Web aplikacija za rezerviranje športskih objekta

Boras, Marko

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA

Sveučilišni studij

Web aplikacija za rezerviranje športskih objekata

Diplomski rad

Marko Boras

Osijek, 2022.

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

Obrazac D1: Obrazac za imenovanje Povjerenstva za diplomski ispit

Osijek, 05.09.2022.

Odboru za završne i diplomske ispite

Imenovanje Povjerenstva za diplomski ispit

Ime i prezime Pristupnika:	Marko Boras
Studij, smjer:	Diplomski sveučilišni studij Računarstvo
Mat. br. Pristupnika, godina upisa:	D-1107R, 13.10.2020.
OIB studenta:	55817592962
Mentor:	Izv. prof. dr. sc. Ivica Lukić
Sumentor:	,
Sumentor iz tvrtke:	
Predsjednik Povjerenstva:	Izv. prof. dr. sc. Mirko Köhler
Član Povjerenstva 1:	Izv. prof. dr. sc. Ivica Lukić
Član Povjerenstva 2:	Izv.prof.dr.sc. Zdravko Krpić
Naslov diplomskog rada:	Web aplikacija za rezerviranje športskih objekta
Znanstvena grana diplomskog rada:	**Informacijski sustavi (zn. polje računarstvo)**
Zadatak diplomskog rada:	U diplomskom radu je potrebno omogućiti registraciju i prijavu korisnika. Prijavljeni korisnici koji posjeduju sportski objekt imaju mogućnost kreiranja i uređivanja vlastitih objekta. Korisnici putem interaktivnog web sučelja mogu pretraživati sportske objekte i rezervirati termin čime vlasniku sportskog objekta dolazi obavijest. Unutar web aplikacije se nalazi chat putem kojega korisnici i vlasnici sportskih objekata mogu komunicirati i izmjenjivati poruke. Svaki registirani korisnik može uređivati svoj korisnički profil. Za pretragu sportskih objekata će postojati pretraga i filtriranje. Diplomski rad će se izraditi koristeći HTML, CSS, React(Typescript), Firebase. Rezervirano za: Marko boras
Prijedlog ocjene pismenog dijela ispita (diplomskog rada):	Izvrstan (5)
Kratko obrazloženje ocjene prema Kriterijima za ocjenjivanje završnih i diplomskih radova:	Primjena znanja stečenih na fakultetu: 3 bod/boda Postignuti rezultati u odnosu na složenost zadatka: 2 bod/boda Jasnoća pismenog izražavanja: 3 bod/boda Razina samostalnosti: 3 razina
Datum prijedloga ocjene od strane mentora:	05.09.2022.

Potvrda mentora o predaji konačne verzije rada:	*Mentor elektronički potpisao predaju konačne verzije.* *Datum*

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

IZJAVA O ORIGINALNOSTI RADA

Osijek, 21.09.2022.

Ime i prezime studenta:	Marko Boras
Studij:	Diplomski sveučilišni studij Računarstvo
Mat. br. studenta, godina upisa:	D-1107R, 13.10.2020.
Turnitin podudaranje [%]:	8

Ovom izjavom izjavljujem da je rad pod nazivom**: Web aplikacija za rezerviranje športskih objekta**

izrađen pod vodstvom mentora Izv. prof. dr. sc. Ivica Lukić

i sumentora ,

moj vlastiti rad i prema mom najboljem znanju ne sadrži prethodno objavljene ili neobjavljene pisane materijale drugih osoba, osim onih koji su izričito priznati navođenjem literature i drugih izvora informacija.
Izjavljujem da je intelektualni sadržaj navedenog rada proizvod mog vlastitog rada, osim u onom dijelu za koji mi je bila potrebna pomoć mentora, sumentora i drugih osoba, a što je izričito navedeno u radu.

Potpis studenta:

Sadržaj

1. UVOD 1
1.1. Zadatak završnog rada 1
2. POSTOJEĆA RJEŠENJA 2
2.1. Playfinder 2
2.2. Pitchbooking 2
2.3. Online Gym Time 3
2.4. Perfect Gym 3
2.5. Bookteq 4
3. KORIŠTENE TEHNOLOGIJE I ALATI 5
3.1. Visual Studio Code 5
3.2. React 6
3.3. TypeScript 6
3.4. Material UI 7
3.5. Firebase 7
3.6. Recoil 8
3.7. Git 9
3.8. Github 10
4. PROGRAMSKO RJEŠENJE 11
4.1. Opis aplikacije 11
4.2. Postavljanje projekta 11
4.3. Reach router 15
4.4. Implementacija Firebase 17
4.6. Konfiguracija Recoila 23
4.7. Moduli 24
5. PRIKAZ NAČINA KORIŠTENJA APLIKACIJE 29

5.1. Prikaz korištenja aplikacije 29
5.2. Prikaz baze podataka Firestore 44
6. ZAKLJUČAK 46
LITERATURA 47
SAŽETAK 48
ABSTRACT 49
ŽIVOTOPIS 50
PRILOZI 51

1. UVOD

Tema diplomskog rada je izrada web aplikacije za rezerviranje športskih objekata. U današnje vrijeme je nezamisliv život bez računala, mobitela. Internet kao masovno sredstvo za prijenos informacija omogućava korisnicima pristup informacijama bilo kada i bilo gdje. U tom smislu je internet moćan marketinški alat. Cilj ovog diplomskog rada napraviti web aplikaciju na kojoj će korisnici imati na jednom mjestu sve športske objekte za različite sportove i lokacije. Korisnici tako ako se nađu u drugome gradu ili državi mogu na jako jednostavan i brz način pronaći športski objekt koji ih zanima. Osim toga svrha ovoga rada je povećati sportsku aktivnost mladih ljudi zbog današnjeg sjedilačkog načina života. Vlasnicima športskih dvorana je u interesu povećati šansu iznajmljivanja svog objekta, pa zbog toga se reklamiraju na više različitih mjesta. Korisnici će na web aplikaciji imati mogućnost iznajmljivanja športskih objekata, kontaktiranja vlasnika športskih objekata, te vide povijest svojih rezervacija.

Web aplikacija u ovome diplomskom radu bit će izrađena koristeći React. React je Javascript biblioteka održavana od strane Facebook-a i zajednice otvorenih developera. Aplikacija je pisana u Typescript programskom jeziku koji je superset Javascripta. Typescript je razvijen od strane Microsoft-a. Programsko okruženje u kojem je razvijena aplikacija se naziva Visual Studio Code razvijen od strane Microsoft-a. React omogućava rastavljanje složenog korisničkog sučelja na manje dijelove čime je olakšan razvoj, no i performanse same aplikacije su mnogo bolje. Serverski dio aplikacije i baza podataka će biti izrađen koristeći Google-ova platforma Firebase koja radi na principu BaaS (engl. *Backend-as-a-Service*).

1.1. Zadatak završnog rada

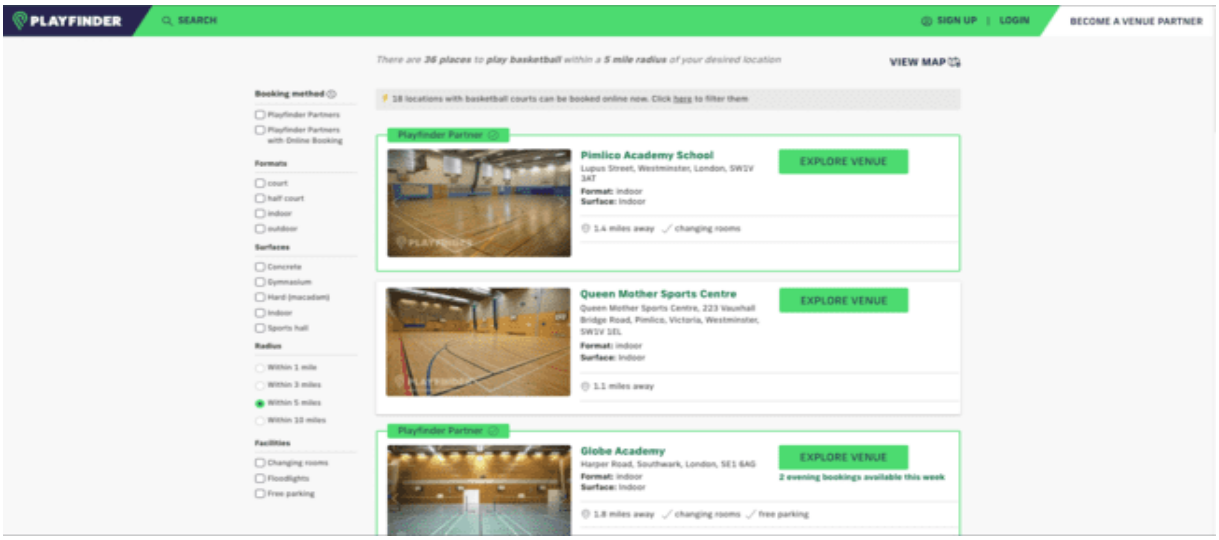
Zadatak završnog rada je putem biblioteke React izraditi web aplikacija za rezerviranje športskih objekata koja korisnicima omogućava jednostavno pretraživanje i iznajmljivanje športskih objekata. Informacije o športskim dvoranama bih unosili vlasnici športskih dvorana kojima je u interesu da se njihov objekt reklamira na što više web lokacija. Unutar aplikacije korisnik treba imati mogućnost komuniciranja s vlasnikom dvorane. Pri stvaranju upita korisnik iz kalendara odabire željeni termin, čime vlasnik športskog objekta dobiva obavijest. Potvrdom obavijesti otvara se chat između korisnika i vlasnike športske dvorane, ako korisnik ima pitanja. U kalendaru je potrebno korisniku vizualno dati informaciju kada može iznajmiti termin.

2. POSTOJEĆA RJEŠENJA

U današnje vrijeme postoji raznolik broj internetskih stranica na kojima je moguće izvršiti rezervaciju športskih termina, no u Republici Hrvatskoj još ne postoji.

2.1. Playfinder

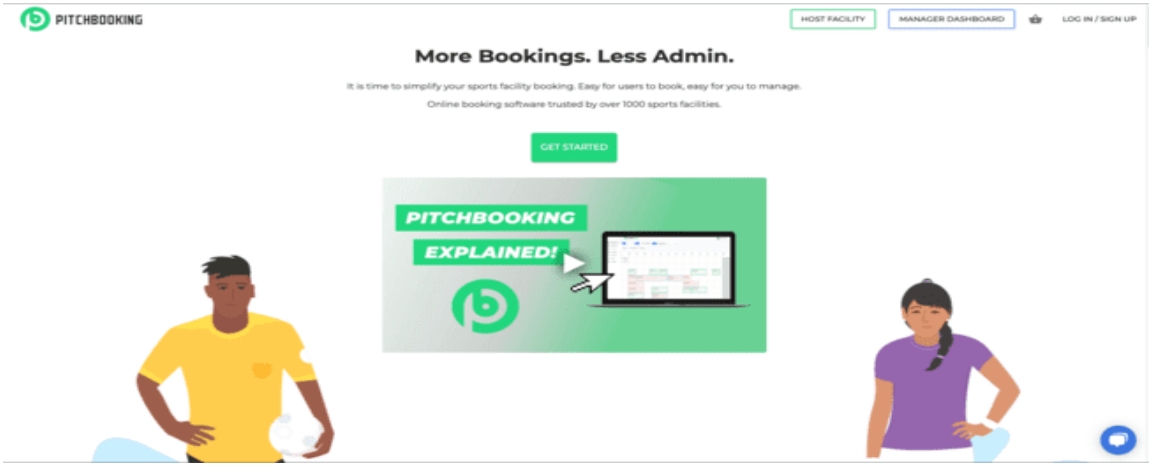
Playfinder [1] je aplikacija dostupna u Ujedinjenom Kraljevstvu gdje korisnici mogu na jednostavan i brz način rezervirati športski objekt. Korisničko sučelje aplikacije je prikazano na slici 2.1.



Slika 2.1. Playfinder korisničko sučelje

2.2. Pitchbooking

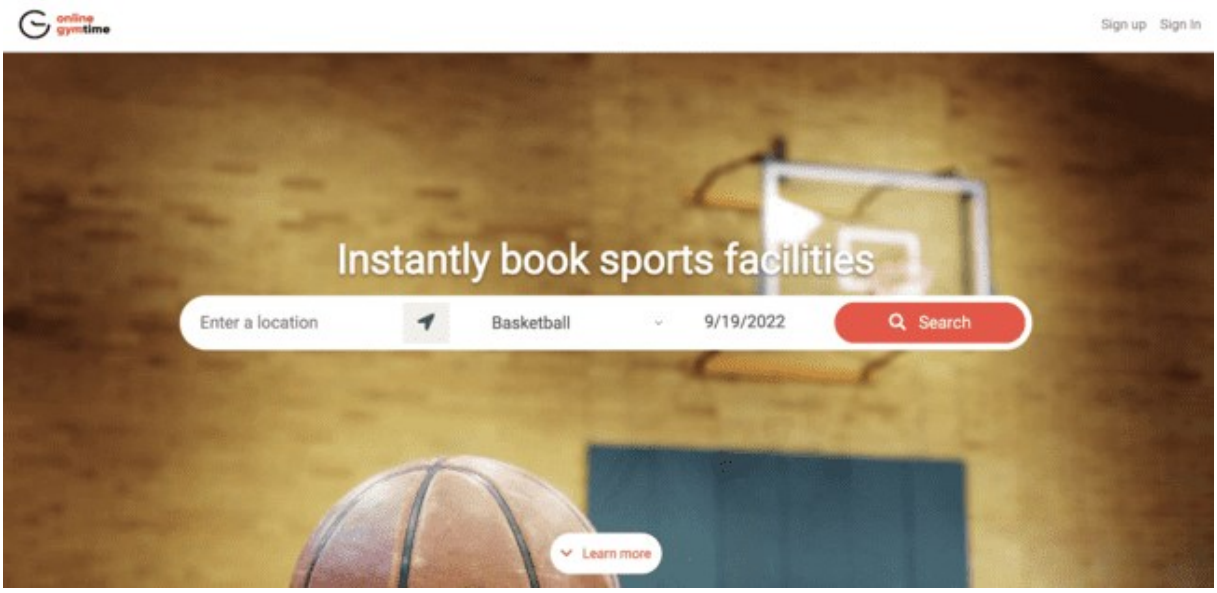
Pitchbooking [2] je web aplikacija za rezerviranje športskih objekata u Ujedinjenom Kraljevstvu i Irskoj. Korisničko sučelje je prikazano na Slici 2.2.



Slika 2.2. Pitchbooking korisničko sučelje

2.3. Online Gym Time

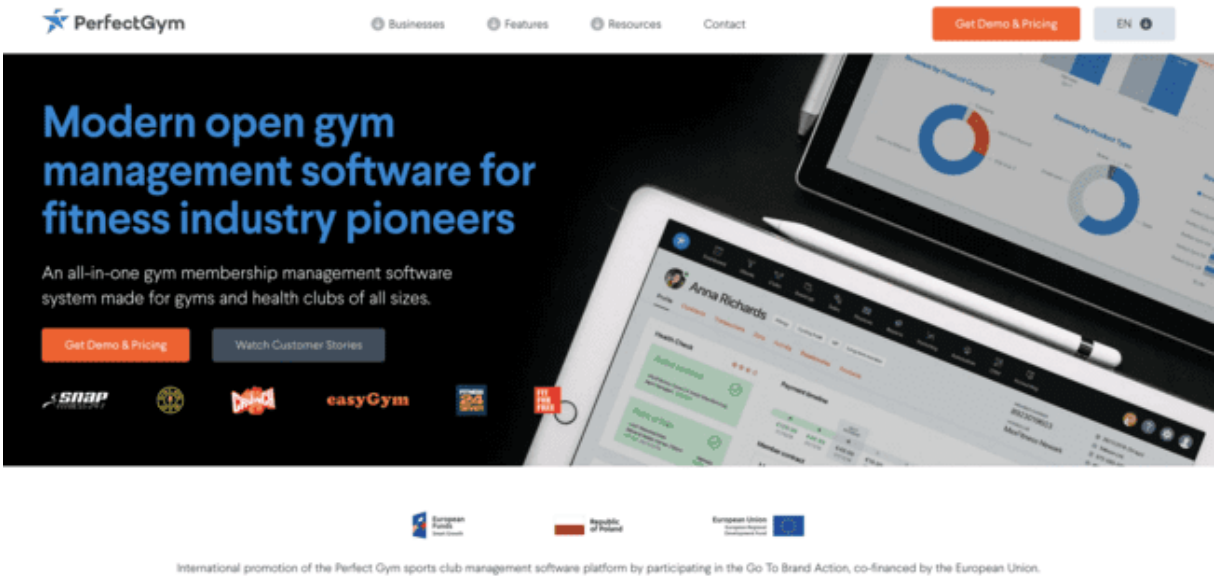
Online Gym Time je web aplikacija koja pojednostavljuje korisnicima i fitness trenerima rezervaciju treninga unutar fitness centara. Korisničko sučelje je prikazano na slici 2.3.



