ulogu upravitelja ili administratora. Korisnik s ulogom upravitelja može dodati plovilo samo na svojoj lokaciji, dok administrator može dodati plovilo na bilo kojoj lokaciji u sustavu. Prilikom dodavanja plovila postoji pet tekstualnih formi te jedna forma s više izbora. To su: registracija plovila, lokacija plovila, model, maksimalna brzina, snaga motora i godina proizvodnje. Prilikom unosa moraju se zadovoljiti određeni uvjeti. Registracija mora biti točnog formata: prefiks od dva slova spojena crtom s tri do šest brojeva, npr. ST-12345. Registracija mora biti jedinstvena. Prilikom dodavanja novog plovila lokacija obavezno mora biti odabrana. Niti jedno polje ne smije ostati prazno. Maksimalna brzina se može unijeti u miljama na sat i kilometrima na sat. Ako je unesena brzina izražena u miljama na sat, prije pohrane u bazu podataka, bit će pretvorena u kilometre na sat. Kod godine proizvodnje postoji provjera da korisnik ne unese plovilo novije od iduće godine ili prije 1950. godine. Svi uneseni brojevi moraju biti pozitivni. Nakon što su svi podatci uneseni pritiskom na tipku `Add a Jet Ski` pokrenuti će se provjera svih unesenih podataka. Novo plovilo u bazi podataka će se stvoriti ako su svi podatci potvrđeni. Prikaz forme stvaranja novog plovila se nalazi na slici 14.



Slika 14: Prikaz dodavanja plovila kao upravitelj lokacije Zenta

5.4.2. Stvaranje lokacija

Stvaranje novih lokacija omogućeno je korisnicima s ulogom administratora ili upravitelja. Postoji razlika u dopuštenim akcijama. Kada korisnik ima ulogu upravitelja, može stvoriti samo lokaciju, bez prava da postavi upravitelja lokacije. Ako administrator stvara lokaciju, on ima dodatnu opciju da odabere upravitelja lokacije. Prilikom stvaranja nove lokacije važno je da ima jedinstveno ime. Kada administrator odabere korisnika s ulogom upravitelja, on se mora pobrinuti da taj upravitelj nema trenutno lokaciju kojom upravlja jer u protivnom neće moći stvoriti novu lokaciju. Prikaz izgleda forme za stvaranje lokacije se nalazi na slici 15. Forma se sastoji od jedne tekstualne forme za unos imena lokacije te izbornika upravitelja ako je korisnik administrator. Na dnu komponente se nalazi tipka `Add a location` koja pokreće postupak provjere točnosti podataka. Ako uneseni podatci odgovaraju svim provjerama stvorit će se nova lokacija. Ispis provjere prilikom stvaranja se nalazi na ispisu 14.



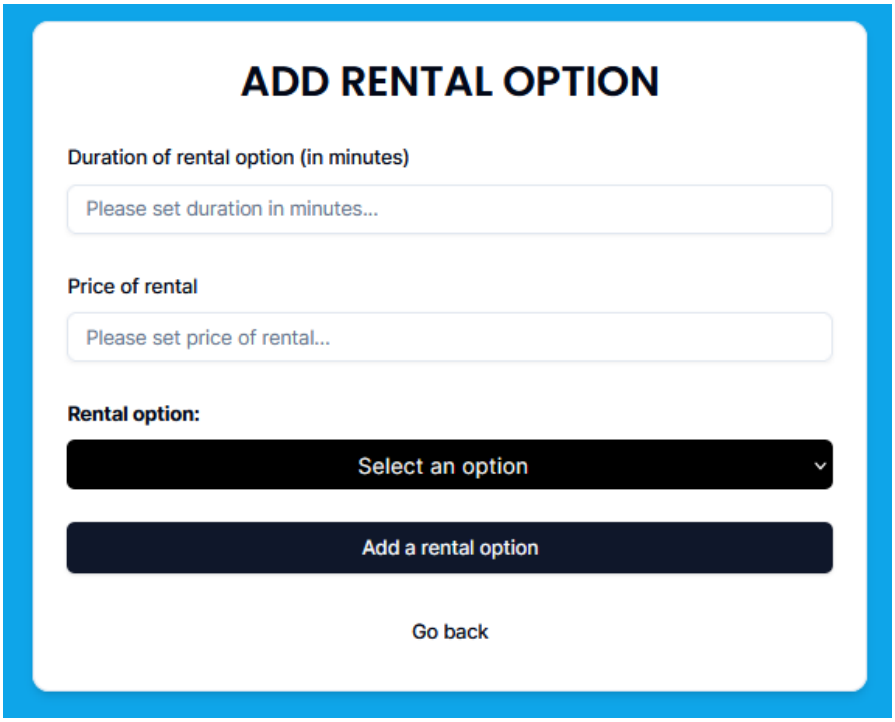
Slika 15: Prikaz stvaranja lokacije kao administrator

```
export const createLocation = async (values: z.infer<typeof LocationSchema>)
=> {
    const validatedField = LocationSchema.safeParse(values);

    if (!validatedField.success) {
        return { error: "Invalid fields" };
    }

    const { location_name, user_id } = validatedField.data;

    const newLocName = location_name.toLowerCase();
    const finalName = newLocName.charAt(0).toUpperCase() +
newLocName.slice(1);

    const existingLocation = await getLocationByName(location_name);

    if (existingLocation) {
        return { error: "Location with that name already exists!" };
    }

    if (user_id !== null) {
        const existingModerator = await db.location.findFirst({
            where: {
                location_manager_id: user_id,
            },
        });

        if (existingModerator) {
            return { error: "User is already a manager of another location!"
};
        }
    }

    const createdLocation = await db.location.create({
        data: {
            location_name: finalName,
            location_manager_id: user_id,
        },
    });

    if (user_id) {
        await db.user.update({
            where: {
                user_id: user_id,
            },
            data: { user_location_id: createdLocation.location_id },
        });
    }

    return {
        success: "Location has been added successfully!"
    };
};
```

