Dizajn grafičkog sučelja mobilne aplikacije

Kočiš, Imra

VELEUČILIŠTE U VIROVITICI

IMRA KOČIŠ

DIZAJN GRAFIČKOG SUČELJA MOBILNE APLIKACIJE

ZAVRŠNI RAD

Virovitica, 2022.

VELEUČILIŠTE U VIROVITICI

Preddiplomski stručni studij Računarstva, smjer Programsko inženjerstvo

Imra Kočiš

DIZAJN GRAFIČKOG SUČELJA MOBILNE APLIKACIJE

ZAVRŠNI RAD

radi stjecanja akademskog zvanja stručnog prvostupnika računarstva

Virovitica, 2022.

Veleučilište u Virovitici

Preddiplomski stručni studij Računarstva - Smjer Programsko inženjerstvo

OBRAZAC 1b

ZADATAK ZAVRŠNOG RADA

Student/ica: **IMRA KOČIŠ** **JMBAG:** **0135256033**

Imenovani mentor: Ivan Heđi, dipl. ing., v. pred.

Imenovani komentor: -

Naslov rada:

Dizajn grafičkog sučelja mobilne aplikacije

Puni tekst zadatka završnog rada:

U Vašem radu opišite principe i mehanizme izrade grafičkog sučelja mobilne aplikacije. Kao praktični primjer implementacije teorijskog dijele osmislite i izradite aplikaciju za rezervaciju stolova ugostiteljskih objekata. Za pohranu podataka koristite Firebase. Detaljno se osvrnite za grafičko sučelje aplikacije. Osim programskog rješenja, u pisanom dijelu završnog rada opišite ukratko korištene tehnologije, opišite detaljno arhitekturu sustava te opišite detaljno odabrane procese i/ili funkcije unutar same aplikacije. Prilikom opisivanja, osim neformalnih koristite i neke od formalnih metoda koje poznajete.

Datum uručenja zadatka studentu/ici: 12.10.2022.
Rok za predaju gotovog rada: 11.11.2022.

Mentor:

Ivan Heđi, dipl. ing., v. pred.

Dostaviti:
1. Studentu/ici
2. Povjerenstvu za završni rad - tajniku

Temeljna dokumentacijska kartica

DIZAJN GRAFIČKOG SUČLJA MOBILNE APLIKACIJE

Sažetak

Svrha ovoga rada je dizajniranje i kreiranje grafičkog korisničkog sučelja mobilne aplikacije. Mobilna aplikacija koja je primjer implementacije teme rada služi za rezerviranje stolova ugostiteljskih objekata. Aplikacija je podijeljena na tri dijela, jedan dio koji predstavlja stranu ugostitelja, a drugi dio koji predstavlja korisničku stranu, te autorizacijski dio koji koriste obije prijašnje navedene strane aplikacije. Ugostitelj ima mogućnost dodavanja jednog ili više ugostiteljskih objekata, a korisnik aplikacije tada ima mogućnost rezerviranja stola ili više stolova u tim ugostiteljskim objektima. Također ugostitelj ima mogućnost prihvaćanja ili odbijanja rezervacije, pregled detalja rezerviranog stola, pregled detalja svojeg odnosno svojih objekta, skeniranja generiranog QR koda koji se generira pri potvrdi rezervacije u svrhu provjere autentičnosti rezervacije. Korisnik ima dodatnu mogućnost pretraživanja objekata po nazivu, pregled detalja pojedinog objekta, pregled svojih rezervacija koje su u stanju obrade te prihvaćenih rezervacija. Svaki korisnik aplikacije ima mogućnost izrade računa za autentifikaciju ili prijave postojećim Google računom. Korištene tehnologije pri izradi aplikacije su React Native, JavaScript, Firebase autentifikacija, NativeBase komponente, za manipulaciju stanjem aplikacije korišten je Redux i React Navigation. Za dohvaćane podataka iz baze podataka korišteno je Aplikacijsko programsko sučelje (API, eng. Application programming interface) koje je izrađeno u svrhu ove aplikacije, za dohvaćanje podataka s API-a korišten je Axios.

(50 stranica, 15 slika, 17 programskih kodova)

Ključne riječi: React, React Native, JavaScript, Redux, Firebase API, Axios, Rezervacija, Ugostiteljski objekt

Mentor: Ivan Heđi, dipl. ing., v. pred

Basic documentation card

GRAPHICAL INTERFACE DESIGN OF A MOBILE APPLICATION

Abstract

The purpose of this work is to design and create a graphical user interface for a mobile application. The mobile application, which is an example of the implementation of the topic of the work, serves to reserve tables in catering establishments. The application is divided into three parts, one part that represents the caterer side, and the second part that represents the user side, and the authentication part that uses both previously mentioned sides of the application. The caterer has the option of adding one or more catering establishments, and the application user then has the option of reserving a table or more tables in those catering establishments. The caterer also has the option of accepting or rejecting the reservation, viewing the details of the reserved table, viewing the details of his or hers facilities, scanning the generated QR code that is generated when the reservation is confirmed for the purpose of verifying the authenticity of the reservation. The user has the additional option of searching for establishments by name, viewing the details of an individual establishment, viewing their reservations that are being processed and accepted reservations. Every application user has the option of creating an account for authentication or signing in to with an existing Google account. The technologies used in creating the application are React Native, JavaScript, Firebase authentication, NativeBase components, Redux and React Navigation are used to manipulate the application state. To retrieve data from the database, the Application Programming Interface (API) was used, which was created for the purpose of this application, and to fetch dana from API was used Axios

(50 pages, 15 figures, 17 code samples)

Keywords: React, React Native, JavaScript, Redux, Firebase API, Axios, Reservation, Catering establishments.

Supervisor: Ivan Heđi, dipl. ing., v. pred

Sadržaj

1. Uvod

Mobilna aplikacija je aplikacija dizajnirana isključivo za pametni telefon ili tablet računalo, iz tog razloga mobilna aplikacija mora biti izrazito optimizirana radi hardverskih ograničenja. Mobilna aplikacija je *native* aplikacija. *Native* aplikacija je izrađena za određeni operacijski sustav, u slučaju mobilnih aplikacija najpopularniji operacijski sustavi su Android i iOS. Također postoje i drugi mobilni operacijski sustavi koji nisu toliko popularni kao što su BlackBerry OS, Symbian, Windows, itd.

Mobilna aplikacija koja je tema ovog rada, izrađena je u svrhu olakšavanja posla pružateljima ugostiteljskih usluga te ubrzavanja procesa rezervacije stola u željenom ugostiteljskom objektu od strane fizičke osobe. Aplikacija omogućuje rezervaciju stola na željeni datum i vrijeme u samo nekoliko dodira putem interaktivnog korisničkog sučelja izrađenog pomoću JavaScript biblioteke React Native.

Glavni cilj koji je ujedno i ideja aplikacije je da centralizira proces rezervacije stolova u ugostiteljskim objektima koji se trenutno najčešće provodi putem telefonskih poziva ili društvenih mreža. Ideja je da vlasnik ili menadžer ugostiteljskog ili ugostiteljskih objekata imaju jednostavan pregled stanja nad svojim trenutnim i nadolazećim rezervacijama, također s druge strane da fizička osoba može na jednostavan način napraviti rezervaciju u željeno vrijeme i biti sigurna da će rezervacija ostati zabilježena.

Ovaj rad opisuje funkcionalnost mobilne aplikacije i sastoji se od tri dijela. Prvi dio opisuje korištene tehnologije prilikom izrade aplikacije, a to su: React odnosno React Native, JavaScript, Redux odnosno Redux Toolkit, Firebase autentifikacija uključujući i Google autentifikaciju, React Navigation, NativeBase komponente i Axios. Drugi dio opisuje arhitekturu aplikacije i kako se aplikacija ponaša u određenim stanjima i glavne funkcionalnosti. Treći dio opisuje navigaciju aplikacije te kako je ona realizirana.

2. Programske tehnologije

Ovaj dio rada u nastavku sadrži objašnjenja svih korištenih programskih tehnologija koje su služile za izradu ove mobilne aplikacije.

2.1. JavaScript

JavaScript programski jezik je uveden 1995. kao način dodavanja programa na web stranice u *Netscape Navigator* pregledniku [1]. JavaScript je važan programski jezik zato što je on jezik web preglednika, i bez njega ne bi postojale moderne i interaktivne web stranice i aplikacije. Njegova povezanost s preglednikom čini ga jednim od najpopularnijih programskih jezika na svijetu. Ujedno u isto vrijeme je jedan od najomraženijih programskih jezika na svijetu [2].

JavaScript je baziran na mnoštvu dobrih ideja, ali također na nekoliko loših. Dobre strane JavaScripta su to što uključuje funkcije, dinamične objekte, ne potrebu definiranja tipa varijabli, loša strana je programiranje bazirano na korištenju globalnih varijabli [2].

JavaScript je lagan, interpretirani programski jezik s funkcijama prvoklasnih objekata s uglavnom leksičkim opsegom, u prijevodu funkcija se tretira kao bilo koja druga varijabla. Manipulacija nad funkcijama je poprilično lagana korištenjem JavaScript-a, funkciju je moguće dodijeliti kao vrijednost varijabli, moguće ju je poslati kao argument drugoj funkciji, također moguće ju je vratiti kao rezultat izvođenja neke funkcije [3].

JavaScript je najpoznatiji jezik za izradu web aplikacija i stranica, ali njegova uporaba doseže daleko u drugim granama programiranja kao što je programiranje na poslužiteljskoj strani, programiranje mobilnih aplikacija, također MongoDB baza podataka koristi JavaScript [1].

2.2. React

React je moćan alat ukoliko je u pitanju programiranje interaktivnih i modernih *single page* web aplikacija i stranica. React je napravljen od strane IT giganta po imenu Meta, svima poznatiji kao Facebook [4].

React je biblioteka, nije programski okvir kao što je naprimjer Angular koji je glavni konkurent React-u. Što bi u prijevodu značilo da je React moguće koristit u nekom od programskih okvira, što inače nije praksa ali je moguće. Samim time React je puno manji po pitanju datoteka, ali ne pruža nam cjelinu kao što pruža programski okvir koji prilikom instalacije dolazi sa svime potrebnim za izradu neke aplikacije ne vezano uz mobilnu, web, ili neku drugu vrstu aplikacije [4].

2.3. React Native

Kao što je spomenuto u prijašnjem odlomku kao i React, tako je i React Native nastao pod istim imenom, ali u svrhu razvoja *native* aplikacija. Korištenjem React Native-a moguće je pisanjem istog koda u JavaScriptu programskom jeziku razviti aplikacije za Android, iOS i Windows [5].

Razlika između React-a i React Native-a je ta što je React Native JavaScript programski okvir, a React je JavaScript biblioteka, također React Native bazira se na *native* aplikacije, a React na web aplikacije [5].

Isto kao i u React-u, React Native aplikacije napisane su pomoću JavaScript programskog jezika i JavaScript XML-a poznatijeg kao JSX-a. JSX nam pruža mogućnost kombiniranja nekog *markup* jezika kao što je HTML, kontrolne logike koju postižemo JavaScript-om, te čak i stiliziranja pisanja CSS-a u web programiranju, ili *StyleSheet* stilova u React Native-u. *StyleSheet* je apstrakcija slična CSS-u.