Slika 2.3. Online Gym Time korisničko sučelje

2.4. Perfect Gym

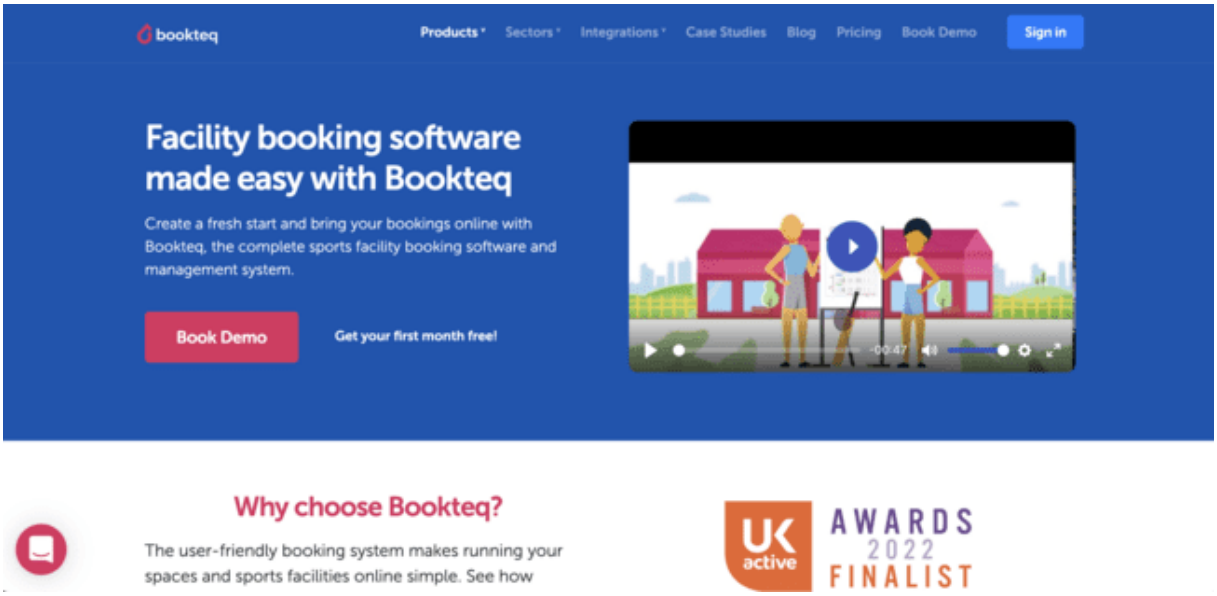
Perfect Gym je web i mobilna aplikacija za rezerviranje fitness centara, trampoline parkova, yoga studia i borilačkih vještina u Poljskoj. Korisničko sučelje je prikazano na slici 2.4.



Slika 2.4. Perfect Gym korisničko sučelje

2.5. Bookteq

Bookteq je web aplikacija dostupna u Ujedinjenom Kraljevstvu i Irskoj koja olakšava vlasnicima športskih objekata upravljanje istih. Korisničko sučelje prikazano je na slici 2.5.

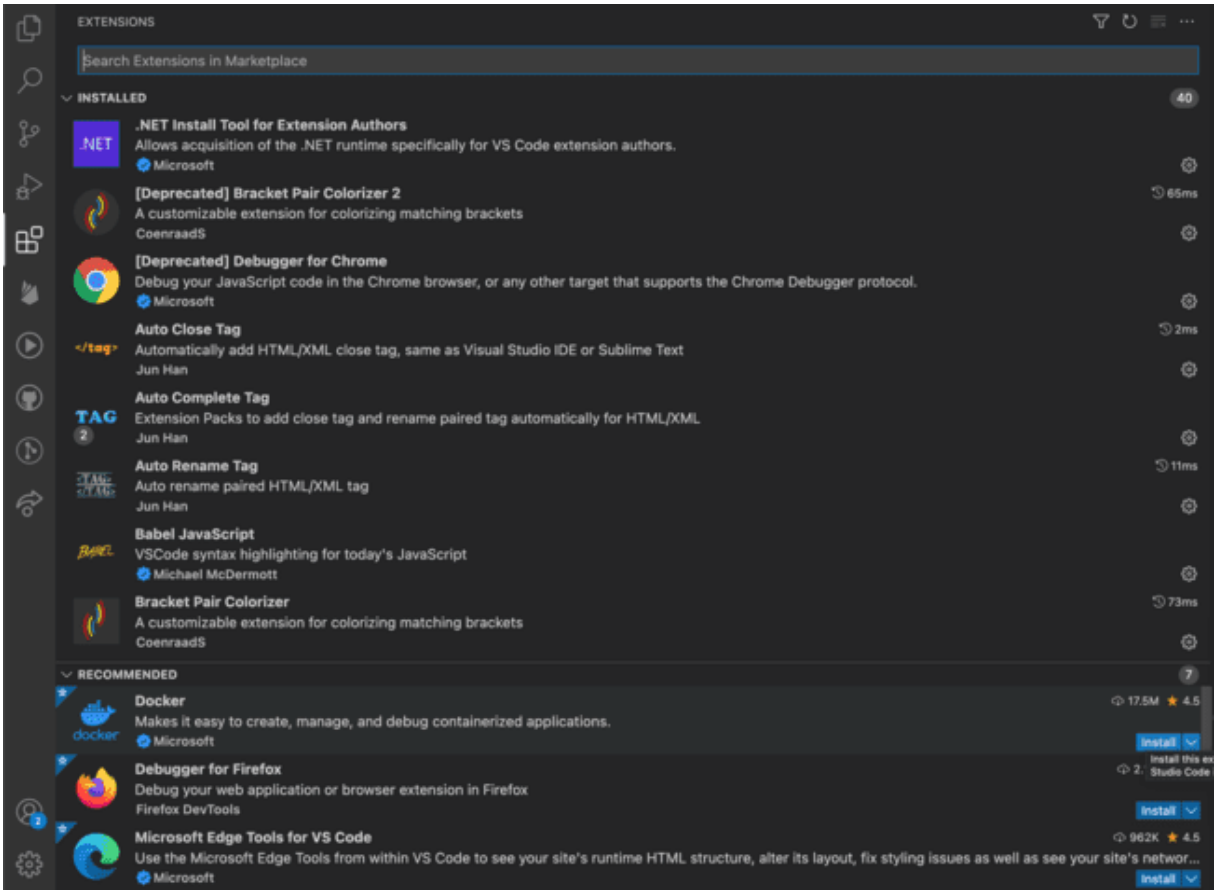


Slika 2.5. Bookteq korisničko sučelje

3. KORIŠTENE TEHNOLOGIJE I ALATI

3.1. Visual Studio Code

Visual Studio Code ili VS Code je besplatni Microsoftov uređivač koda izrađen 2005. godine. Podržava razne programske jezike uključujući Python, Javu, C++, TypeScript i dr. Također moguće je instalirati VS Code na različitim operacijskim sustavima kao npr. Windows, MacOS i Linux. Značajke koje omogućava VS Code su IntelliSense što pojednostavljuje dovršavanje koda, debugiranje, integrirani terminal i integrirani Git, intuitivne prečace na tipkovnici koji značajno pomažu pri razvoju. Naprednije značajke su interaktivni debugger pomoću kojeg se može koračati kroz kod, pregledavati varijable, skupove poziva. Kako bi developeri imali što ugodnije iskustvo rada, VS Code pruža mogućnost developerima izrade i korištenja ekstenzija. Ekstenzije se mogu pretraživati, instalirati koristeći Visual Studio Code Marketplace prikazanog na slici 3.1. Na StackOverflow anketi 2021. godine VS Code je rangiran kao najpopularniji uređivač koda sa 71.06% [3].



Slika 3.1. Visual Studio Code ekstenzije

3.2. React

React ili ReactJS je besplatna front-end JavaScript biblioteka otvorenog koda za izgradnju korisničkih sučelja temeljenih na komponentama korisničkog sučelja [4]. Facebook je stvorio React 2011. godine čime je stvorio novi presedan za razvoj brzih i dinamičnih web stranica, te ima na GitHub repozitoriju 164000 zvjezdica što ga čini top pet repozitorija na GitHubu. Glavna primjena React-a je izgradnja jednostraničnih aplikacija (engl. *Single Page Application/SPA*) gdje prevodi napisani kod u DOM. Komponente su napisane pomoću JavaScript XML (JSX) sintakse. React je biblioteka, ne okvir. Glavna prednost korištenja biblioteka je to što su biblioteke veličinom jako male i developer ima veliku slobodu odabira različitih biblioteka za različite probleme. Koristeći React izrada interaktivnih kompleksnih korisničkih sučelja je jednostavna, a temelji se na komponentama koje imaju vlastito stanje. Kombinacijom više komponenti i njihovih stanja se slaže složenije korisničko sučelje. React kod je deklarativan što čini razvoj jednostavnijim, kod predvidljivijim i lakšim za otklanjanje grešaka. Logika samih komponenti je napisana u JavaScript programskom jeziku čime je jako jednostavno proslijediti podatke. React se može renderirati na poslužitelju (engl. *server*) koristeći Node i pokretati mobilne aplikacije koristeći React Native. React v16.8 donio je značajnu promjenu izgradnje web aplikacija gdje je dodan Hooks API pomoću kojega se logika koja upravlja stanjima nalazi van komponente čime se pojednostavljuje neovisno testiranje i moguća ponovna iskorištenost [5].

3.3. TypeScript

TypeScript je programski jezik otvorenog koda koji je prvi razvio Microsoft 2012. godine kao superset JavaScript programskog jezika. Svaki validan programski kod napisan u JavaScriptu je validan u TypeScriptu. TypeScript kod se transpila u JavaScript, te dodaje dodatnu sintaksu kako bi pojačao integraciju s uređivačem koda. Transpajliranje je process gdje kod napisan u jednom programskom jeziku se konvertira u ekvivalent koda u drugom programskom jeziku. Prednost koju TypeScript pruža u odnosu na JavaScriptu je neobavezno statičko tipkanje što inicijalno usporava razvoj, no znatno pomaže u otkrivanju pogrešaka i debugiranju. Osim toga poboljšava se čitljivost koda, IDE podrška, podrška zajednice, tipiziranje varijabli i funkcija, preopterećivanje funkcija što znači da u kodu može postojati više funkcija s istim imenom, a različitim parametrima. Tipiziranje varijabli i funkcija znatno pomaže u suglasnosti frontend i backend koda gdje developer znaju koji tip podatka mogu očekivati.

3.4. Material UI

Material UI ili MUI je biblioteka React komponenti koja implementira Googleov Material dizajn [6]. Material UI pruža sveobuhvatan paket alata korisničkog sučelja koji znatno pomažu u izradi korisničkog sučelja.

3.5. Firebase

Firebase je platforma koju je razvio Google za razvoj web i mobilnih aplikacija koju koriste milijuni tvrtki diljem svijeta. Firebase je *Backend-as-a-Service* (*BaaS*) što pojednostavljeno znači da je Firebase server, API, baza podataka te sadrži skup alata koji omogućavaju različite usluge koje se nalaze u oblaku i skaliraju se po potrebi korisnika. Firebase usluge se nalaze na slici 3.2.

Slika 3.2. Visual Studio Code ekstenzije

3.5.1. Firebase Authentication

Firebase Authentication pruža end-to-end sigurnosni sustav za autentifikaciju korisnika putem e-maila i zaporke, telefona, Googlea, Facebooka, GitHuba i mnogih drugih providera. Developer ima na raspolaganju razne usluge, *SDK-ove* jednostavne za korištenje za autentifikaciju korisnika u aplikaciji.

3.5.2. Cloud Firestore

Cloud Firestore je NoSQL baza podataka dokumenata izgrađena za automatsko skaliranje, visoku izvedbu koja omogućava jednostavno pohranjivanje, sinkronizaciju i upite podataka za web i mobilne aplikacije. Glavne značajke Firestora su kolekcije i dokumenti. Jedna kolekcija sadrži jedan ili više dokumenata, a svaki dokument može imati potkolekcije. Tako developer

pomoću upita (engl. *queries*) dohvaća podatke čime se rješava problem skaliranja. Firestore isporučuje mobilni i web SDK (engl. *Software Development Kit*), skup sigurnosnih pravila kako bih mogli zaštiti vaše podatke, komunikaciju u stvarnom vremenu putem web socketa ili listenera, izvanmrežnu podršku za mobilne i web uređaje što omogućava rad aplikacija bez obzira na latenciju mreže ili internetsku povezanost. Bazi podataka osim s klijentske strane se može pristupiti koristeći cloud funkcije koje su spomenute u poglavlju 3.5.3.

3.5.3. Cloud Functions

Cloud funkcije su okvir bez poslužitelja koji omogućava pisanje backend koda koji se pokreće putem Firebase događaja ili HTTP zahtjeva. Funkcije napisane u JavaScript ili TypeScript programskom jeziku su pohranjene u Googleovom oblaku gdje rade u sigurnom okruženju. Funkcije se pohranjuju putem Firebase CLI-ja (engl. *Command Line Interface*) gdje su sigurno pohranjene i udaljene od klijentskog koda što omogućava konzistentan rad. Svaka funkcija radi izolirano u vlastitom okruženju s vlastitom konfiguracijom.

3.5.4. Cloud Storage

Cloud Storage omogućava pohranjivanje i posluživanje multimedijskog sadržaja poput fotografija, videozapisa. Firebase SDK osigurava sigurni prijenos i preuzimanje datoteka bez obzira na kvalitetu mreže.

3.6. Recoil

Recoil je biblioteka za upravljanje stanjem za React (engl. *state management*) [7]. Razlog zbog kojeg je potrebna biblioteka za upravljanjem stanja je kada bi se stanje komponente ručno prosljeđivalo kroz stablo nastao bih kaos, a zove se prop-drilling. Recoil omogućava API gdje dijeljeno stanje ima isto jednostavno sučelje kao React lokalno stanje. Osnovni elementi Recoila su atom i selector. Atom je jedinica stanja koja se može ažurirati i na koju se komponente mogu pretplatiti (engl. *subscribe*). Selector je čista funkcija (engl. *pure function*) koja prihvaća atome ili selectore kao inpute te vrši transformacije nad atomima sinkrono ili asinkrono. Pomoću Recoila se kreira graf protoka koji teče od atoma (zajedničkog stanja) preko selektora (čistih funkcija) u React komponente. Kada se atom promijeni, svaka pretplaćena komponenta se re-renderira s novom vrijednosti. Ako se jedan atom koristi u više različitih komponenti, sve komponente imaju isto stanje. Svaki atom zahtijeva jedinstveni identifikator key i zadanu vrijednost (engl. *default value*).

```
const counterState = atom({
  key: 'counterState',
  default: 0,
});
```

Slika 3.3. Primjer jednostavnog atoma

Za čitanje i pisanje stanja atoma iz komponente koristi se hook *useRecoilState* koji ima API kao Reactov *useState.*

```
const CounterButton: React.FC = () => {
  const [counter, setCounter] = useRecoilState(counterState);
  return (
    <button
      onClick={() =>
        setCounter(currentCounterValue => currentCounterValue + 1)
      }
    >
      Increase counter
    </button>
  );
};
```

Slika 3.4. useRecoilState

Selectori se koriste za izračunavanje izvedenih podataka koji se temelje na stanju atoma čime se izbjegava redudantno stanje jer minimalni skup stanja je pohranjen u atomima, dok sve ostalo se izračunava putem selectora. Selectori su definirani putem *selector* funkcije.

```
const doubleCounterState = selector({
  key: 'doubleCounterState',
  get: ({ get }) => {
    const counter = get(counterState);

    return counter * 2;
  },
});
```

Slika 3.5. selector

Stanje selectora se može čitati koristeći *useRecoilValue(selectorName)*.

3.7. Git

Git je besplatan softver otvorenog koda namjenjen za verzioniranje koda koji prati promjene u datotekama tako da developer može pratiti promjene i vratiti se na određenu verziju ako je to potrebno. Osim verzioniranja Git omogućava kolaboraciju više developera da spoje rješenja u jedan izvor. Git je napisao Linus Torvalds 2005. za razvoj Linux kernela. Git radi lokalno.

