Univerzitet u Beogradu

Fakultet organizacionih nauka

Katedra za elektronsko poslovanje

React Native

Seminarski rad iz Internet Tehnologija

Nastavnik: Prof. dr Zorica Bogdanović Studenti: Marina Jevtić 326/2018 Emilija Zdravković 220/2018

Mihailo Ivković 330/2018

Beograd, 2022

SADRŽAJ

[1 UVOD 5](#_Toc108347062)

[2 POSTAVKE I INSTALACIJA 6](#_Toc108347063)

[3 SEARCH BAR 10](#_Toc108347064)

[4 RESTAURANT INFO KARTICA 11](#_Toc108347065)

[5 STILIZACIJA 17](#_Toc108347066)

[5.1 Kreiranje teme 18](#_Toc108347067)

[5.2 Fontovi 21](#_Toc108347068)

[5.3 Spacer 22](#_Toc108347069)

[5.4 Tipography 24](#_Toc108347070)

[5.5 Sređivanje style fajlova 25](#_Toc108347071)

[6 LISTA RESTORANA 27](#_Toc108347072)

[7 NAVIGACIJA 29](#_Toc108347073)

[7.1 Instalacija 29](#_Toc108347074)

[7.2 Postavljanje navigacije 29](#_Toc108347075)

[7.3 Ikonice u okviru navigacije 31](#_Toc108347076)

[7.4 Restrukturiranje navigacije 32](#_Toc108347077)

[7.5 Pojedinačni navigatori 34](#_Toc108347078)

[8 SERVISI 36](#_Toc108347079)

[8.1 Mock-up servisi 36](#_Toc108347080)

[8.2 Geocoding Mock-up 41](#_Toc108347081)

[8.3 Favourites servis 46](#_Toc108347082)

[9 DETALJI RESTORANA 54](#_Toc108347083)

[9.1 Tranzicija sa liste svih restorana 54](#_Toc108347084)

[9.2 Renderovanje detalja restorana 56](#_Toc108347085)

[9.3 Meni restorana 57](#_Toc108347086)

[10 MAPE 60](#_Toc108347087)

[10.1 Postavljanje React Native mape 60](#_Toc108347088)

[10.2 Pretraga u okviru mape 61](#_Toc108347089)

[10.3 Unošenje restorana u okviru mape 62](#_Toc108347090)

[11 AUTENTIKACIJA 68](#_Toc108347091)

[11.1 Postavljanje i testiranje autentikacije 68](#_Toc108347092)

[11.2 Servis za autentikaciju 70](#_Toc108347093)

[11.3 Navigacija na osnovu autentikacije 71](#_Toc108347094)

[11.4 Screen-ovi autentikacije 72](#_Toc108347095)

[11.4.1 AccountScreen 74](#_Toc108347096)

[11.4.2 LoginScreen 77](#_Toc108347097)

[11.4.3 RegisterScreen 80](#_Toc108347098)

[12 SEKCIJA SETTINGS 85](#_Toc108347099)

[12.1 Postavljanje opcija u okviru Settings-a 85](#_Toc108347100)

[12.2 Podaci o korisniku u okviru Settings-a 87](#_Toc108347101)

[12.3 Lista omiljenih restorana 89](#_Toc108347102)

[13 ANIMACIJE 91](#_Toc108347103)

[14 POSTAVLJANJE APLIKACIJE NA EXPO 92](#_Toc108347104)

[15 KAMERA MODUL 93](#_Toc108347105)

[15.1 Povezivanje sa SettingsScreen-om 93](#_Toc108347106)

[15.2 Funkcionalnosti Expo Kamere 93](#_Toc108347107)

[15.3 Kreiranje fotografije 95](#_Toc108347108)

[15.4 Skladištenje slike i postavljanje na profil 95](#_Toc108347109)

[16 FIREBASE FUNKCIJE 97](#_Toc108347110)

[16.1 Location mock fajlovi u okviru firebase funkcija 98](#_Toc108347111)

[16.2 Restaurant mock u okviru firebase funkcija 99](#_Toc108347112)

[16.3 Deployment 100](#_Toc108347113)

[17 INTEGRACIJA SA GOOGLE MAPAMA 102](#_Toc108347114)

[18 NARUČIVANJE 108](#_Toc108347115)

[18.1 Postavljanje Checkout-a u okviru menija 108](#_Toc108347116)

[18.2 Unos kreditne kartice 109](#_Toc108347117)

[18.3 Stripe tokeni 110](#_Toc108347118)

[18.4 Kreiranje korpe 111](#_Toc108347119)

[18.5 Plaćanje 117](#_Toc108347120)

[KORIŠĆENA I REFERENTNA LITERATURA 118](#_Toc108347121)

# 1 UVOD

Kroz istoriju kreiranja mobilnih aplikacije kompanije su nailazile na problem kompatibilnosti operativnih sistema iOS i Android. Dok je iOS napisan u programskom jeziku C, a zatim u Swiftu, a Android u Javi, a zatim u Kotlinu, nije bilo moguće isti kod koristiti za obe platforme, jer je bilo potrebno neophodno da se slažu sa programskim jezikom platforme. To je zahtevalo velike izdatke za prilagođavanje aplikacije drugoj platformi.

Prvo rešenje bio je web development, koji je počeo sa Cordovom i nastavio se sa Ionicom, gde su se aplikacije otvarale u okviru browser-a, tako da nije bilo važno na kojoj smo platformi. Na taj način su kreirane hibridne aplikacije, dostupne na obe platforme. Jedini problem bio je to što su aplikacije zbog prikazivanja u okviru web-view-a bile sporije. Takođe, dizajn ovih platformi je drugačiji, tako da aplikacije nisu isto izgledale na oba uređaja.

React Native predstavlja jedno od rešenja koje spaja najbolje mogućnosti prethodnih rešenja – obezbeđuje lakoću kodiranja kao u okviru Native načina, ali takođe obezbeđuje korišćenje istog koda na obe platforme, kao kod Web načina. Prvo ovakvo rešenje bio je Xamarin, a zatim su naišli React Native i Flutter. Suština ovih rešenja jeste kompajliranje koda u Native za datu platformu, tako da se bez problema pokreće na datom uređaju i ima najbolje performanse. React Native je kreiran od strane Facebook-a.

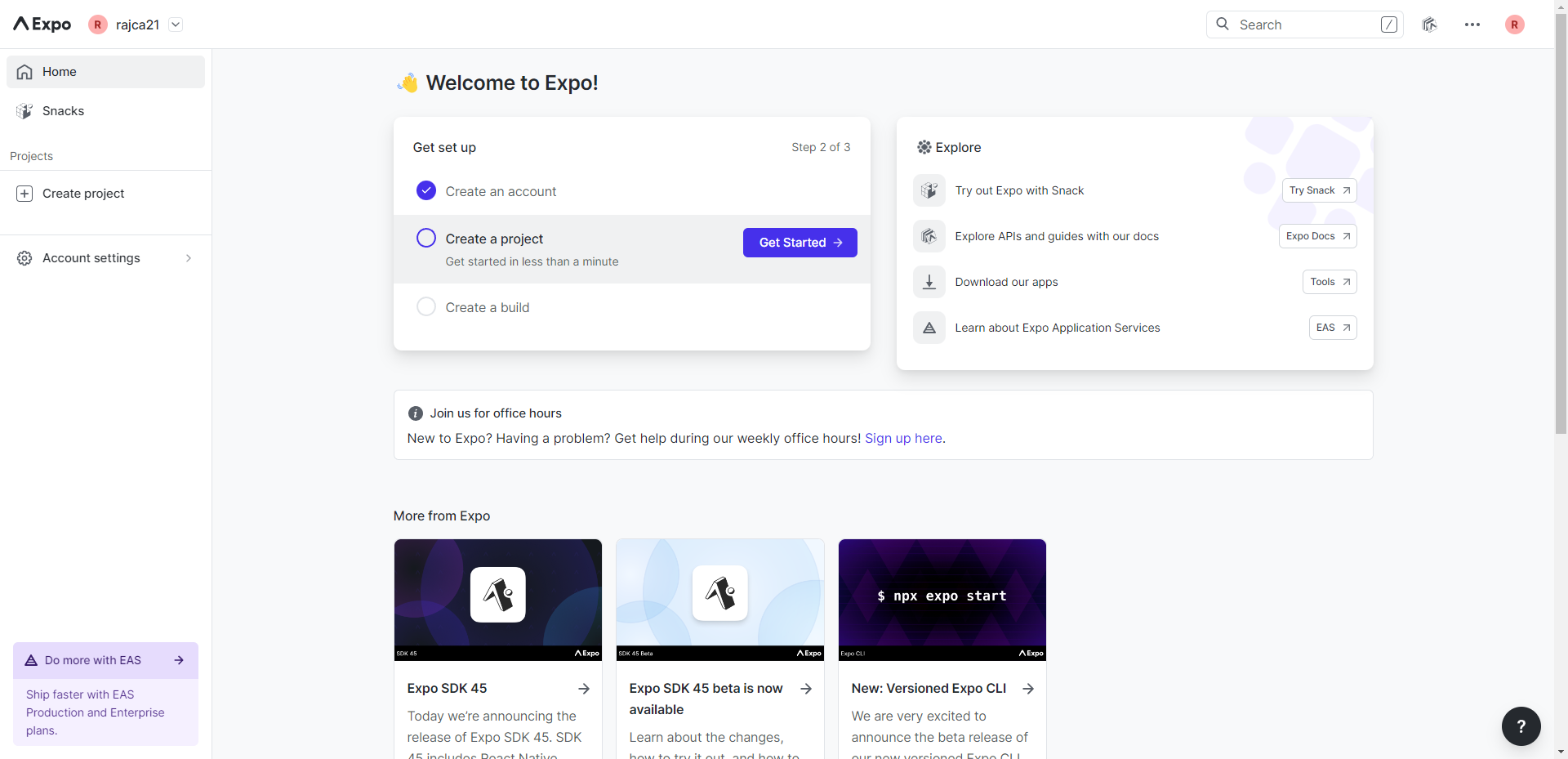


Slika 1: Istorija razvoja mobilnih aplikacija

React Native predstavlja jedno od najboljih rešenja za razvoj mobilnih aplikacija trenutno. Razlozi za to su brojni, kao na primer postojanje široke zajednice koja ga koristi, tako da nije teško naći različita rešenja, veliki broj korisnika je nailazio na slične probleme itd. Takođe, React Native koriste velike IT kompanije (kao npr. Facebook, UberEats, Discord, Instagram, Skype itd.), što znači da je dobro testiran i skaliran s njihove strane. I ono najvažnije je ono što smo već naznačili, kod koji napišemo u okviru React Native-a, možemo koristiti na obe platforme. Takođe, pruža mogućnost pisanja u JavaScirptu, Javi, Swiftu itd.

# 2 POSTAVKE I INSTALACIJA

U okviru React Native dokumentacije imamo dve mogućnosti za instalaciju ovog frameworka, korišćenjem Expo-a ili putem CLI-a. CLI ima određene negativne strane, komplikovanu instalaciju i nemogućnost testiranja iOS aplikacije na Windows-u. Zbog toga ćemo koristiti Expo. Kako bismo lako pokretali aplikaciju koju kreiramo, potrebno je da kreiramo nalog na Expo-u.



Slika 2: Expo Dashboard nakon prijavljivanja

Aplikacije koje pokrećemo ćemo kreirati u okviru opcije Snacks.



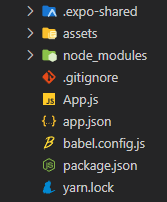
Slika 3: Kod i simulator za pokretanje aplikacije

Expo možemo instalirati i na našem uređaju i ulogovati se sa kreiranim nalogom, kako bismo pokretali aplikacije i sa uređaja. Takođe, neophodno je da instaliramo node.js i yarn na našem uređaju. Prvo treba instalirati node.js i zatim otvoriti node.js command prompt i u okviru njega ukucati komandu npm install –g yarn. Zatim u okviru našeg foldera projekta treba da unesemo komandu yarn. Zatim, kako bismo mogli da pokrećemo našu aplikaciju unosimo komandu yarn add expo-cli –global. Komandom yarn start dobijamo QR kod, pomoću kog možemo pokrenuti aplikaciju na našem uređaju.



Slika 4: QR kod za učitavanje aplikacije

Sam projekat kreiramo u okviru terminala, tako što unesemo komandu expo init order, što će biti naziv naše aplikacije. Izabraćemo opciju blank i expo će u okviru našeg foldera order kreirati sve neophodne foldere i fajlove za početak izrade aplikacije.



Slika 5: Instalirani fajlovi

Početni izgled App.js fajla:

// react imports

import React from 'react';

// react native imports

import { StatusBar } from 'expo-status-bar';

import { StyleSheet, Text, View } from 'react-native';

export default function App() {

  return (

    <View style={{ flex: 1, alignItems: 'center', justifyContent: 'center'}}>

      <Text>React native blank canvas</Text>

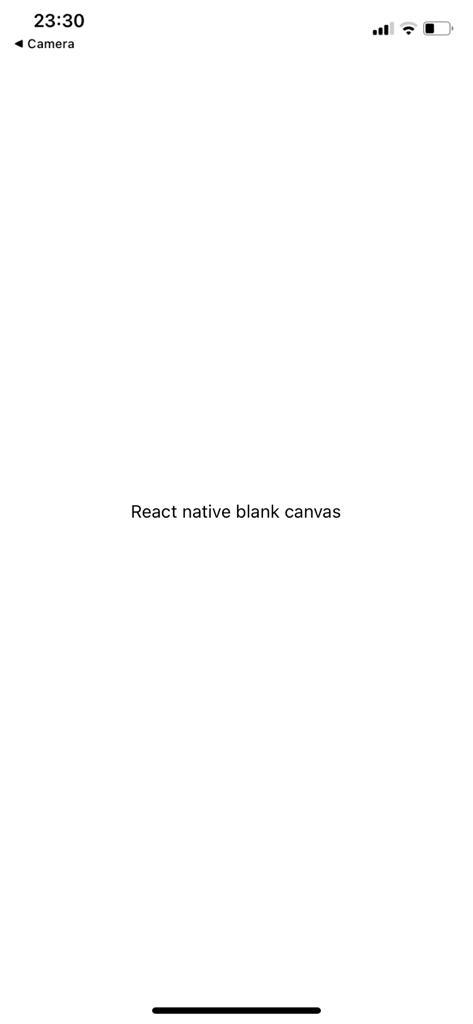
    </View>

  );

}

const styles = StyleSheet.create({

});



Slika 6: Početni izgled aplikacije na našem uređaju

# 3 SEARCH BAR

U okviru naše aplikacije želimo da pretražujemo lokacije i na osnovu te pretrage izbacujemo listu rezultata korisniku. Za polje pretrage koristićemo već ugrađen element Searchbar iz biblioteke react-native-paper. U okviru našeg layout-a unećemo ovaj element i on će izgledati ovako:

// react imports

import React from "react";

// react native imports

import { StatusBar as ExpoStatusBar } from 'expo-status-bar';

import { StatusBar, StyleSheet, SafeAreaView, Text, View, Platform } from 'react-native';

import { Searchbar } from "react-native-paper";

export default function App() {

  return (

    <>

      <SafeAreaView style={{ flex: 1, marginTop: StatusBar.currentHeight}}>

        <View style={{ padding: 16, backgroundColor: 'green'}}>

          <Searchbar />

        </View>

        <View style={{ flex: 1, padding: 16, backgroundColor: 'blue'}}>

          <Text>list</Text>

        </View>

      </SafeAreaView>

      <ExpoStatusBar style="auto" />

    </>

  );

}

const styles = StyleSheet.create({

});



Slika 7: Searchbar

# 4 RESTAURANT INFO KARTICA

Svaki od restorana u okviru aplikacije imaće svoju karticu sa osnovnim informacijama o restoranu. Zbog toga ćemo ovu komponentu smestiti u poseban folder i zatim je importovati u naš fajl App.js radi bolje čitljivosti.

export default function App() {

  return (

    <>

      <RestaurantsScreen />

      <ExpoStatusBar style="auto" />

    </>

  );

}

// react imports

import React from "react";

// react native imports

import { Searchbar } from "react-native-paper";

import { StatusBar, StyleSheet, SafeAreaView, Text, View } from "react-native";

import { RestaurantInfo } from "../components/restaurant-info-card.component";

export const RestaurantsScreen = () => (

  <SafeAreaView style={styles.container}>

    <View style={styles.search}>

      <Searchbar />

    </View>

    <View style={styles.list}>

      <RestaurantInfo />

    </View>

  </SafeAreaView>

);

const styles = StyleSheet.create({

  container: {

    flex: 1,

    marginTop: StatusBar.currentHeight,

  },

  search: {

    padding: 16,

  },

  list: {

    flex: 1,

    padding: 16,

    backgroundColor: "blue",

  },

});

// react imports

import React from "react";

// react native imports

import { Text } from "react-native";

export const RestaurantInfo = () => <Text>Restaurant Info </Text>;

Takođe, u okviru komponente Restaurant info čitaćemo sve podatke o restoranu iz baze, ali ćemo za sada postaviti neke default podatke, samo da bismo videli kako funkcioniše ova komponenta. U okviru template-a koje vraća ova komponenta ćemo ispisivati te podatke.

export const RestaurantInfo = ({ restaurant = {} }) => {

    const {

      name = "Some Restaurant",

      icon,

      photos = [

        "https://www.foodiesfeed.com/wp-content/uploads/2019/06/top-view-for-box-of-2-burgers-home-made-600x899.jpg",

      ],

      address = "100 some random street",

      isOpenNow = true,

      rating = 4,

      isClosedTemporarily,

    } = restaurant;

    return <Text>{name}</Text>;

  };

U okviru komponente Restaurant Info Card napravićemo karticu korišćenjem komponente Card iz biblioteke react-native-paper i u okviru te kartice uneti informacije o restroranu.

    return (

      <Card elevation={5} style={styles.card}>

        <Card.Cover key={name} style={styles.cover} source={{ uri: photos[0] }} />

        <Text style={styles.title}>{name}</Text>

      </Card>

    )

  };

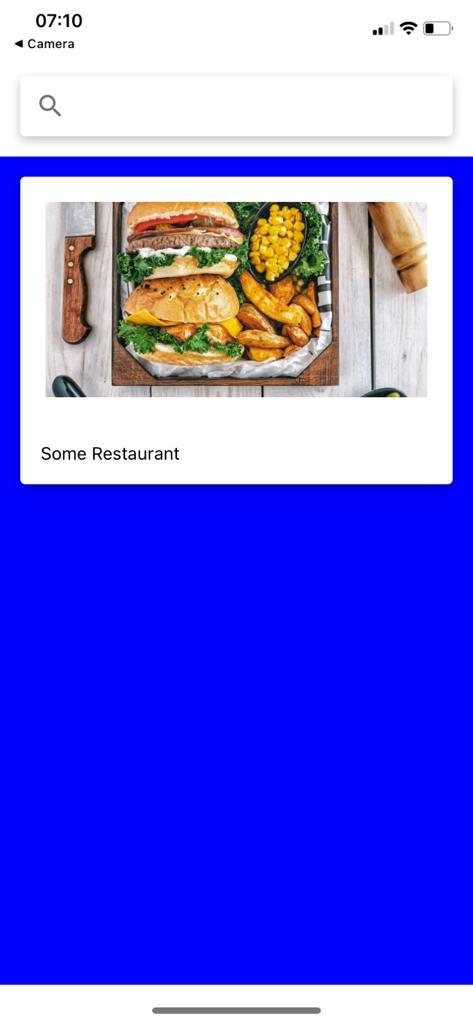
  const styles = StyleSheet.create({

    card: { backgroundColor: "white" },

    cover: { padding: 20, backgroundColor: "white" },

    title: { padding: 16 },

  });



Slika 8: Info kartica restorana

Nakon primene stilova koji su objašnjeni u okviru sledeće sekcije možemo nastaviti sa popunjavanjem naše kartice informacija o restoranu. Adresu restorana unosimo na isti način kao i naziv, samo sa drugačijim vrednostima za font family i font size. Međutim, kada je u pitanju rating restorana, potrebno je uneti svg fajl, koji će biti u stvari zvezdica koja prikazuje kolika je ocena restorana od 1 do 5. Kako bismo koristili svg fajlove potrebno je da instaliramo biblioteku za upravljanje njima i to radimo uz pomoć komande expo install react-native-svg.

// styles

...

import { SvgXml } from 'react-native-svg';

// assets

import star from '../../../../assets/star';

import open from '../../../../assets/open';

Kao što vidimo, potrebno je da importujemo i 2 svg fajla – star i open, gde će star pokazivati rating, a open da li je restoran trenutno otvoren.

Izmene kojima popunjavamo celu Restaurant Info karticu:

const Address = styled(Text)`

  font-family: ${(props) => props.theme.fonts.body};

  font-size: ${(props) => props.theme.fontSizes.caption};

`;

const Info = styled.View`

  padding: ${(props) => props.theme.space[3]};

`;

const Rating = styled.View`

  flex-direction: row;

  padding-top: ${(props) => props.theme.space[2]};

  padding-bottom: ${(props) => props.theme.space[2]};

`;

const Section = styled.View`

  flex-direction: row;

  align-items: center;

`;

const SectionEnd = styled.View`

  flex: 1;

  flex-direction: row;

  justify-content: flex-end;

`;

export const RestaurantInfoCard = ({ restaurant = {} }) => {

    const {

      name = "Some Restaurant",

      icon = "https://maps.gstatic.com/mapfiles/place\_api/icons/v1/png\_71/lodging-71.png",

      photos = [

        "https://www.foodiesfeed.com/wp-content/uploads/2019/06/top-view-for-box-of-2-burgers-home-made-600x899.jpg",

      ],

      address = "100 some random street",

      isOpenNow = true,

      rating = 4,

      isClosedTemporarily,

    } = restaurant;

    const rattingArray = Array.from(new Array(Math.floor(rating)));

    return (

      <RestaurantCard elevation={5}>

        <RestaurantCardCover key={name} source={{ uri: photos[0] }} />

        <Info>

          <Title>{name}</Title>

          <Section>

            <Rating>

              {rattingArray.map(() => (

                <SvgXml  xml={star} width={20} height={20}/>

              ))}

            </Rating>

            <SectionEnd>

              {isClosedTemporarily && (

                <Text variant="label" style={{ color: "red" }}>

                  CLOSED TEMPORARILY

                </Text>

              )}

              <View style={{ paddingLeft: 16 }} />

              {isOpenNow && <SvgXml xml={open} width={20} height={20} />}

              <View style={{ paddingLeft: 16 }} />

              <Image style={{ width: 15, height: 15 }} source={{ uri: icon }} />

            </SectionEnd>

          </Section>

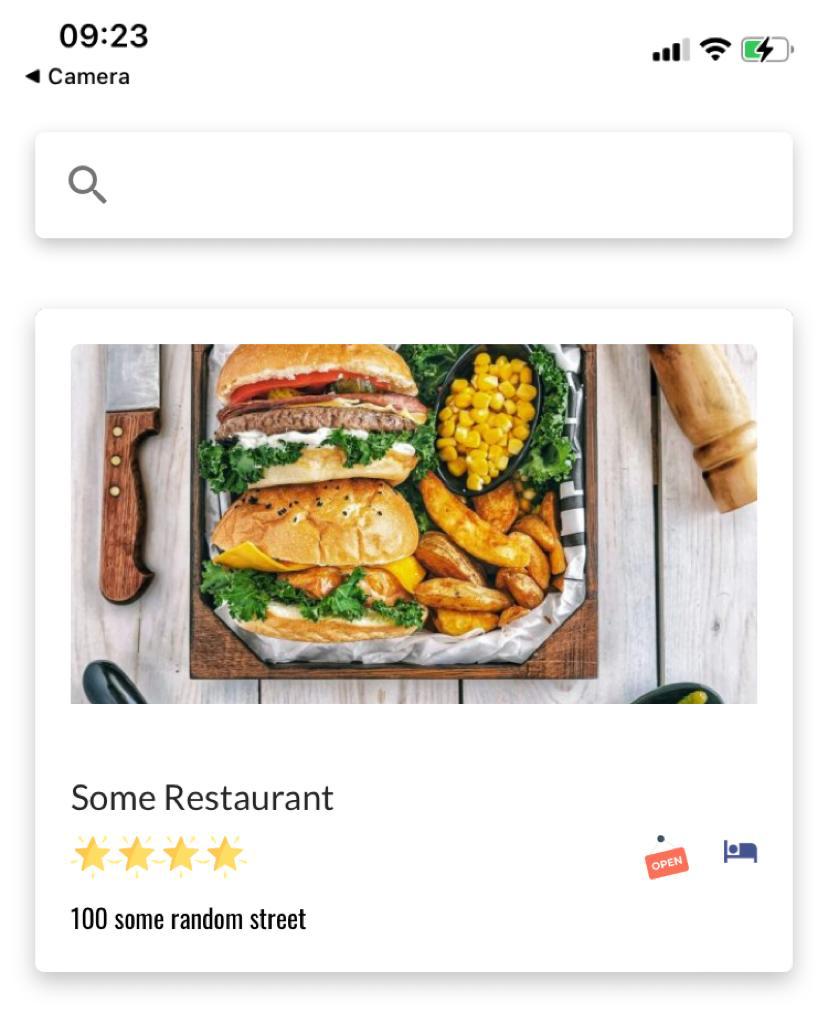
          <Address>{address}</Address>

        </Info>

      </RestaurantCard>

    )

  };

****

Slika 9: Finalni izgled kartice sa informacijama o restoranu

# 5 STILIZACIJA

Do sada smo koristili ili inline style za svaki element ili smo kreirali StyleSheet na dnu svakog fajla. Međutim, postoji bolji način za stilizovanje komponenti u react native-u i naziva se styled-components. Ovu bilioteku unosimo u naš projekat komandom yarn add styled-components. Umesto stilizacije pojedinačnih elemenata kreiraćemo posebne elemente koji primenjuju react-native element i stilove unutar njega i zatim implementirati novi element sa datim osobinama.

import styled from "styled-components/native";

...

const Title = styled.Text`

  padding: 16px;

  color: red;

`;

...