Slika 1. Primjer JSX-a i korištenja StyleSheet stilova [6]

2.4. React Navigation

Najvažniji dio svake velike aplikacije je navigacija, koja je zaslužna za interaktivno ponašanje cijele aplikacije, također za pravilno ponašanje aplikacije te upravo je navigacija zaslužna da korisniku pokaže pravilan ekran.

Gledajući React, u React aplikacijama navigacija se rješava pomoću *React Routera*. *React Router* nam pruža programsko okruženje za manipulaciju rutama aplikacije. *Ruter* nam olakšava rukovanje navigacijom u *single page* aplikacijama [4].

Navigaciju u mobilnim aplikacijama izrađenim u React Native-u rješavamo pomoću React Navigation biblioteke. To je navigacijska biblioteka laka za korištenje, dolazi s ugrađenim navigatorima kao što je *Stack, Tab i Drawer* navigator [7].

Stack navigator je vrsta navigatora koji slaže ekrane jedan na drugi i ovisno koju akciju korisnik napravi taj će ekran biti prikazan, zbog toga svako željeno stanje aplikacije mora biti predefinirano u navigacijskoj datoteci. Korištenjem *Tab* navigatora korisnik ima na raspolaganju pregled ekrana kojima može pristupiti. *Tab* navigator dolazi sa zadanim izgledom i pozicioniranjem koje se nalazi na dnu ekrana, stil navigatora se može jednostavno konfigurirati u navigacijskoj datoteci željenog navigatora, također moguće je navigator smjestit na vrh ekrana. Oba opisana navigatora su korištena u aplikaciji također moguća je kombinacija

navigatora što je isto slučaj u spomenutoj mobilnoj aplikaciji. *Draver* navigator je vrsta navigatora koji se prikazuje ukoliko korisnik pritisne određeno mjesto na ekranu ili klizne prstom s lijeve na desnu stranu ili obrnuto. Ovaj navigator također ima na raspolaganju pregled svih mogućih ekrana kojima se može pristupiti, ali za razliku od *Tab* navigatora nije vidljiv sve dok ga korisnik ne otvori.

2.5. Redux

Prije je spomenuto kako je glavni problem JavaScripta globalne varijable, ali nekada nam je potrebno nešto, neki podatak koji je globalan, na primjer podaci od prijavljenog korisnika, zamislite da za svaki ekran u aplikaciji moramo imati prijavu ili pozvat funkciju koja će provjeriti dali je korisnik prijavljen i dati nam njegove podatke. To bi bio stvarno mukotrpan posao zato nam u pomoć dolazi Redux.

Redux je biblioteka koja nam pomaže da se aplikacije koje radimo ponašaju dosljedno, centralizira stanje i logiku važnih komponenata aplikacije kao što je na primjer autentifikacija. Također Redux moguće je koristit za bilo koji dio aplikacije bio on jednostavan ili složen [8]. Najbolji opis za Redux je taj da Redux pamti stanje aplikacije nevezano koja se akcija dogodila i prati svaki promjenu svog stanja i samim time utječe na ponašanje cijele aplikacije.

Uzmimo u obzir autentifikaciju, ukoliko se korisnik odjavi iz naše aplikacije tada se iz Redux-a uklanja korisnik te će se samim time cijelo stanje aplikacije promijeniti i aplikacija će nas odvesti na početni ekran prijave u aplikaciju. Redux je dosta složen i kompliciran za koristiti, ali nam pruža lakše praćenje i manipulaciju nad cijelom aplikacijom.

2.6. Redux Toolkit

Redux Toolikt je verzija Reduxa, koja olakšava i ubrzava proces korištenja Redux-a. Redux Toolikt je osmišljen da bude standard u pisanju Redux logike. Originalno je kreiran da pomogne programerima oko glavnih zabrinutosti po pitanju Redux-a. To su konfiguracija Redux-a koja je previše komplicirana, također konfiguriranje Redux-a zahtjeva veliku količinu koda koja se mora napisati da Redux funkcionira ispravno [9].

Konfiguracija Redux Toolikta je poprilično jednostavna, prvi i najvažniji dio je konfiguracija Redux *store* objekta. *Store* objekt sadrži svako stanje aplikacije koje je upravljano

Redux-om. Konfiguracija *store* objekta se radi na način da kao argument funkcije pošaljemo objekt koji sadrži sve *reducere*. *Reducer* je funkcija koja sadrži logiku ažuriranja podataka i spremanja istih.

Programski kod 1. Primjer konfiguracije store objekta

```
1.  import { configureStore } from '@reduxjs/toolkit';
2.  import googleReducer from './features/googleSlice';
3.  import userReducer from './features/userSlice';
4.  import formReducer from './features/registrationFormSlice';
5.  import establishmentReducer from './features/establishmentSlice';
6.  import qrReducer from './features/qrSlice';
7.
8.  export default configureStore({
9.    reducer: {
10.     googleKey: googleReducer,
11.     user: userReducer,
12.     form: formReducer,
13.     establishment: establishmentReducer,
14.     qr: qrReducer,
15.   },
16. });
```

Nakon konfiguracije *store* objekta potrebno je konfigurirati funkciju koja kao argumente prihvaća inicijalno stanje određenog objekta u *store* objektu. Objekt koji sadrži *reducer* funckije i naziv tog „*slice-a*“ u Redux Toolkitu *slice* predstavlja prethodno opisanu funkciju koja sadrži inicijalno stanje i svu logiku *reducera* [9].

Programski kod 2. Primjer konfiguracije slice funkcije

```
1.  import { createSlice } from '@reduxjs/toolkit';
2.  import instance from '../../axios/adminInstance';
3.
4.  export const googleSlice = createSlice({
5.    name: 'googleKey',
6.    initialState: {
7.      key: 'x',
8.    },
9.    reducers: {
10.     setGoogleKey: (state, action) => {
11.       state.key = action.payload;
12.     },
13.   },
14. });
15.
16. // Action creators are generated for each case reducer function
17. export const { setGoogleKey } = googleSlice.actions;
18.
19. // selectors
20. export const selectorGoogle = (state) => state.googleKey.key;
21.
22. export default googleSlice.reducer;
```

2.7. Firebase

Firebase kupljen od Google-a u 2014. godini, omogućava programerima korištenje baze podataka koja je ažurna u stvarnom vremenu, opsežne autentifikacije i autorizacije. Firebase programerima aplikacija omogućuje rad bez implementiranja *beckend* aplikacija, sve što bi *backend* dio aplikacije radio za to se pobrinuo Firebase [10].

2.7.1. Firebase Autentifikacija

Firebase autentifikacija pruža *backend* servise vezane uz autentifikaciju, lake za korištenje SDK-eve (eng. Software Development Kit), i UI (eng. User Inteface) biblioteke za autentifikaciju. Firebase autentifikacija nam omogućava prijavu u aplikaciju na razne načine kao što su korištenje email-a i lozinke, telefonskog broja, također omogućava integraciju prijave putem Google-a, Facebook-a i Twiter računa [10].

Firebase autentifikacija je usko povezana s drugim Firebase servisima, i koristi industrijske standarde poput OAuth 2.0 i OpenID Connect-a [10]. OAuth 2.0 je industrijski standardizirani protokol za autentifikaciju, fokusira se na jednostavnost pružajući specifične tokove za web, desktop i mobilne aplikacije [11]. OpneID Conenct je jednostavni identifikacijski sloj na vrhu OAuth 2.0 protokola. Klijentu omogućuje potvrđivanje identiteta krajnjeg korisnika na temelju provjere autentičnosti koju izvodi autorizacijski poslužitelj [12].

Prilikom korištenja Google autentifikacije koju nam pruža Firebase, potrebna su neka dodatna konfiguriranja koda koji omogućava integraciju s autentifikacijom. Osim koda koji omogućuje Firebase autentifikaciju potrebno je dodati Firebase SDK koji nam omogućuje korištenje Google računa za prijavu ili registraciju u aplikaciju. Također moguće je implementiranje *Sign in With Google* biblioteke te ručno napraviti tijek autentifikacije Google računom, samo je nakon toga potrebno povezati Firebase autentifikaciju tako što token koji dobijemo kao rezultat uspješne prijave proslijedimo u Firebase funkciju [10].

2.8. NativeBase komponente

NativeBase je prva mobilna biblioteka komponenti za React i React Native. Trenutno najnovija verzija je 3.0 koja dolazi s kompletnom ARIA (eng. Accessible Rich Internet

Applications) integracijom. Dolazi s velikim brojem komponenti koje je moguće dodatno stilizirati u Android, iOS i Web aplikacijama [13].

2.9. Axios

Axios je *promise-based* HTTP klijent za node.js i preglednik. Neke od pogodnosti koje nam pruža Axios su [14]:

- *XMLHttpRequest* korištenjem preglednika
- HTTP zahtjevi ukoliko se koristi node.js kao programski okvir
- Podržava *Promise* API
- Moguće je presretanje HTTP zahtjeva i odgovora
- Poništavanje HTTP zahtjeva
- Podatci dolaze u JSON (eng. JavaScript Object Notation) formatu
- Klijentska zaštita od XSRF-a (eng. Cross-site request forgery)

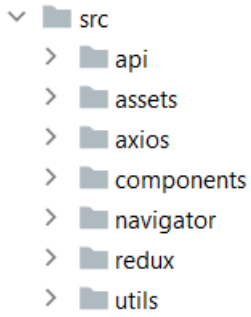
3. Programsko rješenje - mobilna aplikacija za rezervaciju

3.1. Arhitektura

Ovaj dio rada objašnjava kako je aplikacija strukturirana te kako dijelovi programskog koda utječu na rad i ponašanje aplikacije. Također će objasniti kako je riješen problem razdvajanja aplikacije na tri dijela koji su opisani u uvodu ovoga rada točnije objasniti će svrhu navigacije i autentifikacije.

3.1.1. Struktura koda po cjelinama

Struktura koda koji čini ovu aplikaciju, sastoji se od šest glavnih cjelina. Kod je smisleno razdvojen da programeru olakša snalaženje u kodu i da se ostvari mogućnost korištenja jednog koda na više mjesta.

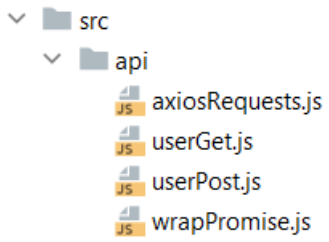


Slika 2. Struktura glavnih cjelina koda

3.1.2. Struktura API-a (eng. Application Programing Interface)

Ovaj dio strukture se brine za pozive prema API-u, tu su smještene funkcije koje u sebi imaju HTTP zahtjeve, također funkcija koja prima *promise* i obrađuje ga, tj. čeka dobivenu vrijednost ili grešku te sukladno tome ažurira korisničko sučelje.

Promise je asinkrona akcija koja će se izvršiti u budućnosti i vratiti nam neki rezultat bio on valjan podatak ili neka vrsta greške (eng. Error) [15].



Slika 3. Struktura koda vezana uz API

Naknadno je opisan primjer glavne funkcije zaslužne za pozive prema API-u. Funkcija ispod služi za pozive prema API-u s jednim parametrom koji se šalje kao *query* parametar. Funkcija prima url, parametar koji se šalje, ključ koji je identifikator navedenog parametra, te autorizacijski token, funkcija nam vraća *promise* koji se kasnije obrađuje.