3.8. Github

Github je usluga temeljena na oblaku koja developerima pomaže u pohranjivanju i upravljanju kodom, kao i praćenju kontroli promjena koda. Nudi uslugu hostinga Git repozitorija što znatno olakšava razvoj aplikacija.

4. PROGRAMSKO RJEŠENJE

U ovome poglavlju opisuje se postupak izrade web aplikacije za rezerviranje športskih objekata, te sve funkcionalnosti koje sama aplikacija pruža.

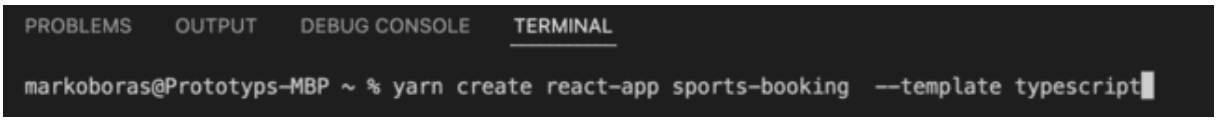
4.1. Opis aplikacije

Aplikacija omogućava vlasnicima športskih objekata kreiranje športskog objekta, a korisnicima koji se žele baviti športom omogućava rezervaciju istih športskih objekata. Kako bih vlasnici mogli kreirati objekte ili korisnici rezervirati, potrebno je imati korisnički račun. Moguće je kreirati račun putem email adrese i lozinke, Googlea ili Facebooka. Nakon što korisnik uspješno kreira račun automatski se preusmjerava na *Onboarding* gdje unosi osobne podatke poput imena, adrese i kontakta. U slučaju da korisnik zaboravi lozinku postoji mogućnost slanja maila pomoću kojeg korisnik može kreirati novu lozinku. Svi podaci koje unese tijekom *Onboarding* koraka se mogu ažurirati kada se korisnik potpuno logira. U slučaju pogrešnog unosa inputa korisnik dobiva kao povratnu informaciju error message ispod navedenog inputa ili putem toastbara. Nakon uspješnog logiranja korisnik koji je vlasnik športskog objekta u navigaciji ima gumb pomoću kojega ga se vodi proces kreiranja športskog objekta gdje je potrebno unijeti sve podatke za dvoranu kao npr. lokaciju, radno vrijeme, cijenu, kontakt i dr. Nakon kreiranja športskog objekta vlasnik uvijek može urediti ili obrisati objekt, te pregledavati svoje objekte. Na početnoj stranici korisnici mogu pretraživati i filtrirati športske objekte, te ponuđeni su športski objekti u njihovom gradu za rezervaciju. U slučaju da korisnik želi rezervirati termin odabirom dvorane u kalendaru bira datum i vrijeme nakon čega vlasniku športskog objekta dolazi notifikacija i zahtjev. Vlasnik športskog objekta može prihvatiti ili odbiti rezervaciju. Ako vlasnik prihvati rezervaciju kreira se chat između vlasnika i osobe koja želi rezervirati športski objekt. Vlasnik i korisnik u bilo kojem trenutku mogu otkazati rezervaciju, te pregledavati rezervacije. Chat je izveden pomoću Firestore listenera. Korisnik na svome profile ima opciju brisanja korisničkog računa. Aplikacija je responzivna i prilagođena svim uređajima te napisana u programskom jeziku TypeScript. Za izradu aplikacije su korištene najnovije značajka React-a, React Hooksa, Recoila, Firebasea, Reach Routera što će biti objašneno u sljedećim poglavljima.

4.2. Postavljanje projekta

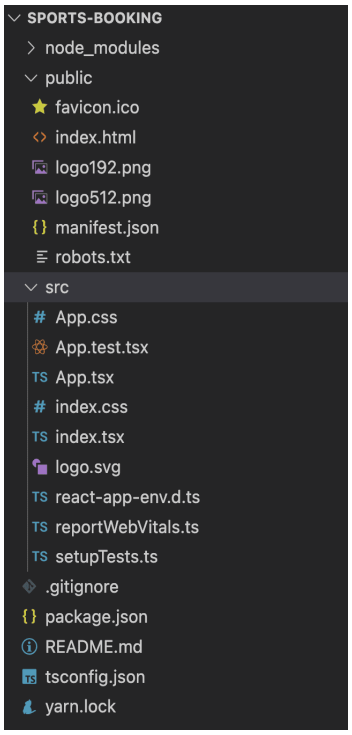
Projekt je kreiran pomoću CRA (engl. *create-react-app*) predloška koji izrađuje osnovnu React aplikaciju te ima na Githubu najviše zvjezdica u kategoriji React predložaka [8]. Instalacija CRA omogućava konfiguraciju webpacka i Babela. Za pokretanje naredbe potrebno je imati

lokalno verziju Node-a veću od 14.0.0. Naredba za kreiranje predloška se unosi unutar terminala koji se vidi na slici 4.1.



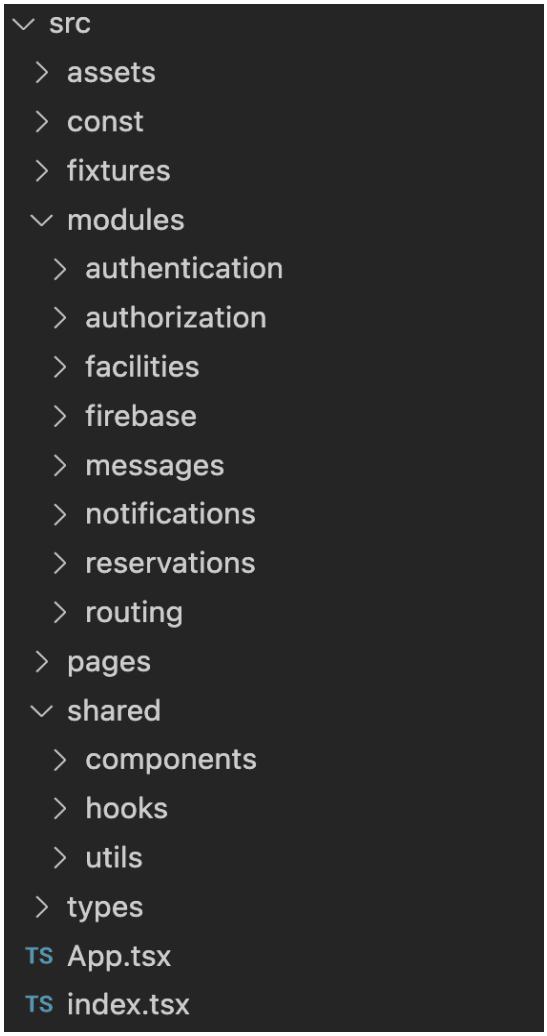
Slika 4.1. Create React App

Prilikom unosa naredbe potrebno je postaviti zastavicu *–template* kako bih projekt bio kreiran u TypeScript programskom jeziku. Struktura foldera nakon uspješnog kreiranja projekta nalazi se na slici 4.2.



Slika 4.2. Struktura foldera nakon kreiranja projekta

Za veću kvalitetu pisanja koda uz što manje grešaka potrebno je instalirati i konfigurirati Eslint, Prettier pakete uz projekt i navedene ekstenzije u VS Codu. Eslint je alat za prepoznavanje čestih sintaktičkih grešaka u kodu, a Prettier je alat za formatiranje koda kako bih kod bio što čitljiviji. Za svaki paket potrebno je postaviti konfiguraciju s ekstenzijom .js, .json ili .rc. Kako bih kvaliteta koda, performanse, standardi unutar aplikacije bili što veći potrebno je jasno definirati arhitekturu gradnje aplikacije. Za izradu ovog diplomskog rada korištena je modularna arhitektura prikazana na slici 4.3. [9].



Slika 4.3. Arhitektura aplikacije

Korijenska komponenta aplikacije je *App.tsx*, a direktorij src se sastoji od više poddirektorija koji su u nastavku objašnjeni:

- Assets direktorij – sastoji se od slika, fontova, svg datoteka
- Const direktorij – sastoji se od konstanti, poput ruta i konfiguracija
- Fixtures direktorij – sastoji se od .json datoteka korištenih za lokalno testiranje
- Modules direktorij – sastoji se od modula, tj. različitih funkcionalnosti aplikacije
- Pages direktorij – sastoji se kreiranih prikaza korisničkog sučelja
- Shared direktorij – sastoji se od ponovno iskoristivih komponenti, hookova i korisnih funkcija (engl. *utility functions*) poput regularnih izraza za validaciju i sl.

Kako bih korisnici u aplikaciji pravovremeno i ispravno dobijali povratne informacije korišten je paket *react-toastify* koji prikazuje prilagođene poruke korisniku putem toastbarova.

Implementiran je hook *useToast* za višekratnu uporabu koji omogućava prikaz različitih toastbarova ovisno o namjeni, a prikazan je na slici 4.4.

```
/**
 * Use Toast Hook
 * @name useToast
 * @description Hook that is used to display toast messages.
 */

export function useToast() {
  return {
    successToast: toast.success,
    warningToast: toast.warning,
    infoToast: toast.info,
    errorToast: toast.error,
    noConnectionToast: toast.dark,
    dissmissToast: toast.dismiss,
  };
}
```

Slika 4.4. useToast hook

Za osiguranje responzivnosti unutar aplikacije kreiran je hook *useDeviceSizes* koji putem media querya prati trenutnu širinu zaslona. Implementacija hooka je prikazana na slici 4.5.

```
/**
 * Use Device Sizes Hook
 * @name useDeviceSizes
 * @description Hook that is used to get current viewport width.
 */

export function useDeviceSizes() {
  const theme = useTheme();

  const mobile = useMediaQuery(theme.breakpoints.down('sm'));
  const smallDeviceSize = useMediaQuery('(max-width:700px)');
  const mediumDeviceSize = useMediaQuery('(max-width:899px)');
  const tablet = useMediaQuery(theme.breakpoints.up('sm'));
  const laptop = useMediaQuery(theme.breakpoints.up('md'));

  return { mobile, tablet, laptop, smallDeviceSize, mediumDeviceSize };
}
```

Slika 4.5. useDeviceSizes hook

Pomoću hookova *useSubmitOnEnter* i *useKeyPress* implementirano je zaključivanje forme korištenjem tipke enter. Prije slanja podataka na backend radi konzistentnosti je potrebno obrisati prazna svojstva objekta, a navedeno je omogućeno putem funkcije *removeEmptyProperties* prikazane na slici 4.6.

```
export const removeEmptyProperties = <T>(object: T) => {
  Object.keys(object).forEach(
    key => object[key] === undefined && delete object[key],
  );
};
```

Slika 4.6. Brisanje praznih svojstava objekta

4.3. Reach router

React biblioteka nema ugrađenu navigaciju, pa je zbog toga potrebno instalirati vanjsku biblioteku. Dvije najčešće korištene biblioteke su React Router i Reach Router. Za potrebe ovog diplomskog rada korišten je Reach Router. Prije postavljanja Routing komponente potrebno je definirati rute koristeći enum. Router odabire koje će se dijete (engl. *children*) renderirati s obzirom na URL. Izvedba Routinga je prikaza na slici 4.7.

```
export const Routing: React.FC = () => {
  return (
    <Router basepath="/">
      <RouterPage path={Routes.Landing} pageComponent={<LandingPage />} />
      <RouterPage path={Routes.Login} pageComponent={<LoginPage />} />
      <RouterPage path={Routes.Register} pageComponent={<RegisterPage />} />
      <RouterPage path={Routes.Onboarding} pageComponent={<OnboardingPage />} />
      <RouterPage path={Routes.Profile} pageComponent={<ProfilePage />} />
      <RouterPage
        path={Routes.FacilityBuilder}
        pageComponent={<HostFacilityPage />}
      />
      <RouterPage
        path={Routes.MySportFacilities}
        pageComponent={<MySportsFacilitiesPage />}
      />
      <RouterPage path={Routes.Inbox} pageComponent={<InboxPage />} />
      <RouterPage path={Routes.Chat} pageComponent={<ChatPage />} />
            //… ostale stranice
      <RouterPage path={Routes.NotFound} pageComponent={<ErrorPage />} />
    </Router>
  );
};
```

Slika 4.7. Arhitektura aplikacije

Kao što je prikazano na slici svakom RouterPageu se prosljeđuje path i komponenta koja će se renderirati. Ako ruta ne odgovara ni jednoj ruti, reach router automatski preusmjerava korisnika na ErrorPage automatski. Komponenta *Routing* se nalazi unutar korijenske komponente *App* kako bih pratila trenutno stanje ruta kao što je prikazano na slici 4.8.

```
export const App: React.FC = () => {
  …
  return (
    <ThemeProvider theme={theme}>
      <FirebaseProvider>
        <Routing />
        <ToastContainer position="bottom-left" />
      </FirebaseProvider>
    </ThemeProvider>
  );
};
```

Slika 4.8. Uporaba komponente Routing

Za programatsku navigaciju moguće je koristiti komponentu *Link* ili metodu *navigate* putem kojih je moguće slati podatke određenoj komponenti. Slanje podataka putem metode *navigate* prikazano je na slici 4.9.

```
navigate(Routes.MakeReservation, {
  state: {
    facilityId: facility.id,
    creatorId: facility.creatorId,
  },
})
```

Slika 4.9. Slanje podataka putem metode navigate

Prije dohvaćanja podataka potrebno je implementirati type guard za provjeru tipa podataka kao na slici 4.10.

```
interface ExtendedLocation extends WindowLocation<unknown> {
  state: {
    creatorId: string;
    facilityId: string;
  };
}

export function isReservationTypeGuard(
  location: WindowLocation<unknown>,
): location is ExtendedLocation {
  return typeof location.state === 'object' && location.state !== null;
}
```

Slika 4.10. Type guard za provjeru tipa podataka

Nakon toga je moguće u komponenti *MakeReservation* dohvatiti podatke koristeći hook *useLocation*. Na ovaj način se pravilno prosljeđuju podaci u aplikaciji koristeći Reach Router.

4.4. Implementacija Firebase

Prije same implementacije Firebasea potrebno je unutar Firebase konzole putem web preglednika kreirati novi web Firebase projekt. Nakon uspješnog kreiranja projekta dobije se konfiguracijska datoteka koja sadrži API ključeve za pristup različitim uslugama. API ključeve je potrebno zaštiti tako što se u korijenskom direktoriju kreiraju varijable okoliša (engl. *environment variables*) unutar .env datoteke koja sadrži navedene ključeve i njihove vrijednosti. Kako bih navedene varijable bile dostupne samo nama unutar .gitignore datoteke potrebno je navesti da se datoteka .env. ignorira.

```
REACT_APP_FIREBASE_API_KEY=value
REACT_APP_FIREBASE_AUTH_DOMAIN=value
REACT_APP_FIREBASE_PROJECT_ID=value
REACT_APP_FIREBASE_STORAGE_BUCKET=value
REACT_APP_FIREBASE_MESSAGING_SENDER_ID=value
REACT_APP_FIREBASE_APP_ID=value
```

Slika 4.11. Varijable okoliša

Zatim je potrebno unutar projekta instalirati Firebase paket pomoću naredbe *firebase init* koja putem CLI-ja daje upite koje Firebase usluge će projekt koristiti. Za potrebe ovog diplomskog rada koristiti će se Firebase Authentication, Cloud Firestore, Cloud Storage i Cloud Functions. Pristup korisnika zaštićenim varijablama okoliša prikazan je na slici 4.12.

```
const clientCredentials: FirebaseOptions = {
  apiKey: process.env.REACT_APP_FIREBASE_API_KEY,
  authDomain: process.env.REACT_APP_FIREBASE_AUTH_DOMAIN,
  projectId: process.env.REACT_APP_FIREBASE_PROJECT_ID,
  storageBucket: process.env.REACT_APP_FIREBASE_STORAGE_BUCKET,
  messagingSenderId: process.env.REACT_APP_FIREBASE_MESSAGING_SENDER_ID,
  appId: process.env.REACT_APP_FIREBASE_APP_ID,
};
```

Slika 4.12. Vjerodajnice klijenta

Povezivanje React i Firebase projekta prikazano je na slici 4.13.

```
/**
 * Initialization of Firebase
 * @name createFirebaseApp
 * @description Function that creates and initializes a firebase instance or retrieves the existing one.
 */

function createFirebaseApp() {
  if (!getApps().length) {
    const app = initializeApp(clientCredentials);

    return app;
  }

  return getApp();
}
```

Slika 4.13. Dohvaćanje Firebase aplikacije

Funkcija *createFirebaseApp* se poziva unutar komponente *FirebaseProvider* koja se nalazi iznad korijenske komponente *App* kako bih Firebase bio dostupan globalno. Za izvanmrežnu podršku rada aplikacije korištena je metoda *enableIndexedDbPersistence.*