Ispis 14: Provjera prilikom stvaranja nove lokacije

5.4.3. Stvaranje opcija najma

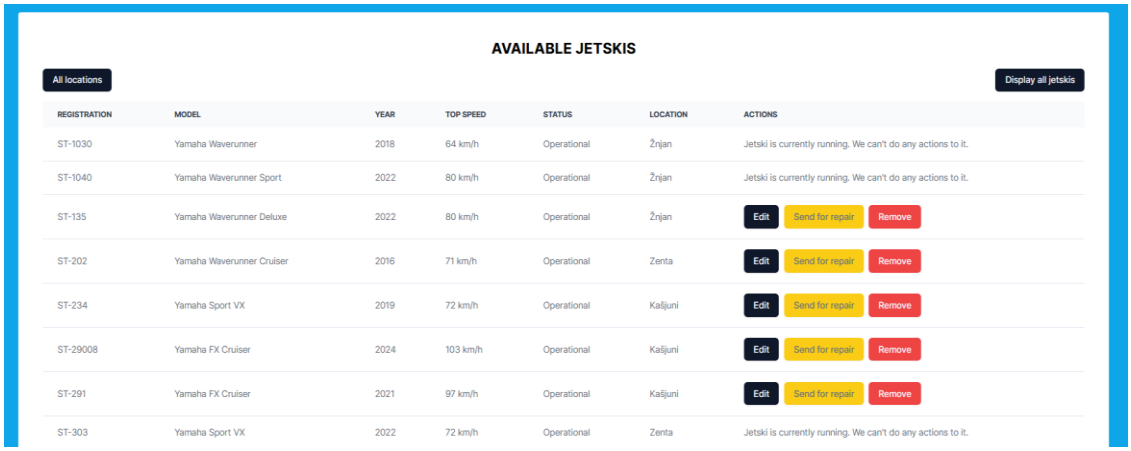
Dodavanje novih opcija najma omogućeno je samo administratoru. Opcije najma preslikavaju ponudu poslovanja. Forma se sastoji od dva tekstualna polja te jednog padajućeg izbornika za odabir tipa rezervacije. Unutar prvog tekstualnog polja unosi se trajanje opcije u minutama, npr. 90 minuta. Drugo tekstualno polje koristi se za unos cijene najma. Cijenu se može unijeti u različitim valutama jer je unos tipa `string`. Izbornik `rental option` označava tip najma. Postoje dva tipa najma: safari najam i regularni najam. Safari najam označava tip najma u kojem gost uzima dodatno plovilo kao pratnju, odnosno turističkog vodiča. Regularni najam predstavlja obični najam u kojem gost iznajmi plovilo samo za sebe. Konačno tipka `Add a rental option` omogućuje stvaranje opcije za iznajmljivanje. Pritiskom na tipku pokreće se provjera postoji li već opcija s istom duljinom trajanja i tipom iznajmljivanja (safari ili regularni najam). Uz to, postoji provjera koja garantira da cijena ne smije biti manja ili jednaka nuli. Izgled forme se nalazi na slici 16.



Slika 16: Prikaz forme za dodavanje nove opcije za iznajmljivanje

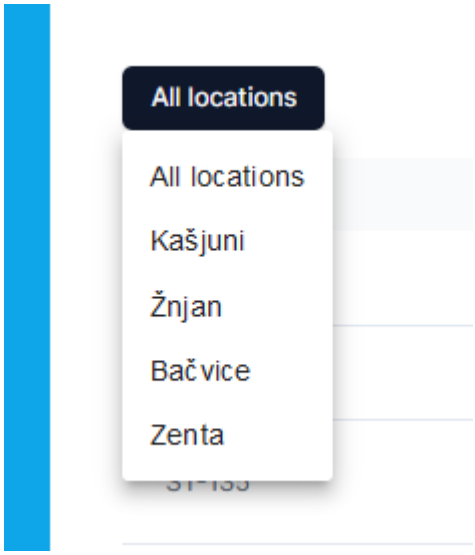
5.5. Prikaz i uređivanje jetskija, lokacija i opcija za iznajmljivanje

Korisnik sa ulogom radnika, upravitelja ili administratora može pristupiti popisu jetskija, lokacija i opcija za iznajmljivanje. Kod prikaza popisa jetskija korisnik može vidjeti sve detalje tih plovila. Jetskiji se mogu sortirati po svim tablicama: registracija, model, godina proizvodnije, maksimalna brzina, status, lokacija kao što je prikazano na slici 17.



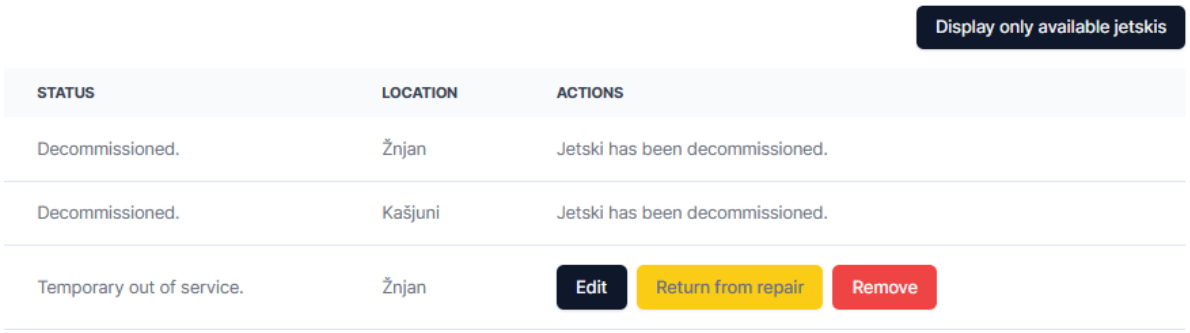
Slika 17: Prikaz dostupnih jetskija kao administrator

Uz popis jetski plovila prikaz sadrži i dvije tipke. Tipka u gornjem lijevom kutu služi za filtraciju lokacija, kao što je vidljivo na slici 18. To je padajući izbornik u kojem se nalaze sve lokacije koje su prethodno dodane. Ako odaberemo jednu od lokacija prikazat će se samo jetskiji s odabrane lokacije.