Programski kod 3. Funkcija za poziv prema API-u s jednim query parametrom

```
1.  import axios from 'axios'
2.
3.  export async function getDataWithQueryParams(url, queryParam, queryKey, jwtToken) {
4.    const config = {
5.      headers: { Authorization: 'Bearer '.concat(jwtToken) },
6.      params: { [queryKey]: queryParam },
7.    };
8.    const promise = await axios
9.      .get(url, config)
10.     .then((res) => res.data)
11.     .catch((err) => console.log('axiosError', err.response.data.errorMessage));
12.
13.   return promise;
14. }
```

Druga funkcija koja je dosta slična prijašnjoj je funkcija koja ima mogućnost napraviti HTTP zahtjev s više *query* parametara. *Query* parametre i identifikacijske ključeve tih parametra prima kao dvije odvojene liste.

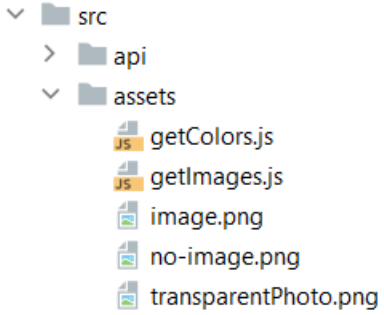
Programski kod 4. Funkcija za poziv prema API-u s više query parametara

```
1.  export async function getDataWithMultipleQueryParams(url, queryParams, queryKey,
    jwtToken) {
2.
3.    const config = {
4.      headers: { Authorization: 'Bearer '.concat(jwtToken) },
5.      params: {},
6.    };
7.
8.    queryParams.forEach((element, index) => {
9.      config.params = { ...config.params, [queryKey[index]]: element };
10.   });
11.
```

```
12.   const promise = await axios
13.     .get(url, config)
14.     .then((res) => res.data)
15.     .catch((err) => console.log('axiosError', err.response.data.errorMessage));
16.
17.   return promise;
18. }
```

3.1.3. Struktura dodataka

Ovaj dio aplikacije sadrži potrebne dodatke kao što su glavne boje aplikacije, statične slike, te dio koda koji dohvaća slike iz baze podataka. Ovaj dio koda i datoteka je vrlo malen, ali je vrlo važan iz razloga što omogućava laku manipulaciju od nekih glavnih dijelova aplikacije kao što su boje i statične slike.

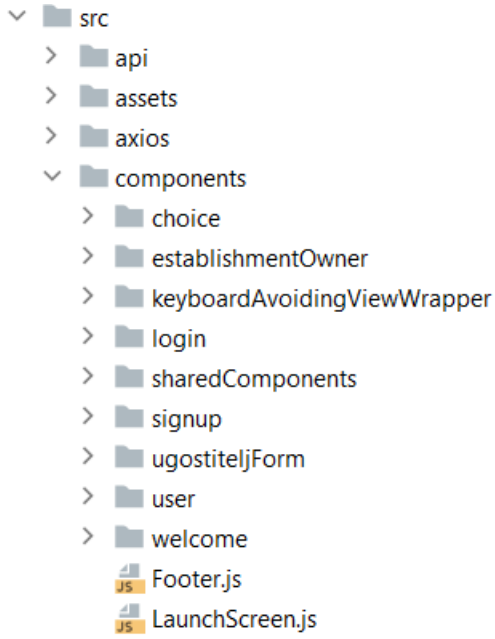


Slika 4. Struktura dodataka aplikacije

3.1.4. Struktura komponenata korisničkog sučelja

Ovaj dio programskog koda je najveći i najvažniji zato što su u ovom dijelu smještene sve komponente i svi ekrani aplikacije. Aplikacija ima nešto više od 20 različitih ekrana ako se zbroje sve strane aplikacije. Ugostiteljska je strana veća i kompleksnija nego korisnička.

Komponente su strukturirane po cjelinama ili ekranima ovisno o slučaju. Svi ekrani za korisnika su centralizirani s pripadajućim komponentama, isto tako i ugostiteljska strana. Autentifikacija i pomoćne komponente su svrstane u druge zasebne cjeline. Jednu cjelinu čini i forma za registraciju novog ugostiteljskog objekta.



Slika 5. Struktura komponenti korisničkog sučelja

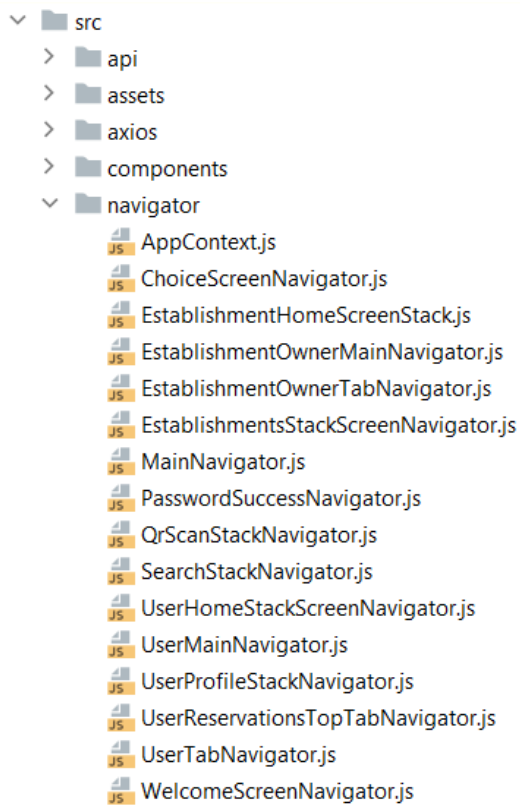
3.1.5. Struktura navigacije

Navigacija aplikacije kao što je prije rečeno rađena je pomoću React Navigation biblioteke. Pri izrade navigacije korišteni su *Stack* i *Tab* navigatori.

Glavni navigator je komponenta koja se poziva pri pokretanju aplikacije, u njemu su smješteni svi drugi navigatori koji čine tri strane aplikacije. Time rečeno glavni navigator ima određenu logiku koja odlučuje koji će se navigator prikazati. Logika svih glavnih navigatora biti će objašnjena u nastavku rada.

Ova aplikacija ima mnoštvo navigatora iz razloga što na svaku akciju koja zahtjeva prikazivanje drugog ekrana potreban je prikladan navigator.

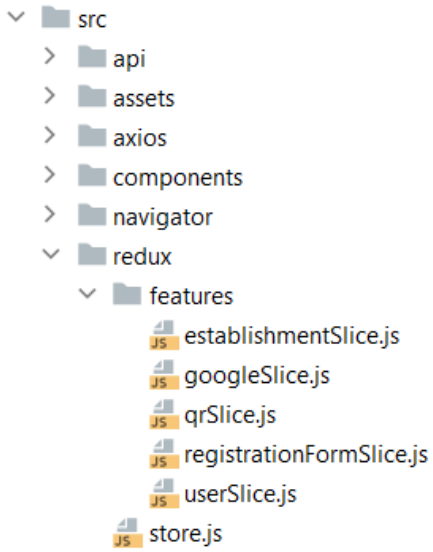
Navigatori nisu svrstani po cjelinama ovisno dali pripadaju korisničkoj, ugostiteljskoj ili autorizacijskoj strani, svi navigatori su svrstani u zasebnu cjelinu.



Slika 6. Struktura navigatora aplikacije

3.1.6. Redux struktura

Redux, objašnjen ranije, strukturiran je po zasebnim *slice* funkcijama, koje su korištene u nekim dijelovima aplikacije, te glavnom *store* objektu.

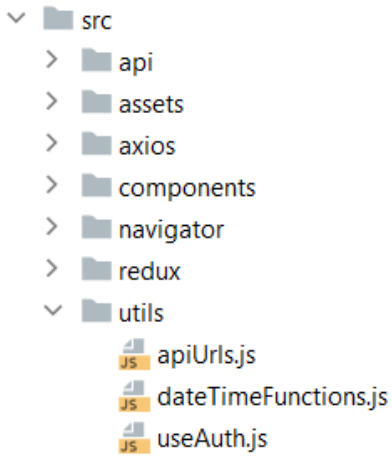


Slika 7. Struktura Redux logike

3.1.7. Struktura pomoćnih funkcija

Pomoćne funkcije znatno olakšavaju pri nepotrebnom ponavljanju iste logike u kodu koje je često slučaj u velikim aplikacijama. Ovaj dio programskog koda sastoji se od varijabla koje predstavljaju krajnje točke na korištenom API-u, funkcije za formatiranje datuma i vremena, te najvažniju funkcionalnost aplikacije *useAuth hook*.

useAuth hook je funkcionalnost koja je zaslužna za autentifikaciju u aplikaciju, te bez ove funkcije dinamičnost i izgled aplikacije ne bi bio moguć. Logika funkcije biti će opisana u nastavku rada.



Slika 8. Struktura pomoćnih funkcija

3.2. Autentifikacija

Kao što je prije rečeno autentifikacija je napravljena pomoću Firebase servisa za autentifikaciju. Aplikacija ima mogućnost registracije i naravno prijave.

3.2.1. Registracija

Registracija tj. izrada novog računa je moguća samo uz klasičnu Firebase autentifikaciju. Ukoliko korisnik odabere izradu novog računa, mora odabrati kojega će tipa biti njegov/njezin račun odnosno da li je on/ona ugostitelj ili korisnik. Prilikom izrade novog računa pozivaju se funkcije od strane Firebase servisa. Uspješna izrada novog računa uzrokuje dvije akcije, prva je izrada novog računa s Firebase funkcijom *createUserWithEmailAndPassword*, [10] koja kao parametre prima email i lozinku, te koja korisnika sprema u autentifikacijsko sučelje Firebase-a.

Programski kod 5. createUserWithEmailAndPassword funckija i primjer korištenja

```
if (email && password && password === confirmPassword) {
      auth()
        .createUserWithEmailAndPassword(email, password)
        .then((userCredentials) => {
          saveUserToFirebase({
            uid: userCredentials.user.uid,
            displayName: userCredentials.user.displayName,
            photoURL: userCredentials.user.photoURL,
            authType: 'emailpassword',
            isNewUser: true,
          });
          setSignUpSuccess(true);
        })
        .catch((error) => {
          if (error.code === 'auth/email-already-in-use') {
            errors.email = 'Odabrani email već postoji';
          }
          if (error.code === 'auth/invalid-email') {
            errors.email = 'Neispravan email';
          }
          if (error.code === 'auth/invalid-password') {
            errors.password = 'Neispravna lozinka';
          }
        });
    }
```

Druga akcija je spremanje podataka u bazu podataka, koja se izvodi s funkcijom *saveUserToFirebase*. Ova funkcija radi HTTP zahtjev s podacima informativnim podacima novo prijavljenog korisnika, kao što su ime i prezime, identifikator, tip računa, itd.