4.4.1. Firebase Authentication

Logika autentifikacije korisnika se nalazi unutar hooka *useAuthentication*. Navedeni hook se poziva unutar *FirebaseProvider* komponente kako bih u svakom trenutku znali stanje korisnika. Sljedeće funkcije se nalaze unutar hooka i importaju se u različitim komponentama ovisno o namjeni. Prije svega potrebno je kreirati korisnički račun što je moguće putem email adrese i lozinke, što je prikazano na slici 4.14.

```
/**
  * Create new account with email and password
  * @name register
  * @description Function that creates new firebase account in Firebase Authentication.
  */

  async function register(email: string, password: string) {
    try {
      await createUserWithEmailAndPassword(auth, email, password);
      successToast('You have successfully registered!');
    } catch (error) {
      if (error instanceof FirebaseError)
        errorToast(getRegisterErrorMessage(error.code));
    }
  }
```

Slika 4.14. Kreiranje korisničkog računa putem emaila i lozinke

Osim registracije putem emaila i lozinke moguće je kreirati račun putem Googlea ili Facbooka. Izvedba Google kreiranja računa prikazana je na slici 4.15.

```
/**
  * Create new account or login user with Google
  * @name loginWithGoogle
  * @description Function that creates new firebase account in Firebase Authentication or logs in user with
Google Provider.
  */

 async function loginWithGoogle() {
   try {
     googleProvider.setCustomParameters({ prompt: 'select_account' });
     await signInWithPopup(auth, googleProvider);
     successToast('You are logged in successfully using Google!');
   } catch (error) {
     if (error instanceof FirebaseError)
       errorToast(getLoginErrorMessage(error.code));
   }
 }
```

Slika 4.15. Kreiranje korisničkog računa Google servisa

Nakon što je korisnik uspješno kreirao korisnički putem funkcija *login* i *logout* je moguće se ulogirati i odlogirati. U slučaju zaboravljene lozinke moguće je zatražiti novu lozinku putem funkcije *resetPassword* koja je prikazana na slici 4.16.

```
/**
  * Send forgot password link to user email
  * @name resetPassword
  * @description Function that sends reset password link to user email.
  */

 async function resetPassword(email: string) {
   await sendPasswordResetEmail(auth, email)
     .then(() => {
       navigate(Routes.Login);
       successToast(
         'Check your email. Reset password mail has been sent successfully!',
       );
     })
     .catch((error: FirebaseError) =>
       errorToast(getLoginErrorMessage(error.code)),
     );
 }
```

Slika 4.16. Zahtjev za kreiranje nove lozinke

Za praćenje stanja korisnika unutar aplikacije implementiran je Firebase listener koji prati i zapisuje stanje trenutnog korisnika. Pravilna implementacija listenera zahtijeva u početnom

dijelu hooka poziv na slušanje stanja korisnika, a na izlazu je potrebno ugasiti konekciju kako bih aplikacija imala što bolje performanse. Implementacija listenera je prikazana na slici 4.17.

```
useEffect(() => {
   const subscription = auth.onAuthStateChanged(onUserAuthStateChange);

   return subscription;
 }, [auth, onUserAuthStateChange]);
```

Slika 4.17. Implementacija Firebase listenera za autentifikaciju

Funkcija *onUserAuthStateChange* provjerava stanje korisnika i ovisno o tome je li logiran ili ne izvršava određene akcije. U slučaju završetka korištenja aplikacije ili potrebom za kreiranje novog korisničko računa korisnik ima mogućnost brisanja korisničkog računa putem funkcije *deleteAccount* koja je prikazana na slici 4.18. Unutar projekta se nalazi hook *useAuthenticationRedirects* koji služi za usmjeravanje korisnika na određene stranice ovisno o stanju autentifikacije. Ako korisnik koji nije logiran pokušava pristupiti dijelu aplikacije koji zahtijeva logiranog usera, istoimeni hook usmjerava korisnika nazad na *LoginPage*. Logirani korisnik ima pristup svim dijelovima aplikacije.

```
/**
  * Delete account for currently logged user
  * @name deleteAccount
  * @description Function that deleted firebase account.
  */

 async function deleteAccount() {
   try {
     const auth = getAuth();
     const user = auth.currentUser;

     if (!user) return;

     await deleteUser(user);
     successToast('User is successfully deleted!');
   } catch (error) {
     if (error instanceof FirebaseError) errorToast(error.message);
   }
 }
```

Slika 4.18. Brisanje korisničkog računa

4.4.2. Cloud Firestore

Pristup bazi podataka se nalazi unutar hooka *useFirestore*. Prije toga kako bih developer imao što ugodnije iskustvo razvoja aplikacija razvijen je hook *useFirestoreUtilities* unutar kojega se nalaze korisne funkcije (engl. *utility functions*) kako se logika ne bih trebala više puta ponavljati za iste stvari, kao npr. dohvaćanje reference za dokument, kolekciju, više dokumenata, provjera postoji li kolekcija od prije. Dohvaćanje reference na kolekciju i dokument prikazano je na slici 4.19.

```
const getCollectionReference = (uid: string) => collection(db, uid);

const getDocumentReference = (uid: string, documentName: string) =>
  doc(db, uid, documentName);
```

Slika 4.19. Dohvaćanje reference na kolekciju i dokument

Osim toga hook *useFirestoreUtilities* sadrži čuvare tipova (engl. *type guards*). Type guardovi znatno olakšavaju razvoj aplikacija kada se šalju i primaju HTTP zahtjevi zato što osiguravaju transparentnost podataka u aplikaciji. Prije svakog slanja na backend i poslije svakog primanja podataka sa backenda potrebno je koristiti type guardove kako bih tipovi podataka u aplikaciji bili konzistentni. Primjer type guarda za provjeru tipa podatka *OnboardingData* prikazan je na slici 4.20.

```
const isOnboardingData = (data?: DocumentData): data is OnboardingData => {
  return data
    ? (data as OnboardingData).isOnboardingInProgress !== undefined
    : false;
};
```

Slika 4.20. Type guard

Hook *useFirestore* dohvaća instancu Firestorea koristeći Reactov hook *useMemo* koji memoizira vrijednost baze podataka i time povećava performanse aplikacije. Razvijene pomagačke funkcije iz *useFirestoreUtilities* hooka se koriste u *useFirestoreu* za različite akcije nad različitim modulima koje će biti opisane u poglavlju moduli. Kako bih korisnik imao podatke za različite module u pravom vremenu kreiran je hook *useFirestoreListeners* koji na svaku promjenu unutar baze podataka dohvaća podatke istovremeno. Hook se poziva unutar korijenske komponente *App* kako bih podaci bili dostupni globalno, a implementacija hooka je prikazana na slici 4.21.

```
export const useFirestoreListeners = () => {
  const user = useRecoilValue(authSelectors.user);

  const { getMyNotifications, getMyFacilities, getMyReservations } =
    useFirestore();
  const { getFacilities, getChats } = useFirebaseFunctions();

  useEffect(() => {
    if (!user?.userUid) return;

    getMyNotifications(user.userUid);
    getMyReservations(user.userUid);
    getMyFacilities(user.userUid);
    getFacilities();
    getChats();
  }, [user]);

};
```

Slika 4.21. Dohvaćanje podataka u stvarnom vremenu

4.4.3. Cloud Storage

Pohrana i dohvaćanje multimedijskih podataka se obavlja putem hooka *useFirebaseStorage* gdje se dohvaća instanca Storaga koristeći *useMemo*. Športski objekti su spremljeni po datotekama, gdje je jedna datoteka ime športskog objekta. Potrebno je dohvatiti referencu na Firebase Storage, a zatim učitati datoteke te dohvatiti URL koji se sprema unutar baze podataka kao na slici 4.22.

```
async function uploadBlobOrFile(
    file: Blob | Uint8Array | ArrayBuffer,
    facilityId: string,
    fileName: string,
  ) {
    const storageReference = getStorageReference(`${facilityId}/${fileName}`);
    try {
      await uploadBytes(storageReference, file);
      const url = await getDownloadURL(storageReference);
      infoToast(
        'Submit the form so you can successfully save updated pictures!',
      );
      return url;
    } catch (error) {
      errorToast('You have failed uploading image!');
    }
    return '';
  }
```

Slika 4.22. Učitavanje podataka

4.4.4. Cloud Functions

Dohvaćanje ID-eva korijenskih kolekcija nije moguće obaviti na klijentskoj strani, no na backend strani koristeći Cloud Functions je moguće. Kako bih korisnici mogli izvršavati određene radnje nad različitim entitetima ili modulima kao npr. objekti, rezervacije, notifikacije i sl. potrebno je imati pristup ID-u. Postoji više vrsta Cloud Funkcija, a za svrhu ovog diplomskog rada je korištena funkcija *onCall* koju je dostupno pozivati sa klijentske strane. Dohvaćanje svih dostupnih kolekcija osim vlastite kolekcije je prikazano na slici 4.23.

```
export const getAvailableCollectionsIDs = region('europe-west2').https.onCall(
  async ({ userUid }: Data) => {
    try {
      const allCollectionsReference = await firestore().listCollections();

      const availableCollectionIDs = allCollectionsReference
        .filter((collection) => collection.id !== userUid)
        .map((collection) => collection.id);
      return availableCollectionIDs;
    } catch (e) {
      return e;
    }
  },
);
```

Slika 4.23. Dohvaćanje ID-eva korijenskih kolekcija

Pozivanje funkcije sa klijentske strane se obavlja koristeći *httpsCallable* kojemu se prosljeđuje referenca na funkcije i ime funkcije, u ovom slučaju *getAvailableCollectionIDs*. Na taj način se posao rasterećuje sa klijentske strane na backend što ubrzava rad aplikacije.

4.6. Konfiguracija Recoila

Svi podaci u aplikaciji koji su potrebni u više različitih komponenti se spremaju u Recoil za globalnu dostupnost. Korijenska komponenta *App* obgrljena je sa *RecoilRoot* komponentom kao na slici 4.24.

```
root.render(
  <RecoilRoot>
    <App />
  </RecoilRoot>,
);
```

Slika 4.24. Postavljanje Recoila

Svaki modul ima zasebne atome i selectore koji su opisani u poglavlju *Korištene tehnologije i alati*. Tako su različita stanja izolirana i samim time se smanjuje broj re-rendera na promjenu

stanja aplikacije. Recoil nudi bogat API čime se upravljanje stanjem aplikacije znatno pojednostavljuje u usporedbi s tržišnim liderom Reduxom [11].

4.7. Moduli

Ultimativna modularna paradigma je da developeri žele izgraditi predvidljive aplikacije sastavljene od funkcija i komponenti gdje svaki dio ima svoju odgovornost. Korištenje modularne arhitekture zahtijeva izgradnju aplikacije oko modula što je veoma jednostavno i samim time kolaboracija više developera se znatno olakšava. Kategoriziranje datoteka na temelju njihovih funkcionalnosti jednostavan je način particioniranja projekta i veoma popularna praksa kod developera koji koriste obrazac kao što je MVC [12]. Razlog zbog kojeg najviše olakšava razvoj je kada se radi na velikom projektu jako je teško identificirati izvor problema i kako developer ne bih trebali kopati po tisućama linija koda dok ne shvate sve međuodnose unutar aplikacije.

4.7.1. Authorization

Authorization modul je namijenjen za *Onboarding* gdje korisnik unosi osobne podatke, a čiji će izgled biti prikazan u sljedećem poglavlju. Glavno stanje authorization modula je *settings* selector koji objedinjuje sve atome kao npr. ime, prezime i sl. Proces je sastavljen od tri dijela unosa podataka, a nakon prvog koraka se unutar baze podataka kreira korijenska kolekcija čiji je ID jednak ID-u korisnika unutar Firebase Authentication servisa kako bih autentifikacija i baza podataka bili povezani. Na svakom sljedećem koraku ažuriraju se podaci unutar dokumenta *settings* kao na slici 4.25.

```
const updateUser = async (
  userUid: string,
  onboardingData: OnboardingData,
) => {
  try {
    const documentReference = getDocumentReference(userUid, 'settings');
    removeEmptyProperties(onboardingData);

    await setUserCollection(documentReference, onboardingData, true);
  } catch (error) {
    console.log(error);
  }
};
```

Slika 4.25. Ažuriranje settings dokumenta

Nakon uspješnog *Onboarding* procesa svi osobni podaci od korisnika se nalaze unutar *settings* dokumenta. Ažuriranje podataka se izvršava koristeći istu funkciju unutar aplikacije.

Dohvaćanje podataka se izvršava putem funkcije getSettings koja se poziva u Authentication listener kako bih uvijek pravovremeno imali podatke o korisniku.

```
const getSettings = async (userUid: string) => {
  try {
    const settingsDocument = doc(db, userUid, 'settings');
    const settingsSnapshot = await getDoc(settingsDocument);

    const onboardingData = settingsSnapshot.data();

    if (isOnboardingData(onboardingData)) {
      return onboardingData;
    }
    return;
  } catch (error) {
    console.log(error);
  }
};
```

Slika 4.26. Dohvaćanje settings dokumenta

Korisničko sučelje za autorizaciju korisnika se sastoji od navigacije i stepera. Steper komponenta omogućava korisniku vizualni pristup na kojem koraku se trenutno nalazi i koje podatke treba unijeti. Ovisno o varijabli *activeStep* se prikazuje različita komponenta. Kod za prikaz korisničkog sučelja prikazan je na slici 4.27.

```
    <form>
      <StepperBuilder
        activeStep={activeStep}
        skipped={skipped}
        steps={onboardingSteps}
      />
      <Container component="main" maxWidth="xl">
        {activeStep === 0 && <UserInfo avatarPhoto={settings?.avatar} />}
        {activeStep === 1 && <OnboardingAddress />}
        {activeStep === onboardingSteps.length && (
          <OnboardingPreview avatarPhoto={settings?.avatar} />
        )}
      </Container>
      <NavigationBuilder
        activeStep={activeStep}
        steps={onboardingSteps}
        onSubmit={onSubmit}
        handleBack={handleBack}
        handleNext={handleNext}
        handleReset={handleReset}
      />
    </form>
```

Slika 4.27. OnboardingBuilder komponenta

4.7.2. Ostali moduli

Facilities modul je glavni entitet nad kojim se vrše sve radnje u aplikaciji, a predstavlja jedan športski objekt. Ako je korisnik vlasnik športskog ima mogućnost kreiranja, ažuriranja, dohvaćanja ili brisanja športskog objekta. Primjer kreiranja športskog objekta prikazan je na slici 4.26.

```
const createFacility = async (
  userUid: string,
  facilityData: Omit<Facility, 'files'>,
) => {
  try {
    removeEmptyProperties(facilityData);
    const subColRef = collection(db, userUid, 'facilities', 'entities');
    const facilityRef = await addDoc(subColRef, facilityData);

    return facilityRef.id;
  } catch (error) {
    console.log(error);
  }
  return;
};
```

Slika 4.26. Kreiranje športskog objekta

Reservations modul predstavlja rezervacije za športske objekte. Korisnik može primati rezervaciju za određenu športsku dvoranu čiji je vlasnik i kreirati rezervacije za druge športske objekte kojih nije vlasnik. Kreiranjem rezervacije na drugi športski objekt kreira se rezervacija za trenutno logiranog korisnika i ažurira se športska dvorana koju korisnik želi rezervirati. Postupak kreiranja rezervacije prikazan je na slici 4.27.

```
const createReservation = async (
  userUid: string,
  reservationData: Omit<Reservation, 'id'>,
  notificationId?: string,
 ) => {
  try {
   if (!notificationId) return;

   removeEmptyProperties(reservationData);
   const reservationsSubColRef = collection(
    db,
    userUid,
    'reservations',
    'entities',
   );
   const reservationRef = await addDoc(reservationsSubColRef, {
    ...reservationData,
    reservationCreatorId: userUid,
    notificationId,
   });

   const documentReference = doc(
    db,
    reservationData.creatorId,
    `facilities/entities/${reservationData.facilityId}`,
   );
   await updateDoc(documentReference, {
    reservedTimes: arrayUnion({
     ...reservationData,
     reservationCreatorId: userUid,
     reservationId: reservationRef.id,
     notificationId,
    }),
   });
   return reservationRef.id;
  } catch (error) {
   console.log(error);
  }
  return;
 };
```

Slika 4.27. Kreiranje rezervacije

Stanje rezervacije kada se kreira je u tijeku (engl. *pending*) sve dok vlasnik športskog objekta ne prihvati ili odbije rezervaciju. Ako se korisnik koji je rezervirao športski objekt predomisli i želi otkazati rezervaciju, sljedeća funkcija se izvršava.

```
const deleteReservation = async (
  userUid: string,
  reservationData: Reservation,
) => {
  try {
    removeEmptyProperties(reservationData);
    const reservationDocRef = doc(
      db,
      userUid,
      'reservations',
      'entities',
      reservationData.reservationId,
    );
    await deleteDoc(reservationDocRef);

    const documentReference = doc(
      db,
      reservationData.creatorId,
      `facilities/entities/${reservationData.facilityId}`,
    );
    await updateDoc(documentReference, {
      reservedTimes: arrayRemove(reservationData),
    });
  } catch (error) {
    console.log(error);
  }
  return;
};
```

Slika 4.28. Otkazivanje rezervacije

Vlasnik športskog objekta ima mogućnost prihvaćanja rezervacije, odbijanja rezervacije. Na događaj kreiranja rezervacije kreira se notifikacija vlasniku športskog objekta. Ako vlasnik športskog objekta prihvati rezervaciju ažurira se notifikacija, rezervacija te korisnik koji je rezervirao športski objekt dobiva notifikaciju da je otvoren chat s vlasnikom dvorane. Ista stvar se događa i ako vlasnik športskog objekta odbije rezervaciju. Chat se kreira jedino ako je prihvaćena rezervacija.