<Title>{name}</Title>

Na sličan način ćemo izmeniti i ostala polja u cilju bolje stilizacije naših elemenata. Takođe, ovaj način stilizacije nam omogućava da koristimo običan css u okviru svakog elementa, što nije slučaj kod StyleSheet-a. Prikazaćemo primer prepravke RestaurantsScreen komponente:

const SafeArea = styled(SafeAreaView)`

  flex: 1;

  ${StatusBar.currentHeight && `margin-top: ${StatusBar.currentHeight}px`};

`;

const SearchContainer = styled.View`

  padding: 16px;

`;

const RestaurantListContainer = styled.View`

  flex: 1;

  padding: 16px;

  background-color: blue;

`;

...

export const RestaurantsScreen = () => (

  <SafeArea>

    <SearchContainer>

      <Searchbar />

      </SearchContainer>

    <RestaurantListContainer>

      <RestaurantInfoCard />

    </RestaurantListContainer>

  </SafeArea>

);

## 5.1 Kreiranje teme

Na ovaj način ćemo vršiti stilizaciju od sada. Takođe, kreiraćemo novi folder – theme, u okviru kog će se nalaziti različiti fajlovi (fonts, colors, sizes itd.), u okviru kog ćemo globalno odrediti neke stilove i zatim ih kao takve primenjivati više puta.

colors.js:

export const colors = {

    brand: {

      primary: "#2182BD",

      secondary: "#5282BD",

      muted: "#C6DAF7",

    },

    ui: {

      primary: "#262626",

      secondary: "#757575",

      tertiary: "#F1F1F1",

      quaternary: "#FFFFFF",

      disabled: "#DEDEDE",

      error: "#D0421B",

      success: "#138000",

    },

    bg: {

      primary: "#FFFFFF",

      secondary: "#F1F1F1",

    },

    text: {

      primary: "#262626",

      secondary: "#757575",

      disabled: "#9C9C9C",

      inverse: "#FFFFFF",

      error: "#D0421B",

      success: "#138000",

    },

  };

fonts.js:

export const fonts = {

    body: "Oswald\_400Regular",

    heading: "Lato\_400Regular",

    monospace: "Oswald\_400Regular",

  };

  export const fontWeights = {

    regular: 400,

    medium: 500,

    bold: 700,

  };

  export const fontSizes = {

    caption: "12px",

    button: "14px",

    body: "16px",

    title: "20px",

    h5: "24px",

    h4: "34px",

    h3: "45px",

    h2: "56px",

    h1: "112px",

  };

sizes.js:

export const sizes = ["8px", "16px", "32px", "64px", "128px"];

spacing.js:

export const lineHeights = {

    title: "28px",

    copy: "20px",

  };

  export const space = ["0px", "4px", "8px", "16px", "32px", "64px"];

index.js (fajl gde unosimo sve što smo napravili u prethodna 4 fajla kako bismo ih lakše koristili u drugim elementima):

import { colors } from "./colors";

import { space, lineHeights } from "./spacing";

import { sizes } from "./sizes";

import { fonts, fontWeights, fontSizes } from "./fonts";

export const theme = {

  colors,

  space,

  lineHeights,

  sizes,

  fonts,

  fontSizes,

  fontWeights,

};

Takođe, kako bismo ovo koristili u okviru cele aplikacije kreiraćemo context i implementirati ga uz pomoć ThemeProvider-a u okviru kog ćemo kao temu uneti onu koju smo upravo kreirali:

// styles

import { ThemeProvider } from 'styled-components/native';

import { theme } from './src/infrastructure/theme';

...

      <ThemeProvider theme={theme}>

        <RestaurantsScreen />

        <ExpoStatusBar style="auto" />

      </ThemeProvider>

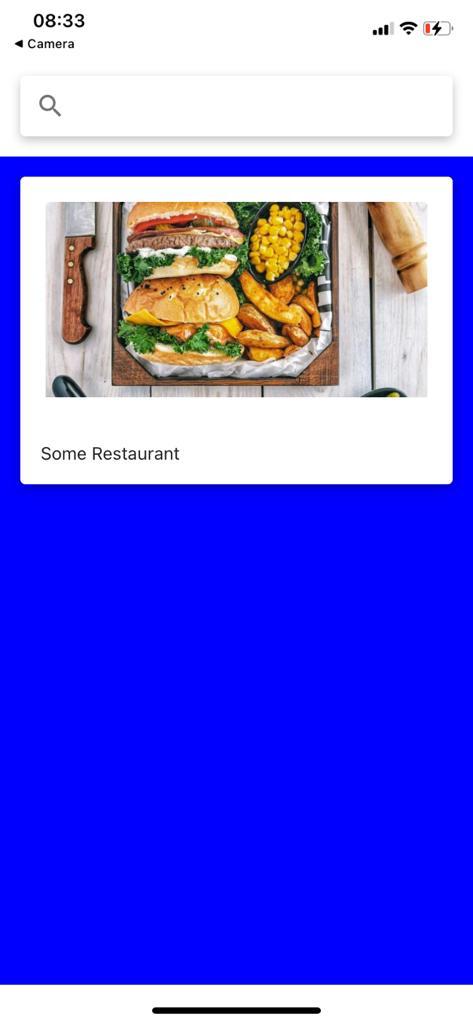
Primer korišćenja teme:

const Title = styled(Text)`

  padding: 16px;

  color: ${props => props.theme.colors.ui.primary};

`;



Slika 10: Izmenjena boja teksta korišćenjem ThemeProvider-a

## 5.2 Fontovi

Već smo kreirali fajl sa fontovima koje ćemo koristiti u okviru sajta, zajedno sa njihovim veličinama. Međutim, potrebno je importovati ove fontove kako bi se primenjivali u okviru aplikacije. Prvo instaliramo fontove pomoću sledećih komandi:

expo install expo-font

expo install @expo-google-fonts/oswald

expo install @expo-google-fonts/lato

Zatim, tako gde želimo da koristimo fontove treba da ih importujemo na sledeći način:

import { useFonts as useOswald, Oswald\_400Regular, } from '@expo-google-fonts/oswald';

import { useFonts as useLato, Lato\_400Regular, } from '@expo-google-fonts/lato';

...

  const [oswaldLoaded] = useOswald({

    Oswald\_400Regular

  });

  const [latoLoaded] = useLato({

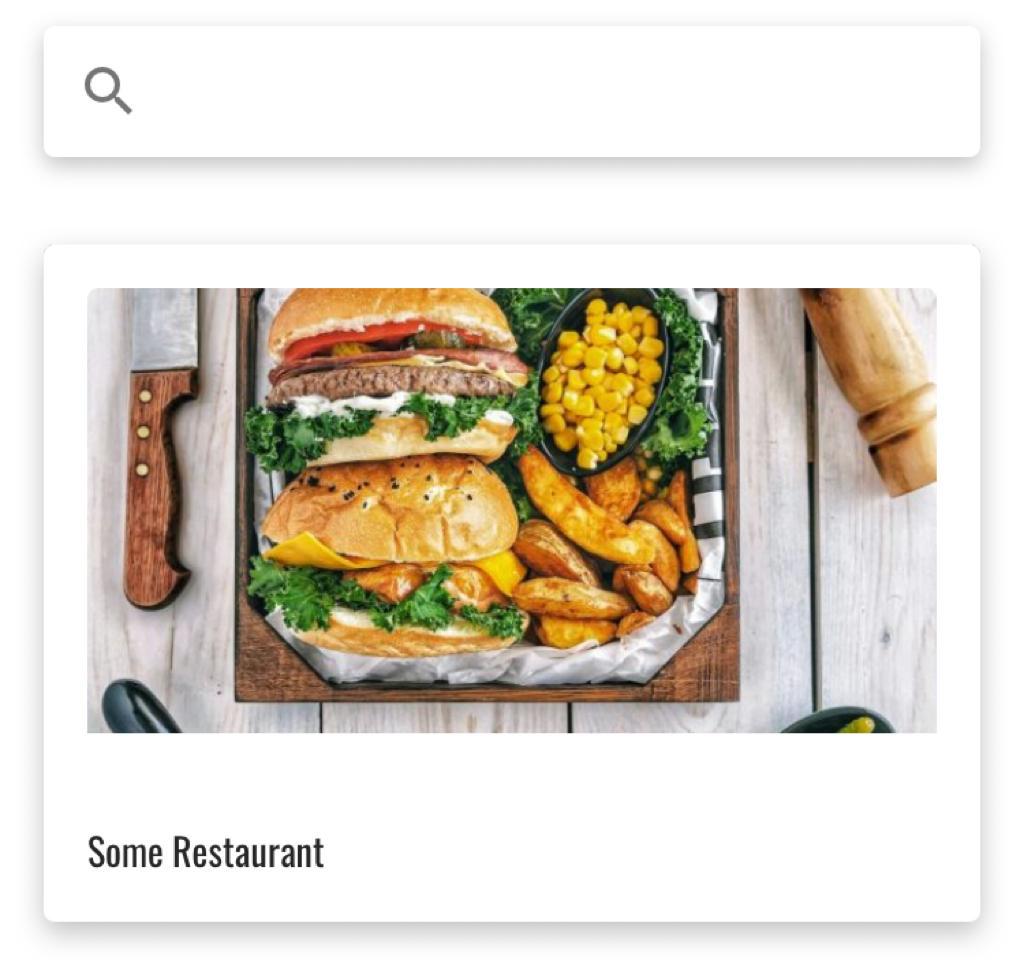
    Lato\_400Regular

  });

.......

const Title = styled(Text)`

  font-family: ${(props) => props.theme.fonts.body};



Slika 11: Primena novog fonta

## 5.3 Spacer

Kreiraćemo komponentu Spacer koja će u sebi imati ispisane margine sa gornje, donje, leve i desne strane i za svaku stranu je podeliti u 3 veličine Small, Medium i Large. Ove komponente ćemo zatim exportovati tako da možemo pozivati samo ove elemente umesto inline style-a u okviru drugih elemenata.

Spacer:

// styles

import styled from 'styled-components/native';

const sizeVariant = {

    small: 1,

    medium: 2,

    large: 3,

};

const positionVariant = {

    top: 'marginTop',

    left: 'marginLeft',

    right: 'marginRight',

    bottom: 'marginBottom',

};

const getVariant = (position, size, theme) => {

    const sizeIndex = sizeVariant[size];

    const property = positionVariant[position];

    const value = theme.space[sizeIndex];

    return `${property}:${value}`;

};

export const Spacer = styled.View`

    ${({ position, size, theme }) => getVariant(position, size, theme)}

`;

Spacer.defaultProps = {

    position: 'top',

    size: 'small'

};

Primena spacer-a:

       <Spacer position="left" size="large">

         {isOpenNow && <SvgXml xml={open} width={20} height={20} />}

       </Spacer>

       <Spacer position="left" size="large">

         <Image style={{ width: 15, height: 15 }} source={{ uri: icon }} />

       </Spacer>

## 5.4 Tipography

U dosadašnjem delu sekcije stilizacije nismo previše obraćali pažnju na Text elemente, već smo za većinu kreirali ili posebnu komponentu ili koristili inline style. Kao i za ostale funkcionalnosti i za ovu ćemo kreirati komponentu koja će obezbediti konzistentnost stilizacije.

// styles

import styled from "styled-components/native";

// if we don't give any props

const defaultTextStyles = (theme) => `

  font-family: ${theme.fonts.body};

  font-weight: ${theme.fontWeights.regular};

  color: ${theme.colors.text.primary};

  flex-wrap: wrap;

  margin-top: 0px;

  margin-bottom: 0px;

`;

const body = (theme) => `

    font-size: ${theme.fontSizes.body};

`;

const hint = (theme) => `

    font-size: ${theme.fontSizes.body};

`;

const error = (theme) => `

    color: ${theme.colors.text.error};

`;

const caption = (theme) => `

    font-size: ${theme.fontSizes.caption};

    font-weight: ${theme.fontWeights.bold};

`;

const label = (theme) => `

    font-family: ${theme.fonts.heading};

    font-size: ${theme.fontSizes.body};

    font-weight: ${theme.fontWeights.medium};

`;

const variants = { // <Text variant="..." />

  body,

  label,

  caption,

  error,

  hint,

};

export const Text = styled.Text`

  ${({ theme }) => defaultTextStyles(theme)}

  ${({ variant, theme }) => variants[variant](theme)}

`;

Text.defaultProps = {

  variant: "body",

};

Primena:

import { Text } from "../../../components/typography/text.component";

...

<Text variant="label">{name}</Text>

...

<Text variant="error">

   CLOSED TEMPORARILY

</Text>

## 5.5 Sređivanje style fajlova

Do sada smo sve nove komponente koje u sebi implementiraju određene stilove unosili u kodu iznad samog template-a koji se renderuje. Međutim, to značajno otežava čitanje koda, zbog čega ćemo sve ove komponente uneti u okviru novog fajla i zatim ih importovati u glavnom fajlu. Svaki fajl koji kreiramo od sada imaće pored glavne komponente sa sufiksom component.js i svoju styles.js komponentu.

restaurant-info-card.styles.js:

// react native imports

import { Card } from "react-native-paper";

// styles

import styled from "styled-components/native";

export const RestaurantCard = styled(Card)`export const

  background-color: ${(props) => props.theme.colors.bg.primary};

`;

export const RestaurantCardCover = styled(Card.Cover)`

  padding: ${(props) => props.theme.space[3]};

  background-color: ${(props) => props.theme.colors.bg.primary};;

`;

export const Address = styled.Text`

  font-family: ${(props) => props.theme.fonts.body};

  font-size: ${(props) => props.theme.fontSizes.caption};

`;

export const Info = styled.View`

  padding: ${(props) => props.theme.space[3]};

`;

export const Rating = styled.View`

  flex-direction: row;

  padding-top: ${(props) => props.theme.space[2]};

  padding-bottom: ${(props) => props.theme.space[2]};

`;

export const Section = styled.View`

  flex-direction: row;

  align-items: center;

`;

export const SectionEnd = styled.View`

  flex: 1;

  flex-direction: row;

  justify-content: flex-end;

`;

export const Icon = styled.Image`

  width: 15px;

  height: 15px;

`;

importovanje u okviru restaurant-info-card.component.js:

import { RestaurantCard, RestaurantCardCover, Address, Info,

          Rating, Section, SectionEnd, Icon } from "./restaurant-info-card.styles";

# 6 LISTA RESTORANA

Restoran iz sekcije 4 predstavlja samo jedan od elemenata koji treba da se prikažu u okviru naše aplikacije. Naime, kada otvorimo listu restorana, prikazivaćemo onoliko kartice RestaurantListCard koliko ih imamo u bazi. Ove elemente ćemo uneti u okviru komponente FlatList koja se importuje iz react-native biblioteke. Mogli smo koristiti i ScrollView, međutim ova komponenta više odgovara kada imamo unapred određen broj elemenata, zbog performansi.

Komponenta FlatList prima više atributa, atribut data predstavlja podatke koji će biti prikazani u okviru te liste i ovde ćemo uneti neke default podatke jer još uvek nismo postavili bazu podataka. U okviru polja renderItem unosimo našu komponentu RestaurantInfoCard, tako da će za svako polje unositi po jednu karticu restorana. Potrebno je uneti i keyExtractor, kako bi react razlikovao elemente međusobno.

export const RestaurantsScreen = () => (

  <SafeArea>

    <SearchContainer>

      <Searchbar />

    </SearchContainer>

    <FlatList

      data={[{ name: 1 }, { name: 2 }, { name: 3 }, { name: 4 }]}

      renderItem={() => <RestaurantInfoCard />}

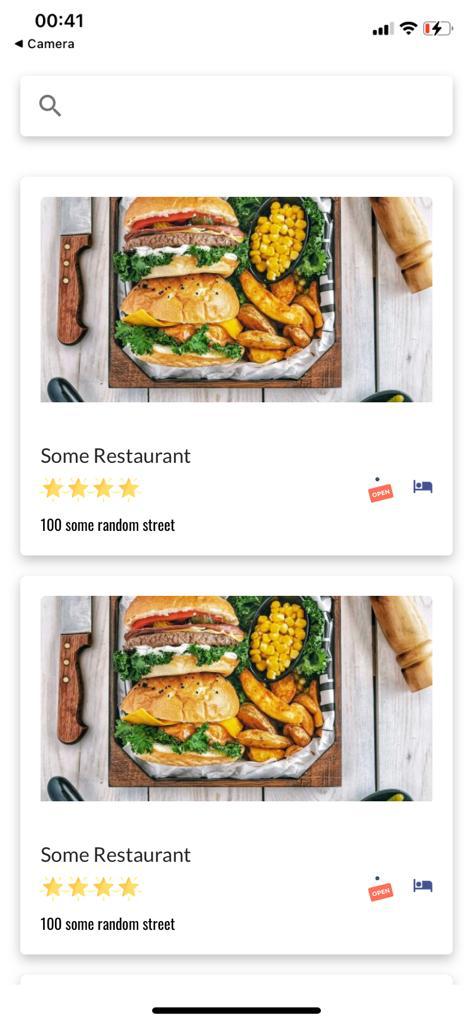
      keyExtractor={(item) => item.name}

      contentContainerStyle={{ padding: 16 }}

    />

  </SafeArea>

);



Slika 12: Dummy lista restorana

# 7 NAVIGACIJA

## 7.1 Instalacija

Navigacija je ono što se veoma često razlikuje kod iOS i Android uređaja. Kod iOS uređaja, navigacija je najčešće u gornjem delu i korisnici idu sa strane na stranu prelaskom prsta s leva na desno ili obrnuto, dok kod Android uređaja ona najčešće stoji u dnu aplikacije. Na adresi <https://reactnavigation.org/> možemo pronaći rešenje koje React Native pruža kao najbolje i veoma ga je lako implementirati u okviru aplikacije.

Prvo je potrebno da instaliramo react-navigation paket pomoću komande yarn add @react-navigation/native. Takođe, uz ovaj paket potrebno je instalirati još nekoliko paketa koji su vezani za react navigaciju: expo install react-native-gesture-handler react-native-reanimated react-native-screens react-native-safe-area-context @react-native-community/masked-view.

U okviru naše aplikacije koristićemo tab-based navigaciju, koja će biti u dnu ekrana. Za to je takođe potrebno instalirati biblioteku yarn add @react-navigation/bottom-tabs. Razlog za ovako veliki broj paketa koje instaliramo jeste to što je react native podelio svoje pakete vezane za navigaciju u veliki broj celina, tako da bi korisnici mogli da preuzmu samo tačno ono što im je potrebno, bez dodatnih paketa koji zauzimaju prostor, a ne koriste se.

## 7.2 Postavljanje navigacije

Našu navigaciju implementiramo na sledeći način u okviru App.js-a:

// navigation

import { NavigationContainer } from "@react-navigation/native";

import { createBottomTabNavigator } from "@react-navigation/bottom-tabs";

...

// navigation setup

const Tab = createBottomTabNavigator();

const Settings = () => (

  <SafeArea>

    <Text>

      Settings

    </Text>

  </SafeArea>

);

const Map = () => (

  <SafeArea>

    <Text>

      Map

    </Text>

  </SafeArea>

)

...

      <ThemeProvider theme={theme}>

        <NavigationContainer>

          <Tab.Navigator>

            <Tab.Screen name="Restaurants" component={RestaurantsScreen} />

            <Tab.Screen name="Map" component={Map}/>

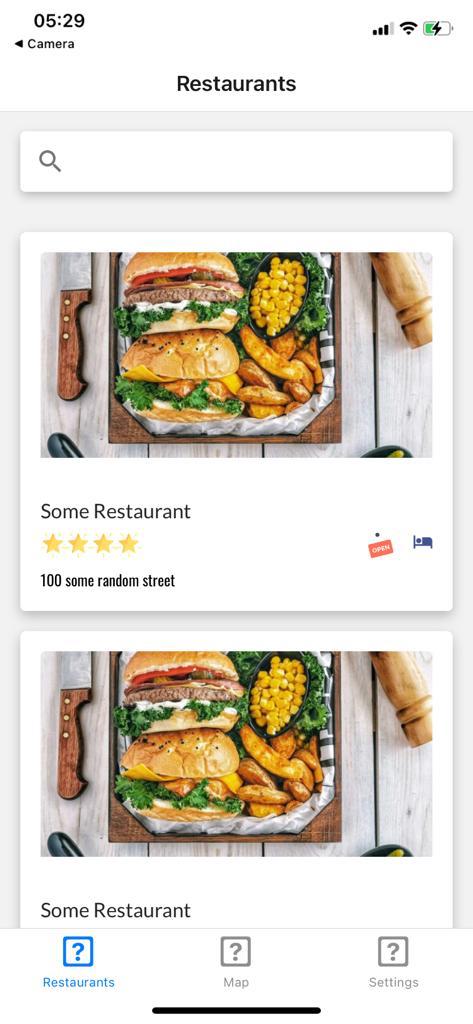
            <Tab.Screen name="Settings" component={Settings}/>

          </Tab.Navigator>

        </NavigationContainer>

        <ExpoStatusBar style="auto" />

      </ThemeProvider>



Slika 13: Početni izgled navigacije

## 7.3 Ikonice u okviru navigacije

Sada smo uneli našu navigaciju, međutim prikazuju nam se default ikonice za svaki od tabova. Ikonice preuzimamo iz Icons Expo dokumentacije, gde možemo pronaći ikonicu koja nama odgovara, kao i način na koji se ona implementira u okviru naših tabova.