Programski kod 6. saveUserToFirebase funkcija

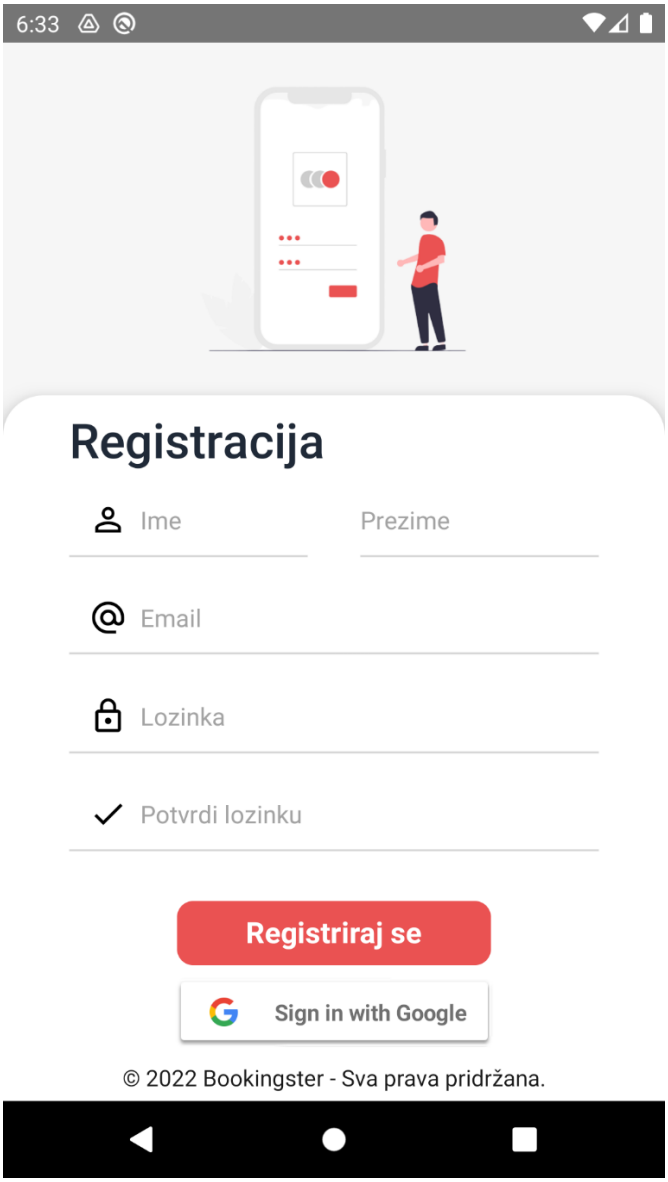
```
const saveUserToFirebase = async (userFB) => {
    if (userFB.isNewUser) {
      await userPost({
        name: userFB.authType === 'google' ? userFB.displayName : ime,
        lastname: userFB.authType === 'google' ? null : prezime,
        authType: userFB.authType,
        photoURL:
          userFB.photoURL == null
            ? `https://ui-
avatars.com/api/?name=${ime}+${prezime}&background=random&rounded=true`
            : userFB.photoURL,
        accountType: signUpType,
        UID: userFB.uid,
      }).catch((error) => {
        console.log(
          'Error in SignUp.js in saveUserToFirebase f:',
          error.response.data.errorMessage
        );
      });
    }
  };

```

Drugi način registracije je pomoću Google računa, ova akcija je ujedno i prijava, iz razloga što je Google račun vanjski dio aplikacije te se na Google autentifikaciju ne može interno utjecati. Uspješna Google autentifikacija nam daje informacije o korisniku kao što su ime i prezime, identifikator, Google profilna fotografija, email, itd. Također nam daje uvid da li je ovaj račun prvi put prijavljen u aplikaciju i to je važna informacija koja se kasnije koristi za manipulaciju navigacijom. Korisnik se također kao i kod obične registracije sprema u bazu podataka s istim podatcima kao u prijašnje objašnjenoj funkciji.

Programski kod 7. onGoogleButtonPress funkcija korištena u registraciji

```
1.  const onGoogleButtonPress = async () => {
2.      const { idToken } = await GoogleSignin.signIn();
3.      const googleCredential = auth.GoogleAuthProvider.credential(idToken);
4.
5.      const credentials = auth().signInWithCredential(googleCredential);
6.      credentials.then((credential) => {
7.        saveUserToFirebase({
8.          uid: credential.user.uid,
9.          displayName: credential.user.displayName,
10.         photoURL: credential.user.photoURL,
11.         authType: 'google',
12.         isNewUser: credential.additionalUserInfo.isNewUser,
13.       });
14.       setSignUpSuccess(true);
15.     });
16.     return credentials;
17.   };
```

Korisničko sučelje za registraciju u aplikaciju je izrađeno pomoću *NativeBase* komponenti i *StyleSheet* stilova.



Slika 9. Korisničko sučelje registracije novog korisnika

3.2.2. Prijava

Prijava na postojeći korisnički račun se također kao i registracija izvršava pomoću Firebase servisa odnosno pozivanjem određenih Firebase funkcija. Korisnik aplikacije se prijavljuje sa svojom email adresom i lozinkom. Postupak prijave izvršava se Firebase funkcijom *singInWithEmailAndPassword*, [10] ta funkcija kao argumente prima email i lozinku unesene od strane korisnika.

Programski kod 8. LoginUser funkcija

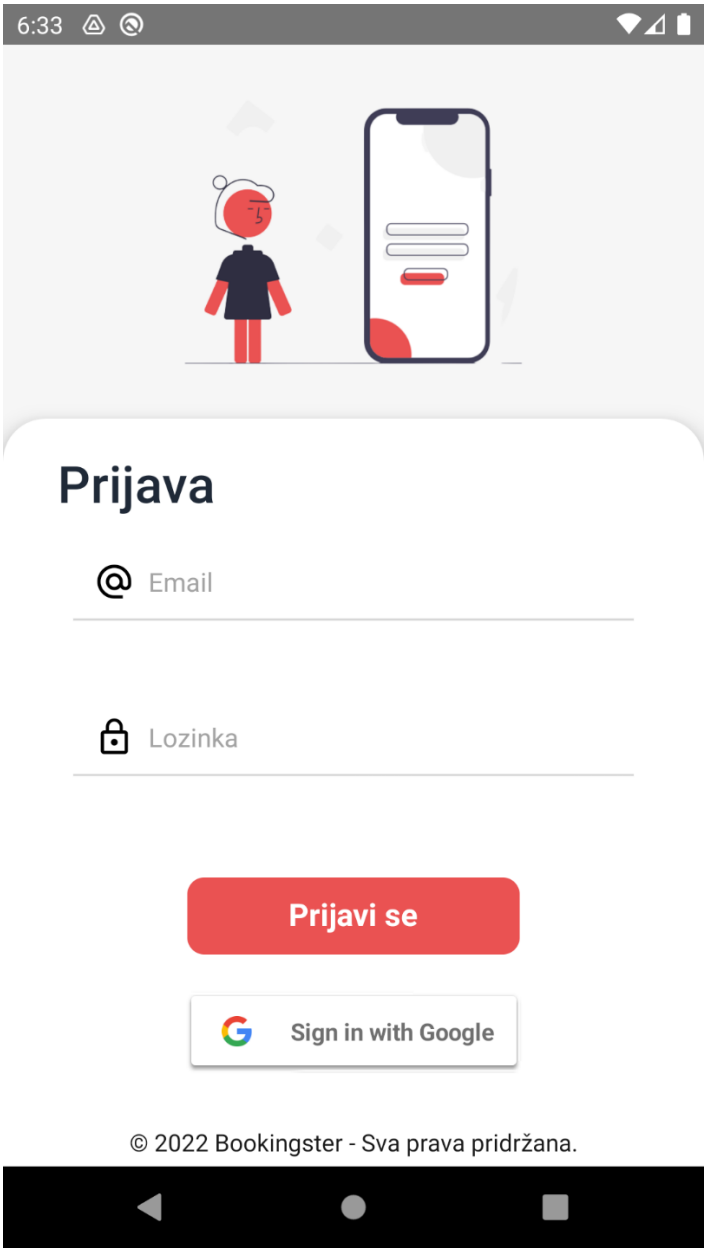
```
1.  const LoginUser = async () => {
2.      const errors = {};
3.      if (!email) errors.email = 'Email je obavezan';
4.      if (!password) errors.password = 'Lozinka je obavezna';
5.      if (email && password) {
6.        await auth()
7.          .signInWithEmailAndPassword(email, password)
8.          .catch((error) => {
9.            if (error.code === 'auth/invalid-email') errors.email = 'Neispravan email';
10.           if (error.code === 'auth/invalid-password') errors.password = 'Neispravna
    lozinka';
11.           if (error.code === 'auth/user-not-found') errors.password = 'Korisnik ne
    postoji';
12.           if (error.code === 'auth/wrong-password') errors.password = 'Kriva lozinka';
13.           console.log(error);
14.           setValidationError(errors);
15.         });
16.     }
17.     setValidationError(errors);
18.   };
```

Kao i u registraciji, prijava je moguća putem Google računa. Za uspješnu provedbu prijave koriste se iste funkcije kao i prilikom registracije, ali se radi dodatni korak provjere, odnosno provjerava se da li je ovo prva prijava u aplikaciju korištenjem tog Google računa. Ukoliko je podaci se privremeno spremaju u Redux store objekt, te se kasnije pri odabiru vrste računa oni obrađuju i spremaju u bazu podataka.

Programski kod 9. onGoogleButtonPress funkcija korištena u prijavi

```
1.  const onGoogleButtonPress = async () => {
2.      const { idToken } = await GoogleSignin.signIn();
3.      const googleCredential = auth.GoogleAuthProvider.credential(idToken);
4.      const credentials = auth().signInWithCredential(googleCredential);
5.      credentials.then((credential) => {
6.        setIsNewUser(credential.additionalUserInfo.isNewUser);
7.        if (credential.additionalUserInfo.isNewUser) {
8.          dispatch(
9.            updateUserInfo({
10.             accountType: null,
11.             isNewUser: credential.additionalUserInfo.isNewUser,
12.             photoURL: credential.user.photoURL,
13.             displayName: credential.user.displayName,
14.             uid: credential.user.uid,
15.           })
16.         );
17.       }
18.     });
19.     return credentials;
20.   };
```

Kao i kod registracije korisničko sučelje prijave je vrlo slično osim što su izostavljena nepotrebna polja. Sučelje je izrađeno pomoću *NativeBase* komponenti i *StyleSheet* stilova.



Slika 10.Izgled korisničkog sučelja prijave

3.2.3. Funkcionalnost useAuth hook

Problem autentifikacije, ujedno i dinamičnosti aplikacije, riješen je pomoću useAuth *hooh-a*. useAuth *hook* je prilagođena funkcionalnost koja koristi radu ove aplikacije.

Prednost ove funkcionalnosti je ta što drastično smanjuje količinu koda potrebnog za svrhu autentifikacije, centralizira logiku autentifikacije, te najvažnija stvar omogućava korištenje React *Suspense* komponente pri pokretanju aplikacije.