5. PRIKAZ NAČINA KORIŠTENJA APLIKACIJE

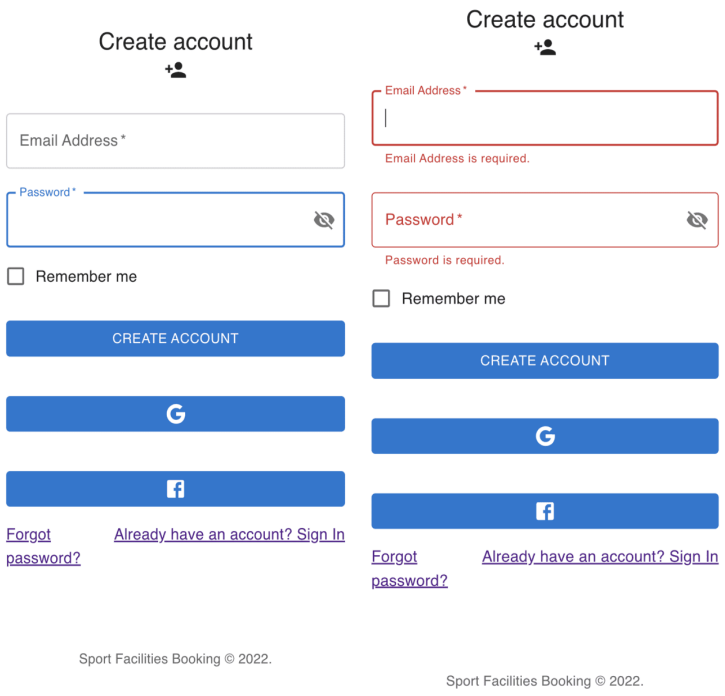
U ovome poglavlju detaljno je prikazan sam rad aplikacije, način korištenja i sve prethodno spomenute funkcionalnosti.

5.1. Prikaz korištenja aplikacije

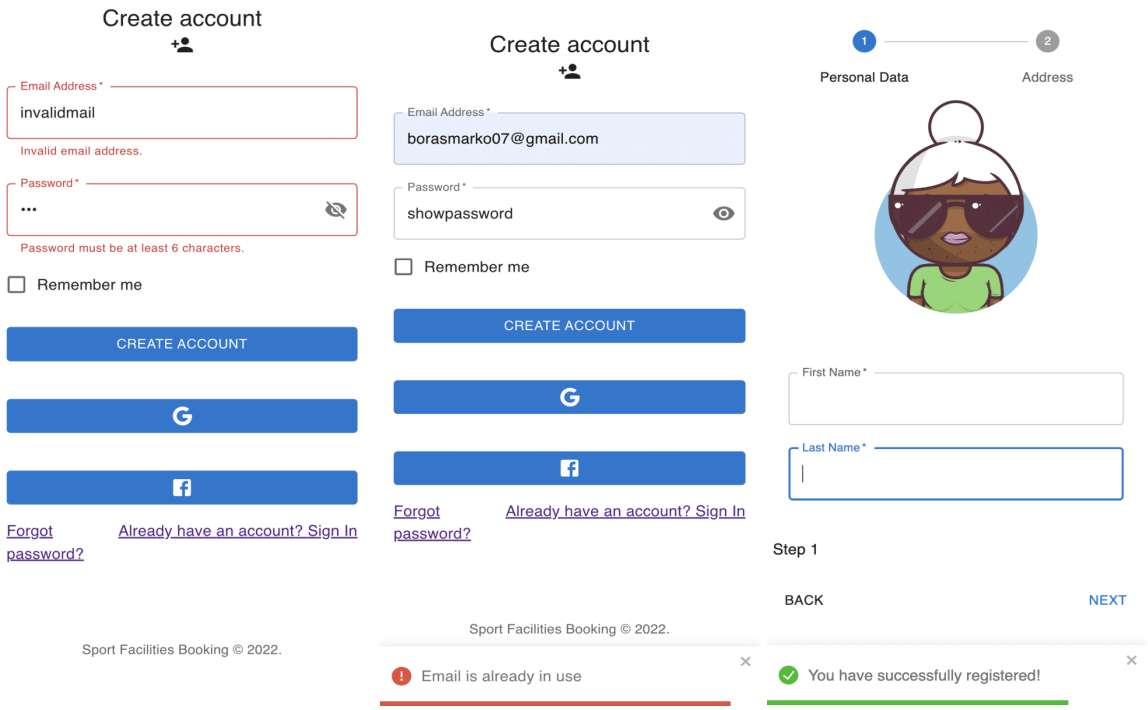
U sljedećim poglavljima će biti opisani i vizualno prikazani zasloni. Svi zasloni unutar aplikacije su responzivni i prilagođeni svim veličinama zaslona.

5.1.1. Registracija korisnika

Prije samog početka korištenja aplikacije potrebno je kreirati korisnički račun koristeći email i lozinku ili putem vanjskog poslužitelja kao što su Google i Facebook što je prikazano na slici 5.1. Na svakom od inputa postoji validacija na klijentskoj i poslužiteljskoj strani kao na slici 5.2. U slučaju pogreške korisnik dobiva putem toastbara povratnu informaciju. Ako korisnik je već kreirao korisnički račun sa jednim od vanjskih poslužitelja i pokuša kreirati korisnički račun sa istom emailom adresom koristeći drugi vanjski poslužitelj dobiva grešku kako već račun postoji.



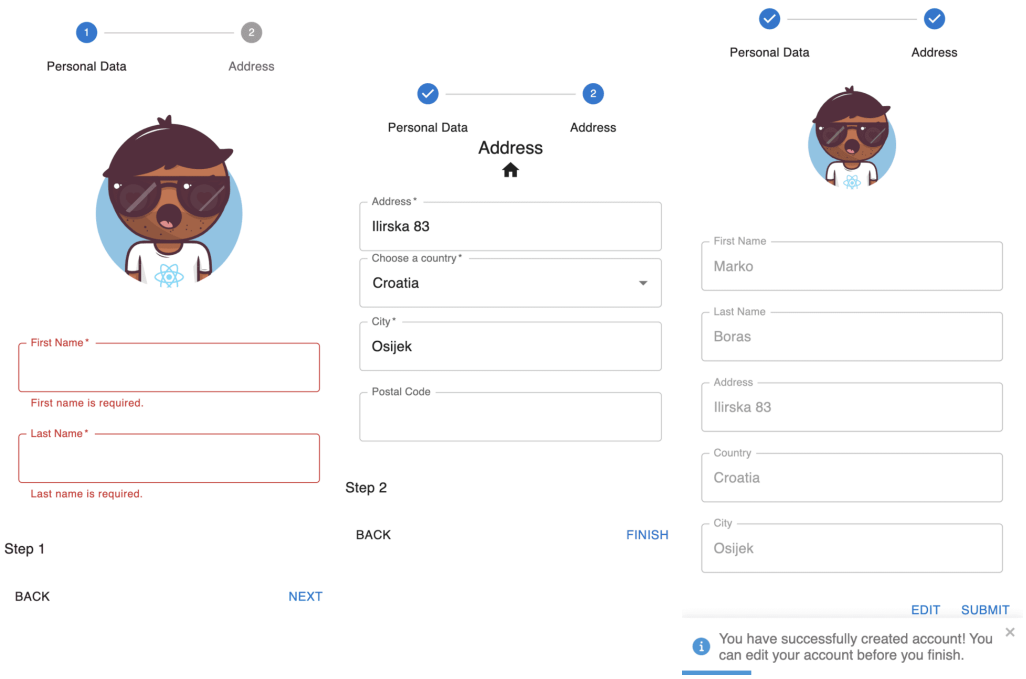
Slika 5.1. Prikaz forme za registraciju



Slika 5.2. Validacija registracije

5.1.2. Onboarding korisnika

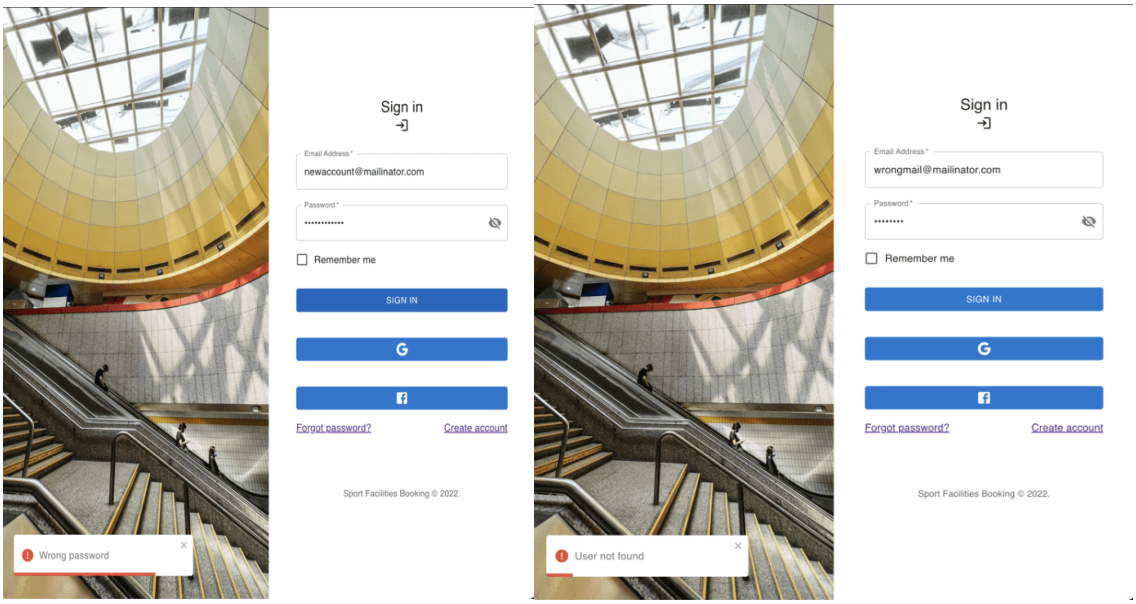
Unutar Onboarding procesa korisnik nasumično dobiva svog avatara kojeg je moguće kasnije izmijeniti unutar postavki korisničkog profila. Kao i unutar registracije postoji validacija svih korisničkih inputa prikazano na slici 5.3. Osim toga, ako korisnik slučajno prekine proces i izađe iz web preglednika, ponovnim ulaskom u aplikaciju dohvaćaju se svi prethodno unešeni podaci. Nakon unosa osobnih podataka kao što su ime, prezime, adresa, država, grad i poštanski broj moguće je pregledavati i uređivati podatke. Na vrhu zaslona se nalazi *stepper* koji služi za informiranje korisnika o određenom koraku unutar kojega se nalazi unutar Onboarding procesa, a na dnu zaslona se nalazi navigacija pomoću koje korisnik može se pomicati sa koraka na korak, što se može vidjeti iz priloženih slika.



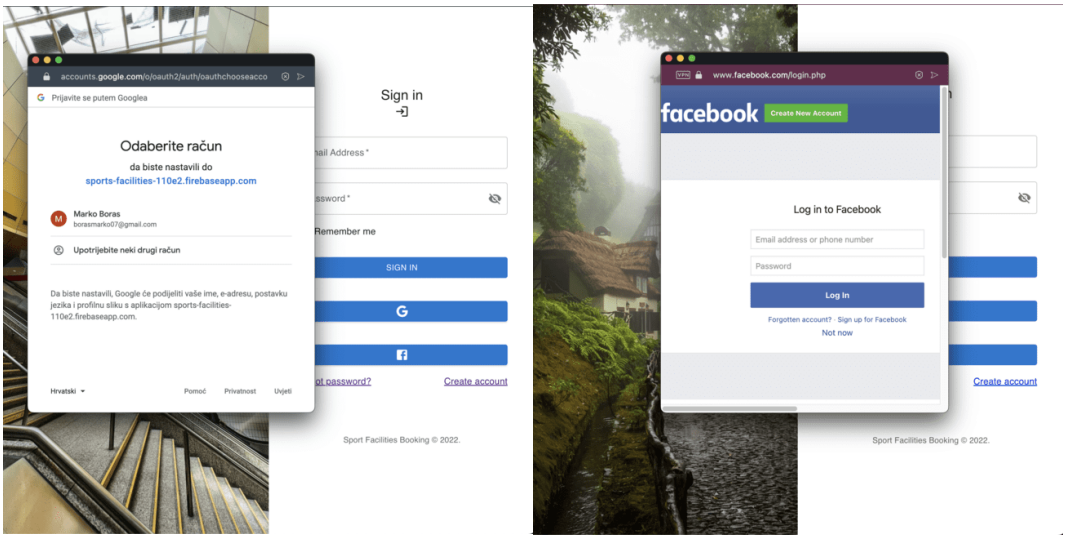
Slika 5.3. Onboarding proces

5.1.3. Prijava korisnika

Za prijavu korisnika unutar aplikacije potrebno je koristiti email i lozinku s kojima je korisnik prethodno kreirao korisnički račun ili vanjski poslužitelj kao što su Google i Facebook unutar ove aplikacije. Za sve pogrešno unešene podatke postoji validacija prikazana na slici 5.4.



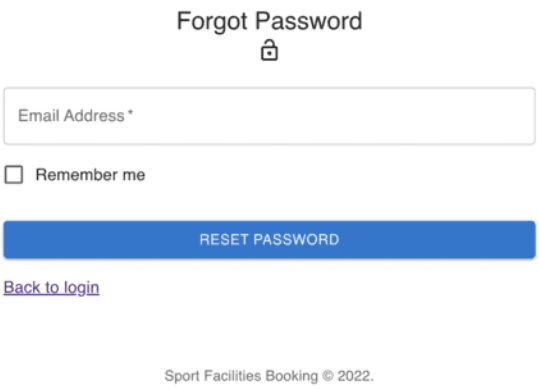
Slika 5.4. Pogrešan unos podataka za prijavu



Slika 5.5. Prijava putem vanjskog poslužitelja

5.1.3. Zaboravljena lozinka

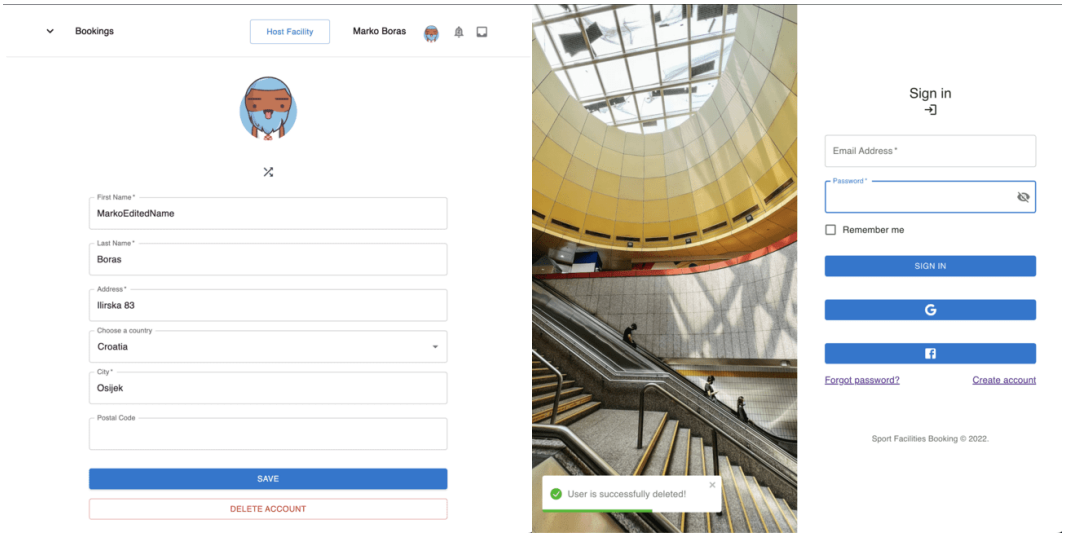
U slučaju da korisnik zaboravi lozinku za istu je moguće dobiti putem email adrese link pomoću kojega se generira nova lozinka. Ako email adresa ne postoji link neće biti poslan.