Slika 18: Dostupne lokacije za filtriranje

Tipka u gornjem desnom kutu radi po principu „upali-ugasi“. Pritiskom na tipku prikazat će se jetski plovila koji više nisu u pogonu. To mogu biti jetski plovila koja se više ne nalaze unutar tvrtke ili jetski plovila koji su privremeno izvan funkcije, npr. nalaze se na servisu. Jetski sa statusom `decommissioned` se ne može ponovno koristiti. Ako je jetski na servisu, imat će status `temporary out of service` i korisnik ga može vratiti jednostavnim klikom na tipku `return from repair`. Tipke za uređivanje, slanje na servis i brisanje jetski plovila će privremeno nestati sve dok se jetski nalazi u najmu. To je prikazano na slici 19.

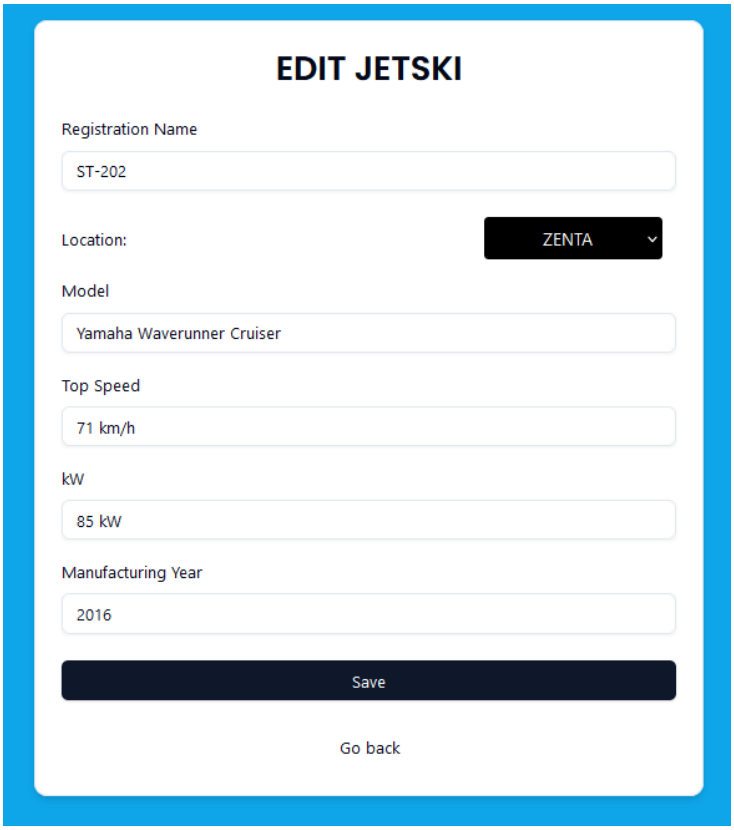


Slika 19: Odabran prikaz nedostupnih jetskija

Kod uređivanja jetski plovila forma izgleda jednako kao i kod dodavanja novih jetski plovila. Podatci forme se automatski popune na temelju podataka plovila koji se uređuje. Uređivanju se pristupa pritiskom na tipku `edit`, koja preusmjerava korisnika na dinamičko generiranu poveznicu. Ta poveznica se generira na temelju identifikatora jetski plovila. Sve forme prolaze kroz provjere. Korisnik može postaviti novu registraciju plovila ako se nova registracija ne koristi, kao što je prikazano na ispisu 15. Izgled forme se nalazi na slici 20.

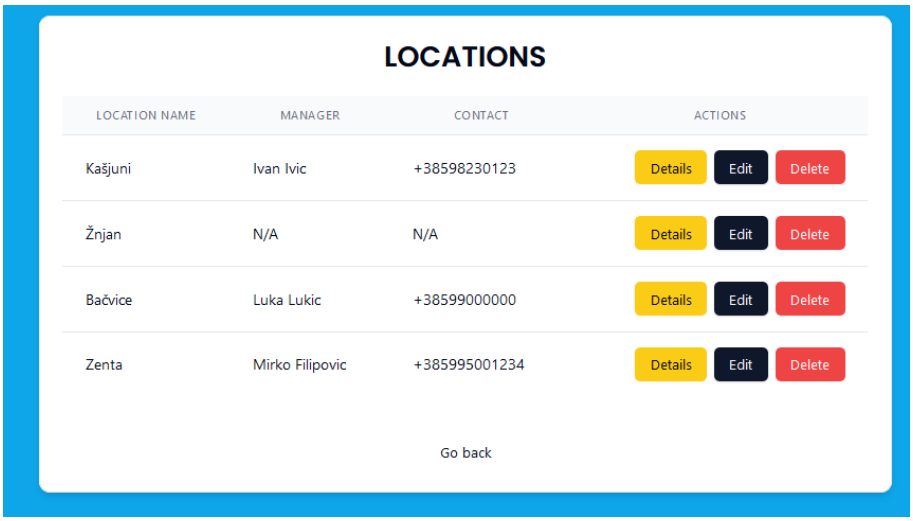
```
const doesJetskiExist = await getJetskiById(jetskiId);

if (!doesJetskiExist) {
    return { error: "Jetski not found" };
}

const existingJetski = await getJetskiByName(jetski_registration);

if (existingJetski && existingJetski.jetski_id !== jetskiId) {
    return { error: "Jetski with that registration already exists };
}
```

Ispis 15: Provjera postoji li registracija već u sustavu, a da nije registracija trenutnog plovila