        <Tab.Navigator

            screenOptions={({ route }) => ({

              tabBarIcon: ({ color, size }) => {

                let iconName;

                if (route.name === "Restaurants") {

                  iconName = "md-restaurant-outline";

                } else if (route.name === "Map") {

                  iconName = "ios-map-outline";

                } else if (route.name === "Settings") {

                  iconName = "settings-outline";

                }

                return <Ionicons name={iconName} size={size} color={color} />;

              },

            })}

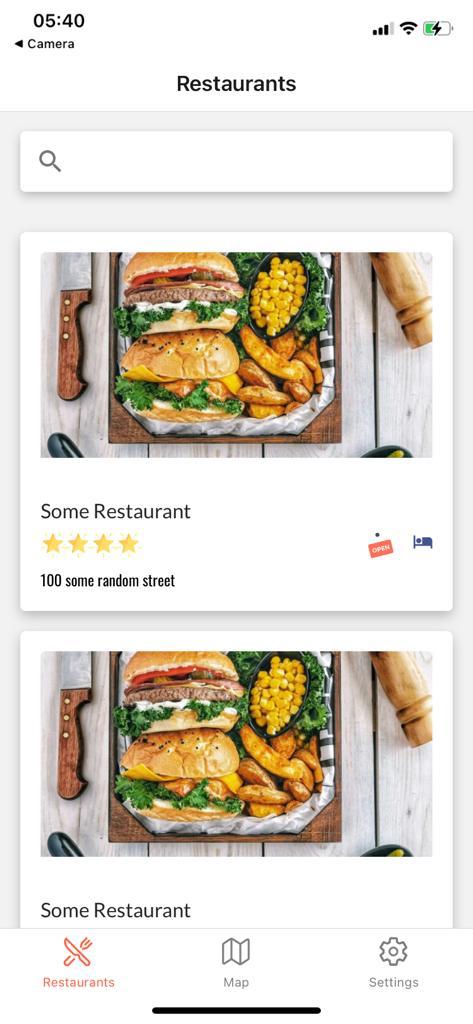
            tabBarOptions={{

              activeTintColor: "tomato",

              inactiveTintColor: "gray",

            }}

          >



Slika 14: Ikonice u navigaciji

## 7.4 Restrukturiranje navigacije

S obzirom na to da navigacija predstavlja jedan od najvažnijih segemenata infrastrukture svake aplikacije, pomerićemo je u okviru njene sopstvene komponente. Takođe, postoji mogućnost da u budućnosti izmenimo našu navigaciju, dodavanjem novih opcija ili nečeg sličnog, zbog čega ćemo je smestiti u okviru nove komponente kako bismo što lakše vršili izmene u budućnosti.

// react imports

import React from "react";

// react native imports

import { Text } from "react-native";

// navigation

import { NavigationContainer } from "@react-navigation/native";

import { createBottomTabNavigator } from "@react-navigation/bottom-tabs";

import { Ionicons } from "@expo/vector-icons";

// features

import { RestaurantsScreen } from "../../features/restaurants/screens/restaurants.screen";

// components

import { SafeArea } from "../../components/utility/safe-area.component";

// navigation setup

const Tab = createBottomTabNavigator();

const TAB\_ICON = {

  Restaurants: "md-restaurant-outline",

  Map: "ios-map-outline",

  Settings: "settings-outline",

};

const Settings = () => (

  <SafeArea>

    <Text>

      Settings

    </Text>

  </SafeArea>

);

const Map = () => (

  <SafeArea>

    <Text>

      Map

    </Text>

  </SafeArea>

)

const createScreenOptions = ({ route }) => {

  const iconName = TAB\_ICON[route.name];

  return {

    tabBarIcon: ({ size, color }) => (

      <Ionicons name={iconName} size={size} color={color} />

    ),

  };

};

export const AppNavigator = () => (

    <NavigationContainer>

      <Tab.Navigator

        screenOptions={createScreenOptions}

        tabBarOptions={{

          activeTintColor: "tomato",

          inactiveTintColor: "gray",

        }}

      >

        <Tab.Screen name="Restaurants" component={RestaurantsScreen} />

        <Tab.Screen name="Map" component={Map} />

        <Tab.Screen name="Settings" component={Settings} />

      </Tab.Navigator>

    </NavigationContainer>

  );

// react imports

import React from "react";

// navigation

import { AppNavigator } from "./app.navigator";

export const Navigation = () => {

    return <AppNavigator />;

};

export default function App() {

...

return (

    <>

      <ThemeProvider theme={theme}>

        <LocationContextProvider>

          <RestaurantsContextProvider>

**<Navigation />**

          </RestaurantsContextProvider>

        </LocationContextProvider>

      </ThemeProvider>

      <ExpoStatusBar style="auto" />

    </>

  );

}

## 7.5 Pojedinačni navigatori

Želimo za svaku od opcija u okviru naše navigacije da kreiramo pojedinačni navigator. Svaki od ovih fajlove će u sebi držati logiku i funkcionalnosti koje se primenjuju kada smo u okviru pojedinačne opcije iz navigacije na koju se odnosi taj fajl. Za sada imamo kreiranu stranicu restorana, tako da ćemo prvo kreirati restaurants.navigator.js fajl. Razlog za ovo je to što ćemo se unutar naše 3 opcije koje trenutno imamo u navigaciji kretati po različitim “stranicama”. Na primer možemo imati listu restorana, ali takođe ćemo imati i stranicu detalja svakog restorana, gde ne želimo da nam u header-u piše Restaurants. Zbog toga ćemo za svaki poseban stack u okviru jednog navigatora imati onoliko opcija koliko “stranica” imamo. Trenutno ćemo uneti samo stack liste restorana.

RestaurantsNavigator:

// react imports

import React from "react";

// navigation

import { createStackNavigator } from "@react-navigation/stack";

// features

import { RestaurantsScreen } from "../../features/restaurants/screens/restaurants.screen";

const RestaurantStack = createStackNavigator();

export const RestaurantsNavigator = () => {

    return (

        <RestaurantStack.Navigator

            screenOptions={{

                headerShown: false

            }}

        >

            <RestaurantStack.Screen

                name="Restaurants"

                component={RestaurantsScreen}

            />

        </RestaurantStack.Navigator>

    )

}

S obzirom na to da nam je za opciju svih restorana headerShown postavljen na false, nećemo videti 2 puta ispis Restaurants.

# 8 SERVISI

## 8.1 Mock-up servisi

Do sada smo kreirali osnovni izgled naše aplikacije, koja međutim još uvek nema prave funkcionalnosti koje bismo želeli, a to je pretraga restorana po mestima. To je prvi servis koji naša aplikacija omogućava. Kreiraćemo mock folder kako bismo u njega uneli veštačke podatke pomoću kojih ćemo definisati funkcionalnosti koje ćemo kasnije koristiti za prave podatke. Napravićemo 4 json fajla za 4 grada (San Francisco, Chicago, Antwerp i Toronto) koji će imitirati našu bazu podataka kako bismo na osnovu njih definisali pravila njihovog preuzimanja i prikazivanja.

Naši podaci su trenutno u formi json-a, a želimo da ih pretvorimo u uobičajeni zapis javascript podataka, što znači da želimo da koristimo camel case. Za to ćemo koristiti paket camelize, koji unosimo komandom yarn add camelize. Takođe, postoje podaci u okviru baze koje je potrebno preimenovati i ekstrahovati kako bismo mogli da ih primenimo u okviru naše aplikacije. Svi ovi podaci unešeni su po uzoru na API koji vraća liste restorana o kom će kasnije biti reči.

Za prikaz restorana koristićemo context, konkretno RestaurantContext preko kog će svi fajlovi unutar naše aplikaciji imati pristup listi restorana. Povezaćemo useContext i servis za mock-up aplikaciju sa našim Restaurant screen-om kako bismo ispisali ove veštačke podatke u okviru aplikacije.

// mocks import

import { mocks, mockImages } from "./mock/index";

// transforming the data

import camelize from "camelize";

export const restaurantsRequest = (location = "137.7749295,-122.494155") => {

    return new Promise((resolve, reject) => {

        const mock = mocks[location];

        if(!mock) {

            reject("not found");

        }

        resolve(mock);

    });

};

export const restaurantsTransform = ({ results = [] }) => {

    const mappedResults = results.map((restaurant) => {

      restaurant.photos = restaurant.photos.map((p) => {

        return mockImages[Math.ceil(Math.random() \* (mockImages.length - 1))];

      });

      return {

        ...restaurant,

        isOpenNow: restaurant.opening\_hours && restaurant.opening\_hours.open\_now,

        isClosedTemporarily: restaurant.business\_status === "CLOSED\_TEMPORARILY",

      };

    });

    return camelize(mappedResults);

  };

// react imports

import React, { useState, createContext, useEffect, useMemo } from "react";

// services

import { restaurantsRequest, restaurantsTransform } from "./restaurants.service";

export const RestaurantsContext = createContext();

export const RestaurantsContextProvider = ({ children }) => {

    const [restaurants, setRestaurants] = useState([]);

    const [isLoading, setIsLoading] = useState(false);

    const [error, setError] = useState(null);

    const retrieveRestaurants = () => {

        setIsLoading(true);

        setTimeout(() => {

            restaurantsRequest().then(restaurantsTransform)

                .then((results) => {

                    setIsLoading(false);

                    setRestaurants(results);

                }).catch(err => {

                    setIsLoading(false);

                    setError(err);

                });

        }, 2000);

    }

    useEffect(() => {

        retrieveRestaurants();

    }, []);

    return (

        <RestaurantsContext.Provider

            value={{

                restaurants,

                isLoading,

                error

            }}

        >

            { children }

        </RestaurantsContext.Provider>

    )

}

export const RestaurantsScreen = () =>  {

  const { restaurants, isLoading, error } = useContext(RestaurantsContext);

  return (

    <SafeArea>

      <SearchContainer>

        <Searchbar />

      </SearchContainer>

      <RestaurantList

        data={restaurants}

        renderItem={({ item }) => {

          return (

            <RestaurantInfoCard restaurant={item}/>

          )

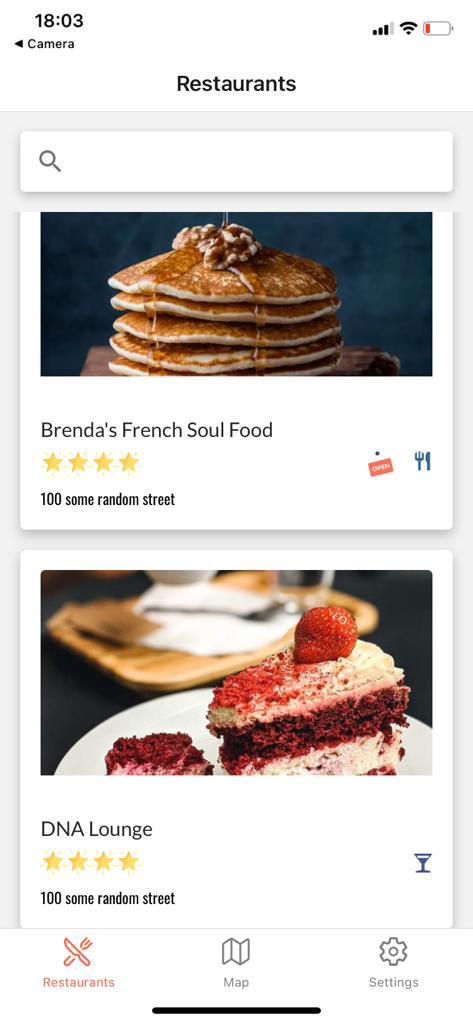
        }}

        keyExtractor={(item) => item.name}

      />

    </SafeArea>

)};



Slika 15: Mock-up izlistavanje restorana

S obzirom na to da imamo opciju Loading, i da smo postaviti timeOut na 2 sekunde, kako bismo imitirali funkciju učitavanja podataka, potrebno je da korisnika obavestimo o tome da se podaci učitavaju. To ćemo uraditi korišćenjem ActivityIndicator-a.

const Loading = styled(ActivityIndicator)`

  margin-left: -25px;

`;

const LoadingContainer = styled.View`

  position: absolute;

  top: 50%;

  left: 50%;

`;

...

export const RestaurantsScreen = () =>  {

  const { restaurants, isLoading, error } = useContext(RestaurantsContext);

  return (

    <SafeArea>

      {isLoading &&

        <LoadingContainer>

          <Loading

            size={50}

            animating={true}

            color={Colors.orange300}

          />

        </LoadingContainer>

      }

...



Slika 16: Učitavanje restorana

## 8.2 Geocoding Mock-up

Geocoding je način na koji Google vrši pretvaranje geografske širine i dužine u konkretnu lokaciju na mapi. Za potrebe lociranja naših restorana koristićemo geocoding umesto mock-up vrednosti koje smo koristili u prethodnoj sekciji. Za početak ćemo kreirati mock-up servis koji će u okviru sebe imati podatke vezane za gradove koje smo uneli kako bismo mogli da vršimo pretragu na osnovu toga.

location.service:

// data transform

import camelize from "camelize";

// mocks imports

import { locations } from "./location.mock";

export const locationRequest = (searchTerm) => {

  return new Promise((resolve, reject) => {

    const locationMock = locations[searchTerm];

    if (!locationMock) {

      reject("not found");

    }

    resolve(locationMock);

  });

};

export const locationTransform = (result) => {

  const formattedResponse = camelize(result);

  const { geometry = {} } = formattedResponse.results[0];

  const { lat, lng } = geometry.location;

  return { lat, lng };

};

location.context:

// react imports

import React, { createContext, useState, useEffect } from "react";

// location service

import { locationRequest, locationTransform } from "./location.service";

export const LocationContext = createContext();

export const LocationContextProvider = ({ children }) => {

    const [location, setLocation] = useState(null);

    const [keyword, setKeyword] = useState("san francisco");

    const [isLoading, setIsLoading] = useState(false);

    const [error, setError] = useState(null);

    const onSearch = (searchKeyWord) => {

        setIsLoading(true);

        setKeyword(searchKeyWord);

        locationRequest(searchKeyWord.toLowerCase()).then(locationTransform)

            .then(result => {

                setIsLoading(false);

                setLocation(result);

            }).catch(err => {

                setIsLoading(false);

                setError(err);

            })

    };

    useEffect(() => {

        onSearch(keyword);

    }, []);

    return (

        <LocationContext.Provider

            value={{

                isLoading,

                error,

                location,

                search: onSearch,

                keyword,

            }}

        >

            { children }

        </LocationContext.Provider>

    )

};

App.js:

  return (

    <>

      <ThemeProvider theme={theme}>

        <LocationContextProvider>

          <RestaurantsContextProvider>

...

          </RestaurantsContextProvider>

        </LocationContextProvider>

      </ThemeProvider>

    </>

Trebalo bi naš Location Context da povežemo sa poljem za pretragu, kako bi se ono što korisnik unese prikazivalo u okviru liste restorana. Korisnik može da pretražuje gradove i na osnovu toga dobija restorane u tom gradu. Za to ćemo kreirati komponentu Search u okviru koje ćemo smestiti kod koji je ranije bio u okviru RestaurantScreen-a vezan za polje pretrage. Takođe ćemo iskoristiti LocationContext.

export const Search = () => {

    const { keyword, search } = useContext(LocationContext);

    const [searchKeyword, setSearchKeyword] = useState(keyword);

    useEffect(() => {

        search(searchKeyword);

    }, [])

    return (

        <SearchContainer>

            <Searchbar

                placeholder="Search for a location"

                value={searchKeyword}

                onSubmitEditing={() => {

                    search(searchKeyword);

                }}

                onChangeText={(text) => {

                    setSearchKeyword(text);

                }}

            />

        </SearchContainer>

    )

}

export const RestaurantsContextProvider = ({ children }) => {

    const [restaurants, setRestaurants] = useState([]);

    const [isLoading, setIsLoading] = useState(false);

    const [error, setError] = useState(null);

    const { location } = useContext(LocationContext)

    const retrieveRestaurants = (loc) => {

        setIsLoading(true);

        setRestaurants([]);

        setTimeout(() => {

            restaurantsRequest(loc)

                .then(restaurantsTransform)

                .then((results) => {

                    setIsLoading(false);

                    setRestaurants(results);

                }).catch(err => {

                    setIsLoading(false);

                    setError(err);

                });

        }, 2000);

    }

    useEffect(() => {

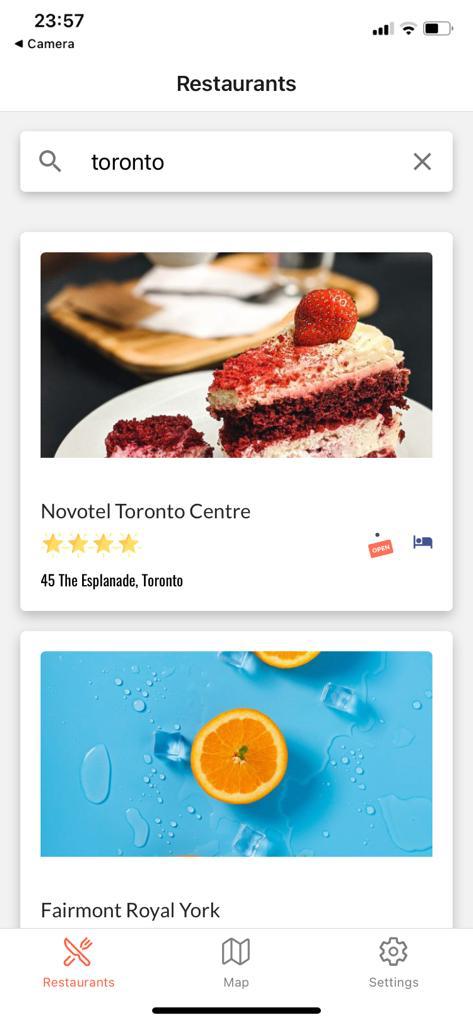
        if (location) {

            const locationString = `${location.lat},${location.lng}`;

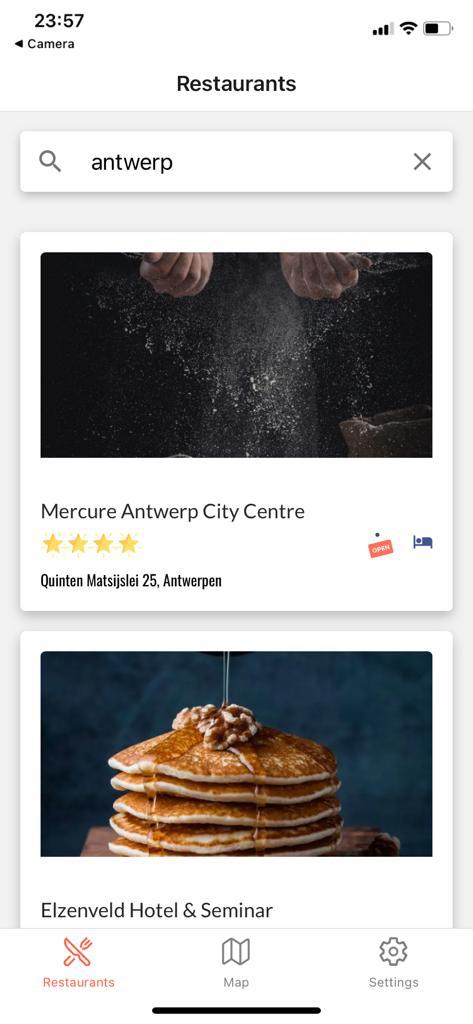
            retrieveRestaurants(locationString);

        }

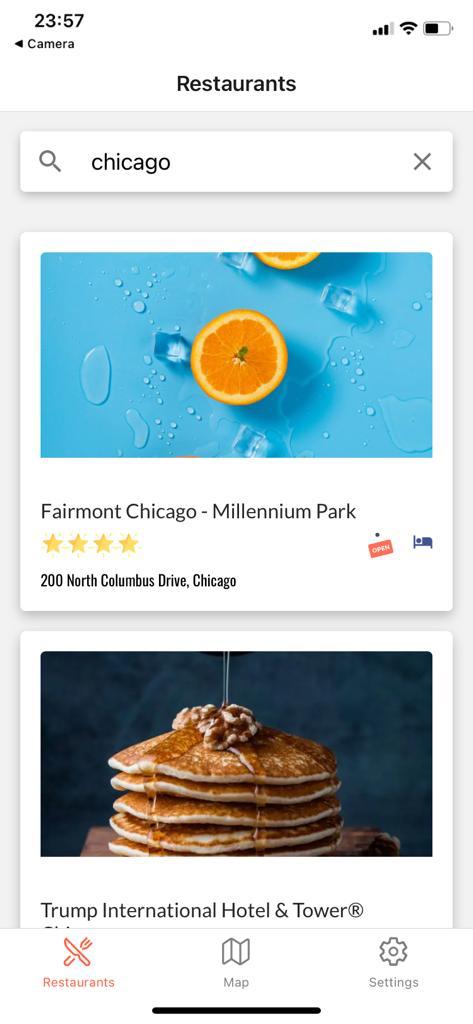
    }, [location]);



Slika 17: Pretraga restorana



Slika 18: Pretraga restorana



Slika 19: Pretraga restorana

## 8.3 Favourites servis

U okviru naše aplikacije želimo da omogućimo korisniku da određene restorane stavi u listu omiljenih restorana. Tu mogućnost bi trebalo da ima kako na stranici svih restorana, tako i sa mape. Zbog toga ćemo prvo kreirati Favourites Context na sledeći način:

// react imports

import React, { createContext, useState } from "react";

export const FavouritesContext = createContext();

export const FavouritesContextProvider = ({ children }) => {

    const [favourites, setFavourites] = useState([]);

    const add = (restaurant) => {

        setFavourites([...favourites, restaurant]);

    };

    const remove = (removedRest) => {

        const newFavourites = favourites.filter(

            (restaurants) => restaurants.placeId !== removedRest.placeId

        );

        setFavourites(newFavourites);

    }

    return (

        <FavouritesContext.Provider

            value={{

                favourites,

                addToFavourites: add,

                removeFromFavourites: remove,

            }}

        >

            { children }

        </FavouritesContext.Provider>

    );

};

Nakon kreiranja context-a potrebno je da ga implementiramo u okviru App.js-a zajedno sa drugim context-ima. Takođe, ubacićemo ovaj context i u okviru RestaurantsScreen-a, jer će tu biti prikazana opcija za dodavanje u ovu listu. To ćemo uraditi u okviru komponente RestaurantCardInfo. Za to ćemo kreirati novu komponentu – FavouriteComponent:

// react imports

import React, { useContext } from "react";

// react native imports

import { TouchableOpacity } from "react-native";

// context

import { FavouritesContext } from "../../services/favourites/favourites.context";

// styles

import styled from "styled-components/native";

import { AntDesign } from "@expo/vector-icons";

const FavouriteButton = styled(TouchableOpacity)`

    position: absolute;

    top: 25px;

    right: 25px;

    z-index: 9;

`;

export const Favourite = ({ restaurant }) => {

    const { favourites, addToFavourites, removeFromFavourites } = useContext(FavouritesContext);

    const isFavourite = favourites.find(

        (r) => r.placeId === restaurant.placeId

    );

    return (

        <FavouriteButton

            onPress={() => !isFavourite

                ? addToFavourites(restaurant)

                : removeFromFavourites(restaurant)

            }

        >

            <AntDesign

                name={

                    isFavourite ? "heart" : "hearto"

                }

                size={24}

                color={

                    isFavourite ? "red" : "white"

                }

            />

        </FavouriteButton>

    );

};

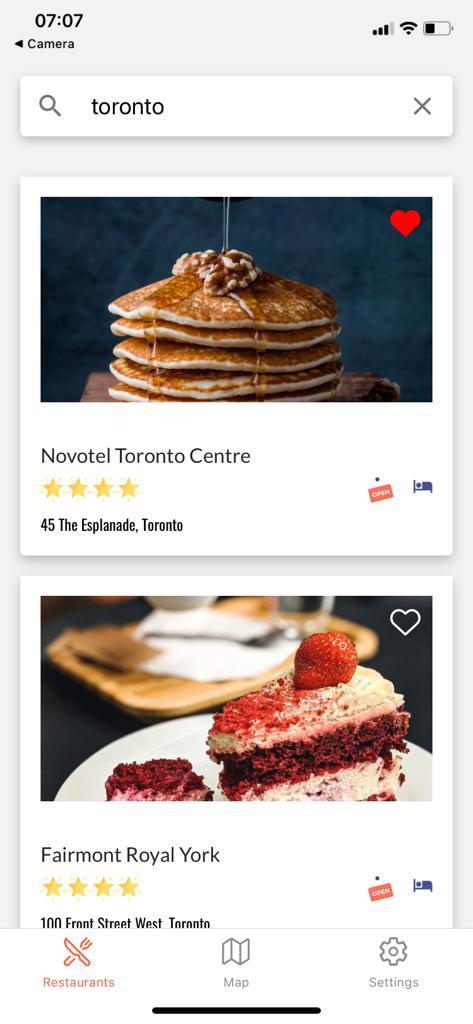
U okviru kartice restorana ćemo primeniti ovu komponentu:

    return (

      <RestaurantCard elevation={5}>

        <Favourite restaurant={restaurant}/>

...