Slika 5.6. Generiranje nove lozinke

5.1.4. Korisnički profil

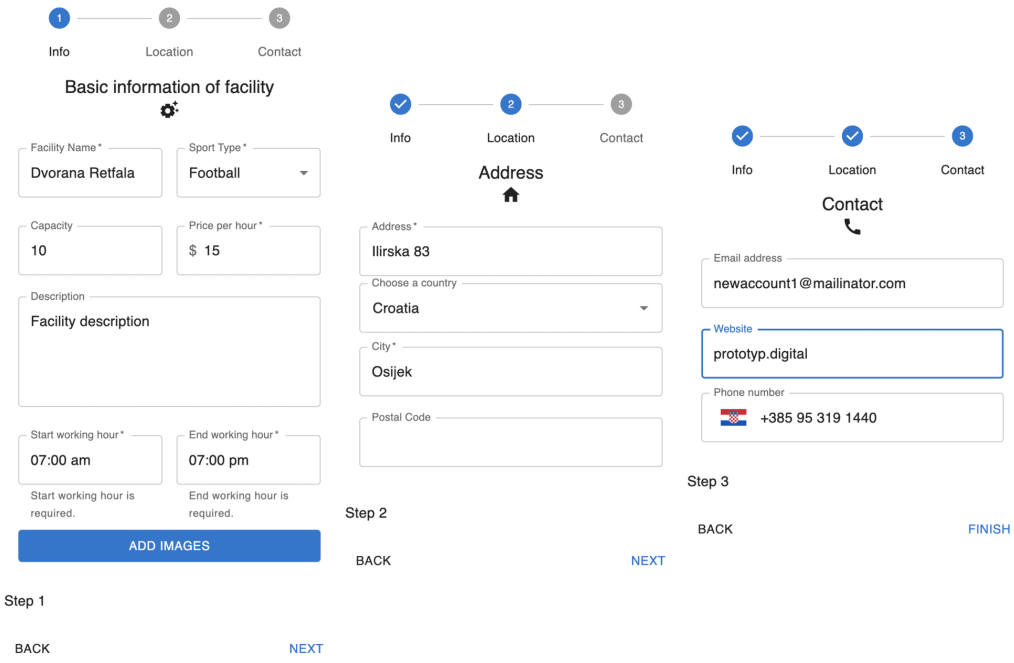
Korisnik ima mogućnost uređivanja osobnih podataka koje je unijeo na Onboarding procesu na stranici *Profile* prikazano na slici 5.7. Postoji mogućnost brisanja korisničkog računa putem gumba *Delete account* nakon čega se brišu svi podaci o korisniku i korisnik se automatski odjavljuje s aplikacije i vraća na zaslon prijave. Kada korisnik obriše korisnički račun u donjem lijevom zaslonu dobiva povratnu informaciju da je korisnički račun uspješno obrisan prikazano na slici 5.7.



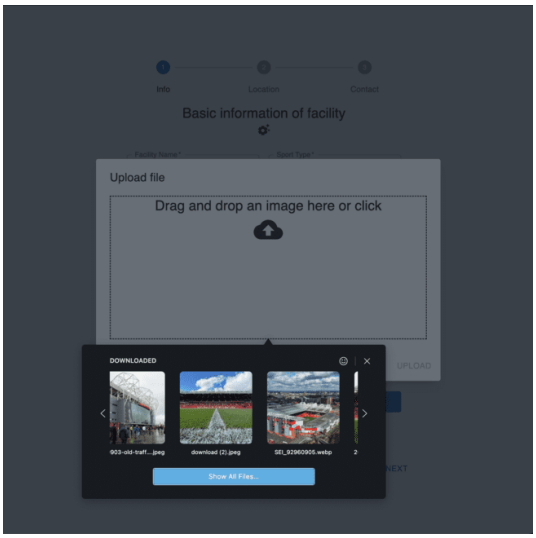
Slika 5.7. Uređivanje i brisanje korisničkog računa

5.1.5. Kreiranje športskog objekta

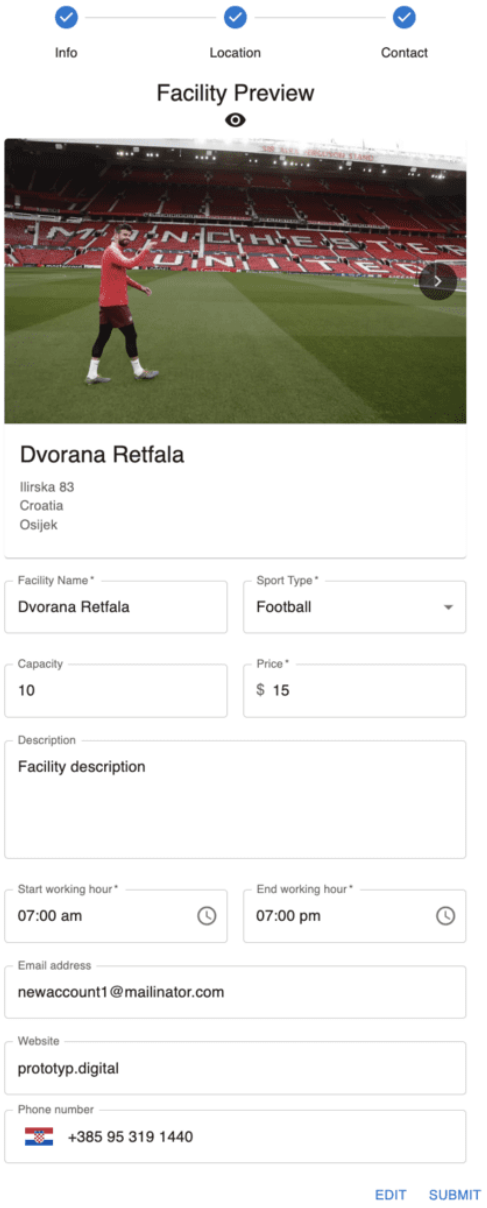
Vlasnici športskih objekata kreiraju športski objekt pritiskom na gumb *Host facility* koji usmjerava korisnika na stranicu za kreiranje objekta. Postupak izrade športskog objekta je sličan Onboardingu korisnika gdje je potrebno unijeti obavezne i opcionalne podatke o dvorani unutar više koraka. Prilikom izrade športskog objekta obavezno je navesti ime dvorane, tip sporta, cijenu po satu, radno vrijeme, adresu i kontakt, a ako korisnik želi može unijeti i dodatne informacije o dvorani, email adresu, broj mobitela, web adresu. Korisnik ima mogućnost dodavanja fotografija športskog objekta što je prikazano na slici 5.9., a ako ne želi automatski će mu biti dodjeljena nasumična fotografija. Nakon što korisnik unese sve informacije o športskom objektu, postoji mogućnost pregladavanja i uređivanja unešenih podataka unutar procesa kreiranja objekta kao na slici 5.10. Na vrhu zaslona se nalazi *stepper*, a na dnu se nalazi navigacija kao na Onboarding procesu.



Slika 5.8. Koraci pri kreiranju športskog objekta



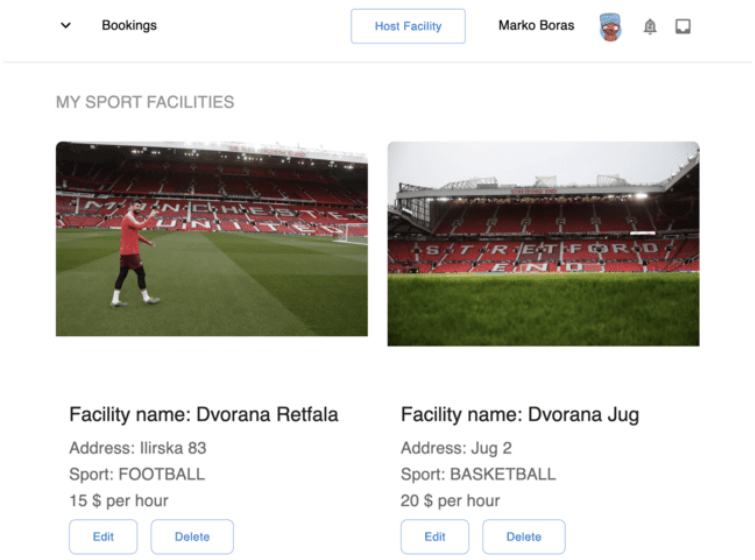
Slika 5.9. Odabir fotografija za športski objekt



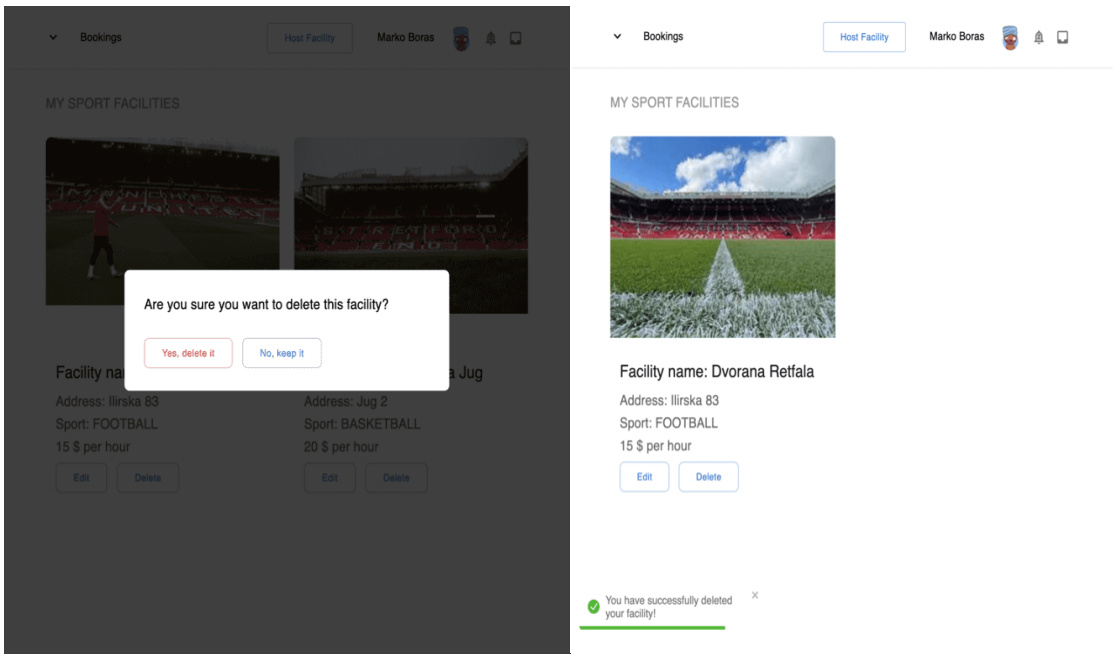
Slika 5.10. Pregled športskog objekta

5.1.6. Pregled kreiranih športskih objekata

Na stranici *My Sport Facilities* korisnik može pregledavati, uređivati i brisati vlastite športske objekte što je prikazano na slici 5.11. Do same stranice moguće je doći putem navigacije. Ako korisnik želi uređivati športski usmjerava ga se na istu stranicu gdje je kreirao športski objekt. U slučaju brisanja športskog objekta prikazan je modal prije samog brisanja kako bih korisnik potvrdio željenu akciju što je prikazano na slici 5.12.



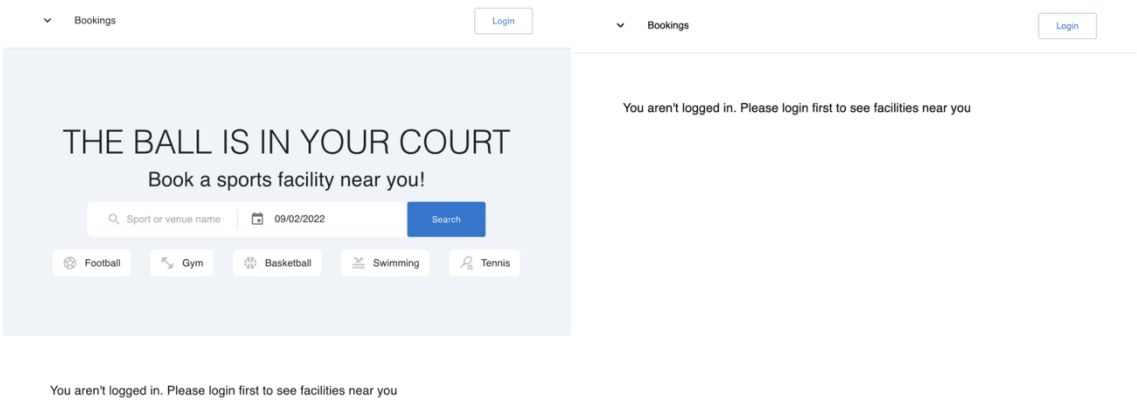
Slika 5.11. Pregled kreiranih športskih objekata



Slika 5.12. Brisanje športskog objekta

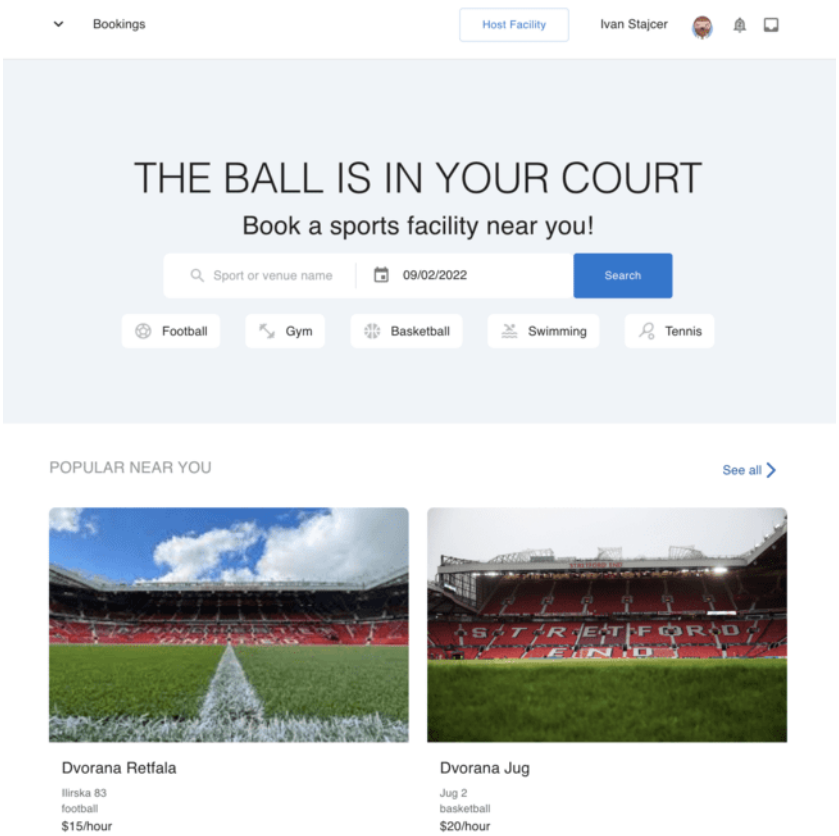
5.1.7. Pretraživanje drugih športskih objekata

Kada korisnik nije prijavljen u aplikaciju ne može kreirati rezervacije i pregledavati trenutno dostupne športske objekte što je prikazano na slici 5.13.