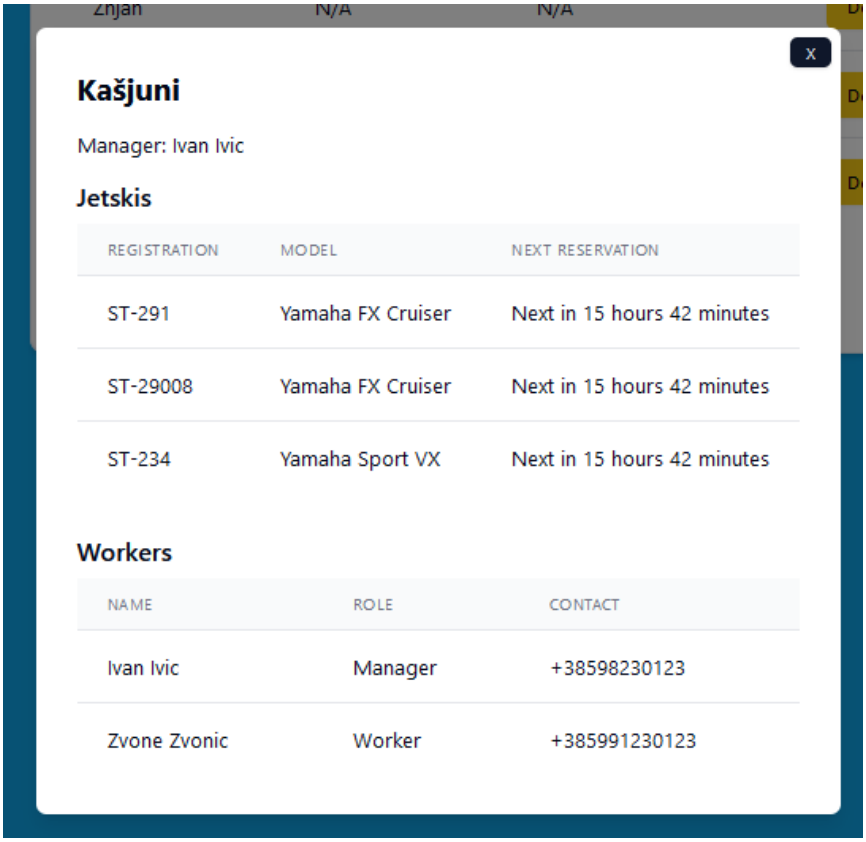
Slika 20: Prikaz uređivanja plovila registracije ST-202

Prikazu lokacija mogu pristupiti svi korisnici osim korisnika s ulogom gosta. Popis sadrži četiri tablice. Korisnik može poredati popis po svim tablicama. Tablice su: ime lokacije, ime voditelja lokacije, kontakt broj voditelja i akcije. Posljednja tablica akcije (engl. *actions*) sadrži tipke koje korisnik može pritisnuti. Ako je korisnik administrator bit će mu dodatno prikazane tipke za uređivanje i brisanje lokacija. Svi korisnici imaju pristup tipki detalji (engl. *details*). Lokacija koja sadrži rezervacije se ne može izbrisati dok se sve rezervacije povezane s tom lokacijom ne uklone. Također, ako lokacija ima upravitelja, ona se ne može izbrisati. Kod uređivanja lokacija administrator može promijeniti upravitelja lokacije te ime same lokacije, ako ne postoji druga lokacija s tim imenom. Prikaz popisa lokacija nalazi se na slici 21.



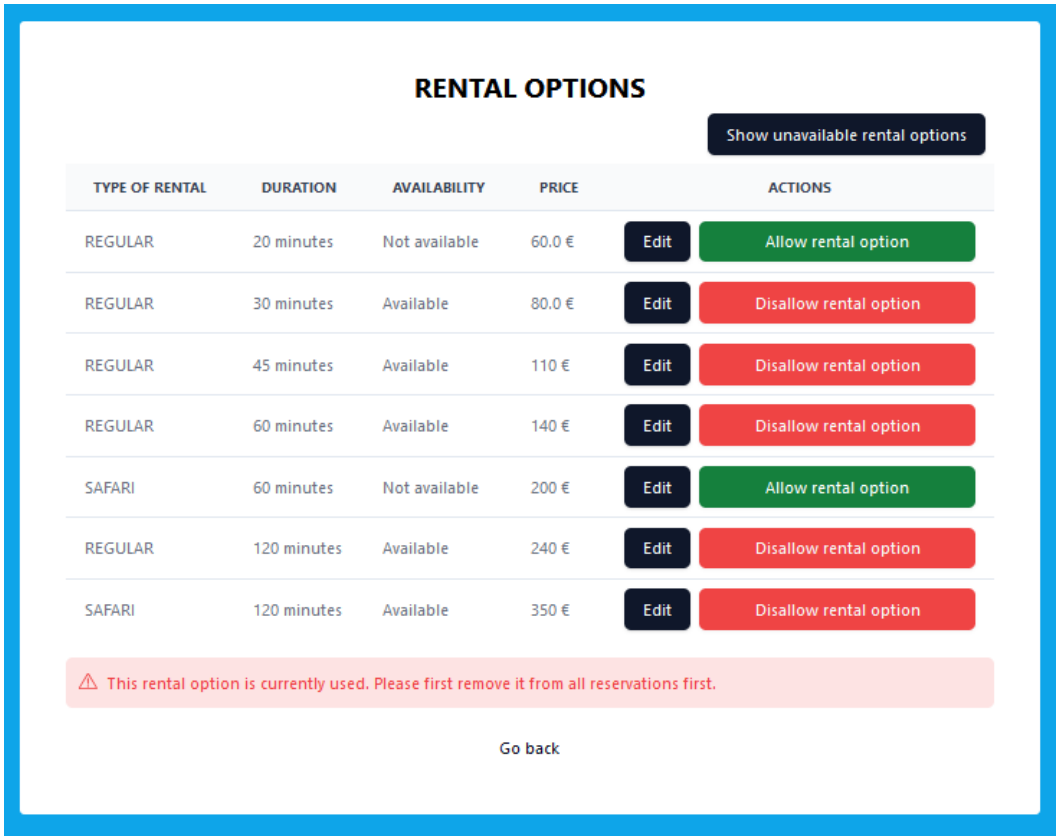
Slika 21: Izgled popisa lokacija

Pritiskom na tipku detalji prikazuje se skočni prozor, prikazan na slici 22. Na skočnom prozoru su prikazani dodatni podatci o lokaciji. Tu se nalazi popis plovila koji su registrirani na toj lokaciji, odnosno njihove registracije i modeli tih plovila. Također se nalazi informacija koliko je vremena preostalo do iduće rezervacije. Na dnu se nalazi popis radnika, njihove uloge i kontakt brojevi.