Slika 20: Favourites na listi svih restorana

Sada bi trebalo da naše favourite restorane stavimo negde u okviru aplikacije gde bi korisnici mogli da vide koje restorane su stavili u tu listu. Međutim, kako ne bismo kreirali potpuno novi segment naše aplikacije, iskoristićemo listu svih restorana. Postavićemo za ikonicu našeg searchbar-a heart-icon i kada korisnik klikne na nju prikazaće mu se restorani koje je uneo u listu omiljenih u formi CompactRestaurantInfo kartice iznad ostalih restorana horizontalno.

// react imports

import React from "react";

// react native imports

import { ScrollView, TouchableOpacity } from "react-native";

// styles

import styled from "styled-components/native";

import { Text } from "../typography/text.component";

// components

import { CompactRestaurantInfo } from "../restaurant/compact-restaurant-info.component";

import { Spacer } from "../spacer/spacer.component";

const FavouritesWrapper = styled.View`

    padding: 10px;

`;

export const FavouritesBar = ({ favourites, onNavigate }) => {

    if(!favourites.length) {

        return null;

    }

    return (

        <FavouritesWrapper>

            <Spacer variant="left.xlarge">

                <Text variant="caption">Favourites</Text>

            </Spacer>

            <ScrollView

                horizontal // left to right

                showsHorizontalScrollIndicator={false}

            >

                {favourites.map((restaurant) => {

                    const key = restaurant.name.split(" ").join("");

                    return (

                        <Spacer position="left" size="medium" key={key}>

                            <TouchableOpacity

                                onPress={() => onNavigate("RestaurantDetail", {

                                    restaurant,

                                })}

                            >

                                <CompactRestaurantInfo restaurant={restaurant} />

                            </TouchableOpacity>

                        </Spacer>

                    );

                })}

            </ScrollView>

        </FavouritesWrapper>

    );

};

RestaurantScreen:

  const { favourites } = useContext(FavouritesContext);

  const [isToggled, setIsToggled] = useState(false);

....

<Search

        isFavouritesToggled={isToggled}

        onFavouritesToggle={() => setIsToggled(!isToggled)}

      />

      {isToggled &&

        <FavouritesBar

          favourites={favourites}

          onNavigate={navigation.navigate}

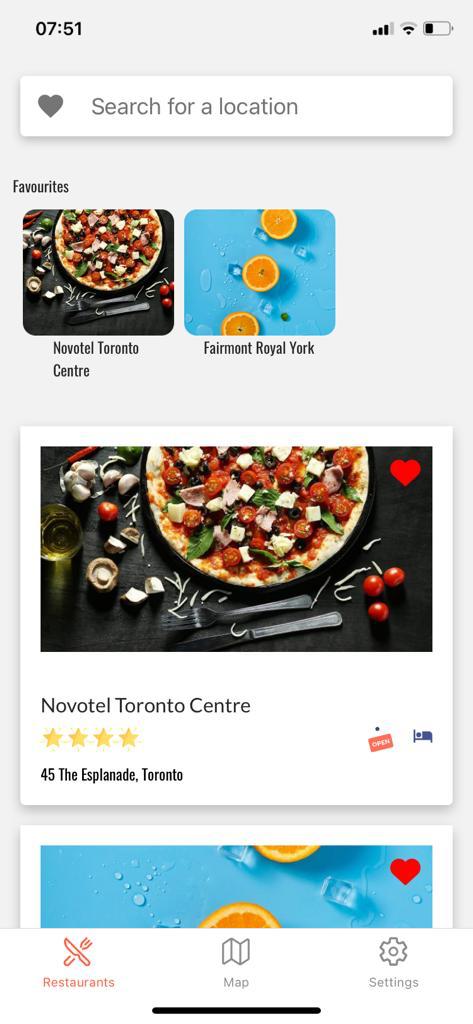
        />

      }

SearchComponent:

     icon={ isFavouritesToggled ? "heart" : "heart-outline"}

     onIconPress={onFavouritesToggle}



Slika 21: Favourites Bar

Trenutno, kada god ponovo učitamo stranicu, gube se svi elementi iz liste omiljenih, tako da nemamo sačuvane one koje smo uneli u nju u prethodnoj sesiji. Zbog toga bi trebalo ove restorane skladištiti negde. S obzirom na to da korisnici neće čuvati veliki broj restorana u toj listi, niti da postoji toliko veliki broj restorana u sistemu, ove podatke čuvaćemo lokalno na mobilnom telefonu za sada. Za to ćemo koristiti react-native koncept AsyncStorage.

// react native imports

import AsyncStorage from "@react-native-async-storage/async-storage";

...

    const saveFavourites = async (value) => {

        try {

            const jsonValue = JSON.stringify(value);

            await AsyncStorage.setItem('@favourites', jsonValue);

        } catch (err) {

            console.log("Saving error", err);

        }

    };

    const loadFavourites = async () => {

        try {

            const value = await AsyncStorage.getItem('@favourites');

            if (value != null) {

                setFavourites(JSON.parse(value));

            }

        } catch (err) {

            console.log("Loading error", err);

        }

    }

...

    useEffect(() => {

        loadFavourites();

    }, [])

    useEffect(() => {

        saveFavourites(favourites);

    }, [favourites])

Trenutno imamo listu omiljenih restorana, međutim ona se ne povezuje sa konkretnim korisnikom koji ju je uneo, već ostaje ista za svakog koji se prijavi na aplikaciju. Zbog toga, nakon što postavimo autentikaciju u okviru naše aplikacije, trebalo bi da povežemo korisnike sa njihovim listama omiljenih restorana. Za ovo ćemo u našim funkcijama loadFavourites i saveFavourites kao identifikatore koristiti i id korisnika.

    const { user } = useContext(AuthenticationContext);

    const saveFavourites = async (value, uid) => {

        try {

            const jsonValue = JSON.stringify(value);

            await AsyncStorage.setItem(`@favourites-${uid}`, jsonValue);

        } catch (err) {

            console.log("Saving error", err);

        }

    };

    const loadFavourites = async (uid) => {

        try {

            const value = await AsyncStorage.getItem(`@favourites-${uid}`);

            if (value != null) {

                setFavourites(JSON.parse(value));

            }

        } catch (err) {

            console.log("Loading error", err);

        }

    }

...

    useEffect(() => {

        if (user) {

            loadFavourites(user.uid);

        }

    }, [user])

    useEffect(() => {

        if (user) {

            saveFavourites(favourites, user.uid);

        }

    }, [favourites, user])

Na ovaj način povezujemo listu omiljenih restorana sa svakim korisnikom.

# 9 DETALJI RESTORANA

## 9.1 Tranzicija sa liste svih restorana

Za početak ćemo postaviti opciju u okviru navigacije za ovu komponentu. Klikom na karticu restorana trebalo bi da budemo prebačeni na stranicu detalja, a ovo je omogućeno korišćenjem navigatora i komponente Pressable iz osnovnog paketa react-native.

            <RestaurantStack.Screen

                name="RestaurantsAll"

                component={RestaurantsScreen}

            />

            <RestaurantStack.Screen

                name="RestaurantDetail"

                component={() => <Text>Restaurant Details</Text>}

            />

Prebacivanje ćemo izvršiti pomoću funkcije navigate na sledeći način:

    <Pressable onPress={() => navigation.navigate("RestaurantDetail")}>

       <RestaurantInfoCard restaurant={item}/>

    </Pressable>

Zahvaljajući korišćenju stack navigacije iz sekcije 7.5, tranzicija između stranice svih restorana i pojedinačne stranice restorana se vrši s leva na desno preklapanjem ekrana. Takođe, da bismo se vratili na prethodnu stranicu treba samo da prevučemo prstom. Ono što želimo jeste da se to događa odozdo na gore, što ćemo postići korišćenjem TransitionPresets-a iz iste biblioteke iz koje dobijamo StackNavigation. Takođe, komponentu Pressable zamenićemo komponentom TouchableOpacity, kako bismo videli koju karticu smo kliknuli:

<RestaurantStack.Navigator

            screenOptions={{

                ...TransitionPresets.ModalPresentationIOS,

                headerShown: false,

            }}

        >

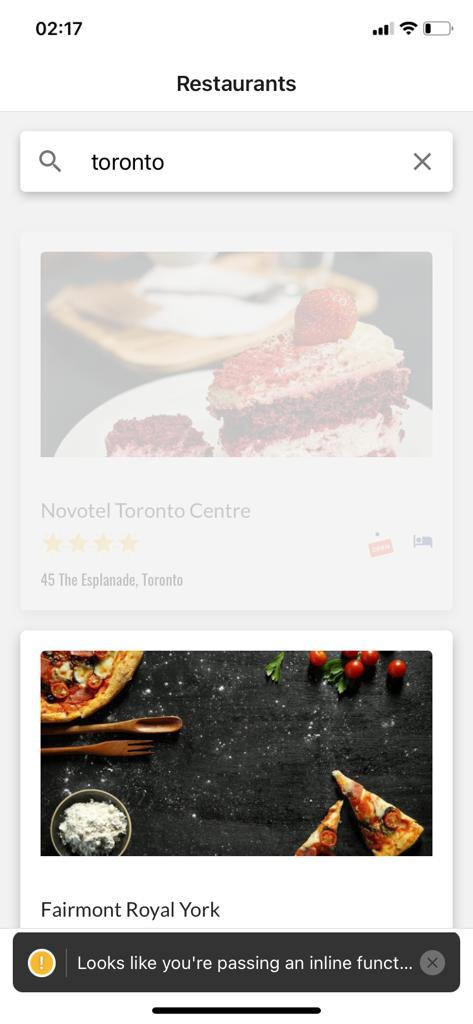
<TouchableOpacity

   onPress={() => navigation.navigate("RestaurantDetail")}

>

       <RestaurantInfoCard restaurant={item}/>

</TouchableOpacity>



Slika 22: Zamagljena kartica kada kliknemo na nju – TouchableOpacity

## 9.2 Renderovanje detalja restorana

Kreiraćemo novi screen, ovog puta za detalje restorana – restaurant-detail.screen.js. Za početak u ovu komponentu unećemo samo InfoCard tog restorana na sledeći način:

// react imports

import React from "react";

// components

import { RestaurantInfoCard } from "../components/restaurant-info-card.component";

import { SafeArea } from "../../../components/utility/safe-area.component";

export const RestaurantDetailScreen = ({ route }) => {

  const { restaurant } = route.params;

  return (

    <SafeArea>

      <RestaurantInfoCard restaurant={restaurant} />

    </SafeArea>

  );

};

Povezivanje ovog screen-a sa detaljima restorana vršimo u okviru restaurants.screen-a i u okviru navigatora na sledeći način:

            <TouchableOpacity

              onPress={() =>

                navigation.navigate("RestaurantDetail", {

                  restaurant: item,

                })

              }

            >

              <RestaurantInfoCard restaurant={item}/>

            </TouchableOpacity>

            <RestaurantStack.Screen

                name="RestaurantDetail"

                component={RestaurantDetailScreen}

            />

U okviru detalja restorana pored ove osnovne kartice trebalo bi ubaciti u meni tog restorana, čime ćemo se baviti u nastavku.

## 9.3 Meni restorana

Za kreiranje menija restorana koristićemo komponentu iz biblioteke react native paper koja se zove List.Accordion. Za 4 opcije (Breakfast, Lunch, Dinner, Drinks) ćemo postaviti po jednu dropdown meni koji će moći da se otvara kako bi prikazao opciju u okviru ovih. Svaka od ove 4 opcije predstavljaće jedan List.Accordion elemnt sa nazivom, ikonicom, oznakom da li je proširena lista i logikom proširivanja liste kada se klikne na nju.

U okviru svakog List.Accordion elementa unećemo još nekoliko List.Item elemenata koji će predstavljati konkretne stavke menija. Takođe, da bismo mogli da vidimo sve stavke povlačenjem prsta na gore po ekranu sve ove elemente ćemo ubaciti u okviru ScrollView elementa.

      <SafeArea>

        <RestaurantInfoCard restaurant={restaurant} />

         <ScrollView>

            <List.Accordion

              title="Breakfast"

              left={(props) => <List.Icon {...props} icon="bread-slice" />}

             expanded={breakfastExpanded}

            onPress={() => setBreakfastExpanded(!breakfastExpanded)}

          >

               <List.Item title="Eggs Benedict" />

              <List.Item title="Classic Breakfast" />

           </List.Accordion>

           <List.Accordion

               title="Lunch"

               left={(props) => <List.Icon {...props} icon="hamburger" />}

              expanded={lunchExpanded}

              onPress={() => setLunchExpanded(!lunchExpanded)}

            >

                <List.Item title="Burger with Fries" />

                <List.Item title="Steak Sandwich" />

                <List.Item title="Mushroom Soup" />

          </List.Accordion>

          <List.Accordion

            title="Dinner"

            left={(props) => <List.Icon {...props} icon="food-variant" />}

            expanded={dinnerExpanded}

            onPress={() => setDinnerExpanded(!dinnerExpanded)}

           >

             <List.Item title="Spaghetti Bolognese" />

             <List.Item title="Steak Frites" />

           </List.Accordion>

                <List.Accordion

                    title="Drinks"

                    left={(props) => <List.Icon {...props} icon="cup" />}

                    expanded={drinksExpanded}

                    onPress={() => setDrinksExpanded(!drinksExpanded)}

                >

                    <List.Item title="Coffee" />

                    <List.Item title="Tea" />

                    <List.Item title="Coca-Cola" />

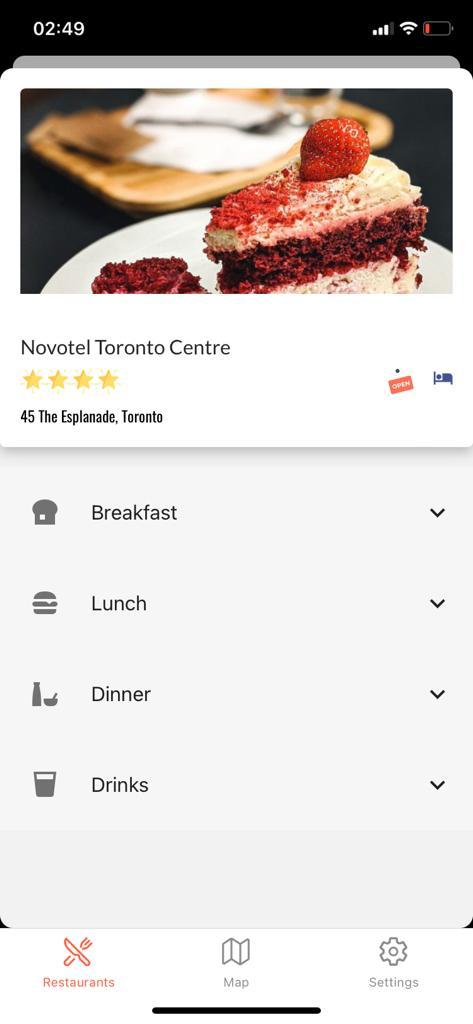
                    <List.Item title="Fanta" />

                    <List.Item title="Orangina" />

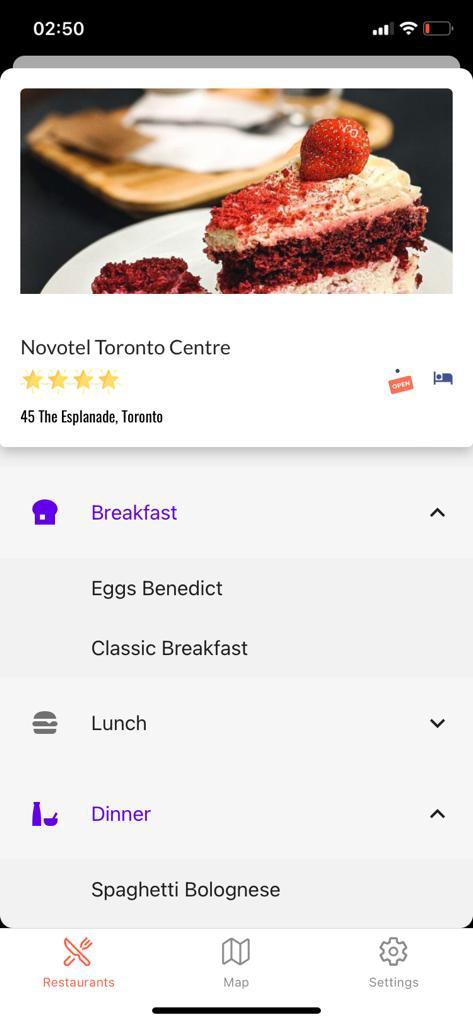
                </List.Accordion>

            </ScrollView>

        </SafeArea>



Slika 23: Detalji restorana sa opcijama iz menija



Slika 24: Otvaranje opcija iz menija

# 10 MAPE

## 10.1 Postavljanje React Native mape

Za početak mape ćemo implementirati korišćenjem MapView-a koji dobijamo iz biblioteke react-native-maps. Pomoću MapView-a možemo uneti mapu date platforme, tako da na Androidu možemo videti GoogleMaps, a na iOS-u AppleMaps.

// react imports

import React, { useContext, useState, useEffect } from "react";

// react maps imports

import MapView from "react-native-maps";

export const MapScreen = () => <MapView style= {{ height: "100%"}} />

....

<Tab.Screen name="Map" component={MapScreen} />



Slika 25: React Native default mapa (MapView)

## 10.2 Pretraga u okviru mape

Podesićemo pretragu mape, gde će korisnik moći da unese gradove i u okviru mape vidi lokacije restorana. Međutim, ovaj searchbar neće biti isti kao kod restorana, zbog čega ćemo kreirati potpuno novu komponentu za njega, po uzoru na postojeću. Ono što ćemo izmeniti jeste stilizacija ovog searchbar-a kao i logika u okviru map.js fajla.

Stilizacija novog searchbar-a:

const SearchContainer = styled.View`

  padding: ${(props) => props.theme.space[3]};

  position: absolute;

  z-index: 999;

  top: 40px;

  width: 100%;

`;

Takođe, kako bi se pretraga prenosila iz searchbar-a restorana na searchbar mape i obrnuto uvešćemo useEffect funkciju u obe komponente:

    useEffect(() => {

        setSearchKeyword(keyword);

    }, [keyword]);

MapScreen:

const Map = styled(MapView)`

    height: 100%;

    width: 100%;

`;

export const MapScreen = () => (

    <>

        <Search />

        <Map />

    </>

)



Slika 26: Pretraga mape

## 10.3 Unošenje restorana u okviru mape

Kako bismo uneli restorane u okviru naše mape, potrebno je da iskoristimo naše context-e za lokaciju i restorane. Za označavanje svakog restorana na mapi koristićemo element MapView.Marker. Međutim, da bi korisnici što lakše koristili ovu mapu potrebno je da se ona otvori u okviru određenog regiona, a ne na default poziciju kao do sada. Opcija viewport iz našeg fajla location.mock nam pruža onaj deo mape koji treba da se prikaže kada pretražimo neki od gradova.

Kako bismo odredili koliko treba da zumiramo ekran na određenu poziciju koristićemo delta latitude, tj. razliku između severo-istočnog latitude-a i jugo-zapadnog. Sve ove podatke dobijamo iz json fajlova koje smo kreirali za testiranje funkcionalnosti naše aplikacije.

U okviru opcije region unosimo geografsku širinu i dužinu za datu pretragu, tj. grad koji pretražujemo i razliku po dijagonali.

export const MapScreen = () => {

    const { location } = useContext(LocationContext);

    const { restaurants = [] } = useContext(RestaurantsContext);

    const [deltaLat, setDeltaLat] = useState(0);

    const { lat, lng, viewport } = location;

    useEffect(() => {

        const northeastLat = viewport.northeast.lat;

        const southwestLat = viewport.southwest.lat;

        setDeltaLat(northeastLat - southwestLat);

    }, [location, viewport]);

    return (

        <>

            <Search />

            <Map

                region={{

                    latitude: lat,

                    longitude: lng,

                    latitudeDelta: deltaLat,

                    longitudeDelta: 0.02,

                }}

            >

                {restaurants.map((restaurant) => {

                    return null;

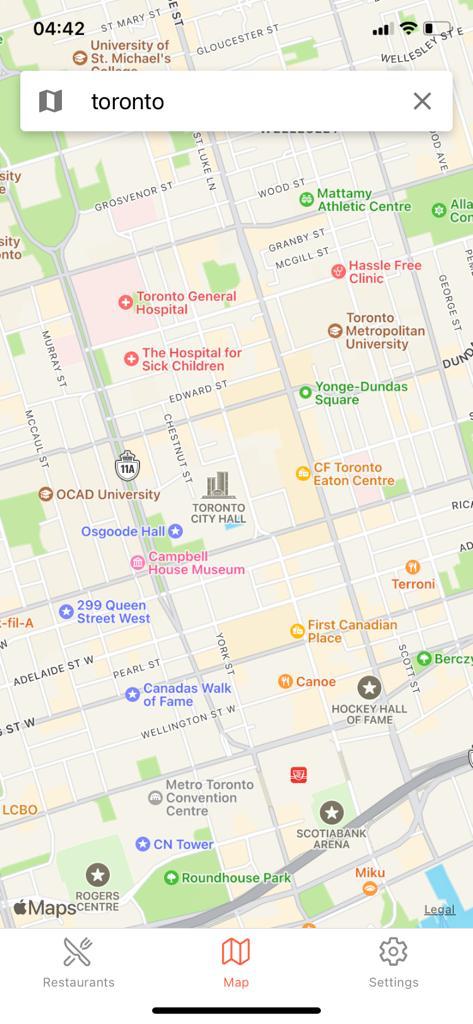
                })}

            </Map>

        </>

    );

};



Slika 27: Prikaz regiona koji pretražujemo

Pošto se sada otvaraju odgovarajući delovi mape na našu pretragu, možemo na osnovu liste restorana koju imamo da prikažemo markere za svaki od njih. Ove markere prikazujemo tako što funkcijom map prolazimo kroz niz restorana i za svaki izbacujemo po jedan marker na odgovarajuću poziciju.

     <MapView.Marker

           key={restaurant.name}

           title={restaurant.name}

           coordinate={{

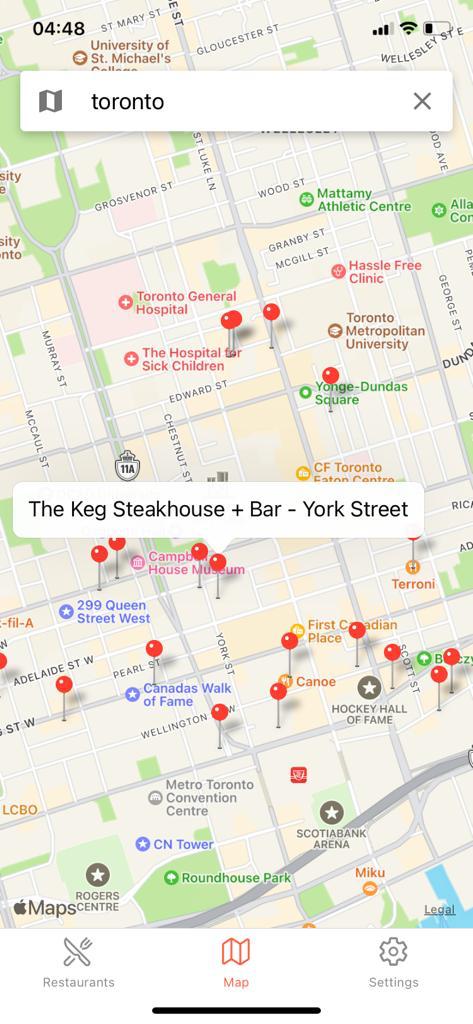
               latitude: restaurant.geometry.location.lat,

               longitude: restaurant.geometry.location.lng,

             }}

      >

      </MapView.Marker>



Slika 28: Markeri na mapi za svaki restoran u regionu sa ispisom imena onog na koji kliknemo

Kada kliknemo na markere trenutno dobijamo samo naziv restorana, međutim ono što želimo jeste da nam se prikaže slika restorana sa nekim informacijama. To ćemo uraditi pomoću MapView.Callout elementa unutar kog možemo uneti bilo koji layout i on će se prikazati na markeru kada kliknemo na njega. Za to ćemo kreirati nove komponente – MapCallout i CompactRestaurantInfo u okviru kog ćemo smestiti parametre koje želimo da prikažemo.