React *Suspense* [16]komponenta nam pruža mogućnost „čekanja“ podatka koje ćemo u nekom vremenu u budućnosti dobiti od strane API-a. *Suspense* nije biblioteka za dohvaćanje podataka, to je mehanizam koji koriste upravo takve biblioteke. *Suspense* komunicira s React-om i prenosi informaciju da li su neki podaci spremni ili ne, s tom spoznajom React može ažurirati korisničko sučelje ili biti u stanju čekanja dok određeni podaci ne dođu.

useAuth *hook* se aktivira svaki puta kada se nešto primjeni po pitanju autentifikacije te samim time odrađuje neku funkcionalnost, npr. dohvaća prijavljenog korisnika iz baze podataka i Firebase autentifikacijskog sučelja, provodi akciju spremanja tokena uređaja u bazu podataka. Kada se te akcije uspješno obave *promise* se „riješi“ (eng. *resolve*) te *Suspense* ažurira korisničko sučelje.

Programski kod 10. useAuth hook

```
1.  import auth from '@react-native-firebase/auth';
2.  import messaging from '@react-native-firebase/messaging';
3.  import axios from 'axios';
4.  import wrapPromise from '../api/wrapPromise';
5.  import instance from '../axios/adminInstance';
6.  import { userSlice } from '../redux/features/userSlice';
7.  import store from '../redux/store';
8.
9.  let storeState = store.getState();
10. let resolve;
11. let currentPromise;
12.
13. const initialCurrentUser = new Promise((r) => {
14.   resolve = r;
15. });
16.
17. const saveTokenToDatabase = async (token, userUid) => {
18.   instance.patch(`user/deviceToken?deviceToken=${token}&UID=${userUid}`,
    null).catch((error) => {
19.     console.log('PatchError', error);
20.   });
21. };
22.
23. auth().onIdTokenChanged(async (firebaseUser) => {
24.   if (firebaseUser) {
25.     messaging()
26.       .getToken()
27.       .then((token) => saveTokenToDatabase(token, firebaseUser.uid));
28.     await auth()
29.       .currentUser.getIdToken(true)
30.       .then(async (idToken) => {
31.         store.dispatch(userSlice.actions.updateUserInfo({ jwt: idToken }));
32.
33.         currentPromise = await instance
34.           .get(`user?UID=${firebaseUser.uid}`)
35.           .then((response) => {
36.             store.dispatch(userSlice.actions.login(response.data.user));
37.             return response;
38.           })
39.           .catch((error) => {
```

```
40.             console.log('Error in useAuth.js onIdTokenChanged, get request',
    error.response);
41.           });
42.       })
43.       .catch((error) => {
44.         console.log('Error fetching JWT from firebase =>', error.code);
45.         if (error.code === 'auth/user-token-expired') {
46.           store.dispatch(userSlice.actions.logout());
47.           auth().signOut();
48.         }
49.       });
50.
51.     messaging().onTokenRefresh((token) => {
52.       saveTokenToDatabase(token, firebaseUser.uid);
53.     });
54.   }
55.   resolve(currentPromise);
56. });
57.
58. export default function useAuth() {
59.   const currentUserState = wrapPromise(initialCurrentUser);
60.
61.   if (!(currentUserState.read() instanceof Error)) {
62.     storeState = store.getState();
63.     return storeState.user.user;
64.   }
65.   throw currentUserState.read();
66. }
67.
```

U kodu je prikazana funkciju po imenu *wrapPromise*, ta funkcija kao argument prima *promise* te ga obrađuje i čeka njegovo rješavanje. Funkcija je u petlji sve dok se *promise* ne izvrši s podacima ili s greškom. Ova funkcija vraća *read* funkciju s kojom je moguće „pročitati“ stanje izvršenog *promise-a*.

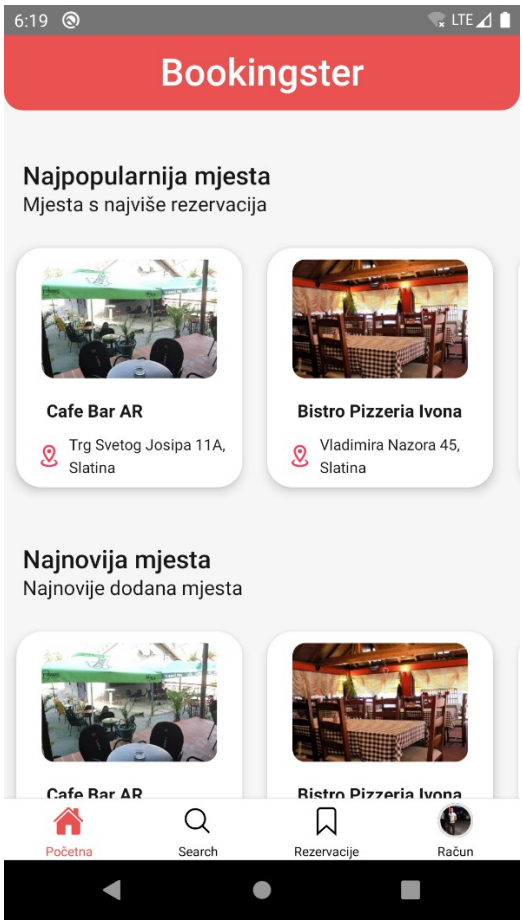
3.3. Korisnički dio

Korisnički dio aplikacije koji se prikazuje onima koji su se uspješno registrirali i prijavili kao korisnik aplikacije. Korisnik aplikacije ima mogućnosti pretraživanja ugostiteljskih objekata prema nazivu, dostupan pregled pojedinih objekata koji su najpopularniji, pregleda detalja pojedinog objekta, rezervacije stola u željenom ugostiteljskom objektu, pregled svojih rezervacija koje su u statusu obrade i one koje su potvrđene i ima mogućnost pregleda svojeg računa.

3.3.1. Početni zaslon korisničkog dijela

Ovaj ekran se prikazuje uvijek prilikom rezervacije ili prijave u račun. Glavni dio početnog ekrana su dvije horizontalne liste objekata kroz koje se kreće pomoću klizanja po ekranu s lijeve na desno ili obrnuto. Ove dvije liste predstavljaju najpopularnije i najnovije

dodane ugostiteljske objekte. Opće informacije pojedinog ugostiteljskog objekta smještene su u karticu koja se sadrži od glavne fotografije objekta, njegova naziva i adrese.



Slika 11. Početni ekran korisničkog dijela aplikacije

Podaci koje čine ugostiteljske objekte dohvaćaju se iz baze podataka korištenjem HTTP zahtijeva na određenu krajnju točku API-a. Podaci se dohvaćaju uz pomoć Axios funkcije koja se poziva preko *useSWR hook-a* [17]. *useSWR hook* je alat za dohvaćanje podataka, upravo onaj alat koji koriste mehanizmi poput React *Suspense* komponente. Axios funkcija koja dohvaća podatke kao argumente prima krajnju točku API-a kao URL i autorizacijski token od prijavljenog korisnika, te kao rezultat vraća *promise* koji obrađuje *useSWR hook.*

Komponenta koja prikazuje kartice objekata smještena je u *Suspense* komponentu kao „dijete“. Pošto *Suspense* komponenta nema nikakav izgled i ne zauzima nikakvo mjesto na ekranu moramo specificirati neku zamjenu (eng. *fallback*) za komponentu koja je na čekanju dok podatci ne budu spremni. Zamjena za ovu komponentu je „kostur“ kartice.

Podatke koje dobijemo iz baze podataka smješteni su u listu objekata. Radi učinkovitosti koda i brzine aplikacije koristimo komponentu koja služi upravo za takvu vrstu uporabe. Komponenta koja nam pomaže u realizacije velike ili male liste podataka zove se *FlatList*.

FlatList [18] je komponenta koja pruža lagan način izrade efikasne klizajuće liste podataka. Prilikom korištenja ove komponente moramo specificirati podatke koji trebaju biti smješteni u listu, nije bitno da li je lista velika i ima ogromne objekte koji sadrže dodatne objekte i liste ili je lista s jednom vrstom podataka kao što je na primjer jedna riječ ili broj. Također moramo specificirat kako će izgledati svaka komponenta koju će činiti podaci dobiveni iz liste podataka.

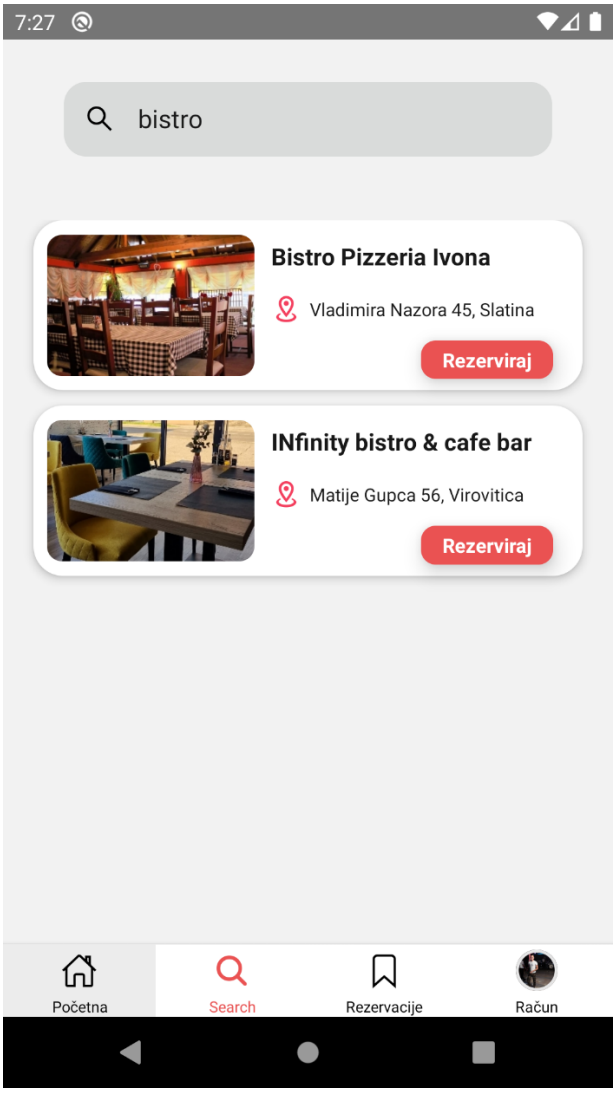
Programski kod 11. Funkcija koja vraća horizontalnu listu ugostiteljskih objekata

```
1.  function CustomFlatList({ navigation }) {
2.    const { data } = useSWR([establishmentUrl, ADMIN_TOKEN], getData);
3.    return (
4.      <View style={{ marginVertical: 5, flex: 1 }}>
5.        <FlatList
6.          nestedScrollEnabled
7.          horizontal
8.          showsHorizontalScrollIndicator={false}
9.          data={data.establishments}
10.         renderItem={({ item }) => <CubeCard navigation={navigation} item={item} />}
11.         keyExtractor={(item) => item.oib}
12.       />
13.     </View>
14.   );
15. }
```

U kodu je prikazana *CubeCard* komponenta. Ova komponenta kao parametre prima objekt iz liste podataka dobivenih iz baze podataka, te objekt stanja navigacije koji služi za navigaciju na neki drugi ekran, kao rezultat vraća pojedinu karticu koja se kao takva nalazi na početnom ekranu. Navigacija će biti detaljno objašnjena u nastavku rada.

3.3.2. Pretraživanje ugostiteljskih objekata

Pronaći željeni ugostiteljski objekt moguće je obaviti traženjem naziva objekta. Algoritam na strani API-a pretražuje nazive objekata i vraća sva mjesta koja se podudaraju s unosom od strane korisnika. Objekt je moguće pronaći unosom dijela ili cijelog naziva.