Slika 22: Skočni meni o detaljima lokacije

Popisu opcija najma može pristupiti korisnik čija uloga nije gost. Kod pristupanja administratora prikazat će se dodatna tablica koja omogućuje uređivanje opcija najma te onemogućavanje tih opcija. Kod prikaza imamo pet stupaca: tip najma, trajanje, dostupnost, cijena i konačno akcije. U gornjem desnom kutu nalazi se tipka koja omogućuje prikazivanje onemogućenih opcija najma. Opcija najma se ne može onemogućiti ako postoji rezervacija koja se trenutno izvodi ili se nalazi u budućnosti s tom opcijom. Ako je opcija onemogućena njena dostupnost će biti postavljena kao `not available`. Tipka uz nju će postati zelena da označi korisniku da je može omogućiti. Na slici 23 prikazan je izgled popisa opcija za najam uz poruku koja iskoči ako korisnik pokuša onemogućiti opciju najma koja se koristi. Kod uređivanja administrator može promijeniti trajanje najma i cijenu najma.

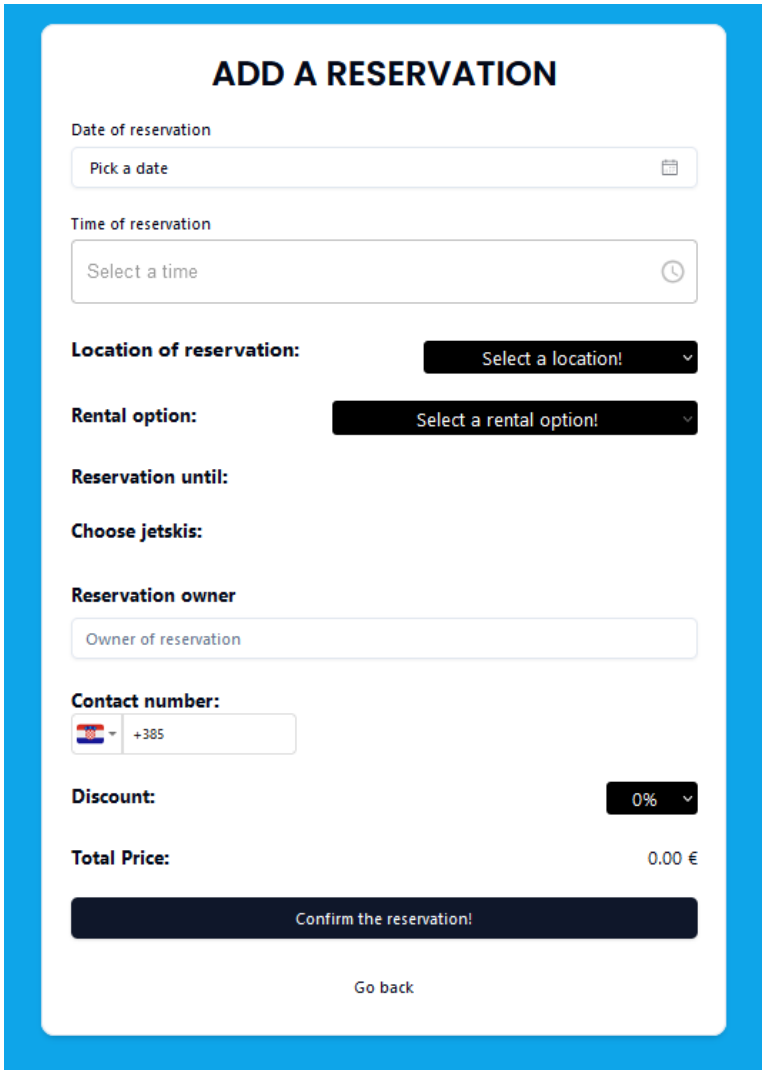


Slika 23: Popis opcija najma

5.6. Stvaranje, uređivanje i prikaz rezervacija

5.6.1. Stvaranje rezervacija

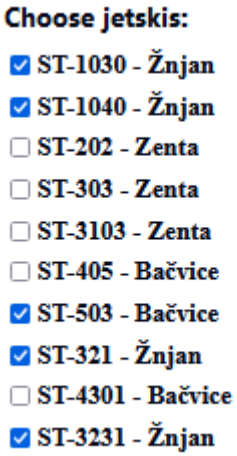
Stvaranje nove rezervacije rezervirano je za voditelje lokacija i administratora. Voditelj lokacije može stvoriti rezervaciju samo na svojoj lokaciji. Administrator nije ograničen po pitanju lokacije te može stvoriti rezervaciju na bilo kojoj lokaciji. Prilikom stvaranja rezervacije korisnik ima više izborničkih stupaca. Novootvorena forma bit će potpuno prazna. Prikaz forme za stvaranje rezervacija je na slici 24.