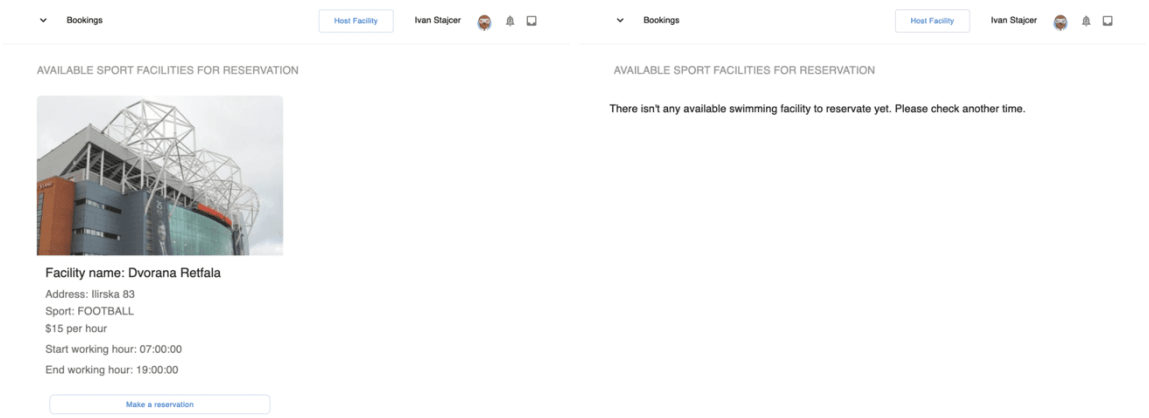
Slika 5.13. Prikaz aplikacije korisniku koji nije prijavljen

U svrhu prikaza korištenja aplikacije kreiran je drugi korisnički račun s kojim će se pregledavati športski objekti korisnika "Marko Boras", pod imenom "Ivan Štajcer". Ispod navigacije se nalazi slogan i opis aplikacije. Korisnik ima više načina putem kojih može pretraživati športske objekte što je prikazano na slici 5.14.



Slika 5.14. Prikaz aplikacije korisniku koji je logiran

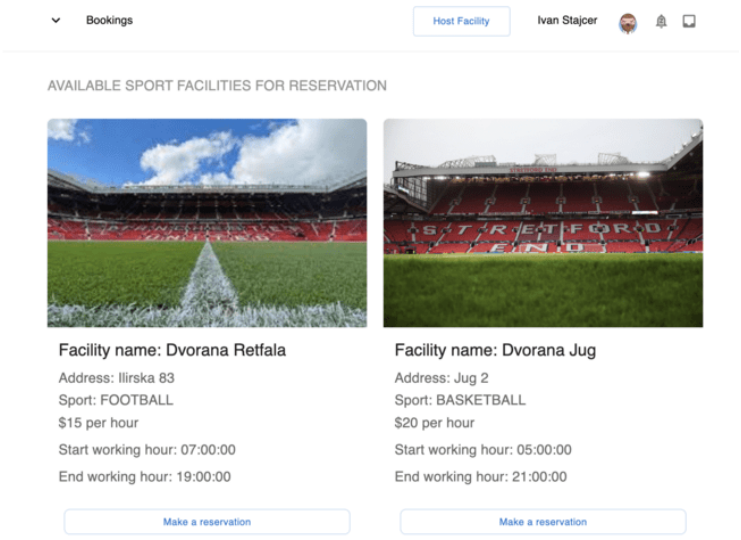
Značajka *Popular near you* prikazuje športske objekte u gradu u kojem se korisnik nalazi. Klikom na gumb *See all* korisnik se usmjerava na stranicu sa svim dostupnim športskim objektima za pretraživanje. Korisnik može pretraživati športske objekte prema sportu. Ako ne postoji ni jedan športski objekt odabranog tipa sporta prikaže se odgovarajuća poruka. Osim navedenih metoda pretraživanja korisnik može pretraživati športske objekte prema imenu dvorane ili sporta kao na slici 5.15.



Slika 5.15. Pretraživanje prema tipu športa

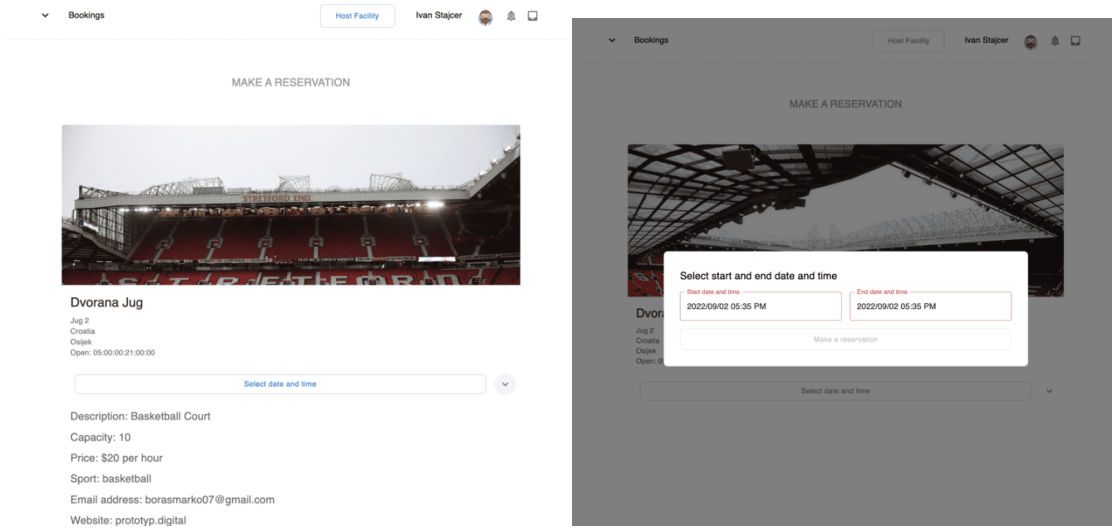
5.1.8. Rezerviranje športskog objekta

Odlaskom na *Available sports facilities* prikazuje se lista svih dostupnih športskih objekata za rezerviranje kao na slici 5.16.



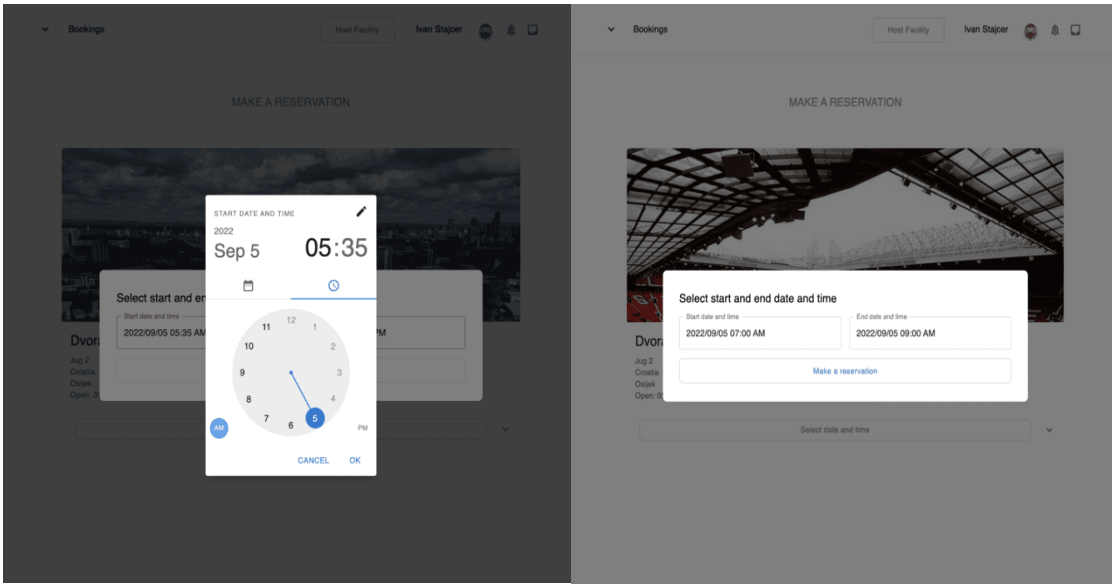
Slika 5.16. Prikaz dostupnih športskih objekata za rezerviranje

Klikom na gumb *Make a reservation* usmjerava se na stranicu za rezerviranje športskog objekta gdje korisnik može pregledati sve dostupne podatke o športskom objektu. Ako se korisnik odluči na rezerviranje športskog objekta pritiskom gumba *Select date and time* se pojavljuje modal unutar kojega se bira početno i završno vrijeme rezervacije kao na slici 5.17.



Slika 5.17. Rezerviranje športskog objekta

Minimalno dostupni datum početka rezervacije je trenutni datum, a za vrijeme se uzima radno vrijeme odabranog športskog objekta što je prikazano na slici 5.18.



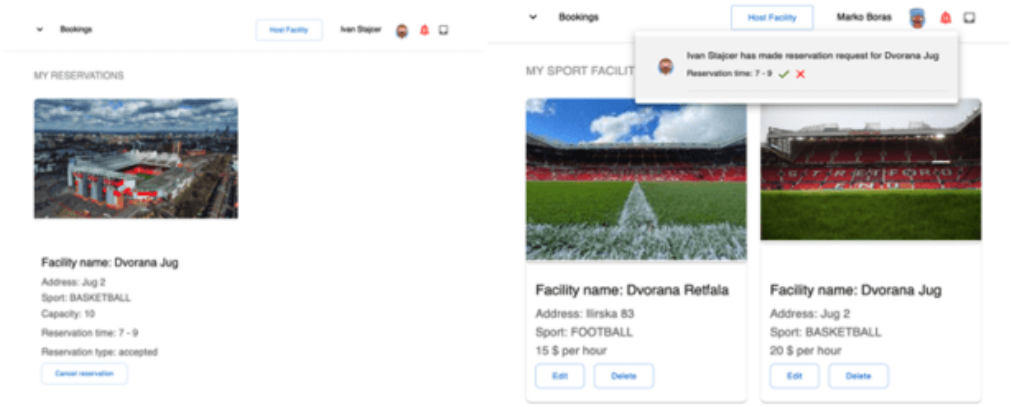
Slika 5.18. Odabir vremena i datuma rezervacije

Korisniku je dozvoljeno rezervirati športski objekt unutar radnog vremena, te postoji validacija odabira datuma i vremena. Nakon što korisnik uspješno rezervira termin preusmjerava se na

My Reservations stranicu na kojoj se nalaze sve rezervacije koje je korisnik napravio. Trenutno stanje rezervacije koju je napravio je u tijeku (engl. *pending*). Ako korisnik želi otkazati rezervaciju postoji gumb *Cancel reservation.* Vlasnik športskog objekta u trenutku kreiranja rezervacije prima notifikaciju što se može vidjeti na slici u navigacijsko traci.

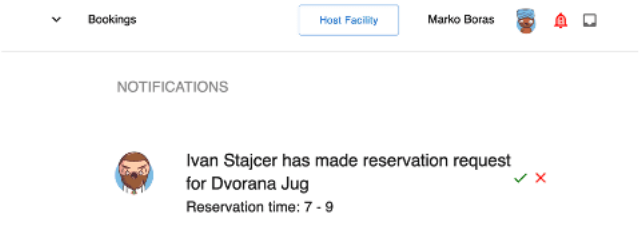
5.1.9. Notifikacije

Sa lijeve strane slike se nalazi korisnik koji želi kreirati rezervaciju, a sa desne se nalazi vlasnik športskog objekta što je prikazano na slici 5.19



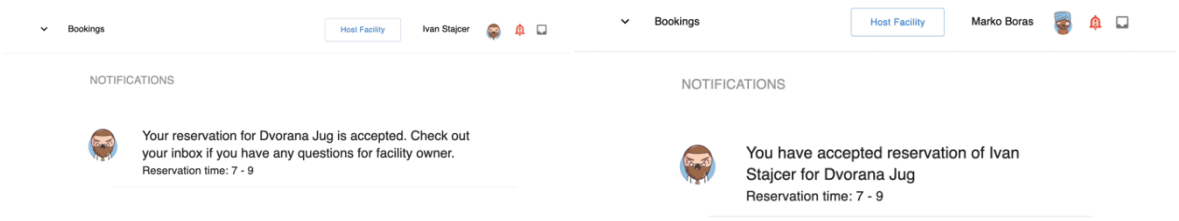
Slika 5.19. Slanje rezervacije i prijem notifikacije

Na stranici *Notifications* se nalaze sve obavijesti koje korisnik ima što je vidljivo na slici 5.20.



Slika 5.20. Notifications stranica

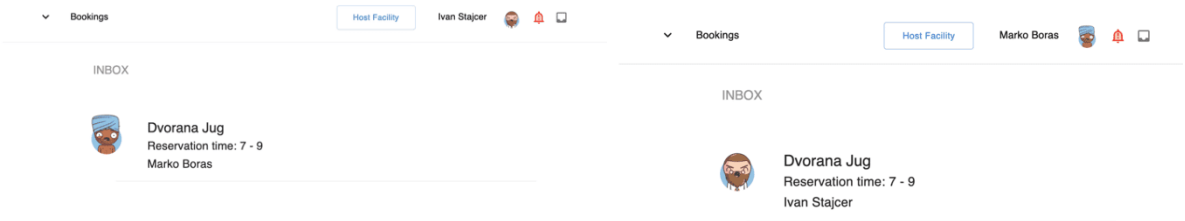
Ako vlasnik športskog objekta prihvati rezervaciju oba korisnika dobivaju notifikaciju što je prikazano na slici 5.21.



Slika 5.21. Notifications stranica nakon prihvaćanja rezervacije

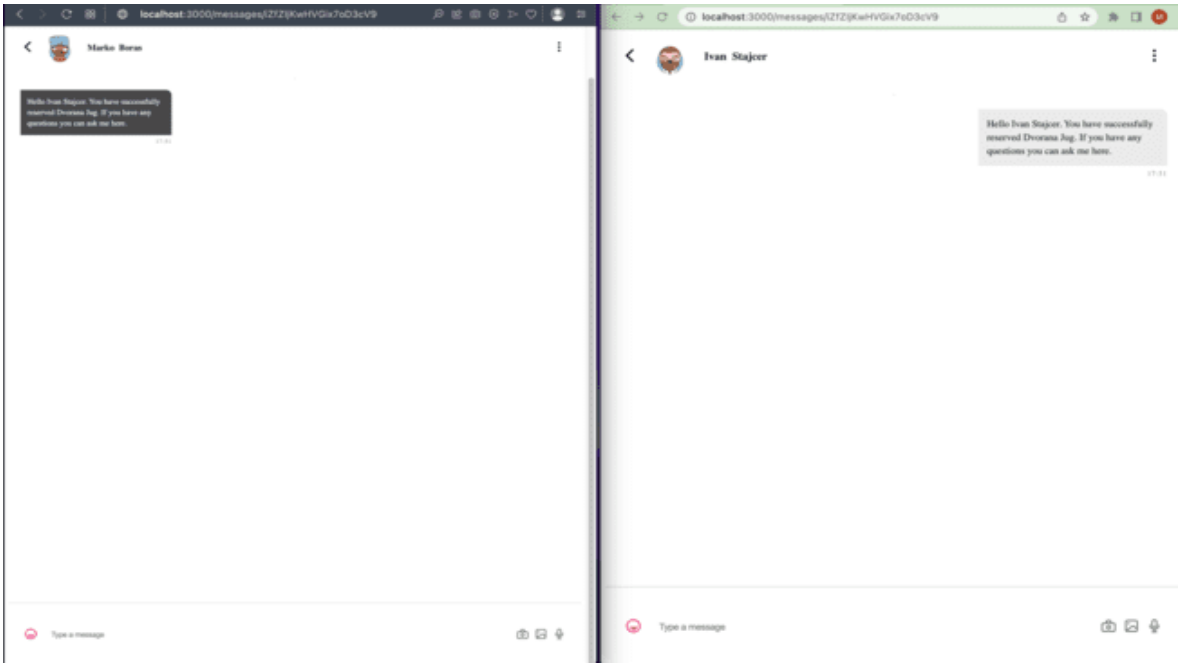
5.1.10. Chat

Kada vlasnik športskog objekta prihvati rezervaciju kreira se chat. Odlaskom u *Inbox* prikazan je nastali chat za oba korisnika što je prikazano na slici 5.22.

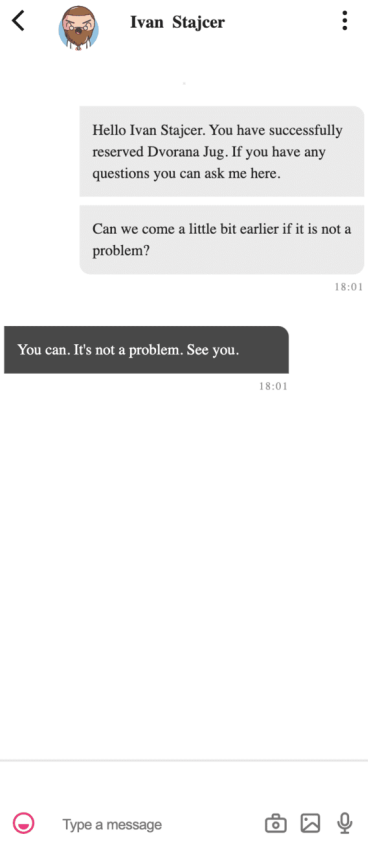


Slika 5.22. Inbox

Vlasnik športskog objekta automatski šalje generičku poruku korisniku koji je rezervirao športski objekt. Unutar poruke se dobiva obavijest da je uspješno potvrđena rezervacija. Sve poruke unutar chata su u stvarnom vremenu. Nakon svakog bloka poruka se nalazi vrijeme zadnje poslane poruke. Korisnici mogu slati multimedijske datoteke unutar chata što je prikazano na slici 5.23.