// MapCallout unutar MapView.Marker

<MapView.Callout>

     <MapCallout restaurant={restaurant} />

</MapView.Callout>

MapCallout:

export const MapCallout = ({ restaurant }) => {

    return (

        <CompactRestaurantInfo

            restaurant={restaurant}

        />

    );

};

CompactRestaurantInfo:

const CompactImage = styled.Image`

    border-radius: 10px;

    width: 120px;

    height: 100px;

`;

const Item = styled.View`

    padding: 10px;

    max-width: 120px;

    align-items: center;

`;

export const CompactRestaurantInfo = ({ restaurant }) => {

    return (

        <Item>

            <CompactImage source={{ uri: restaurant.photos[0] }} />

            <Text center variant="caption" numberOfLines={3}>

                {restaurant.name}

            </Text>

        </Item>

    );

};



Slika 29: Novi prikaz kada kliknemo na marker

Međutim, sliku restorana možemo videti samo na iOS uređajima sa trenutnom postavkom, zbog čega ćemo morati da optimizujemo naš kod i za Android uređaje, a to ćemo uraditi korišćenjem WebView-a na sledeći način:

// react native imports

import { Platform } from "react-native";

import WebView from "react-native-webview";

...

const CompactWebview = styled(WebView)`

    border-radius: 10px;

    width: 120px;

    height: 100px;

`;

...

const isAndroid = Platform.OS === "android";

...

    const Image = isAndroid ? CompactWebview : CompactImage;

    return (

        <Item>

            <Image source={{ uri: restaurant.photos[0] }} />

            <Text center variant="caption" numberOfLines={3}>

                {restaurant.name}

            </Text>

        </Item>

    );

Takođe, potrebno je da povežemo naše markere sa stranicama detalja ovih restorana, tako da kada kliknemo na marker i zatim na karticu detalja dobijemo sve detalje sa menijem. To ćemo uraditi tako što unutar MapView.Callout taga unesemo i funkciju u okviru onPress opcije.

 <MapView.Callout

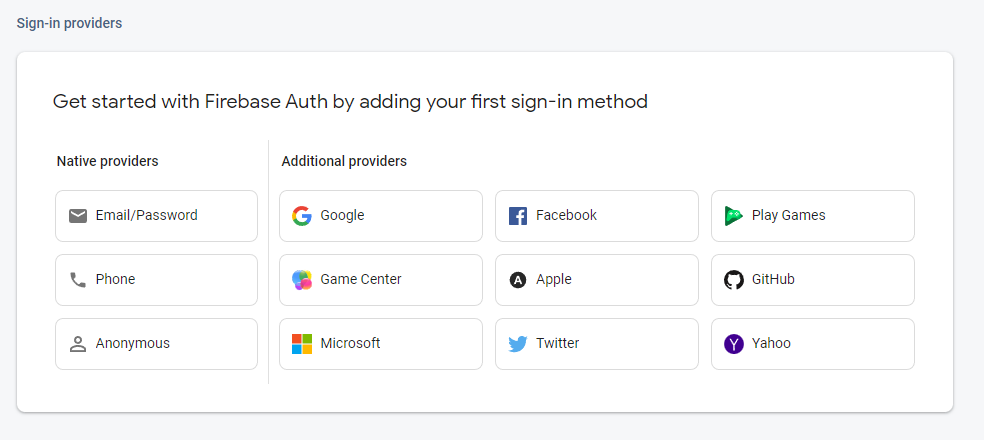
     onPress={() => navigation.navigate("RestaurantDetail", { restaurant})}

 >

# 11 AUTENTIKACIJA

## 11.1 Postavljanje i testiranje autentikacije

Za kreiranje korisnika i logike prijavljivanja na aplikaciju koristićemo Firebase. Na Firebase se logujemo sa našim Google nalogom. U okviru Firebase-a kreiraćemo novi projekat, kom ćemo dati naziv Order. Od servisa koje pruža Firebase trenutno nam treba servis Authentication, tako da ćemo ga izabrati iz menija sa leve strane i kliknuti na dugme Get Started. Firebase nam ovde pruža različite načine za autentikaciju, mi ćemo odabrati osnovni način, e-mail i password.



Slika 30: Moduli autentikacije

Sada je potrebno da spojimo naš Firebase projekat sa projektom na našem računaru. Za početak ćemo instalirati biblioteku komandom expo install firebase. Zatim ćemo povezati našu aplikaciju sa firebaseom u okviru dashboard-a i kopirati kredencijale naše baze u naš projekat kako bismo izvršili povezivanje.

// firebase import

import firebase from 'firebase/app';

import 'firebase/auth';

const firebaseConfig = {

    apiKey: "AIzaSyCwssn0r41pYSiWp8kcVx3mrLai2HJgYuY",

    authDomain: "order-54af9.firebaseapp.com",

    projectId: "order-54af9",

    storageBucket: "order-54af9.appspot.com",

    messagingSenderId: "227177403767",

    appId: "1:227177403767:web:7ea992a8c4ab3975463d72"

};

// firebase initialization

if(!firebase.apps.length) {

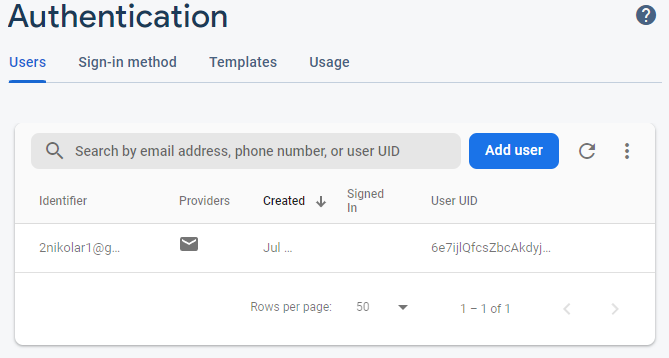
    firebase.initializeApp(firebaseConfig);

}

const projectAuth = firebase.auth();

export { projectAuth };

Sada ćemo u okviru našeg fajla App.js testirati prijavljivanje na aplikaciju. Za to ćemo u okviru manuelno kreirati novog korisnika i uneti njegove informacije:



Slika 31: Kreirani korisnik

Testiranje prijavljivanja:

  const [isAuthenticated, setIsAuthenticated] = useState(false);

  useEffect(() => {

    projectAuth.signInWithEmailAndPassword("2nikolar1@gmail.com", "test1234")

      .then((user) => {

        console.log(user);

        setIsAuthenticated(true);

      }).catch((err) => {

        console.log(err);

      });

  }, []);

Nakon pokretanja aplikacije u terminalu nam se ispisuju podaci o ovom korisniku. Međutim, ovu funkciju nećemo uneti u okviru našeg App.js fajla već ćemo kreirati novi servis koji će se baviti autentikacijom.

## 11.2 Servis za autentikaciju

Kao i kod svih servisa u okviru aplikacije kreiraćemo novi kontekst i u okviru service fajla ćemo napraviti funkciju za logovanje na platformu.

// react imports

import React, { useState, createContext } from "react";

// authentication

import { loginRequest } from "./authentication.service";

export const AuthenticationContext = createContext();

export const AuthenticationContextProvider = ({ children }) => {

    const [isLoading, setIsLoading] = useState(false);

    const [user, setUser] = useState(null);

    const [error , setError] = useState(null);

    const onLogin = (email, password) => {

        setIsLoading(true);

        loginRequest(email, password).then((u) => {

            setIsLoading(false);

            setUser(u);

        }).catch((err => {

            setIsLoading(false);

            setError(err);

        }));

    };

    return (

        <AuthenticationContext.Provider

            value={{

                user,

                isLoading,

                error,

                onLogin,

            }}

        >

            { children }

        </AuthenticationContext.Provider>

    )

}

// firebase

import { projectAuth } from "../firebase/config";

export const loginRequest = (email, password) => {

    projectAuth.signInWithEmailAndPassword(email, password);

}

        <AuthenticationContextProvider>

          <FavouritesContextProvider>

            ...

          </FavouritesContextProvider>

        </AuthenticationContextProvider>

## 11.3 Navigacija na osnovu autentikacije

U ovom trenutno autentikacija se vrši automatski i korisnik se odmah prijavljuje na aplikaciju. Međutim, trebalo bi uvesti mogućnost unošenja e-mail adrese i lozinke kako bi se korisnik prijavio pre nego što mu se otvori aplikacija. Za to ćemo kreirati komponentu AccountNavigator, na koju će korisnik biti preusmeren ukoliko autentikacija nije izvršena, tj. ukoliko nije prijavljen.

// react imports

import React from "react";

// react native imports

import { Text, View } from "react-native";

// navigation

import { createStackNavigator } from "@react-navigation/stack";

const Stack = createStackNavigator();

export const AccountNavigator = () => (

    <Stack.Navigator headerMode="none">

      <Stack.Screen

        name="Main"

        component={() => (

          <View>

            <Text>Account Screen</Text>

          </View>

        )}

      />

      <Stack.Screen

        name="Login"

        component={() => (

          <View>

            <Text>Login Screen</Text>

          </View>

        )}

      />

    </Stack.Navigator>

  );

    return  (

        <NavigationContainer>

            {isAuthenticated ? <AppNavigator /> : <AccountNavigator />}

        </NavigationContainer>

    );

## 11.4 Screen-ovi autentikacije

Stranica autentikacije će zapravo imati tri screen-a, AccountScreen, LoginScreen i RegisterScreen, što treba da implementiramo u okviru Account.Navigator-a:

export const AccountNavigator = () => (

    <Stack.Navigator screenOptions={{headerShown: false}}>

      <Stack.Screen

        name="Main"

        component={AccountScreen}

      />

      <Stack.Screen

        name="Login"

        component={LoginScreen}

      />

      <Stack.Screen

        name="Register"

        component={RegisterScreen}

      />

    </Stack.Navigator>

  );

Takođe, svaki Screen treba da kreiramo u okviru foldera Account u features. Pored toga, želimo da prikažemo određenu pozadinu koja će se nalaziti iza formi za kreiranje naloga i prijavljivanje, što ćemo sada uraditi korišćenjem ImageBackground komponente iz React Native-a.

// styles

import styled from "styled-components/native";

export const AccountBackground = styled.ImageBackground.attrs({

  source: require("../../../../assets/home\_background.jpg"),

})`

  flex: 1;

  align-items: center;

  justify-content: center;

`;

export const AccountCover = styled.View`

  position: absolute;

  width: 100%;

  height: 100%;

  background-color: rgba(255, 255, 255, 0.3);

`;

Početni prikaz Account.Screen-a:

// react imports

import React from "react";

// components - styles

import { AccountBackground, AccountCover } from "../components/account.styles";

export const AccountScreen = () => {

    return (

        <AccountBackground>

          <AccountCover />

        </AccountBackground>

      );

};

Na ovaj način implementirani su i Login i Register Screen-ovi.



Slika 32: Pozadina u okviru AccountScreen-a

### 11.4.1 AccountScreen

U okviru AccountScreen-a imaćemo 2 dugmeta koja će voditi na stranice Login i Register. Njih ćemo implementirati korišćenjem komponente Button iz React Native-a. Međutim, Button komponenta koristi default prikaz dugmića na iOS-u i Androidu, zbog čega će na iOS-u prikazati samo text kao dugme. Zbog ovoga je bolje koristi React Native Paper Button.

// react imports

import React from "react";

// components - styles

import { AccountBackground, AccountCover, AccountContainer, AuthButton } from "../components/account.styles";

import { Spacer } from "../../../components/spacer/spacer.component";

export const AccountScreen = ({ navigation }) => {

    return (

        <AccountBackground>

          <AccountCover />

          <AccountContainer>

            <AuthButton

                icon="login"

                mode="contained"

                onPress={() => navigation.navigate("Login")}

            >

              Login

            </AuthButton>

            <Spacer size="large">

            <AuthButton

                icon="badge-account-horizontal-outline"

                mode="contained"

                onPress={() => navigation.navigate("Register")}

            >

              Register

            </AuthButton>

            </Spacer>

          </AccountContainer>

        </AccountBackground>

      );

};

// buttons container

export const AccountContainer = styled.View`

  background-color: rgba(255, 255, 255, 0.7);

  padding: ${(props) => props.theme.space[4]};

  margin-top: ${(props) => props.theme.space[2]};

`;

// buttons

export const AuthButton = styled(Button).attrs({

  color: colors.brand.primary,

})`

  padding: ${(props) => props.theme.space[2]};

`;



Slika 33: Account Screen

### 11.4.2 LoginScreen

LoginScreen će se sadržati od login forme, koja će imati dva polja – e-mail i password, za koje će biti iskorišćen TextInput iz React Native Paper-a. Kada korisnik klikne na dugme Login, ukoliko je uneo odgovarajuće podatke, biće prebačen na stranicu svih restorana.

// react imports

import React, { useContext, useState } from "react";

// context

import { AuthenticationContext } from "../../../services/authentication/autentication.context";

// components - styles

import { AccountBackground, AccountCover, AccountContainer,

          AuthButton, AuthInput } from "../components/account.styles";

import { Spacer } from "../../../components/spacer/spacer.component";

import { Text } from "../../../components/typography/text.component";

export const LoginScreen = () => {

  const [email, setEmail] = useState("");

  const [password, setPassword] = useState("");

  const { onLogin, error} = useContext(AuthenticationContext);

  return (

    <AccountBackground>

      <AccountCover />

      <AccountContainer>

        <AuthInput

          label="E-mail"

          value={email}

          textContentType="emailAddress"

          keyboardType="email-address"

          autoCapitalize="none"

          onChangeText={(ea) => setEmail(ea)}

        />

        <Spacer size="large">

          <AuthInput

            label="Password"

            value={password}

            textContentType="password"

            secureTextEntry

            autoCapitalize="none"

            secure

            onChangeText={(pass) => setPassword(pass)}

          />

        </Spacer>

        {error && (

          <Spacer size="large">

            <Text variant="error">{error}</Text>

          </Spacer>

        )}

        <Spacer size="large">

          <AuthButton

            icon="login"

            mode="contained"

            onPress={() => onLogin(email, password)}

          >

            Login

          </AuthButton>

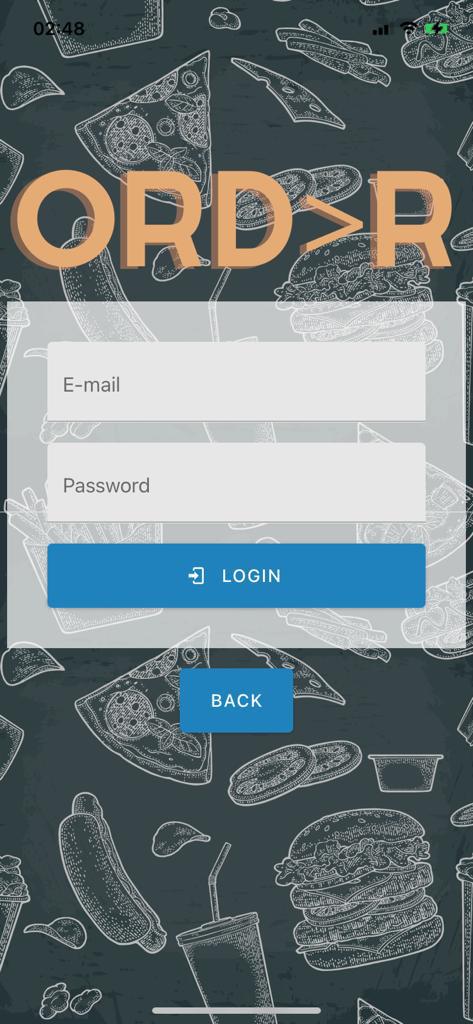
        </Spacer>

      </AccountContainer>

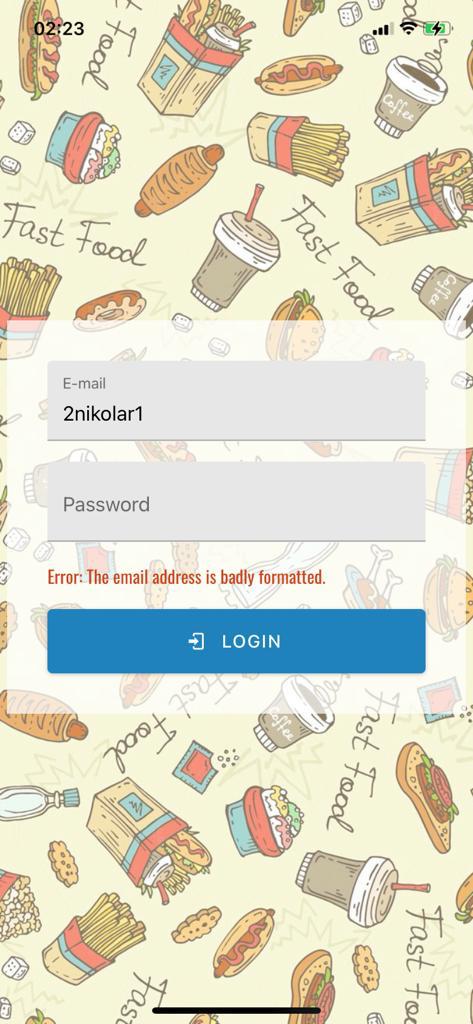
    </AccountBackground>

  );

}



Slika 34: Login Screen



Slika 35: Pogrešan format e-mail adrese



Slika 36: Pogrešna lozinka ili nepostojeći korisnik

### 11.4.3 RegisterScreen

// react imports

import React, { useContext, useState } from "react";

// context

import { AuthenticationContext } from "../../../services/authentication/autentication.context";

// components - styles

import { AccountBackground, AccountCover, AccountContainer,

          AuthButton, AuthInput, ErrorContainer, ErrorText } from "../components/account.styles";

import { Spacer } from "../../../components/spacer/spacer.component";

import { Text } from "../../../components/typography/text.component";

export const RegisterScreen = ({ navigation }) => {

  const [email, setEmail] = useState("");

  const [password, setPassword] = useState("");

  const [repeatedPassword, setRepeatedPassword] = useState("");

  const { onRegister, error} = useContext(AuthenticationContext);

  return (

    <AccountBackground>

      <AccountCover />

      <AccountContainer>

        <AuthInput

          label="E-mail"

          value={email}

          textContentType="emailAddress"

          keyboardType="email-address"

          autoCapitalize="none"

          onChangeText={(ea) => setEmail(ea)}

        />

        <Spacer size="large">

          <AuthInput

            label="Password"

            value={password}

            textContentType="password"

            secureTextEntry

            autoCapitalize="none"

            onChangeText={(pass) => setPassword(pass)}

          />

        </Spacer>

        <Spacer size="large">

          <AuthInput

            label="Confirm password"

            value={repeatedPassword}

            textContentType="password"

            secureTextEntry

            autoCapitalize="none"

            onChangeText={(pass) => setRepeatedPassword(pass)}

          />

        </Spacer>

        {error && (

          <ErrorContainer size="large">

            <ErrorText>

              <Text variant="error">{error}</Text>

            </ErrorText>

          </ErrorContainer>

        )}

        <Spacer size="large">

          <AuthButton

            icon="badge-account-horizontal-outline"

            mode="contained"

            onPress={() => onRegister(email, password, repeatedPassword)}

          >

            Register

          </AuthButton>

        </Spacer>

      </AccountContainer>

      <Spacer size="large">

          <AuthButton

            mode="contained"

            onPress={() => navigation.goBack()}

          >

            Back

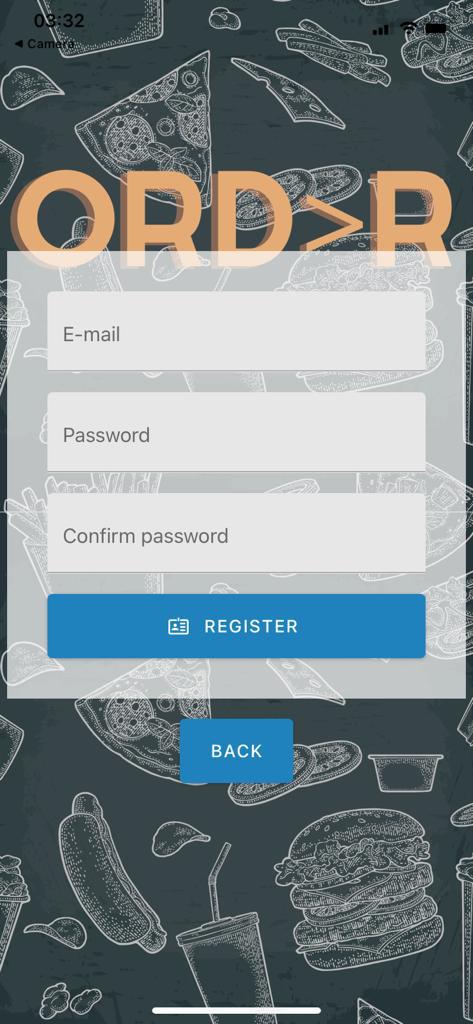
          </AuthButton>

      </Spacer>

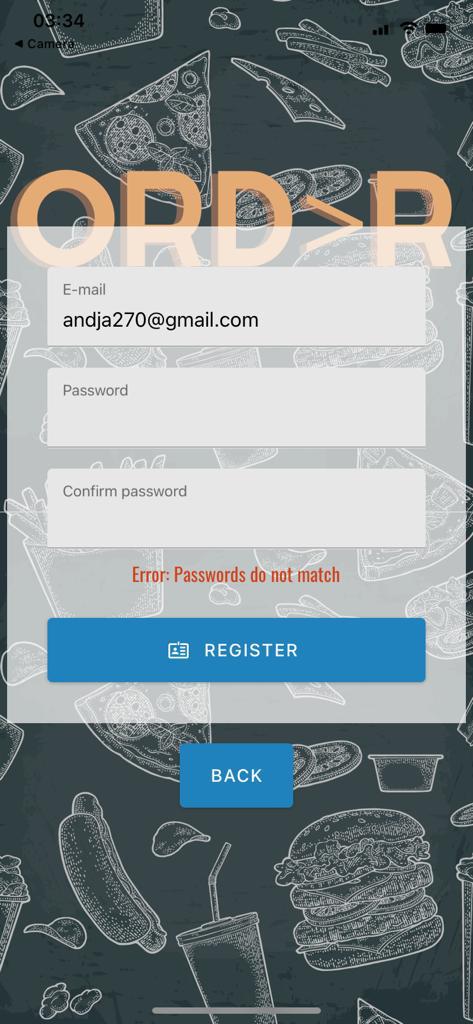
    </AccountBackground>

  );

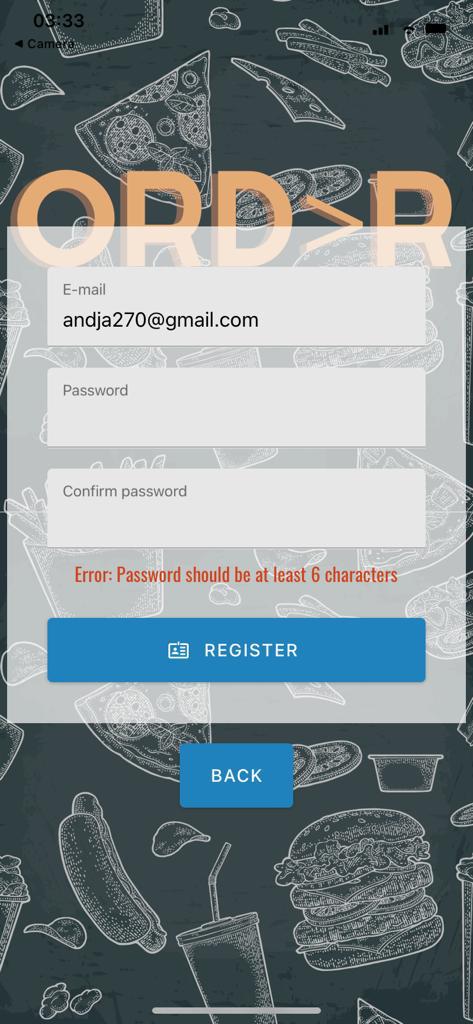
}



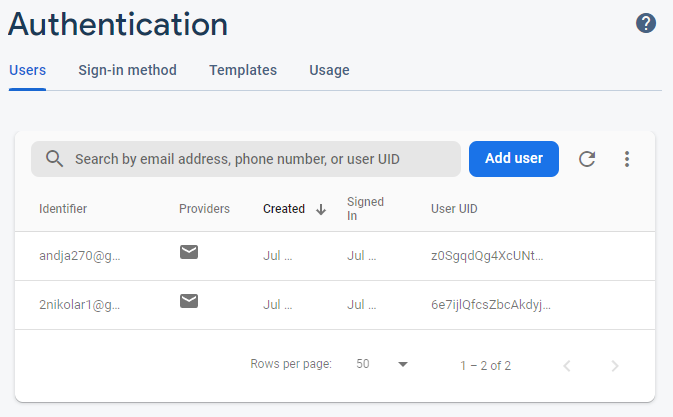
Slika 37: Register Screen



Slika 38: Lozinke se ne poklapaju



Slika 39: Kratka lozinka



Slika 40: Novi korisnik na Firebase-u

# 12 SEKCIJA SETTINGS

## 12.1 Postavljanje opcija u okviru Settings-a

U okviru sekcije Settings, koja predstavlja poslednju opciju u našoj navigaciji, prikazaćemo detalje o korisniku koji je trenutno ulogovan, listu omiljenih restorana i opciju za odjavljivanje sa aplikacije. Prvo kreiramo navigator za ovu sekciju i zatim poseban screen.