Slika 24: Prikaz stvaranja rezervacije pri otvaranju forme

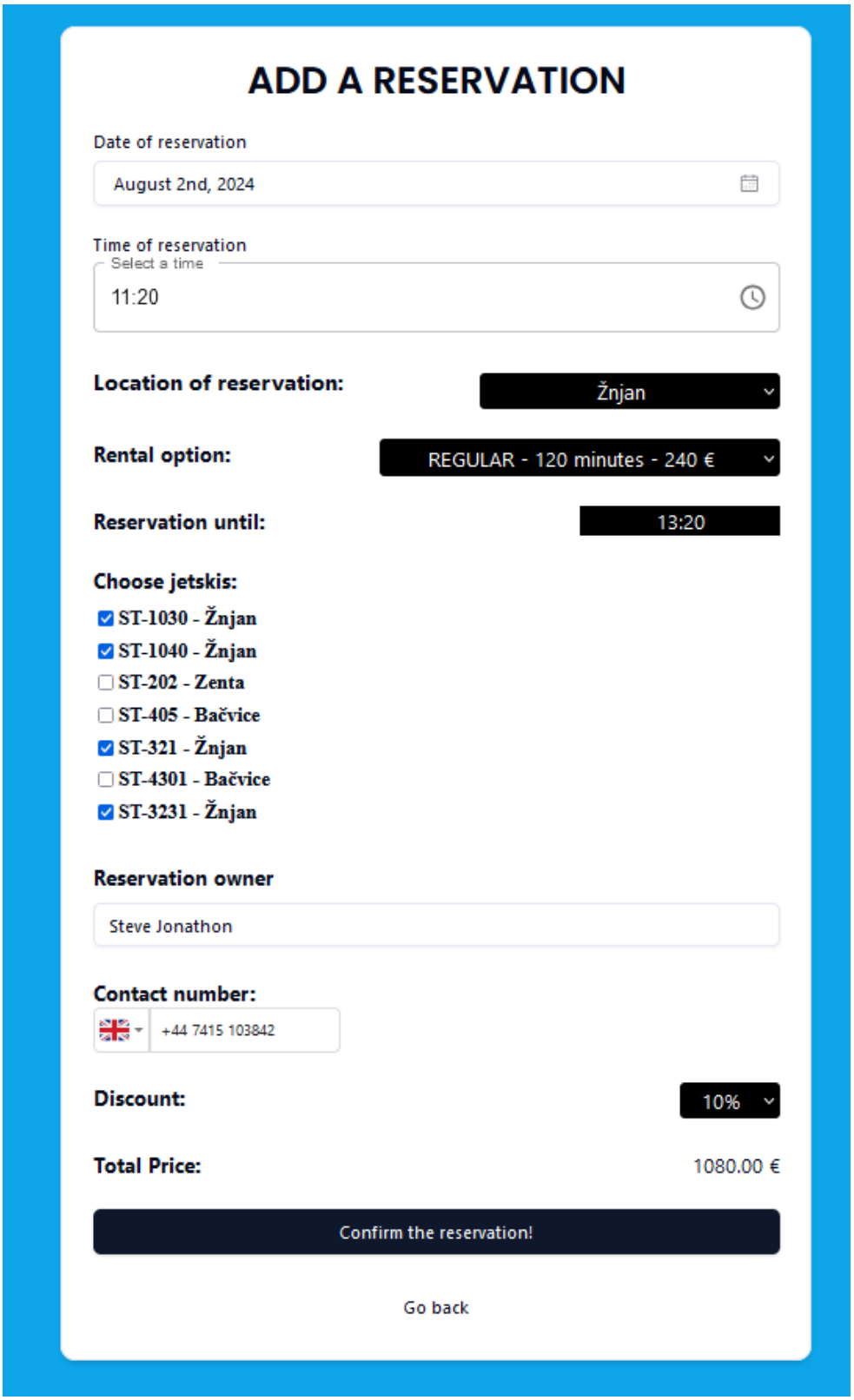
Korisnik prvo mora odabrati datum rezervacije. Izbornik datuma rezervacije je padajući izbornik. Onemogućen je izbor datuma koji je već prošao kako bi se izbjeglo stvaranje rezervacija u prošlosti. Nakon što korisnik odabere datum, omogućeno mu je

postaviti vrijeme rezervacije. Vrijeme rezervacije se unosi putem tipkovnice ili pritiskom na ikonicu sata, koja otvara padajući izbornik na kojem se odabire vrijeme. Ako je korisnik odabrao današnji datum i unio vrijeme koje je već prošlo, rezervacija se neće moći spremiti. Potom korisnik odabire lokaciju rezervacije, ovisno o zahtjevu gosta. Nakon što je odabrao lokaciju, korisnik može otvoriti padajući izbornik za opciju najma. Ovisno o korisnikovu izboru prikazat će se dostupna jetski plovila. Ako je odabrana opcija najma safari, korisnik mora odabrati minimalno dva jetski plovila. Dakle, nakon što korisnik odabere datum, vrijeme, lokaciju i opciju najma prikazat će se dostupna jetski plovila koja se mogu odabrati klikom u mali kvadratić uz njihovu registraciju. Prikaz odabiranja jetski plovila prikazan je na slici 25.



Slika 25: Izbornik dostupnih jetskija

Ako korisnik nakon što je već odabrao plovila odluči promijeniti jednu od unesenih stavki, pokrenut će se ponovno dohvaćanje dostupnih plovila. Korisnik mora upisati na čije ime glasi rezervacija te kontakt broj gosta. Kontakt broj se provjerava preko `npm` (`node package manager`) paketa koji provjerava postoji li taj broj. Konačno, korisnik može postaviti popust na cjelokupnu cijenu. Jetski plovila nisu ograničena lokacijom, odnosno unatoč njihovoj zadanoj lokaciji mogu se koristiti u rezervacijama na drugim lokacijama. Lokacija jetski plovila primarno služi da se prikaže lokaciji na kojoj jetski plovilo mora biti vezano. Korisnik pritiskom na tipku `Confirm the reservation` pokreće provjeru podataka na poslužiteljskoj strani. Nova rezervacija će se stvoriti ako se svi podatci verificiraju. Forma za dodavanje nove rezervacije prikazana je na slici 26. Kôd za provjeru valjanosti podataka rezervacije prikazan je u ispisu 16.