Slika 12. Izgled sučelja za pretragu

Podaci koje dobijemo slanjem HTTP istog su oblika kao i podaci koje smo dobili za izradu horizontalne liste na početnom ekranu. Također problem dinamičnosti i sigurnosti dobivanja podataka rješavamo pomoću kombinacije *Suspense* komponente *useSWR hook-a* i Axios funkcije. Zbog kompleksnijeg zahtjeva problematika dohvaćanja podataka uključuje neke dodatne korake kao što su spremanje podataka u Redux store objekt. Te *FlatList* koji nam ovog puta vraća vertikalnu klizajuću listu podataka dobiva podatke upravo iz Redux-a.

Komponenta u kojoj se nalazi ova *FlatList-a* specifična je zato što se koristi i na korisničkoj, ali i na ugostiteljskoj strani aplikacije. Funckija koja čini ovu komponentu napravljena je dinamički i podržava takav način implementacije. Funkcija odnosno komponenta kao parametre prima mnoštvo podataka neki od tih su parametri koji se šalju

putem API zahtijeva, objekt navigacije i podaci za navigaciju, itd. Sastoji se od *useSWR hook-a*, Redux selektora i JSX dijela koda.

Programski kod 12. Komponenta koja vraća vertikalnu listu ugostiteljskih objekata na korisničkoj i ugostiteljskoj strani aplikacije

```
1.  function CustomLongCardFlatList({
2.    navigation,
3.    queryParams,
4.    queryKey,
5.    screenToNavigate,
6.    nestedScreenToNavigate,
7.    buttonText,
8.    buttonNavigationScreen,
9.    apiUrl,
10. }) {
11.   const user = useSelector(selectUser);
12.   const establishments = useSelector(selectEstablishment);
13.   const dispatch = useDispatch();
14.   const effectTriggeredRef = useRef(false);
15.
16.   const { data, error } = useSWR([apiUrl, queryParams, queryKey, user.jwt],
    getDataWithQueryParams);
17.
18.   useEffect(() => {
19.     if (!effectTriggeredRef.current && data) {
20.       effectTriggeredRef.current = true;
21.       dispatch(updateEstablishmentState(data.establishments
22.     } else if (!effectTriggeredRef.current && error) {
23.       Alert.alert('Nešto je pošlo po zlu', error, [{ text: 'Zatvori' }]);
24.     }
25.   }, [data, dispatch, error]);
26.   return (
27.     <View flex={1} my={5}>
28.       <FlatList
29.         ListHeaderComponent={
30.           buttonText === 'Detalji' && (
31.             <HStack paddingBottom={5} flex={1} justifyContent="center">
32.               <VectorIcon
33.                 onPress={() => navigation.navigate('AddNewEstablishment')}
34.                 name="plus-circle"
35.                 size={60}
36.                 color={primary}
37.               />
38.             </HStack>
39.           )
40.         }
41.         nestedScrollEnabled
42.         showsVerticalScrollIndicator={false}
43.         ListEmptyComponent={
44.           queryKey === 'keyword' && (
45.             <VStack flex={1} justifyContent="center" alignItems="center">
46.               <Text fontSize="md">Nema rezultata</Text>
47.             </VStack>
48.           )
49.         }
50.         data={establishments}
51.         renderItem={({ item }) => (
52.           <LongCard
53.             navigation={navigation}
54.             buttonText={buttonText}
55.             item={item}
```

```
56.             screenToNavigate={screenToNavigate}
57.             nestedScreenToNavigate={nestedScreenToNavigate}
58.             buttonNavigationScreen={buttonNavigationScreen}
59.           />
60.         )}
61.         keyExtractor={(item) => item.oib}
62.       />
63.     </View>
64.   );
65. }
```

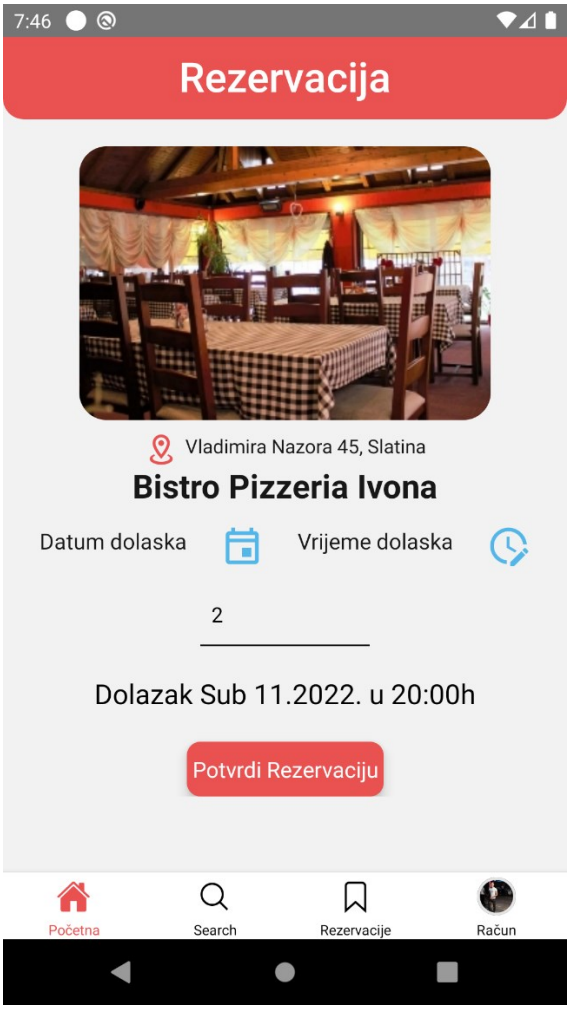
3.3.3. Rezervacije

Mogućnost rezervacije stola dostupna je kada odaberemo željeno mjesto. Odabirom rezervacije prikazuje se sučelje s mogućnošću odabira datuma i vremena dolska te je također potrebno odabrati broj mjesta. Koju nam prikazuje (Slika 13)

Problematika rezervacije se rješava na API strani, na nama je da pošaljemo valjane podatke, te onda nam API daje odgovor koji sadrži grešku tj. neuspješni pokušaj rezervacije ili uspješnu rezervaciju i potvrdu.

Programski kod 13. Funkcija za slanje zahtjeva rezervacije

```
1.  const handleReservationRequest = async () => {
2.      const apiObj = {
3.        establishmentOIB: route.params.oib,
4.        establishmentOwner: route.params.owner,
5.        reservedBy: user.UID,
6.        nameOnReservation: user.fullName,
7.        reservedFrom: {
8.          year: fromDate.getFullYear(),
9.          month: fromDate.getMonth(),
10.         day: fromDate.getDate(),
11.         hours: from.getHours(),
12.         minutes: from.getMinutes(),
13.       },
14.       places,
15.     };
16.     const response = await handleApiCall(apiObj);
17.     if (response) {
18.       Alert.alert(
19.         'Zahtjev uspješno poslan',
20.         'Vaša rezervacija je zabilježena kod ugostitelja, detalje rezervacije možete
    provjerit pod rezervacijama u obradi',
21.         [{ text: 'OK', onPress: () => console.log('OK Pressed') }]
22.       );
23.     }
24.   };
```



Slika 13. Sučelje za podnošenje zahtjeva rezervacije

Nakon uspješno podnesenog zahtjeva za rezervaciju, rezervaciju je moguće pronaći pod dijelom u navigatoru „Rezervacije“, te pod „Rezervacije u obradi“.

3.4. Ugostiteljski dio

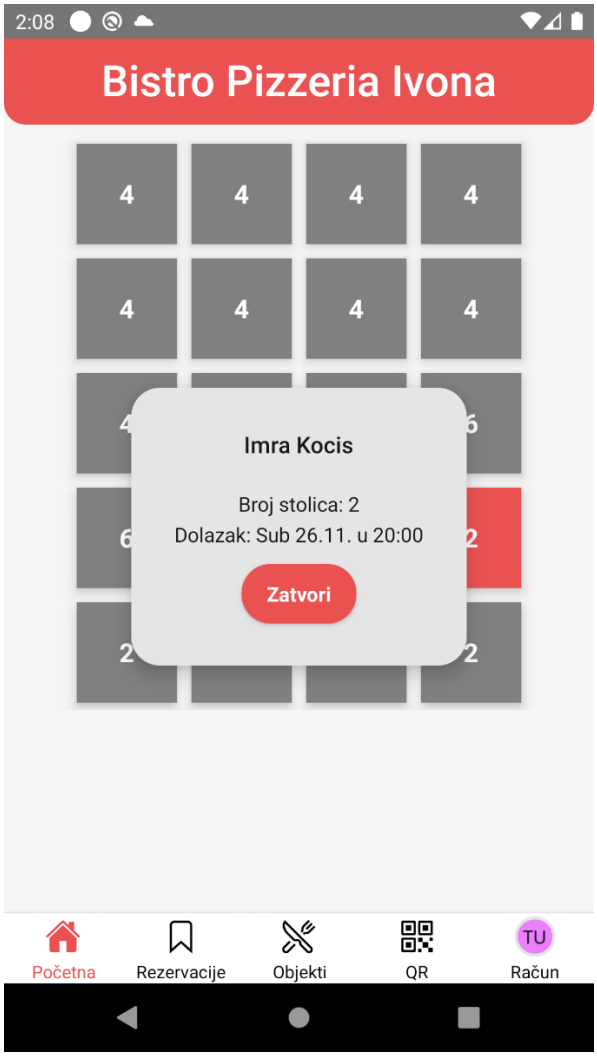
Ovaj dio aplikacije prikazuje se osobama koje su se registrirale odnosno prijavile kao ugostitelj. Sastoji se od ekrana za pregled svojih objekata i stolova u odabranom objektu, ekran za pregled nadolazećih rezervacija, ekran za pregled detalja svojeg objekata, kamera za skeniranje QR koda, te ekrana za pregled svojeg korisničkog računa.

3.4.1. Početni zaslon ugostiteljskog dijela

Početni ekran ugostiteljskog dijela prikazuje objekt/e koji su registrirani od strane prijavljenog ugostitelja. Nakon odabira željenog objekta prikazuje se ekran sa svim stolovima tog objekta, koji nudi pregled stanja svih stolova.

Komponenta koja nam daje pregled svih objekata registriranih na prijavljenog ugostitelja je ona ista komponenta korištena na korisničkoj strani za pregled objekata prilikom pretrage (Programski kod 12).

Ukoliko je stol rezerviran označen je drugačijom bojom, također moguć je pregled detalja rezervacije klikom na rezervirani stol.



Slika 14. Sučelje pregleda stanja rezervacija u odabranom ugostiteljskom objektu

Podatke za prikaz ovih informacija ugostitelju dobivamo iz baze podatka. Kao i na korisničkoj strani problematiku slanja zahtjeva na API rješavamo korištenjem *Suspense* komponente, *useSWR hook-a* i *Axios* funkcija za slanje HTTP zahtijeva.

Sve komponente na ugostiteljskoj strani kontrolirane su Redux-om, tj. svi dinamični podaci koje koriste komponente na ugostiteljskoj strani povezane su i akcija na jednom ekranu utjecati će na drugi ekran.