// react imports

import React from "react";

// navigation imports

import { createStackNavigator, CardStyleInterpolators } from "@react-navigation/stack";

// features

import { SettingsScreen } from "../../features/settings/screens/settings.screen";

const SettingsStack = createStackNavigator();

export const SettingsNavigator = ({ route, navigation }) => {

  return (

    <SettingsStack.Navigator

      headerMode="screen"

      screenOptions={{

        cardStyleInterpolator: CardStyleInterpolators.forHorizontalIOS,

        headerShown: false,

      }}

    >

      <SettingsStack.Screen

        name="SettingsScreen"

        component={SettingsScreen}

      />

      <SettingsStack.Screen name="Favourites" component={() => null} />

    </SettingsStack.Navigator>

  );

};

// react imports

import React, { useContext } from "react";

// react native imports

import { List } from "react-native-paper";

// context

import { AuthenticationContext } from "../../../services/authentication/autentication.context";

// components

import { SafeArea } from "../../../components/utility/safe-area.component";

export const SettingsScreen = ({ navigation }) => {

    const { onLogout } = useContext(AuthenticationContext);

    return (

      <SafeArea>

        <List.Section>

            <List.Item

                style={{ padding: 16 }}

                title="Favourites"

                description="View your favourites"

                left={(props) =>

                    <List.Icon {...props} color="black" icon="heart" />}

                onPress={() => navigation.navigate("Favourites")}

            />

            <List.Item

                style={{ padding: 16 }}

                title="Logout"

                left={(props) =>

                    <List.Icon {...props} color="black" icon="exit-run" />}

                onPress={onLogout}

            />

        </List.Section>

      </SafeArea>

    );

  };

     <Tab.Navigator

       screenOptions={createScreenOptions}

      >

        <Tab.Screen name="Restaurants" component={RestaurantsNavigator} />

        <Tab.Screen name="Map" component={MapScreen} />

        <Tab.Screen name="Settings" component={SettingsNavigator} />

      </Tab.Navigator>



Slika 41: Settings – početni izgled

## 12.2 Podaci o korisniku u okviru Settings-a

    <AvatarContainer>

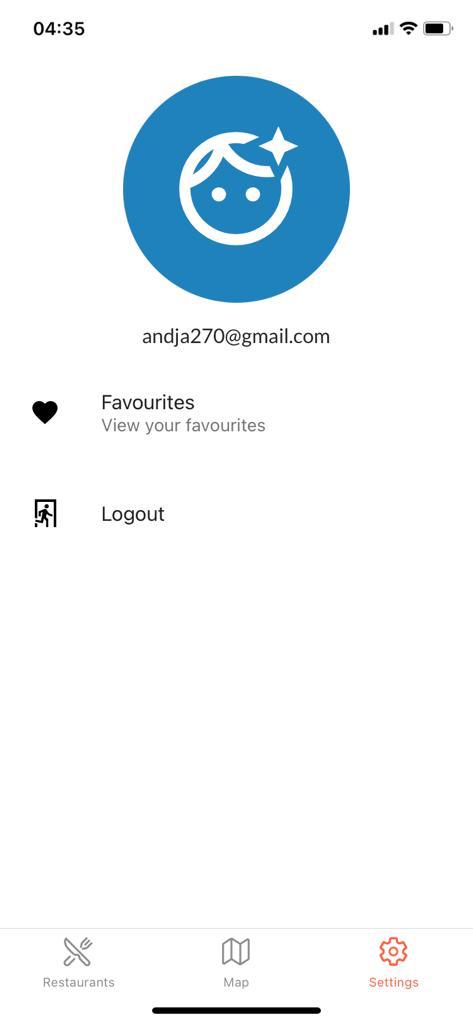
<Avatar.Icon size={180} icon="face-man-shimmer-outline" backgroundColor="#2182BD" />

      <Spacer position="top" size="large">

      <Text variant="label">{user.email}</Text>

      </Spacer>

    </AvatarContainer>



Slika 42: Settings sa e-mailom korisnika

## 12.3 Lista omiljenih restorana

Listu omiljenih restorana kreiraćemo u okviru posebnog screen-a – FavouritesScreen, tako što ćemo iskoristiti istu logiku kao kod kreiranja liste svih restorana, sa filtriranjem onih koji su u listi omiljenih.

// react imports

import React, { useContext } from "react";

// react native imports

import { TouchableOpacity } from "react-native";

// context

import { FavouritesContext } from "../../../services/favourites/favourites.context";

// styles

import styled from "styled-components/native";

// components

import { SafeArea } from "../../../components/utility/safe-area.component";

import { Text } from "../../../components/typography/text.component";

import { Spacer } from "../../../components/spacer/spacer.component";

import { RestaurantList } from "../../restaurants/components/restaurant-list.styles";

import { RestaurantInfoCard } from "../../restaurants/components/restaurant-info-card.component";

const NoFavouritesArea = styled(SafeArea)`

  align-items: center;

  justify-content: center;

`;

export const FavouritesScreen = ({ navigation }) => {

  const { favourites } = useContext(FavouritesContext);

  return favourites.length ? (

    <SafeArea>

      <RestaurantList

        data={favourites}

        renderItem={({ item }) => {

          return (

            <TouchableOpacity

              onPress={() =>

                navigation.navigate("RestaurantDetail", {

                  restaurant: item,

                })

              }

            >

              <Spacer position="bottom" size="large">

                <RestaurantInfoCard restaurant={item} />

              </Spacer>

            </TouchableOpacity>

          );

        }}

        keyExtractor={(item) => item.name}

      />

    </SafeArea>

  ) : (

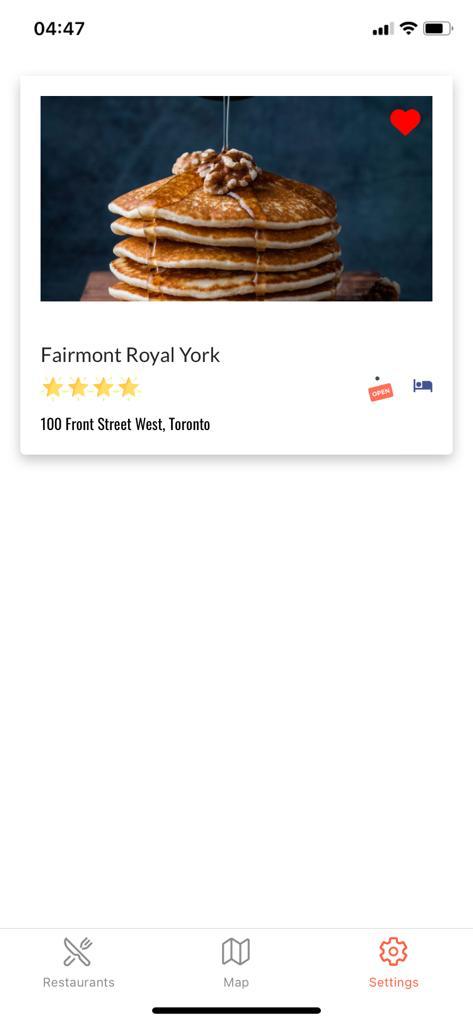
    <NoFavouritesArea>

      <Text center>No favourites yet</Text>

    </NoFavouritesArea>

  );

};



Slika 43: FavouritesScreen

# 13 ANIMACIJE

Za kreiranje animacija koristićemo React Native Animations biblioteku.

// react imports

import React, { useRef, useEffect } from "react";

// react native imports

import { Animated } from "react-native";

export const FadeInView = ({ duration = 1500, ...props }) => {

    const fadeAnim = useRef(new Animated.Value(0)).current;

    useEffect(() => {

        Animated.timing(fadeAnim, {

            toValue: 1,

            duration: duration,

            useNativeDriver: true,

        }).start();

    }, [fadeAnim, duration])

    return (

        <Animated.View

            style={{

                ...props.style,

                opacity: fadeAnim

            }}

        >

            { props.children }

        </Animated.View>

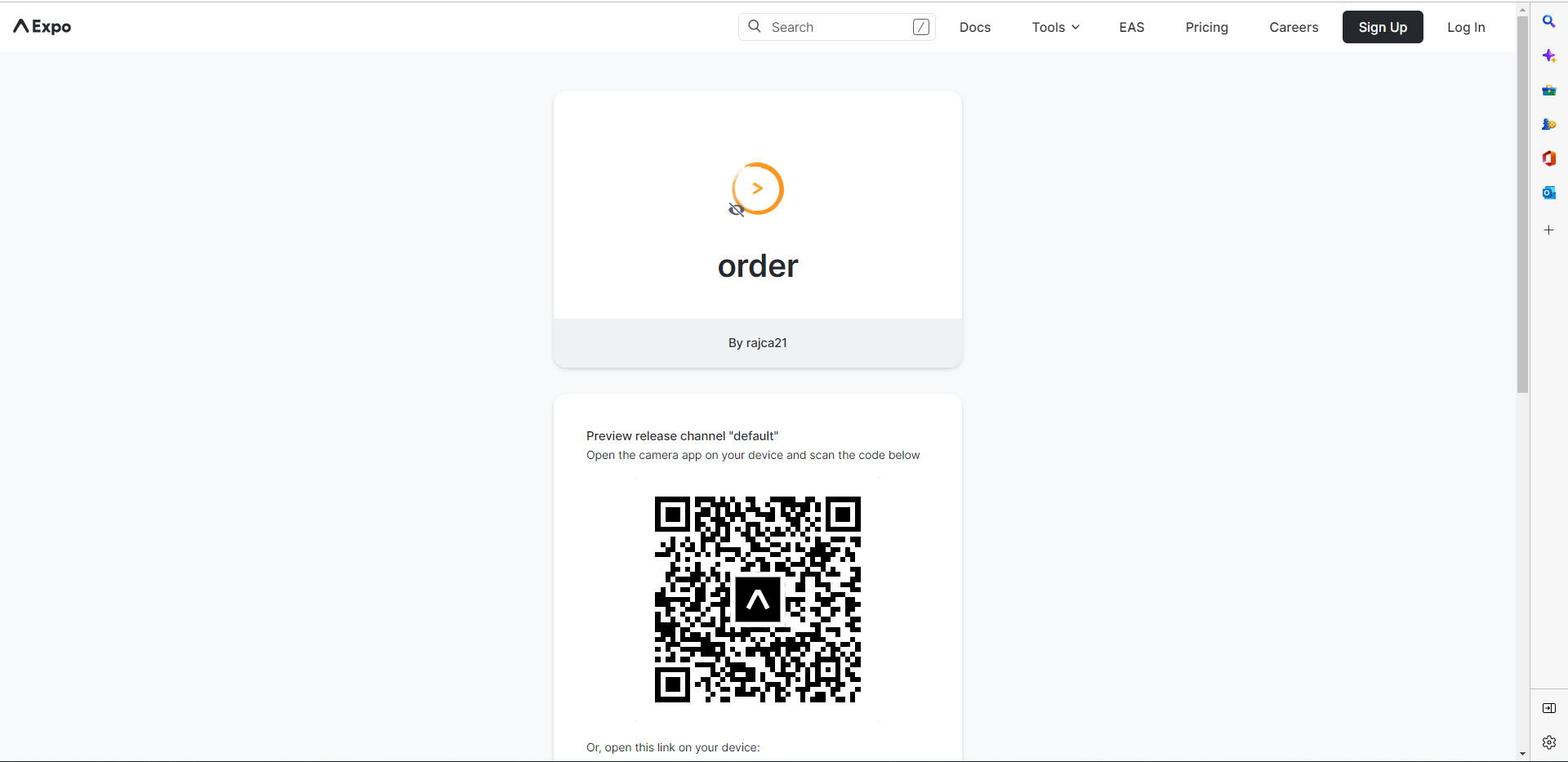
    );

};

Ovu komponentu implementiraćemo oko RestaurantInfo kartice.

# 14 POSTAVLJANJE APLIKACIJE NA EXPO

Kreiranu osnovnu verziju aplikacije možemo postaviti na naš Expo nalog kako bi bila dostupna i van pokretanja direktno sa našeg računara. Kako bismo to uradili u okviru terminala treba da unesemo komandu expo publish i da se ulogujemo na naš expo nalog ukoliko to već nije urađeno.



Slika 44: Postavljen projekat

# 15 KAMERA MODUL

## 15.1 Povezivanje sa SettingsScreen-om

Trenutno korisnici ne mogu postaviti svoju profilnu sliku, već se ona postavlja kao default React Native Paper Icon. Kako bi korisnici mogli da postavljaju svoje profilne slike, trebalo bi omogućiti korišćenje kamere u okviru aplikacije. S obzirom na to da ćemo sa SettingsScreen-a prelaziti na CameraScreen, potrebno je kreirati navigation fajl za kameru i povezati sa Settings Navigatorom.

   <SettingsStack.Screen name="Favourites" component={FavouritesScreen} />

   <SettingsStack.Screen name="Camera" component={CameraScreen} />

 </SettingsStack.Navigator>

    <TouchableOpacity onPress={() => navigation.navigate("Camera")}>

                <Avatar.Icon

                    size={180}

                    icon="face-man-shimmer-outline"

                    backgroundColor="#2182BD"

                />

    </TouchableOpacity>

## 15.2 Funkcionalnosti Expo Kamere

Da bismo koristili kameru u našoj aplikaciji treba da instaliramo paket expo-camera komandom expo install expo-camera. Na ovaj način možemo koristiti komponentu Camera, koja otvara našu kameru na mobilnom uređaju. Ovde možemo postaviti tip kamere i mi ćemo postaviti samo prednju kameru. Takođe, kada otvorimo kameru, potrebno je da damo dozvolu aplikaciji da pristupa našoj kameri, što će se proveravati prilikom svakog poziva kamere i ukoliko ta dozvola ne postoji, kamera se neće otvoriti.

// react imports

import React, { useRef, useState, useEffect } from "react";

// react native imports

import { View } from "react-native";

import { Camera } from "expo-camera";

// styles

import styled from "styled-components/native";

// components

import { Text } from "../../../components/typography/text.component";

const ProfileCamera = styled(Camera)`

    width: 100%;

    height: 100%;

`;

export const CameraScreen = () => {

    const [hasPermission, setHasPermission] = useState(null);

    const cameraRef = useRef();

    useEffect(() => {

        (async () => {

          const { status } = await Camera.requestCameraPermissionsAsync();

          setHasPermission(status === "granted");

        })();

    }, []);

    if (hasPermission === null) {

        return <View />;

    }

    if (hasPermission === false) {

        return <Text>No access to camera</Text>;

    }

    return (

        <ProfileCamera

            ref={(camera) => (cameraRef.current = camera)}

            type={Camera.Constants.Type.front}

        >

        </ProfileCamera>

    )

};

## 15.3 Kreiranje fotografije

Sa postojećom logikom, korisnici nemaju mogućnost da zapravo slikaju korišćenjem kamere, jer nemaju tu funkcionalnost ugrađenu u kodu. To ćemo postići tako što oko ProfileCamera elementa postavimo TouchableOpacity koji će pozivati funkciju snap.

        <TouchableOpacity onPress={snap}>

            <ProfileCamera

                ref={(camera) => (cameraRef.current = camera)}

                type={Camera.Constants.Type.front}

            >

            </ProfileCamera>

        </TouchableOpacity>

    const snap = async () => {

        if (cameraRef) {

            const photo = await cameraRef.current.takePictureAsync();

            console.log(photo);

        }

    };

## 15.4 Skladištenje slike i postavljanje na profil

Slično kao kod čuvanja liste omiljenih restorana, profilnu sliku ćemo čuvati lokalno, pošto će svaki korisnik moći da ima samo jednu profilnu sliku. Za ovo ćemo koristiti useFocusEffect hook kako bismo pokrenuli funkciju za vraćanje profilne slike iz lokalne memorije. Zatim, ukoliko korisnik ima fotografiju, ubacićemo je u okviru Avatar.Image elementa.

export const SettingsScreen = ({ navigation }) => {

    const { onLogout, user } = useContext(AuthenticationContext);

    const [photo, setPhoto] = useState(null);

    const getProfilePicture = async (currentUser) => {

        const photoUri = await AsyncStorage.getItem(`${currentUser.uid}-photo`);

        setPhoto(photoUri);

    }

    useFocusEffect(

        useCallback(() => {

            getProfilePicture(user);

        }, [user])

    );

...

     <TouchableOpacity onPress={() => navigation.navigate("Camera")}>

          {!photo && (

              <Avatar.Icon

                 size={180}

                 icon="face-man-shimmer-outline"

                 backgroundColor="#2182BD"

               />

          )}

           {photo && (

              <Avatar.Image

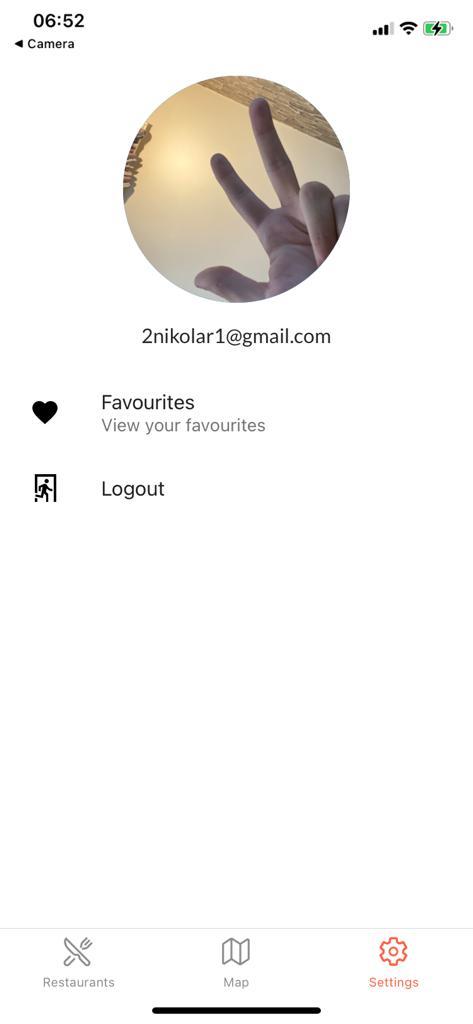
                  size={180}

                  source={{ uri: photo }}

                  backgroundColor="#2182BD"

               />

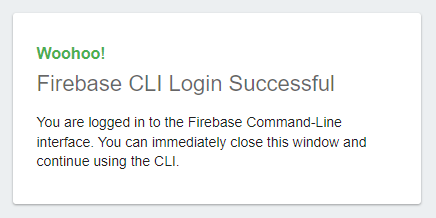
           )}



Slika 45: Postavljanje profilne slike

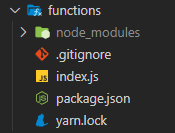
# 16 FIREBASE FUNKCIJE

U terminalu unosimo komandu firebase init functions. Ovo radimo kako bismo postavili naš projekat na Cloud i zatim koristili različite servise Firebase-a, kao što je na primer Geocoding o kome će kasnije biti reči. Nakon unošenja ove komande treba da povežemo direktorijum sa postojećim projektom na Firebase-u. Zatim ćemo za jezik kodiranja izabrati JavaScript i odbiti korišćenje EsLint-a.



Slika 46: Uspešno povezivanje sa Firebase CLI-om

Dobijamo novi folder functions:



Slika 47: Firebase functions folder

Kako bismo pokretali Firebase funkcije moramo biti u okviru ovog foldera, tako što ćemo u našem terminalu uneti komandu cd functions. Funckije koje možemo pokretati nalaze se u okviru fajla package.json:

  "scripts": {

    "serve": "firebase emulators:start --only functions",

    "shell": "firebase functions:shell",

    "start": "npm run shell",

    "deploy": "firebase deploy --only functions",

    "logs": "firebase functions:log"

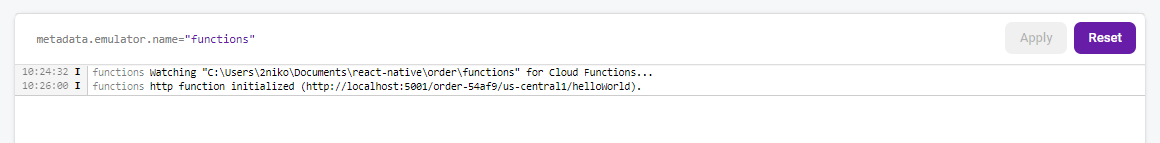
Pomoću funckije yarn serve, pokrećemo našu aplikaciju lokalno, na portu 4000. Sada Firebase Emulator pregleda fajl index.js i pokreće funkcije koje se u njemu nalaze. Možemo otkomentarisati Hello World funkciju u okviru ovog fajla i zatim će se ona prikazati na Firebase Emulatoru:

exports.helloWorld = functions.https.onRequest((request, response) => {

   functions.logger.info("Hello logs!", {structuredData: true});

   response.send("Hello from Firebase!");

});



Slika 48: Funkcija koju možemo da pokrenemo – drugi unos

Ukoliko kopiramo link koji se nalazi u zagradi kod unosa funkcije, možemo otvoriti tu stranicu. Ove funkcije koristićemo kako bismo prikupljali informacije o lokacijama naših restorana, bez korišćenje mock fajlova kao do sada.