Slika 26: Popunjena forma za stvaranje rezervacija

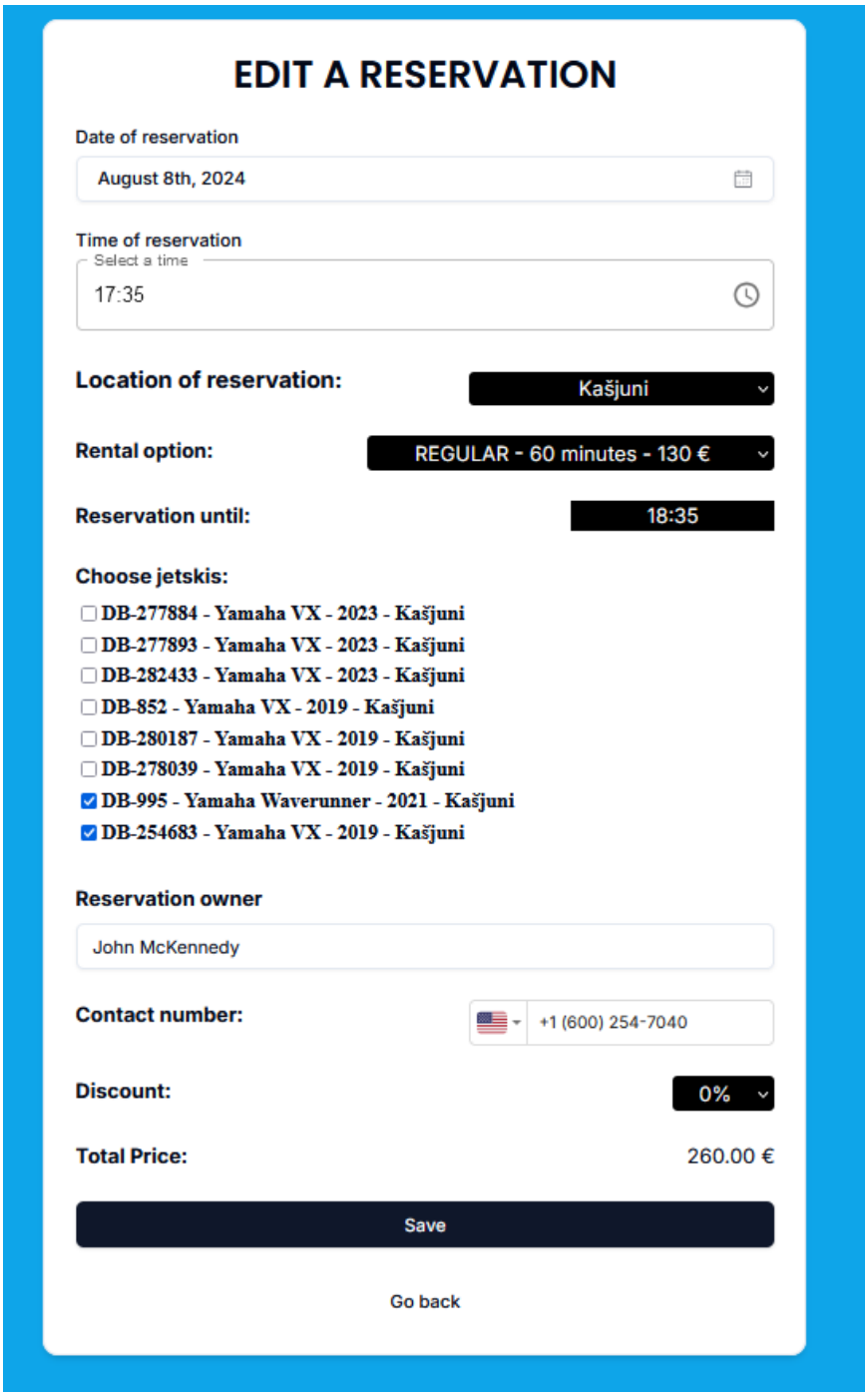
```
    if(rentalOption?.rentaloption_description === "SAFARI" &&
reservation_jetski_list.length < 2)
    {
        return {error: "Safari tour needs minimum two jetskis. One for guide
and one for the guest."};
    }

    if(rentalOption?.rentaloption_description === "REGULAR" &&
reservation_jetski_list.length < 1)
        {
            return {error: "Regular tour needs minimum one jetski."};
        }

    if (startDateTime < now) {
        return { error: "You have selected a starting time that has already
passed!" };
    }

    if (startDateTime >= endDateTime) {
        return { error: "End time must be after start time." };
    }

    const locationExists = await db.location.findUnique({
        where: { location_id: reservation_location_id },
    });
    if (!locationExists) {
        return { error: "The specified location does not exist." };
    }

    const jetskiAvailabilityChecks = reservation_jetski_list.map(async jetski
=> {
        const reservations = await db.reservation.findMany({
            where: {
                reservation_jetski_list: {
                    some: {
                        jetski_id: jetski.jetski_id,
                    },
                },
                NOT: [
                    { endTime: { lte: newStartTime } },
                    { startTime: { gte: endTime } },
                ],
            },
        });
        return reservations
    });

    const results = await Promise.all(jetskiAvailabilityChecks);
    if (results.some(isAvailable => !isAvailable)) {
        return { error: "One or more jet skis are not available in the chosen
time slot." };
    }
};
```

Ispis 16: Provjere poslužiteljske strane prilikom stvaranja rezervacija

Nakon što je rezervacija stvorena, prikazat će se na popisu rezervacija. Na popisu rezervacija se nalazi više tipki. Omogućeno je sortiranje po svim stupcima. Tipka u gornjem lijevom kutu je padajući kalendar s pomoću kojeg korisnik odabire datum za koji želi prikazati rezervacije. Korisnik može odabrati datum u prošlosti. Ispod te tipke nalazi se tipka `Copy to Clipboard` koja omogućuje korisniku kopiranje svih rezervacija u tekstualnom obliku. Tipka `Save as` omogućuje spremanje podataka o rezervacijama u tekstualnu datoteku. S desne strane se nalaze dvije tipke koje služe za filtriranje rezervacija po lokaciji ili po jetskiju. Upravitelj ili administrator može pokrenuti rezervaciju pritiskom na tipku `Start` te završiti rezervaciju pritiskom na tipku `End`. Ako je rezervacija pokrenuta, njen status će se promijeniti te neće biti moguće uređivanje ni brisanje rezervacije dok god ona traje. Prije nego što rezervacija počne ili nakon što rezervacija završi, korisnik je može izbrisati iz baze podataka. Prikaz popisa rezervacija se nalazi na slici 27.