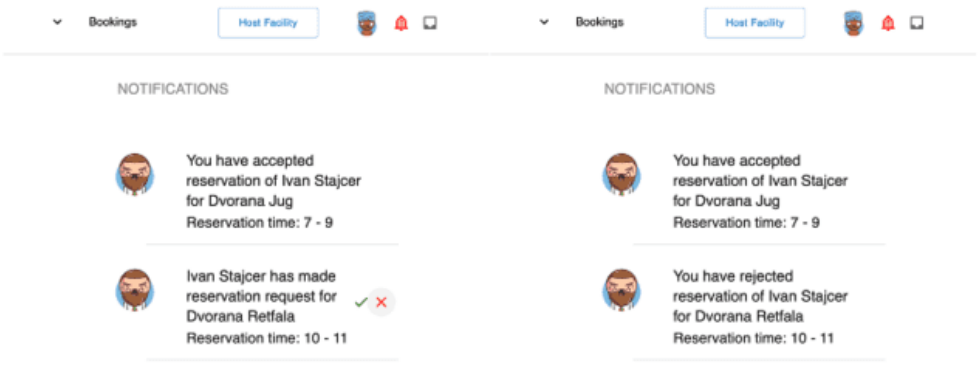
Slika 5.23. Chat



Slika 5.24. Slanje poruka

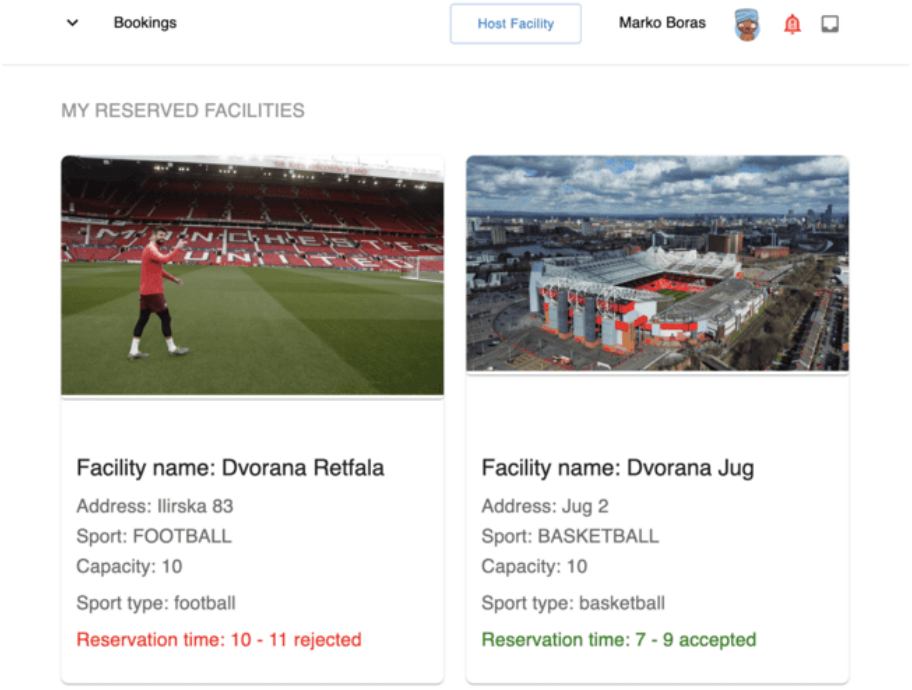
5.1.11. Odbijanje rezervacije

Vlasnik športskog objekta u nekim slučajevima može odbiti rezervaciju. Ako korisnik odbije rezervaciju oba korisnija dobivaju obavijest što je prikazano na slici 5.25.



Slika 5.25. Notifikacije

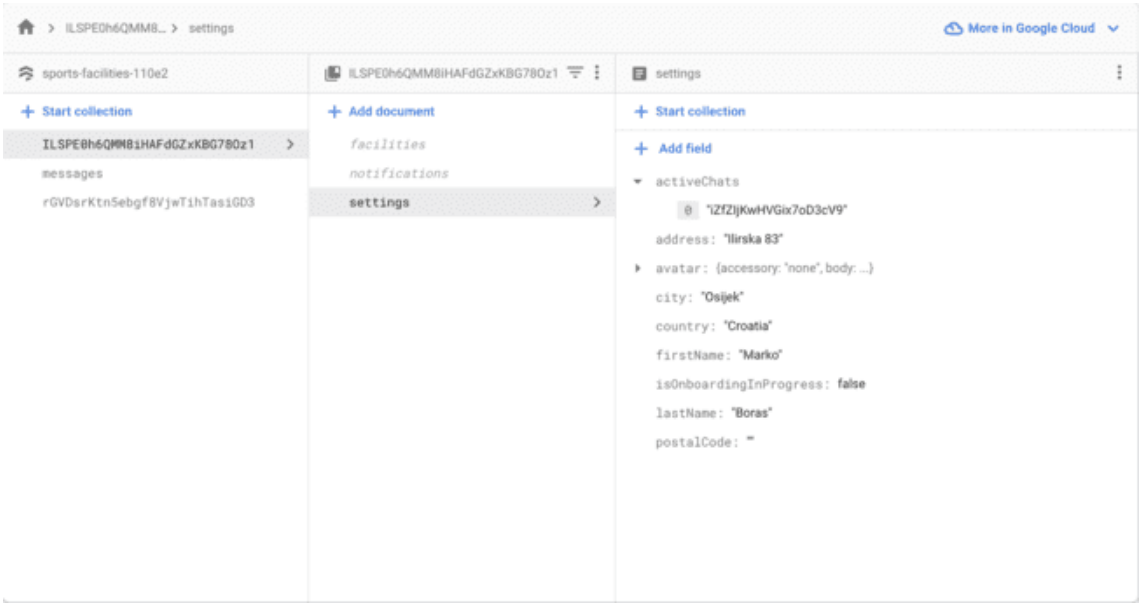
Na stranici *My reserved facilities* vlasnik športskog objekta ima arhivu svih rezervacija.



Slika 5.26. Arhiva rezervacija

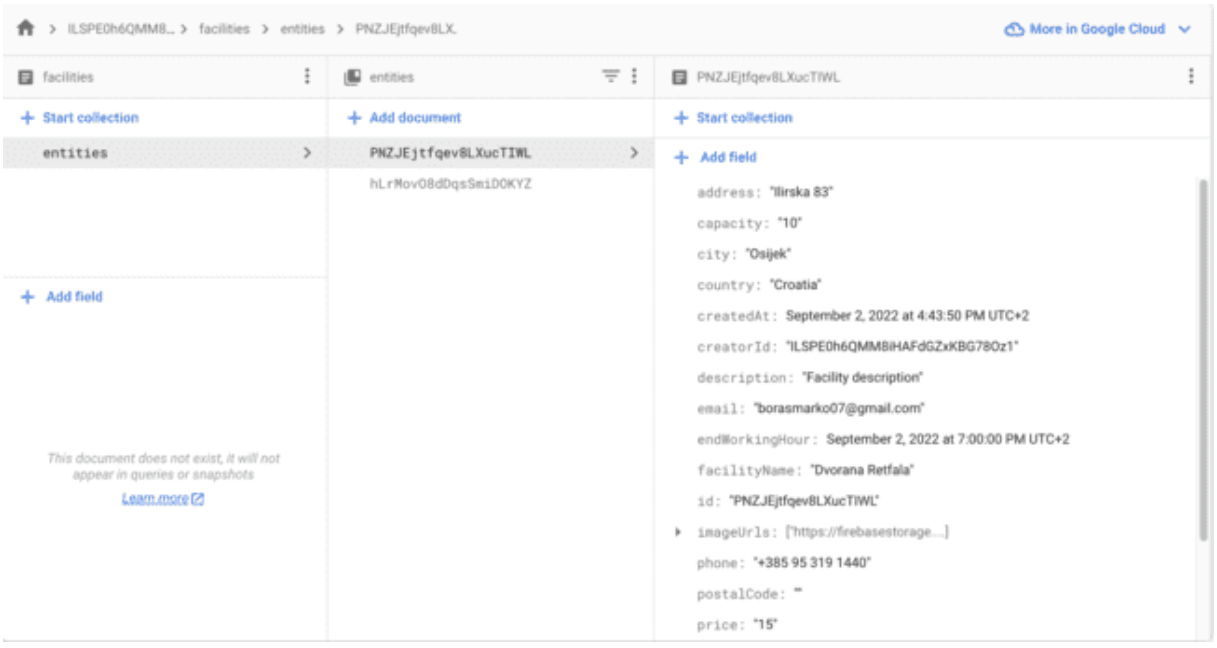
5.2. Prikaz baze podataka Firestore

Korisnički ID se sprema kao korijenska kolekcija. Korisnici ovisno o korištenju aplikacije sadrže dokumente *facilities*, *notifications*, *reservations*, a uvijek sadrže dokument *settings* unutar kojega se nalaze podaci o korisniku kao na slici 5.27.



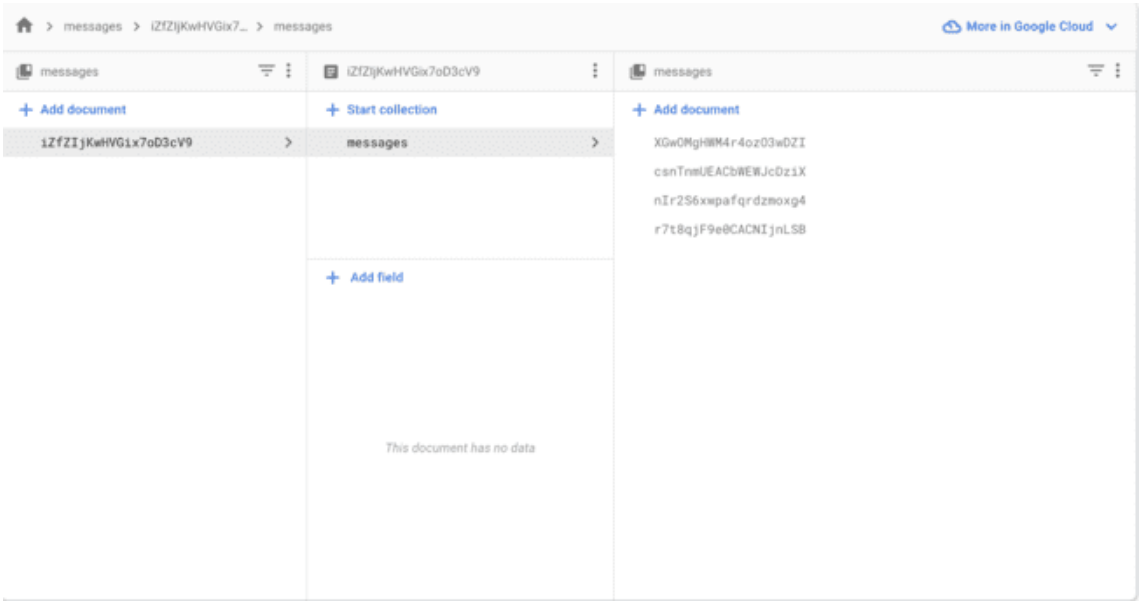
Slika 5.27. Dokument *settings*

Dokument *facilities* sastoji se od potkolekcije *entities* unutar koje su spremljeni športski objekti kao dokumenti, kaon a slici 5.28.



Slika 5.28. Prikaz športskih objekata unutar baze podataka

Dokumenti *reservations* i *notifications* su izvedeni na isti način kao i *facilities.* Kolekcija *messages* se sastoji od dokumenata koji sadrže potkolekcije unutar kojih se kao dokumenti nalaze pojedine poruke. ID chat-a se sprema u dokument *settings* određenog korisnika i tako korisnici dohvaćaju poruke kojima ima pristup.



Slika 5.29. Prikaz poruka unutar baze podataka

6. ZAKLJUČAK

U ovome diplomskom radu detaljno je prikazana izrada web aplikacije za rezerviranje športskih objekata koristeći JavaScript biblioteku React i Firebase platformu. Uporaba navedene dvije tehnologije developeru daje veliku moć iz čega možemo zaključiti da su React i Firebase napredni alati za razvoj web aplikacija. Velika pažnja prilikom izrade aplikacije je na arhitekturu aplikacije kako bih performanse i skalabilnost aplikacije bili što veći. Iste komponente su korištene na više mjesta kako bih iskoristivost koda bila što veća, a redundancije što manje. Aplikaciju je moguće koristiti na bilo kojem pregledniku, uređaju i operacijskom sustavu.

LITERATURA

[1] Google, Playfinder | Book sports facilities, https://www.playfinder.com/

[2] Google, Pitchbooking: Bookings and payments for your sports facility, https://pitchbooking.com/

[3] Ramel, David, Stack Overflow: Old .NET Framework Usage Still Beats 'Most Loved' .NET Core/.NET 5, https://visualstudiomagazine.com/articles/2021/08/03/so-survey-2021.aspx - 3. Ožujak, 2021.

[4] Google, React docs, https://reactjs.org/

[5] Google, Introducing Hooks, https://reactjs.org/docs/hooks-intro.html

[6] Google, Material Design, https://material.io/design

[7] Google, Core Concepts, https://recoiljs.org/docs/introduction/core-concepts

[8] Google, Create React App, https://github.com/facebook/create-react-app

[9] Babu, Rajesh, Building a Scalable and Modular Architecture for React-TS Applications, https://levelup.gitconnected.com/building-a-scalable-and-modular-architecture-for-react-ts-applications-e1d917250e04 - 5. Ožujak, 2022.

[10] Using media queries, https://developer.mozilla.org/en-US/docs/Web/CSS/Media_Queries/Using_media_queries -16. Rujan, 2022.

[11] Ariscrisna, Anjali, Recoil vs Redux, https://www.imaginarycloud.com/blog/recoil-vs-redux/ - 7. Travanj, 2022.

[12] Google, How React and Redux brought back MVC and everyone loved it, https://rangleio.medium.com/how-react-and-redux-brought-back-mvc-and-everyone-loved-it-rangle-io-2b35e87f2295 - 27. Travanj, 2021.

SAŽETAK

U ovome diplomskom radu izrađena je web aplikacija za rezerviranje športskih objekata. Korisnik može kreirati i stavljati na oglašavanje vlastite športske objekte ili rezervirati druge športske objekte za rekreaciju. Osim toga u aplikaciji postoje notifikacije i poruke u stvarnom vremenu kako bih korisnici pravovremenu dobili povratnu informaciju. Za izradu aplikacije potrebno je imati iskustva u Reactu, Firebaseu, HTMLu, CSSu i TypeScriptu.

Ključne riječi: CSS, Firebase, HTML, React, šport, TypeScript, web aplikacija.

ABSTRACT

For this master thesis, a web application for booking sports facilities was created. The user can create and advertise his own sports facilities or reserve other sports facilities for recreation. Also, there are real-time notifications and messages in the application for getting real-time feedback. Requirements for creating this applications are experience in React, Firebase, HTML, CSS and TypeScript.

Keywords: CSS, Firebase, HTML, React, sport, TypeScript, web application.

ŽIVOTOPIS

Marko Boras rođen je 07. prosinca 1998.godine u Žepču. Osnovnu školu upisao je i završio u Žepču. Nakon završene osnovne škole upisao je srednju školu u Katoličkom školskom centru "Don Bosco" Žepče, smjer tehničar za mehatroniku, koju završava 2017. godine i iste te godine upisuje preddiplomski studij računarstva na Fakultetu elektrotehnike, računarstva i informacijskih tehnologija u Osijeku. Nakon završenog preddiplomskog sveučilišnog studija računarstva iste godine upisuje diplomski studij računarstva na Fakulteru elektrotehnike, računarstva i informacijskih tehnologija u Osijeku, smjer Programsko inženjerstvo. Za vrijeme diplomskog studija odlazi na studijski boravak u Konya, Turska na Selcuk University. Od ožujka 2022. radi u Osječkoj tvrtci Prototyp kao web developer.

_____

Marko Boras

PRILOZI

Prilog 1. Diplomski rad u datoteci docx

Prilog 2. Diplomski rad u datoteci pdf

Prilog 3. Programsko rješenje *React* aplikacije: https://github.com/markoboras0712/Sports-Facilities-Booking