Komponenta koja nam prikazuje stolove je također *FlatList* kao i u većini primjera na korisničkoj strani. *FlatList* ima razne mogućnosti stiliziranja i utjecanja na položaj komponenti koje ona prikazuje.

Programski kod 14. FlatList komponenta koja prikazuje sve stolove u pojedinom objektu

```
1.  <FlatList
2.    style={{ paddingTop: '2%' }}
3.    onRefresh={onRefresh}
4.    refreshing={refreshing}
5.    numColumns={4}
6.    columnWrapperStyle={{ justifyContent: 'center' }}
7.    data={establishment.filter((obj) => obj.oib === route.params.oib)[0].tables}
8.    renderItem={({ item }) => <Table item={item} />}
9.    keyExtractor={(item) => item.id}
10.  />
```

U gore navedenom kodu možemo vidjeti vrijednost, koju prima *FlatList* komponenta, *numColumns* (hrv. broj kolona) koja ima vrijednost četiri. To znači da će se sve komponente koje će *FlatList* napraviti biti raspoređene u četiri kolone.

3.4.2. Zahtjev za rezervacijom

Ovaj ekran prikazuje sve rezervacije koje su u stanju obrade, točnije koje ugostitelj može prihvatiti ili odbiti. Prilikom potvrde ugostitelj dobiva potvrdu da je rezervacija uspješno potvrđena i detalje rezerviranog stola može provjeriti na početnom ekranu. Ukoliko odluči odbiti rezervaciju ugostitelju se otvara modal s porukom da li je siguran da želi odbiti rezervaciju. Bilo koja odluka ugostitelja biti će prikazana kao obavijest na korisničkoj strani aplikacije.

Problematiku dobivanja podataka rješavamo na isti način kao i u prije objašnjenim primjerima. Kod ovog ekrana postoji dodatna ,ali jako bitna stavka a to je ažuriranje stanja rezervacije. Logika aplikacije baziranja je na tome da postoji pet stanja rezervacije, a to su: stanje 0 što znači da je rezervacija u procesu obrade, to su sve rezervacije koje će vidjeti

ugostitelj na ovome ekranu, nakon toga rezervacija sa stanjem 1 ovo stanje inicira da je rezervacija odobrena, stanje 2 koje označava odbijenu rezervaciju, stanje 3 koje označava završenu rezervaciju i posljednje stanje 4 koje označava otkazanu rezervaciju od strane korisnika.

Prilikom obrade rezervacije tj. kada ugostitelj odobri ili dobije rezervaciju radi se isti poziv na API koji šalje informaciju o novom stanju rezervacije koja se kao takva obrađuje na poslužiteljskoj strani aplikacije. Pošto je manipulacija rezervacijama u obradi napravljena da bude kompletno ažurna logika u pozadini je kompliciranija nego ostatak logike koji se bavi slanjem poziva na poslužiteljsku stranu aplikacije.

Programski kod 15. Funkcija koja obrađuje odliku ugostitelja oko stanja rezervacije

```
1.  const handleOwnerDecision = async (decision, establishment) => {
2.      isPressed.current = true;
3.      const config = {
4.        headers: { Authorization: 'Bearer '.concat(user.jwt) },
5.        params: {
6.          newStatus: decision,
7.          reservationId: establishment.id,
8.          establishmentOIB: establishment.establishmentOIB,
9.        },
10.     };
11.     await axios
12.       .patch(reservationUrlStatus, null, config)
13.       .then(async (res) => {
14.         if (res.data.success) {
15.           if (decision === 1) modalHeader.current = 'Rezervacija prihvaćena';
16.           if (decision === 2) modalHeader.current = 'Rezervacija odbijena';
17.           await mutate({ ...data }).then((response) => {
18.             dispatch(updateEstablishmentReservationState(response.reservations));
19.           });
20.
21.           const configEsth = {
22.             headers: { Authorization: 'Bearer '.concat(user.jwt) },
23.             params: {
24.               UID: user.UID,
25.             },
26.           };
27.
28.           await axios
29.             .get(establishmentOwnerUrl, configEsth)
30.             .then((response) => {
31.               dispatch(updateEstablishmentState(response.data.establishments));
32.               isPressed.current = false;
33.               setModalVisible(true);
34.             })
35.             .catch((error) => {
36.               isPressed.current = false;
37.               console.log(
38.                 'Error in ReservationFlatList, establishments get in
    handleOwnerDecision',
39.                 error.response.data
40.               );
41.             });
```

```
42.         }
43.       })
44.       .catch((error) => {
45.         setModalVisible(false);
46.         if (error.response.data.errorMessage) {
47.           isPressed.current = false;
48.           Alert.alert('Nije moguće potvrditi rezervaciju',
    error.response.data.errorMessage, [
49.             { text: 'Zatvori', style: 'cancel' },
50.           ]);
51.         }
52.       });
53.   };
54.
```

U gore navedenom primjeru možemo vidjeti dva API poziva od kojih je jedan ugniježđen u drugi. Razlog takve implementacije je ukoliko je poziv izvršen uspješno tada moramo ažurirati stanje ugostiteljskih objekata koji ujedno sadrže stanja i od svih stolova koji se nalaze u pojedinom objektu. Također to moramo napraviti kako se ugostitelju više ne bi prikazivale rezervacije koje je potvrdio ili odbio.

3.4.3. Ugostiteljski objekti

Ovaj ekran ugostitelju prikazuje sve njegove objekte gdje ima mogućnosti pregleda detalja pojedinog objekta dodavanja novog ili brisanja pojedinog objekta. Ova komponenta je ista onda komponenta korištena na korisničkoj strani za pretragu objekata i na ugostiteljskoj strani za pregled objekata na početnom ekranu. Razlika je što ova komponenta ima dodatni dio u sučelju za dodavanje novog ugostiteljskog objekta.

Dodavanje ugostiteljskog objekta je prvi korak koji ugostitelj mora proći prilikom izrade ovog računa. Dakle aplikaciju ne može koristiti ugostitelj ukoliko nema niti jedan ugostiteljski objekt. Novi ugostiteljski objekt dodaje se putem forme koja je podijeljena na pet ekrana. Koraci dodavanja su unos općih podataka kao što je naziv objekta OIB, broj telefona, adresa, nakon toga unos radnog vremena, dodavanje naslovne fotografije objekta, te dodavanje stolova. Ukoliko su svi podaci uneseni ispravno korisnik je preusmjeri na početni ekran aplikacije.

Slika 15. Detalji ugostiteljskog objekta

Detalji ugostiteljskog objekta se mogu pronaći na korisničkoj i ugostiteljskoj strani aplikacije. Jedina razlika je što je na ugostiteljskoj strani dostupna akcija brisanja objekta.

Informacije dobivene za izradu ovog ekrana dobivene su s poslužiteljske strane aplikacije na isti način rješavanja problematike ako i u prijašnjim primjerima.

3.4.4. QR skener

QR skener daje ugostitelju mogućnost pregleda autentičnosti rezervacije. Svaka potvrđena rezervacija na ugostiteljskoj strani utječe na izradu QR koda koji se kao takav može pronaći na korisničkoj strani pod potvrđenim rezervacijama.

Prilikom uspješnog skeniranja QR koda ugostitelju se prikazuju detalji rezervacije te dali je rezervacija valjana.

QR skener je izrađen pomoću *react-native-vision-camera* biblioteke i detektora QR kodova iz bibliteke *vision-camera-code-scanner*. Skener funkcionira tako što konstantno skenira i pokušava očitati kod, kada ga očita automatski daje rezultat skeniranja.

Programski kod 16. Funkcija koja omogućuje korištenje kamere kao QR skener

```
1.  function QrScanScreen({ navigation }) {
2.    const [hasPermission, setHasPermission] = useState(false);
3.    const [cameraPermission, setCameraPermission] = useState(true);
4.    const [barcodes, setBarcodes] = useState([]);
5.    const isFocused = useIsFocused();
6.    const devices = useCameraDevices();
7.    const device = devices.back;
8.
9.    const onError = useCallback((error) => {
10.     console.log(error);
11.   }, []);
12.   const dispatch = useDispatch();
13.   const qrData = useSelector(selectQrData);
14.   const camera = useRef();
15.   const handleSettingOpen = async () => {
16.     await Linking.openSettings();
17.     const status = await Camera.getCameraPermissionStatus();
18.     setHasPermission(status === 'authorized');
19.     setCameraPermission(status === 'authorized');
20.   };
21.   const frameProcessor = useFrameProcessor((frame) => {
22.     'worklet';
23.     const detectedBarcodes = scanBarcodes(frame, [BarcodeFormat.QR_CODE]);
24.     runOnJS(setBarcodes)(detectedBarcodes);
25.   }, []);
26. }
27. if (!cameraPermission)
28.     return (
29.       <View flex={1} justifyContent="center" alignItems="center">
30.         <Heading textAlign="center" size="lg" fontWeight={400}>
31.           Molimo odobrite kameru u opcijama aplikacije za nastavak
32.         </Heading>
33.         <Button my={10} size="md" backgroundColor={primary} onPress={handleSettingOpen}>
34.           Otvori postavke
35.         </Button>
36.       </View>
37.     );
38.   return (
39.     cameraPermission &&
40.     hasPermission &&
41.     isFocused &&
42.     device != null && (
43.       <>
44.         <Camera
45.           onError={onError}
46.           style={StyleSheet.absoluteFill}
47.           device={device}
48.           isActive
49.           frameProcessor={frameProcessor}
50.         />
51.       </>
52.     )
53.   );
```

3.5. Navigacija

Navigacija je odigrala veliku ulogu u izradi ove aplikacije iz razloga što bilo kakva interakcija koja korisnika vodi na neki drugi ekran ne bi bila moguća. Kao što je rečeno prije problem navigacije riješen je pomoću ReactNavigation bibliteke.

Ukoliko se vratimo nazad i pogledamo arhitekturu aplikacije pod sekcijom strukture navigacije možemo vidjeti mnoštvo datoteka od koje je svaka jedan navigator osim *AppContext.js* datoteke.

AppContext.js skripta nam isporučuje instancu React Context [19] funkcije. Context nam pruža način slanja podataka kroz stablo komponenti bez potrebe za ručnim prosljeđivanjem podataka tj. vrijednosti nekih varijabli na svakoj razini. Kako bi Context funkcionirao moramo postaviti neke inicijalne vrijednosti koje mogu biti varijable, funkcije, itd. Također moramo „registrirati“ komponentu koja će pružati korištenje svih tih inicijalnih vrijednosti. U našem slučaju to je bio navigator koji je bio prvi u hijerarhiji stabla navigatora. Nakon toga preostaje nam napraviti komponentu koja će koristit funkcionalnosti izrađenog Context-a.

Prilikom izrade navigacije ova funkcija nam je došla u pomoć kada smo trebali proslijediti neke vrijednosti u komponentu navigatora. Mogli smo to napraviti ručno, ali dokumentacija ReactNavigatio-a preporučuje upravo ovaj pristup slanja nekih vrijednosti u komponentu navigatora.

3.5.1. Glavni navigator

Ovaj navigator je glavna komponenta aplikacije, tj. ova komponenta je na vrhu stabla komponenti cijele aplikacije. Komponenta MainNavigator sadrži navigatore glavnih strana aplikacije, kao što je glavni navigator korisničke i ugostiteljske strane, te navigator koji vodi do autentifikacije. Navigator radi na način da poziva funkciju useAuth te dobiva stanje prijavljenog korisnika. Nakon toga po objektu vraćanog od strane useAuth funkcije odlučuje koji će navigator vratiti kao rezultat izvođenja funkcije.