Slika 27: Prikaz rasporeda rezervacija

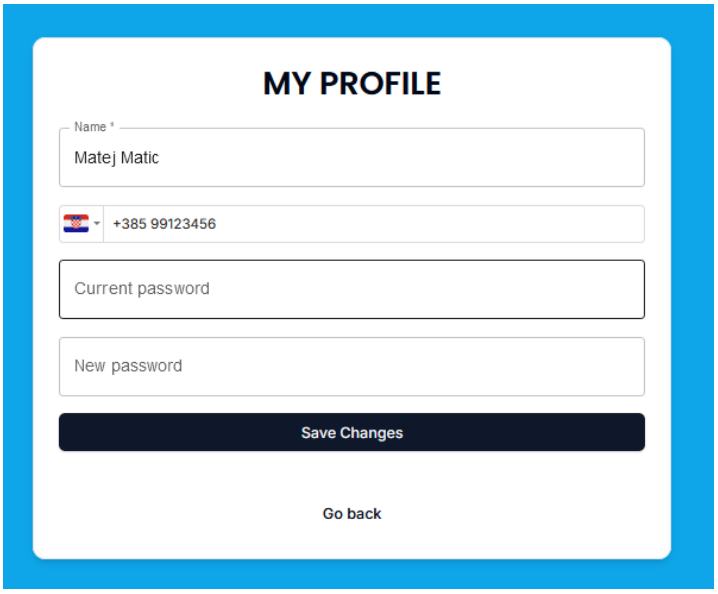
Korisnik može odlučiti urediti rezervaciju pritiskom na zelenu tipku s ikonom olovke koja će otvoriti formu ispunjenu podatcima te rezervacije. Može promijeniti bilo koji podatak o rezervaciji. Ako određeni podatak ne zadovoljava postavljene uvjete prikazat će se poruka s opisom pogreške koja se dogodila. Na slici 28 prikazana je forma za uređivanje rezervacija.



Slika 28: Prikaz forme uređivanja rezervacije

5.7. Korisničke postavke

U postavkama, korisnik može promijeniti svoje ime, kontakt broj te trenutnu zaporku. Podatci se mogu promijeniti neovisno jedno o drugima. To je moguće zahvaljujući TypeScriptu koji omogućuje da bilo koji tip podataka budu opcionalan pri unosu. Ako se unesu zaporke, izvršit će se provjera trenutne. Kada se trenutna zaporka podudara s onom u sustavu, pokrenut će se postavljanje nove lozinke. Ako sve provjere prolaze, nova zaporka će se hashirati. Uneseni kontakt broj mora biti jedinstven. Ime korisnika može se preklapati s imenom drugog korisnika. Pritiskom na tipku `Save Changes` promjene se spremaju u bazu podataka. Izgled forme za postavke prikazan je na slici 29.



Slika 29: Prikaz forme postavki

6. Zaključak

Digitalizacija je obilježje suvremenog doba. Cilj svake tvrtke je unaprijediti svoju uslugu. Tvrtke korisne razne aplikacije kako bi ubrzale ili automatizirale pojedine dijelove poslovanja. Ovaj projekt za cilj ima automatizirati raspored dostupnosti plovila i pružiti platformu za praćenje najmova osiguravajući operateru brzinu i pouzdanost. Djelomična automatizacija se postiže kroz korisnički unos podataka, dok aplikacija garantira točnost podataka kroz razne provjere. Aplikacija pruža kalendar koji se stvara na temelju unesenih podataka.

Tijekom izrade aplikacije korištene su tehnologije Next.js, TypeScript, React.js, Prisma i Tailwind CSS. Pomoću njih stvorena je dinamička i funkcionalna aplikacija koja ima daljnji prostor za razvoj na temelju zahtjeva i potreba klijenata. Aplikacija se vrlo jednostavno može prilagoditi za ostala plovila, pa čak i vozila, ako je to potrebno. Moguće je implementirati sustav koji će automatski stvarati rezervacije u aplikaciji na temelju podataka koje povlači s vanjskih prodajnih platformi poput TripAdvisora ili CheckYeti. Osim toga, jednostavno se može implementirati novi tip korisnika, agent, koji bi mogao samo stvarati rezervacije.

Kako bi unaprijedili aplikaciju moguće je optimizirati kôd, npr. unutar baze podataka za tablicu `Reservations` može se postaviti atribut koji će sadržavati status rezervacije kako bi se smanjio broj potrebnih atributa, odnosno izbrisati `isCurrentlyRunning` i `hasFinished`. Također, može se aplikaciju napraviti prilagodljivom, kako bi se mogla koristiti i na mobilnim uređajima. Može se implementirati nova tablica s postavkama vlasnika tvrtke koja bi omogućila prilagođavanje početnog vremena poslovanja kao i konfiguriranje uloga.

Najveći izazov tijekom razvoja projekta se odnosio na implementaciju sigurnosnih mehanizama. Aplikacija sadrži ključne poslovne informacije i stoga je osiguranje podataka od neovlaštenog pristupa od iznimne važnosti. Također, potrebno je ograničiti mogućnosti običnog radnika kako bi se spriječilo stvaranje problema za upravitelje i administratore. U pogledu funkcionalnosti, najveći izazovi bili su skalabilnost i provjera dostupnosti, kao i rad s vremenskim zonama. Budući da poslužitelj i klijent mogu biti u različitim vremenskim zonama, važno je osigurati da korisnik odabere ispravno vrijeme kako bi se izbjegli problemi

s dostupnošću rezervacija. Za kraj se može reći da je projekt uspješno izveden te da se može koristiti u stvarnom svijetu.

7. Literatura

[1] McCue I., *„The History of ERP“*, https://www.netsuite.com/portal/resource/articles/erp/erp-history.shtml (posjećeno 29.8.2024)

[2] *„Next.js Tutorial“,* https://www.geeksforgeeks.org/nextjs/ (posjećeno 2.7.2024)

[3] *„Što je Server-Side Rendering (SSR)?“*, https://loadfocus.com/hr-hr/glossary/what-is-server-side-rendering-ssr (posjećeno 2.7.2024)

[4] *„React“,* https://programiranje.com.hr/React (posjećeno 3.7.2024)

[5] *„React Community*“, https://react.dev/community (posjećeno 4.7.2024)

[6] Kazmi A., *„All React hooks in one short.“,* https://medium.com/@AbidKazmi/all-react-hooks-in-one-short-4b0ed4b5a6e4 (posjećeno 4.7.2024)

[7] *„TypeScript docs“*, https://www.typescriptlang.org/docs/ (posjećeno 3.7.2024)

[8] *„Zod“,* https://zod.dev/ (posjećeno 7.8.2024)

[9] *„What is Prisma ORM?“*, https://www.prisma.io/docs/orm/overview/introduction/what-is-prisma (posjećeno 5.7.2024)