## 16.1 Location mock fajlovi u okviru firebase funkcija

Sada ćemo u okviru index.js fajla kreirati funkcije za upravljanje mock lokacijama.

const functions = require("firebase-functions");

const { geocodeRequest } = require("./geocode");

exports.geocode = functions.https.onRequest((request, response) => {

    geocodeRequest(request, response);

});

Takođe, kreiraćemo novi folder gecode u okviru functions foldera gde ćemo kreirati naš objekat geocodeRequest:

const { locations: locationsMock } = require("./geocode.mock");

const url = require("url");

module.exports.geocodeRequest = (request, response) => {

  const { city } = url.parse(request.url, true).query;

  const locationMock = locationsMock[city.toLowerCase()];

  response.json(locationMock);

};

Takođe, prebacićemo naš location.mock file iz foldera service u ovaj novi folder i izmeniti funkciju za location.service na sledeći način kako bismo mogli da pretražujemo restorane po lokacijama:

export const locationRequest = (searchTerm) => {

  return fetch(

    `http://localhost:5001/order-54af9/us-central1/geocode?city=${searchTerm}`

    ).then((res) => {

      return res.json();

    });};

## 16.2 Restaurant mock u okviru firebase funkcija

Ovo je drugi deo naše aplikacije koji smo implementirali kao mock-up, zbog čega ćemo i njega uneti u okviru naših firebase funkcija.

exports.placesNearby = functions.https.onRequest((request, response) => {

    placesRequest(request, response);

});

const { mocks, addMockImage } = require("./mock");

const url = require("url");

module.exports.placesRequest = (request, response) => {

    const { location } = url.parse(request.url, true).query;

    const data = mocks[location];

    if(data) {

        data.results = data.results.map(addMockImage);

    }

    response.json(data);

}

module.exports.addMockImage = (restaurant) => {

  const randomImage =

    mockImages[Math.ceil(Math.random() \* (mockImages.length - 1))];

  restaurant.photos = [randomImage];

  return restaurant;

};

export const restaurantsRequest = (location) => {

  return fetch(

    `http://localhost:5001/order-54af9/us-central1/placesNearby?location=${location}`

    ).then((res) => {

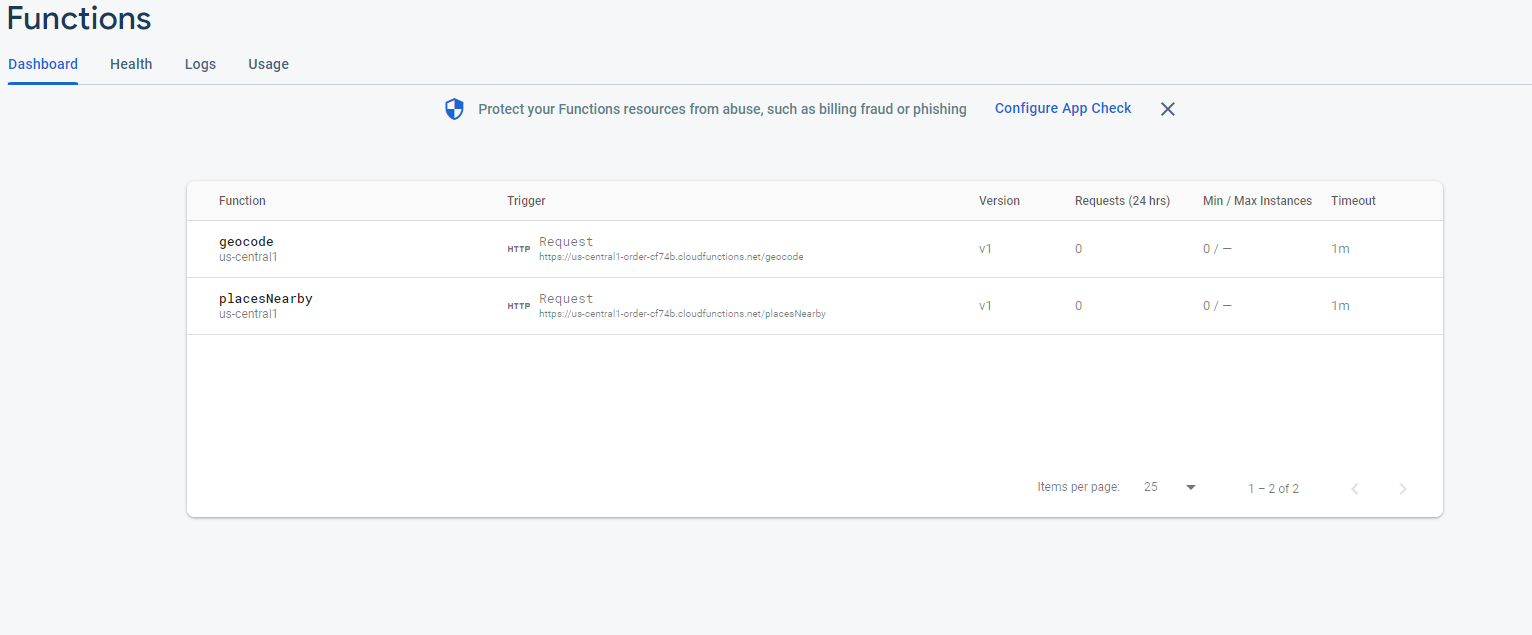
      return res.json();

    });

};

## 16.3 Deployment

Do sada smo ove izmene mogli da vidimo samo lokalno i one služe za testiranje prilikom izrade nekog projekta, zbog čega smo koristili opciju serve. Sada ćemo koristiti komandu deploy, kako bismo ove izmene povezali sa našim Firebase serverom i mogli da prikažemo ove izmene i van našeg računara. To radimo pomoću komande yarn deploy.



Slika 49: Kreirane funkcije na Firebase-u

URL-ove koje vidimo ispod naših funkcija bi sada trebalo da ubacimo u naše fetch funkcije kako bi funkcionisale.

Takođe, kreiraćemo upit koji će proveravati da li se trenutno nalazimo u fazi development-a ili smo postavili naš kod preko deploy-a, kako bismo znali koji URL da prosledimo:

export const liveHost = 'https://us-central1-order-cf74b.cloudfunctions.net';

const localHost = 'http://localhost:5001/order-cf74b/us-central1';

export const isDevelopment = process.env.NODE\_ENV === 'development';

export const host = isDevelopment ? localHost : liveHost;

Međutim, problem kod ovog koda je to što na Android uređajima ne rade http zahtevi, već samo https, zbog čega nećemo moći da pokrećemo lokalno našu aplikaciju. Zbog toga ćemo svugde, za sad, proslediti liveHost i pokretati našu aplikaciju live.

export const locationRequest = (searchTerm) => {

  return fetch(

    `${liveHost}/geocode?city=${searchTerm}`

  ).then((res) => {

    return res.json();

  });

};

export const restaurantsRequest = (location) => {

  return fetch(

    `${liveHost}/placesNearby?location=${location}`

    ).then((res) => {

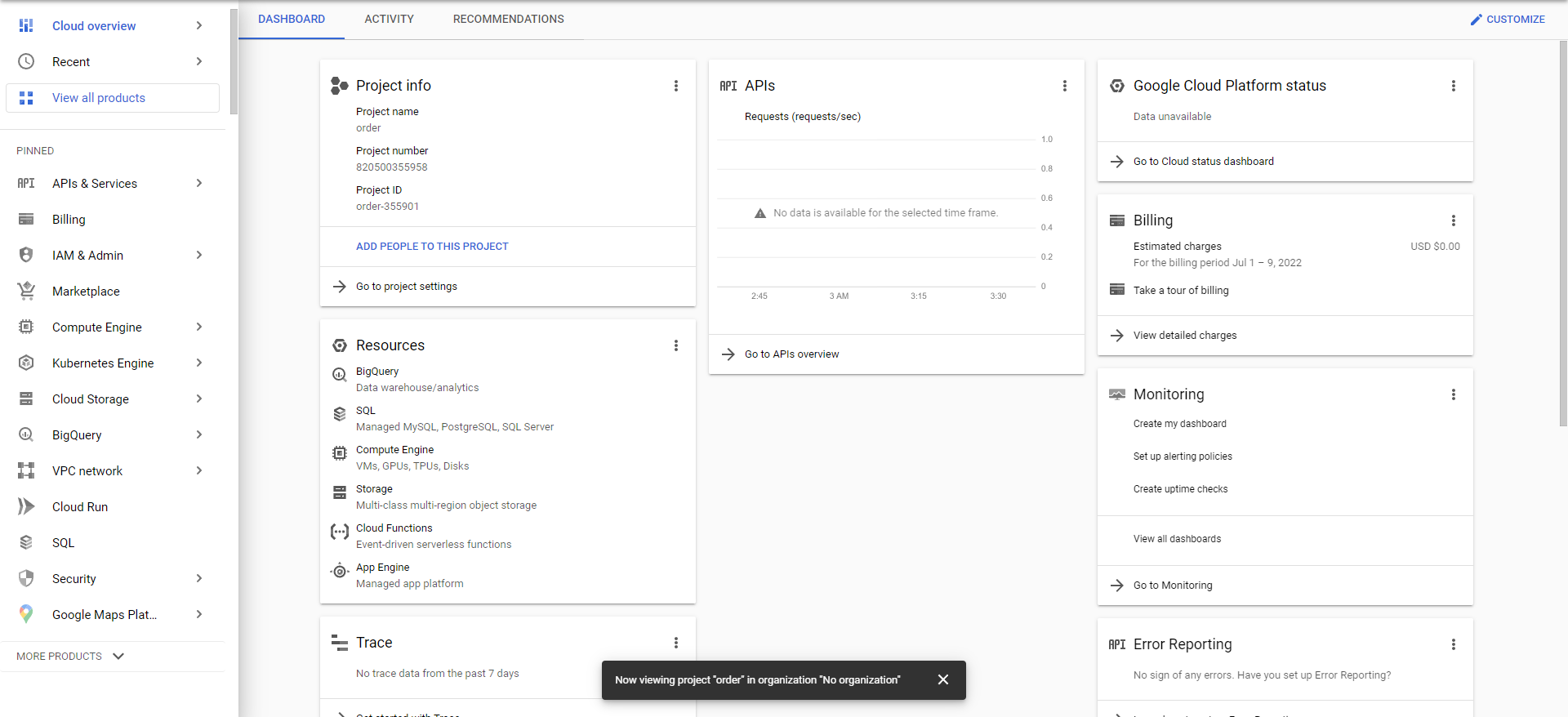
      return res.json();

    });

};

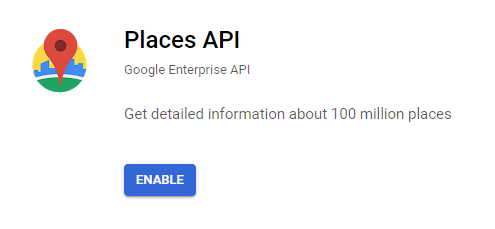
# 17 INTEGRACIJA SA GOOGLE MAPAMA

U prethodnoj sekciji postavili smo osnovu za integraciju naše aplikacije sa google servisima, a jedan od onih koje ćemo koristiti su Google mape, kako bismo prikupili prave, real-time podatke o našim restoranima i implementirali ih u aplikaciju umesto mock podataka. Zbog ovoga bi trebalo da kreiramo nalog na Google Cloud-u. Zatim ćemo kreirati naš projekat order u okviru Google Cloud-a.

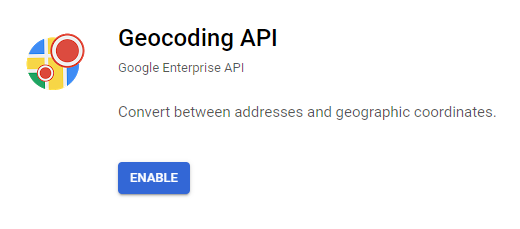


Slika 50: Kreiran projekat

Potrebno je da unesemo određene servise i prve koje ćemo uneti su Places API i Geocoding API.



Slika 51: Places API



Slika 52: Geocoding API

Takođe, kako ne bi došlo do velikog broja zahteva, koji se zatim naplaćuju u okviru opcije Quotas ćemo ograničiti broj zahteva za oba API-a na 1000 dnevno. Uz to, u okviru opcije Credentials kreiraćemo naš API key, koji ćemo ograničiti samo na Geocoding i Places API.

Da bismo na kraju ovo implementirali u okviru našeg projekta potrebno je da instaliramo biblioteku Node.js Client for Google Maps Services. Trenutno u našoj aplikaciji još uvek koristimo mock data, međutim, trebalo bi da prebacimo to na realne podatke koji će se prikupljati korišćenjem geocoding-a. Zbog toga ćemo poslati klijentski zahtev sa adresom (tj. gradom) i API ključem koji smo kreirali ranije.

const functions = require("firebase-functions");

const { geocodeRequest } = require("./geocode");

const { placesRequest } = require("./places");

const { Client } = require("@googlemaps/google-maps-services-js");

const client = new Client({});

exports.geocode = functions.https.onRequest((request, response) => {

  geocodeRequest(request, response, client);

});

exports.placesNearby = functions.https.onRequest((request, response) => {

  placesRequest(request, response, client);

});

const { mocks, addMockImage } = require("./mock");

const url = require("url");

const functions = require("firebase-functions");

const addGoogleImage = (restaurant) => {

  const ref = restaurant.photos[0].photo\_reference;

  if (!ref) {

    restaurant.photos = [

      "https://www.foodiesfeed.com/wp-content/uploads/2019/06/top-view-for-box-of-2-burgers-home-made-600x899.jpg",

    ];

    return restaurant;

  }

  restaurant.photos = [

    `https://maps.googleapis.com/maps/api/place/photo?maxwidth=400&photoreference=${ref}&key=${

      functions.config().google.key

    }`,

  ];

  return restaurant;

};

module.exports.placesRequest = (request, response, client) => {

  const { location, mock } = url.parse(request.url, true).query;

  if (mock === "true") {

    const data = mocks[location];

    if (data) {

      data.results = data.results.map(addMockImage);

    }

    return response.json(data);

  }

  client

    .placesNearby({

      params: {

        location: location,

        radius: 1500,

        type: "restaurant",

        key: functions.config().google.key,

      },

      timeout: 1000,

    })

    .then((res) => {

      res.data.results = res.data.results.map(addGoogleImage);

      return response.json(res.data);

    })

    .catch((e) => {

      response.status(400);

      return response.send(e.response.data.error\_message);

    });

};

const { locations: locationsMock } = require("./geocode.mock");

const url = require("url");

const functions = require("firebase-functions");

module.exports.geocodeRequest = (request, response, client) => {

  const { city, mock } = url.parse(request.url, true).query;

  if (mock === "true") {

    const locationMock = locationsMock[city.toLowerCase()];

    return response.json(locationMock);

  }

  client

    .geocode({

      params: {

        address: city,

        key: functions.config().google.key,

      },

      timeout: 1000,

    })

    .then((res) => {

      return response.json(res.data);

    })

    .catch((e) => {

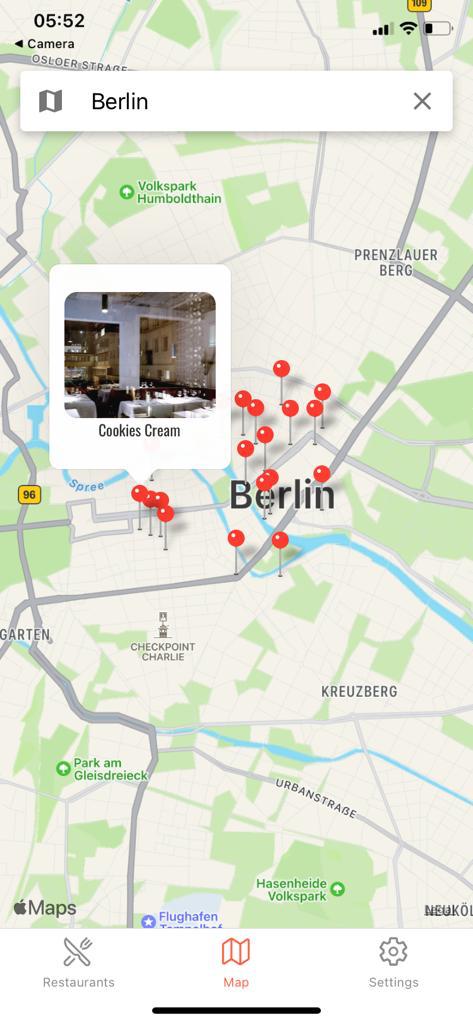
      response.status(400);

      return response.send(e.response.data.error\_message);

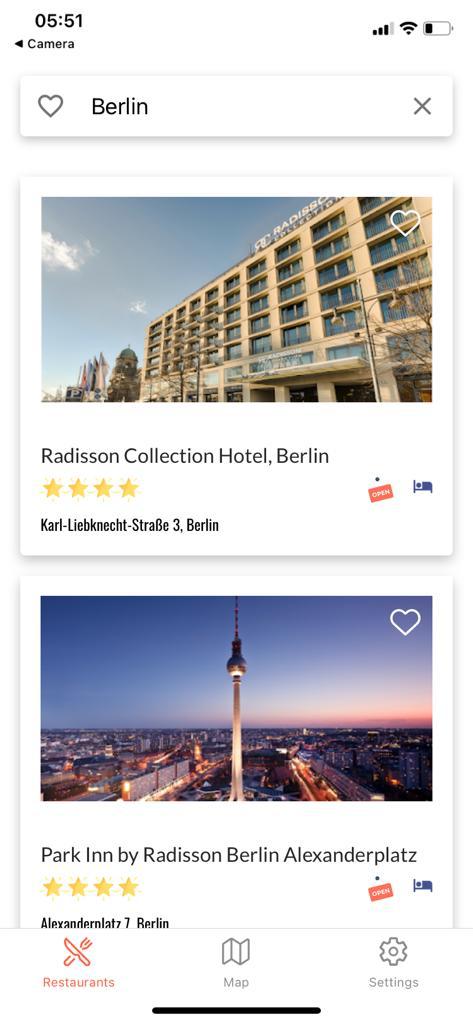
    });

};

Sada u okviru naše aplikacije možemo pretražiti bilo koje mesto na svetu i biće nam prikazani restorani povezani sa njim, barem oni koji su uneti u okviru Places API-a kao restaurants.



Slika 53: Mapa za pretragu Berlin



Slika 54: Pretraga restorana za Berlin

Na ovaj način isključili smo korišćenje mock podataka i sada koristimo prave podatke za restorane širom sveta. Od naših mock parametara ostaju još slike koje će se primeniti ukoliko neki restoran nema svoju sliku.

# 18 NARUČIVANJE

Za svaki restoran u okviru naše aplikacije implementiraćemo mogućnost naručivanja specijaliteta kuće, koji će imati istu cenu za svaki restoran. Korisnici će zahvaljujući integraciji aplikacije sa Stripe modulom moći da unesu broj svoje kreditne kartice i izvrše plaćanje onih specijaliteta koje su dodali u korpu. Stripe je kompanija koja nudi mogućnost outsource-ovanja funkcionalnosti plaćanja raznim kompanijama, kao što je npr. Amazon. Korišćenjem Stripe-a olakšava se implementacija ovog servisa u okviru aplikacije, uz naknadu za svaku transakciju koja se izvrši. Unećemo biblioteku stripe-client kako bismo započeli sa kreiranjem payment tokena za naše korisnike.

## 18.1 Postavljanje Checkout-a u okviru menija

Postavićemo opciju checkout u okviru menija naše aplikacije, tako da ćemo za početak u fajlu app.navigator implementirati CheckoutScreen, na isti način kao i ostale screen-ove do sada.

import { CheckoutScreen } from "../../features/checkout/screens/checkout.screen";

...

const TAB\_ICON = {

  Restaurants: "md-restaurant-outline",

  Checkout: "cart-outline",

  Map: "ios-map-outline",

  Settings: "settings-outline",

};

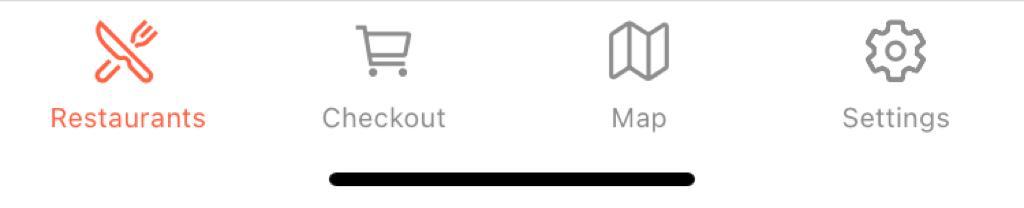
...

       <Tab.Screen name="Restaurants" component={RestaurantsNavigator} />

       <Tab.Screen name="Checkout" component={CheckoutScreen} />

       <Tab.Screen name="Map" component={MapScreen} />

       <Tab.Screen name="Settings" component={SettingsNavigator} />



Slika 55: Checkout opcija u meniju

## 18.2 Unos kreditne kartice

Za kreiranje forme za unos kreditne kartice koristićemo biblioteku react-native-credit-card-input, konkretno komponentu LiteCreditCardInput. Ova komponenta prima atribut onChange, koji će pokrenuti funkciju koja će za početak vršiti validaciju unosa, tj. proveravaće da li smo uneli sve podatke.

// react imports

import React from "react";

// react native imports

import { LiteCreditCardInput } from "react-native-credit-card-input";

export const CreditCardInput = () => {

    const onChange = (formData) => {

        const { values, status } = formData;

        const isIncomplete = Object.values(status).includes("incomplete");

    }

    return (

        <LiteCreditCardInput onChange={onChange}/>

    );

};

export const CheckoutScreen = () => {

    return(

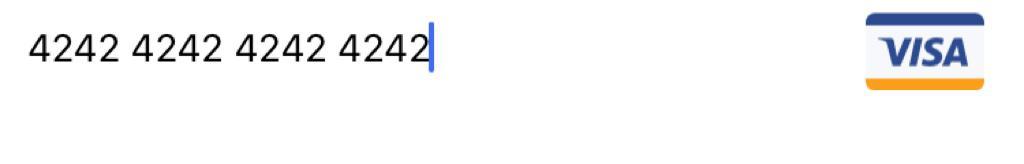
        <SafeArea>

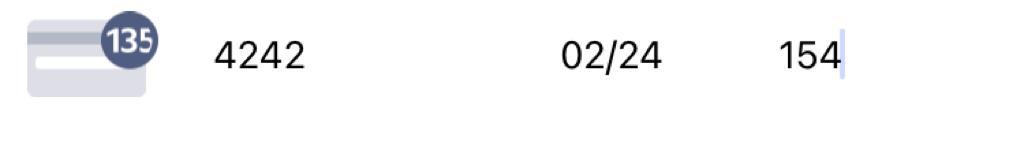
            <CreditCardInput />

        </SafeArea>

    );

};





Slika 56: Unos podataka u LiteCreditCardInput

## 18.3 Stripe tokeni

Potrebno je kreirati stripe biblioteku korišćenjem funkcije createStripe, koja kao parametar prima API key koji možemo pronaći u okviru našeg Stripe naloga. Sada ćemo na osnovu toga pokrenuti funkciju createToken, sa dummy karticom koju nam daje Stripe za testiranje na sledeći način:

// stripe imports

import createStripe from "stripe-client";

// publishable API key

const stripe = createStripe(

"pk\_test\_51LJrSlEYB36uQTpUCzWP2qZtzn9n9EqCoZKUQpIsdXoq2jWYP4EeiVwzCnGgsHizgdq6eJxUMf854NIvq0us9IPD00C5wpVXQV"

);

...

const card = {

            number: '4242424242424242',

            exp\_month: '02',

            exp\_year: '24',

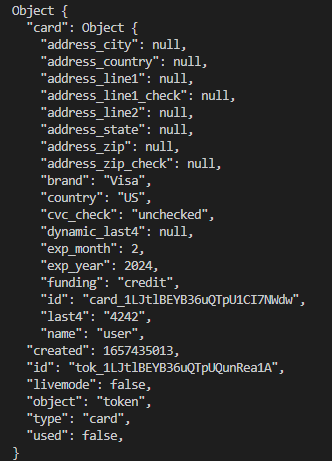
            cvc: '242',

            name: 'user',

        };

const info = await stripe.createToken({ card });

console.log(info);



Slika 57: Kreirani token za plaćanje

Umesto ovih upisanih vrednosti unosićemo ono što je korisnik uneo tokom checkout-a, a to se čuva u okviru promenljive values, koju smo ranije kreirali.

        const expiry = values.expiry.split('/');

        const card = {

            number: values.number,

            exp\_month: expiry[0],

            exp\_year: expiry[1],

            cvc: values.cvc,

            name: name,

        };

## 18.4 Kreiranje korpe

Za implementiranje korpe prvo ćemo kreirati cart context, koji će u okviru sebe sadržati logiku dodavanja proizvoda u korpu, kao i uklanjanja iz korpe. S obzirom na to da će korisnici naručivati samo iz jednog restorana primenićemo funkciju clear, koja briše sve proizvode iz korpe.