Programski kod 17. Komponenta MainNavigator

```
1.  function MainNavigator() {
2.    const userFb = useAuth();
3.    const user = useSelector(selectUser);
4.    const dispatch = useDispatch();
5.    const [isNewUser, setIsNewUser] = useState(false);
```

```
6.    const LogoutUserFromFirebase = () => {
7.      dispatch(logout());
8.      auth().signOut();
9.    };
10.   const contextValue = useMemo(
11.     () => ({
12.       isNewUser,
13.       setIsNewUser,
14.       LogoutUserFromFirebase,
15.     }),
16.     [isNewUser, setIsNewUser, LogoutUserFromFirebase]
17.   );
18.   const renderNavigator = () => {
19.     if (userFb) {
20.       if (user.accountType === null) {
21.         return <ChoiceScreenNavigator />;
22.       }
23.       if (user.accountType === 0) {
24.         return <UserMainNavigator />;
25.       }
26.       if (user.isNewUser && user.accountType === 1) {
27.         return <EstablishmentRegistrationForm />;
28.       }
29.       if (user.accountType === 1) {
30.         return <EstablishmentOwnerMainNavigator />;
31.       }
32.     }
33.     return <WelcomeScreenNavigator />;
34.   };
35.
36.   return <AppContext.Provider
    value={contextValue}>{renderNavigator()}</AppContext.Provider>;
37. }
```

Objekt vraćen od strane useAuth funkcije sadrži sve podatke o korisniku, kao što je tip njegova računa. U aplikaciji postoje dva različita tipa, a to su ugostitelj i korisnik, korisnik se označava s brojem 0, a ugostitelj s brojem 1. Na taj način dolazimo do pravilnog funkcioniranja glavnog navigatora. Ukoliko korisnik ne postoji točnije ukoliko korisnik nije prijavljen u aplikaciju, prikazat će se ekran koji osobi nudi izbor prijave ili registracije. Ukoliko se osoba odlučila prijavit u aplikaciju, i odabere prijavu s Google računom, a prije toga nije nikada bila prijavljena u aplikaciju, osobi će se prikazati navigator koji prikazuje ekran s mogućnošću odabira tipa računa.

3.5.2. Glavni navigator korisničke strane

Glavni navigator korisničke strane možemo vidjeti u glavnom navigatoru u jednom od povratnih parametara funkcije. UserMainNavigator sastoji se od dva navigatora, jedan koji služi za korisnički račun i njegovu manipulaciju, a drugi je uistinu glavni navigator korisničke strane, a to je UserTabNavigator.

Ovaj navigator sačinjen je od svih pod navigatora koji čine cijelo korisničko sučelje korisničke strane aplikacije. U ovom navigatoru nalaze se četiri pod navigatora, a to su: navigator početnog ekrana, navigator koji služi za pretragu ugostiteljskih objekata, navigator rezervacija na korisničkoj strani i navigator korisničkog računa.

UserTabNavigator je *Tab* navigator što znači da korisnik ima pregled svih ekrana, u ovom slučaju drugih navigatora na koje se može pozicionirati. Svaki ekran je označen ikonom i nazivom, osim navigatora korisničkog računa kojeg karakterizira profilna slika prijavljenog korisnika.

3.5.3. Glavni navigator ugostiteljske strane

Glavni navigator ugostiteljskog dijela je EstablishmentOwnerMainNavigator komponenta, ali kao i kod korisničke stane ovaj navigator ima ispod sebe dva navigatora jedan za korisnički račun, a drugi navigator je onaj koji čini cijelo korisničko sučelje ugostiteljskog dijela aplikacije.

EstablishmentOwnerTabNavigator komponenta sastoji se od *tab* navigatora kojeg čine četiri pod navigatora i jednog ekrana, a to su: navigator početnog ekrana ugostiteljske strane, ekran dolazećih rezervacija, navigator ugostiteljskih objekata registriranih na prijavljenu osobu, navigator koji predstavlja QR skener i navigator korisničkog računa.

Kao i na korisničkoj strani EstablishmentOwnerTabNavigator je *tab* navigator.

4. Zaključak

Početak razvijanja ove aplikacije krenuo je tako što smo prvo morali osmisliti izgled svih ekrana, što uključuje pozicioniranje komponenti na ekranu i njihov izgled, boje koje će se koristiti kao primarne i sekundarne kao pozadina, itd. Također bilo je potrebno osmisliti atraktivnu i interaktivnu aplikaciju. Najveći zadatak je bio riješiti problematiku logike aplikacije, tj. kako će sve biti povezano na što moguće jednostavniji način korisniku aplikacije. Naravno sam dizajn se mijenjao prilikom razvoja aplikacije radi olakšavanja korisniku korištenje određenih stavki aplikacije.

Razvoj ove aplikacije postignut je korištenjem JavaScipt programskog jezika uz pomoć React Native programskog okruženja. Također velik doprinos atraktivnom korisničkom sučelju donijele su NativeBase komponente. Pri izradi apliakcije korištene su i neke druge biblioteke koje su omogućile izradu interaktivnog sučelja. Radi korištenja React Native programskog okruženja ova aplikacija se može koristiti na Android uređajima, ali i na iOS uređajima.

Izrada ovog programskog rješenja doprinijelo je velikim dijelom boljeg razumijevanja React Native-a i React-a u kombinaciji s JavaScript programskim jezikom. Također pomogla je u osmišljavanju riješenja i boljem razmišljanju pri rješavanju problema tijekom razvoja velikih i kompleksnih aplikacija.

Popis literature

[1] M. Haverbeke, Eloquent JavaScript 3rd edition, 2018.

[2] D. Crockford, JavaScript: The Good Parts, O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472, 2008.

[3] »MDN Web Docs,« [Mrežno]. Dostupno: https://developer.mozilla.org/en-US/docs/Web/JavaScript. [Pokušaj pristupa 26. listopad 2022.].

[4] E. P. Alex Banks, Learning React Functional Web Development with React and Redux, O'Reilly Media, Inc. , 1005 Gravenstein Highway North, Sebastopol, CA 95472., 2016.

[5] B. Eisenman, Learning React Native Building Native Mobile Apps with JavaScript, O'Reilly Media, Inc. , 1005 Gravenstein Highway North, Sebastopol, CA 95472., 2018.

[6] »React Native,« Meta Platforms, Inc., [Mrežno]. Dostupno: https://reactnative.dev/docs/stylesheet. [Pokušaj pristupa 27. listopad 2022.].

[7] »React Navigation,« Expo, Software Mansion, Callstack,, [Mrežno]. Dostupno: https://reactnavigation.org/. [Pokušaj pristupa 27. listopad 2022.].

[8] »Redux,« Dan Abramov and the Redux documentation authors, 2015. [Mrežno]. Dostupno: https://redux.js.org/. [Pokušaj pristupa 27. listopad 2022.].

[9] »Redux Toolkit,« Dan Abramov and the Redux documentation authors, 2015. [Mrežno]. Dostupno: https://redux-toolkit.js.org/. [Pokušaj pristupa 27. listopad 2022.].

[10] »Firebase,« Google, [Mrežno]. Dostupno: https://firebase.google.com/. [Pokušaj pristupa 30. listopad 2022.].

[11] »OAuth,« Stytch, [Mrežno]. Dostupno: https://oauth.net/2/. [Pokušaj pristupa 30. listopad 2022.].

[12] »OpenID,« OpenID, [Mrežno]. Dostupno: https://openid.net/connect/. [Pokušaj pristupa 30. listopad 2022.].

[13] »NativeBase,« GeekyAnts, [Mrežno]. Dostupno: https://nativebase.io/. [Pokušaj pristupa 30. listopad 2022.].

[14] »Axios,« [Mrežno]. Dostupno: https://axios-http.com/docs/intro. [Pokušaj pristupa 2. studeni 2022.].

[15] »MDN Web Docs,« [Mrežno]. Dostupno: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise. [Pokušaj pristupa 2 studeni 2022].

[16] »React,« Meta Platforms, Inc., [Mrežno]. Dostupno: https://17.reactjs.org/docs/concurrent-mode-suspense.html. [Pokušaj pristupa 2. studeni 2022.].

[17] »SWR,« Varcel Inc., [Mrežno]. Dostupno: https://swr.vercel.app/. [Pokušaj pristupa 3. studeni 2022.].

[18] »React Native,« Meta Platforms, Inc., [Mrežno]. Dostupno: https://reactnative.dev/docs/flatlist. [Pokušaj pristupa 3 studeni 2022].

[19] »React,« Meta Platforms, Inc., [Mrežno]. Dostupno: https://reactjs.org/docs/context.html. [Pokušaj pristupa 5 studeni 2022].

Popis slika

Popis programskih kodova

Veleučilište u Virovitici

OBRAZAC 5

IZJAVA O AUTORSTVU

Ja, IVA Kočiš

izjavljujem da sam autor/ica završnog/diplomskog rada pod nazivom

DIZAJN GRAFIČKOG SUČELJA MOBILNE APLIKACIJE

Svojim vlastoručnim potpisom jamčim sljedeće:
- da je predani završni/diplomski rad isključivo rezultat mog vlastitog rada koji se temelji na mojim istraživanjima i oslanja se na objavljenu literaturu, a što pokazuju korištene bilješke i bibliografija,
- da su radovi i mišljenja drugih autora/ica, koje sam u svom radu koristio/la, jasno navedeni i označeni u tekstu te u popisu literature,
- da sam u radu poštivao/la pravila znanstvenog i akademskog rada.

Potpis studenta/ice

Kočiš

Veleučilište u Virovitici

OBRAZAC 6

ODOBRENJE ZA POHRANU I OBJAVU ZAVRŠNOG/DIPLOMSKOG RADA

Ja IMRA KOŠIŠ

dajem odobrenje za objavljivanje mog autorskog završnog/diplomskog rada u javno dostupnom digitalnom repozitoriju Veleučilišta u Virovitici te u javnoj internetskoj bazi završnih radova Nacionalne i sveučilišne knjižnice bez vremenskog ograničenja i novčane nadoknade, a u skladu s odredbama članka 83. stavka 11. Zakona o znanstvenoj djelatnosti i visokom obrazovanju (NN 123/03, 198/03, 105/04, 174/04, 02/07, 46/07, 45/09, 63/11, 94/13, 139/13, 101/14, 60/15, 131/17).

Potvrđujem da je za pohranu dostavljena završna verzija obranjenog i dovršenog završnog/diplomskog rada. Ovom izjavom, kao autor navedenog rada dajem odobrenje i da se moj rad, bez naknade, trajno javno objavi i besplatno učini dostupnim:

a) široj javnosti
b) studentima i djelatnicima ustanove
c) široj javnosti, ali nakon proteka 6 / 12 / 24 mjeseci (zaokružite odgovarajući broj mjeseci).

Potpis studenta/ice

Košiš

U Virovitici, 07. 11. 2022.

**U slučaju potrebe dodatnog ograničavanja pristupa Vašem završnom/diplomskom radu, podnosi se pisani obrazloženi zahtjev.*