// react imports

import React, { createContext, useState, useEffect, useContext } from "react";

// react native imports

import AsyncStorage from "@react-native-async-storage/async-storage";

// context

import { AuthenticationContext } from "../authentication/autentication.context";

export const CartContext = createContext();

export const CartContextProvider = ({children}) => {

    const { user } = useContext(AuthenticationContext);

    const [cart, setCart] = useState([]);

    const [restaurant, setRestaurant] = useState(null);

    const add = (item, rst) => {

        if(!restaurant || restaurant.placeId !== rst.placeId) {

            setRestaurant(rst);

            setCart([item]);

        }

        setCart([...cart, item]);

    };

    const clear = () => {

        setCart([]);

        setRestaurant(null);

    };

    return (

        <CartContext.Provider value={{

            addToCart: add,

            clearCart: clear,

            restaurant,

            cart,

        }}>

            { children }

        </CartContext.Provider>

    );

}

Nakon povezivanja našeg context-a sa CheckoutScreen-om ovu komponentu bi trebalo implementirati u okviru screen-a RestaurantDetail, koju smo ranije kreirali za prikaz menija restorana.

export const CheckoutScreen = () => {

    const { cart, restaurant } = useContext(CartContext);

    return(

        <SafeArea>

            <Text>{JSON.stringify(cart)}</Text>

            <Text>restaurant:{JSON.stringify(restaurant)}</Text>

            <CreditCardInput />

        </SafeArea>

    );

};

U RestaurantDetail screen unosimo sledeći kod:

            <Spacer position="bottom" size="large">

                <OrderButton

                    mode="contained"

                    icon="cash-multiple"

                    onPress={() => {

                        addToCart({

                            item: `${restaurant.name}'s Special`,

                            price: 799

                        }, restaurant);

                    }}

                >

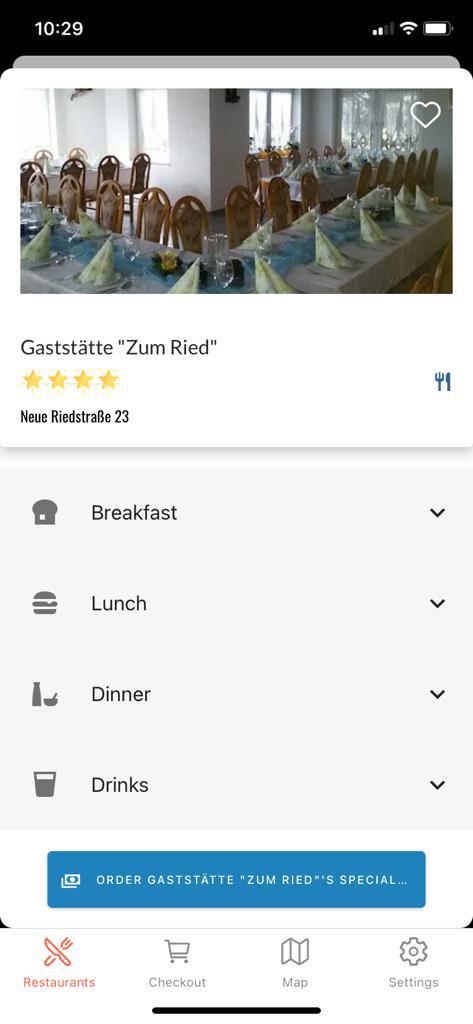
                    <OrderButtonText>

                        Order {restaurant.name}'s Special for $7.99

                    </OrderButtonText>

                </OrderButton>

            </Spacer>



Slika 58: Dugme za dodavanje u korpu

U situaciji kada nismo uneli ništa u našu korpu, ne bi trebalo da nam se prikazuje forma za karticu, već obaveštenje o tome da je korpa prazna, što ćemo uraditi na sledeći način:

    if(!cart.length || !restaurant) {

        return (

            <SafeArea>

                <CartIconContainer>

                    <CartIcon icon="cart-off" />

                    <Text>Your cart is empty!</Text>

                </CartIconContainer>

            </SafeArea>

        );

    }

export const CartIconContainer = styled.View`

    align-items: center;

    justify-content: center;

    flex: 1;

`;

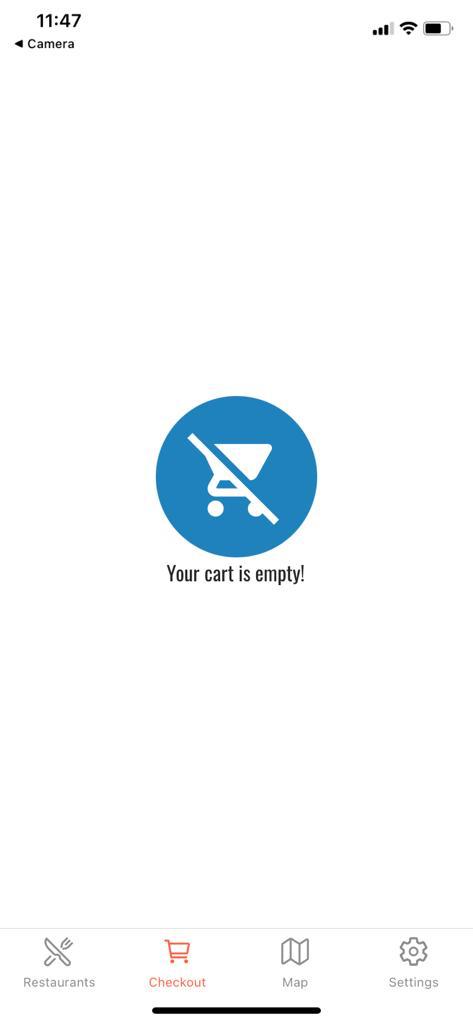
export const CartIcon = styled(Avatar.Icon).attrs({

    size: 128,

})`

    background-color: ${(props) => props.bg || props.theme.colors.brand.primary};

`;



Slika 59: Prazna korpa

Kada unesemo novi proizvod u korpu trebalo bi da nam se prikaže restoran iz kog naručujemo specijalitet, za šta ćemo koristiti RestaurantCardInfo, kao i svaki specijalitet koji smo naručili i ukupna cena.

    return(

        <SafeArea>

            <RestaurantInfoCard restaurant={ restaurant } />

            <ScrollView>

                <Spacer position="left" size="large">

                    <Spacer position="top" size="large">

                        <Text>Your Order</Text>

                    </Spacer>

                    <List.Section>

                        {cart.map(({item,price}) => {

                            return (

                                <List.Item

                                    title={`${item} - $${price / 100}`}

                                />

                            )

                        })}

                    </List.Section>

                    <Text>Total: ${sum / 100} </Text>

                </Spacer>

                <CreditCardInput />

            </ScrollView>

        </SafeArea>

    );

    const [sum, setSum] = useState(0);

    useEffect(() => {

        if(!cart.length) {

            setSum(0);

            return;

        }

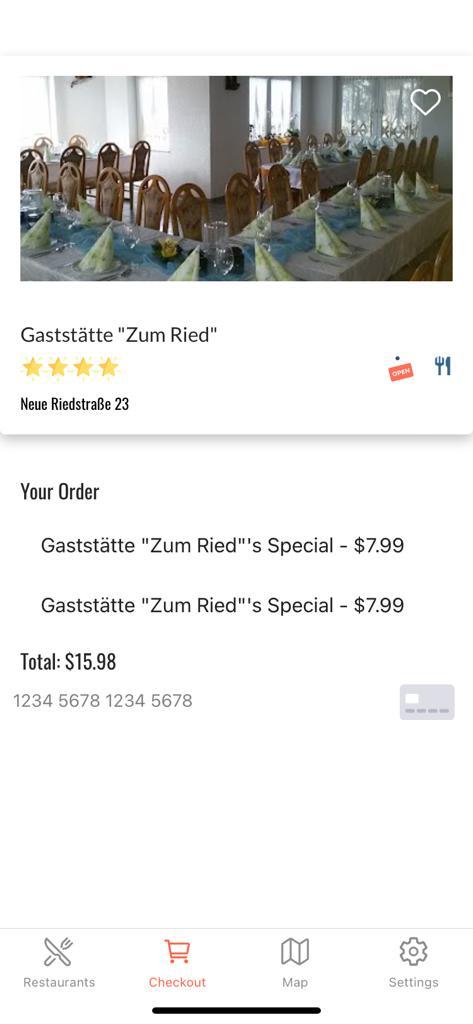
        const newSum = cart.reduce((acc, { price }) => {

            return (acc += price);

        }, 0);

        setSum(newSum);

    }, [cart]);



Slika 60: Checkout sa popunjenom korpom

## 18.5 Plaćanje

Kao što znamo, prilikom unosa kreditne kartice za plaćanje u okviru neke aplikacije, pored podataka koje imamo u okviru LiteCreditCardForm-e, trebalo bi uneti i ime na kartici. U ovom trenutku to ime je postavljeno na default vrednost “user”, međutim, ono što želimo jeste da korisnici unesu svoje ime zajedno sa drugim podacima. Za ovo ćemo kreirati novu komponentu NameInput, kao i varijablu name i u slučaju da nismo uneli ništa u ovo polje, neće nam se prikazati polje za unos kartice.

export const NameInput = styled(TextInput)`

    margin: ${props => props.theme.space[3]};

`;

<NameInput

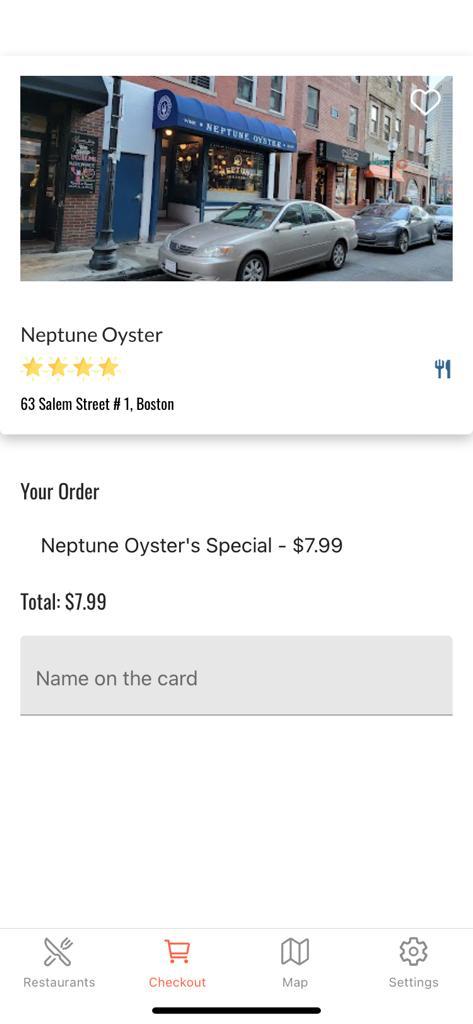
     label="Name on the card"

     value={name}

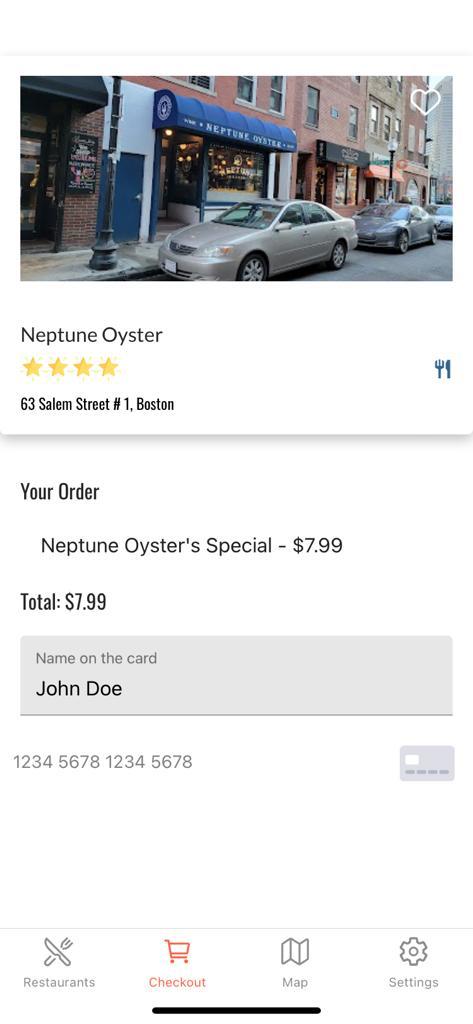
     onChangeText={(t) => setName(t)}

/>

{name.length > 0 && <CreditCardInput />}



Slika 61: Polje za unos imena



Slika 62: Otvaranje opcije za unos podataka za karticu nakon unosa imena

Ispod forme za unos kreditne kartice i imena unećemo dva dugmeta, jedno za plaćanje i jedno za brisanje unosa u korpi.

                <Spacer position="top" size="xlarge">

                    <PayButton

                        icon="cash-multiple"

                        mode="contained"

                        onPress={() => console.log("paying")}

                    >

                        Order

                    </PayButton>

                    <Spacer position="top" size="large" />

                    <ClearButton

                        icon="cart-off"

                        mode="contained"

                        onPress={clearCart}

                    >

                        Clear Cart

                    </ClearButton>

                </Spacer>

export const PayButton = styled(Button).attrs({

    color: colors.brand.primary,

})`

    width: 80%;

    align-self: center;

    padding: ${props => props.theme.space[2]};

`;

export const ClearButton = styled(Button).attrs({

    color: colors.ui.error,

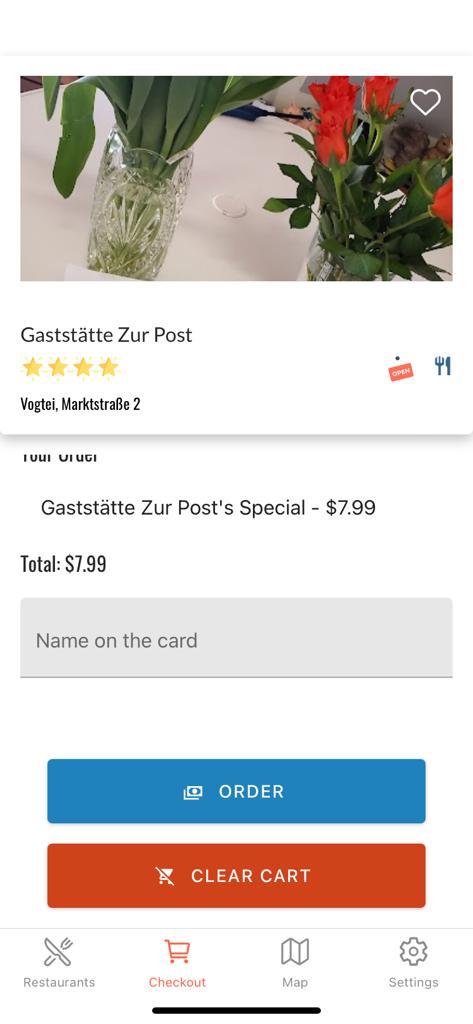
})`

    width: 80%;

    align-self: center;

    padding: ${props => props.theme.space[2]};

`;



Slika 63: Dugmići za naručivanje i brisanje korpe

Ono što je preostalo jeste uvođenje funkcionalnosti u dugme Order. Kako bismo to uradili prvo ćemo implementirati Stripe SDK među naše firebase funkcije. Za početak ćemo u okviru foldera functions uneti komandu yarn add stripe. Takođe, na isti način na koji smo unosili API key za google mape, unećemo i stripe key. Kreiraćemo payment request i proslediti ga od checkout screen-a nadalje do naše funckije pay.

    const [card, setCard] = useState(null);

    const onPay = () => {

        if(!card || !card.id) {

            console.log("Error");

            return;

        }

        payRequest(card.id, sum, name);

    }

        if(!isIncomplete) {

            const info = await cardTokenRequest(card);

            onSuccess(info);

        }

export const payRequest = (token, amount, name) => {

    return fetch(`${liveHost}/pay`, {

        body: JSON.stringify({

            token,

            name,

            amount,

        }),

        method: "POST",

    }).then(res => {

        if(res.status > 200) {

            return Promise.reject("Something went wrong processing your payment");

        }

        return res.json();

    });

}

module.exports.payRequest = (request, response, stripeClient) => {

    const body = JSON.parse(request.body);

    const { token, amount } = body;

    stripeClient.paymentIntents.create({

        amount,

        currency: "USD",

        payment\_method\_types: ["card"],

        payment\_method\_data: {

            type: "card",

            card: {

                token,

            },

        },

        confirm: true,

    }).then((paymentIntent) => {

        response.json(paymentIntent);

    }).catch((err) => {

        response.status(400);

        response.send(err);

    });

};

Kreiraćemo i pojedinačne screen-ove za uspešnu i neuspešnu transakciju, tako da ćemo morati da inicijalizujemo CheckoutNavigator, koji će upravljati ovim slučajevima.

// react imports

import React from "react";

// navigation imports

import { createStackNavigator, CardStyleInterpolators } from "@react-navigation/stack";

// screens

import { CheckoutScreen } from "../../features/checkout/screens/checkout.screen";

import { CheckoutErrorScreen } from "../../features/checkout/screens/checkout-error.screen";

import { CheckoutSuccessScreen } from "../../features/checkout/screens/checkout-success.screen";

const CheckoutStack = createStackNavigator();

export const CheckoutNavigator = () => {

    return (

        <CheckoutStack.Navigator

            screenOptions={{

                cardStyleInterpolator: CardStyleInterpolators.forHorizontalIOS,

                headerShown: false,

            }}

        >

            <CheckoutStack.Screen

                name="CheckoutScreen"

                component={CheckoutScreen}

            />

            <CheckoutStack.Screen

                name="CheckoutSuccess"

                component={CheckoutSuccessScreen}

            />

            <CheckoutStack.Screen

                name="CheckoutError"

                component={CheckoutErrorScreen}

            />

        </CheckoutStack.Navigator>

    )

}

// react imports

import React from "react";

// components

import { Text } from "../../../components/typography/text.component";

import { SafeArea } from "../../../components/utility/safe-area.component";

import { CartIconContainer, CartIcon } from "../components/checkout.style";

export const CheckoutSuccessScreen = () => (

  <SafeArea>

    <CartIconContainer>

      <CartIcon icon="check-bold" />

      <Text variant="label">Success!</Text>

    </CartIconContainer>

  </SafeArea>

);

// react imports

import React from "react";

// styles

import { colors } from "../../../infrastructure/theme/colors";

// components

import { Text } from "../../../components/typography/text.component";

import { SafeArea } from "../../../components/utility/safe-area.component";

import { CartIconContainer, CartIcon } from "../components/checkout.style";

export const CheckoutErrorScreen = ({ route }) => {

  const { error = "" } = route.params;

  return (

    <SafeArea>

      <CartIconContainer>

        <CartIcon icon="close" bg={colors.ui.error} />

        <Text variant="label">{error}</Text>

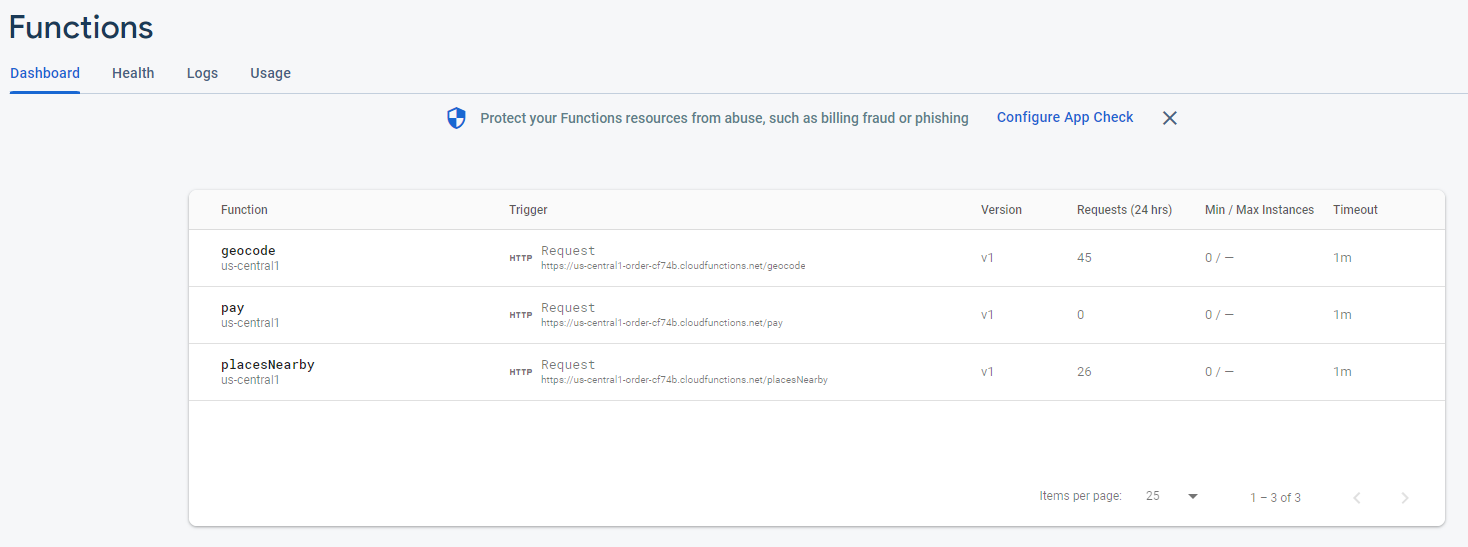
      </CartIconContainer>

    </SafeArea>

  );

};

Izvršićemo deploy funkcije pay na naš firebase projekat kako bismo mogli da je koristimo u okviru aplikacije.



Slika 64: Firebase funkcije

# KORIŠĆENA I REFERENTNA LITERATURA

1. ActivityIndicator – React Native <https://reactnative.dev/docs/activityindicator>
2. Animations – React Native <https://reactnative.dev/docs/animations>
3. AsyncStorage – Expo Documentation <https://docs.expo.dev/versions/latest/sdk/async-storage/>
4. Avatar.Icon – React Native Paper <https://callstack.github.io/react-native-paper/avatar-icon.html>
5. Basics – styled-components <https://styled-components.com/docs/basics>
6. Button – React Native <https://reactnative.dev/docs/button>
7. Button – React Native Paper <https://callstack.github.io/react-native-paper/button.html>
8. Camera – Expo Documentation <https://docs.expo.dev/versions/latest/sdk/camera/>
9. Context – React <https://reactjs.org/docs/context.html>
10. Documentation – Yoga Layout <https://yogalayout.com/docs/>
11. Expo - <https://expo.dev/>
12. Expo Cli – Expo Documentation <https://docs.expo.dev/workflow/expo-cli/>
13. Firebase <https://firebase.google.com/>
14. Firebase CLI reference - https://firebase.google.com/docs/cli
15. Firebase Deploying – TutorialsPoint <https://www.tutorialspoint.com/firebase/firebase_deploying.htm>
16. Get Started | Geocoding API – Google Developers <https://developers.google.com/maps/documentation/geocoding/start#:~:text=Geocoding%20is%20the%20process%20of,into%20a%20human%2Dreadable%20address>.
17. Google Cloud: Cloud Computing Services <https://cloud.google.com/>
18. Google Fonts <https://fonts.google.com/>
19. History: Mystery of Mobile Application Development <https://www.appschopper.com/blog/history-mystery-of-mobile-application-development-revealed-here/>
20. How to Write Better Code: 3 Levels of Code Consistency <https://css-tricks.com/write-better-code-3-levels-code-consistency/>
21. Icons – Expo Documentation <https://docs.expo.dev/guides/icons/>
22. Icons – React Native Paper <https://callstack.github.io/react-native-paper/icons.html>
23. ImageBackground – React Native <https://reactnative.dev/docs/imagebackground>
24. Introduction – React Native <https://reactnative.dev/docs/getting-started>
25. Layout with Flexbox – React Native <https://reactnative.dev/docs/flexbox>
26. List.Accordion – React Native Paper <https://callstack.github.io/react-native-paper/list-accordion.html>
27. Map View - Expo Documentation <https://docs.expo.dev/versions/latest/sdk/map-view/>
28. Node.js <https://nodejs.org/en/>
29. Node.js Client for Google Map Services – Github <https://github.com/googlemaps/google-maps-services-js>
30. Organizing your CSS <https://developer.mozilla.org/en-US/docs/Learn/CSS/Building_blocks/Organizingv>
31. Pressable – React Native <https://reactnative.dev/docs/pressable>
32. React Navigation <https://reactnavigation.org/>
33. React Native Credit Card Input <https://www.npmjs.com/package/react-native-credit-card-input>
34. Stripe | Payment Processing Platform for the Internet <https://stripe.com/>
35. TextInput – React Native Paper <https://callstack.github.io/react-native-paper/text-input.html>
36. Using Firebase – Expo Documentation <https://docs.expo.dev/guides/using-firebase/>
37. Web View – React Native <https://reactnative.dev/docs/0.61/webview>
38. Yarn – Package Manager <https://yarnpkg.com/>