REALIZACIJA SUSTAVA IZNAJMLJIVANJA KORISTEĆI HIBRIDNI RAZVOJNI OKVIR

Martinić, Tin

Undergraduate thesis / Završni rad

2023

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **University of Split / Sveučilište u Splitu**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:228:193315

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

Download date / Datum preuzimanja: **2024-05-13**

Repository / Repozitorij:

Repository of University Department of Professional Studies

SVEUČILIŠTE U SPLITU

SVEUČILIŠNI ODJEL ZA STRUČNE STUDIJE

Stručni prijediplomski studij Informacijska tehnologija

TIN MARTINIĆ

Z A V R Š N I R A D

REALIZACIJA SUSTAVA IZNAJMLJIVANJA KORISTEĆI HIBRIDNI RAZVOJNI OKVIR

Split, rujan 2023.

SVEUČILIŠTE U SPLITU

SVEUČILIŠNI ODJEL ZA STRUČNE STUDIJE

Stručni prijediplomski studij Informacijska tehnologija

Predmet: Uvod u programiranje

Z A V R Š N I R A D

Kandidat: Tin Martinić

Naslov rada: Realizacija sustava iznajmljivanja koristeći hibridni okvir

Mentor: Teo Žuljević, viši predavač

Split, rujan 2023

Sadržaj

SADRŽAJ I

SAŽETAK 1

SUMMARY 1

1. UVOD 2

2.TEHNOLOGIJE 3

2.1 FIGMA 3

2.2 RAZVOJNO OKRUŽENJE 4

2.2.1 Visual studio code 4

2.2.2 Expo 5

2.2.3 Android studio 6

2.2.4 Yarn 7

2.3 REACT I REACT NATIVE 8

2.3.1 React 8

2.3.2 React Native 9

2.4 JAVASCRIPT 10

2.5 REACT ZAKAČKE 11

2.5.1 UseState 12

2.5.2 UseEffect 12

2.5.3 UseContext 13

2.6 NASLJEĐIVANJE I KOMPOZICIJA 15

2.7 REACT NATIVE BIBLIOTEKE 15

3. FIREBASE, FIRESTORE I ASYNC STORAGE 17

3.1 FIREBASE 17

3.1.1 Firebase autorizacija i autentifikacija 18

3.2 FIRESTORE 19
3.3 ASYNC STORAGE 21
4. ALTERNATIVNE TEHNOLOGIJE 22
4.1 FLUTTER 23
4.2 XAMARIN 24
5. APLIKACIJA SETTLEME 25
5.1 FUNKCIONALNOSTI APLIKACIJE 25
5.2 ARHITEKTURA APLIKACIJE 26
5.3 POPIS VANJSKIH PAKETA 27
5.4 RAZVOJ APLIKACIJE 29
5.4.1 Postavljanje temelja 29
5.4.2 Priključak na Firebase 30
5.4.3 Korištenje konteksta 32
5.4.4 Početni zaslon 34
5.4.5 Zaslon informacije proizvoda 36
5.4.6 Zaslon rezervacija 40
5.4.7 Zaslon omiljenih 43
5.4.8 Zaslon postavki 45
5.4.9 Ocjenjivanje proizvoda 48
6. ZAKLJUČAK 52
LITERATURA 1

Sažetak

Završni rad je podijeljen na teorijski i praktični dio. U teoretskom dijelu se opisuju svrha i rad alata i programskih podrški okvira React, React Native, dizajnerskog alata Figma te Firebase i Firestore NoSQL skup usluga poslužitelja za manipulaciju nad podacima. Također, završni rad opisuje razvoj aplikacije u React Native okruženju koja olakšava proces rezervacije mjesta u restoranima i noćnim klubovima. Aplikacija omogućuje korisnicima da pregledaju ponudu različitih restorana i klubova, odaberu željeno mjesto i vrijeme te jednostavno naprave rezervaciju. Aplikacija također sadrži opciju pregleda recenzija drugih korisnika kako bi korisnici mogli odabrati najbolje mjesto za sebe. Aplikacija ima mogućnost dostavljanja potvrde o rezervaciji putem elektroničke pošte. Završni rad opisuje detalje implementacije aplikacije. Kroz ovaj projekt, stvorena je funkcionalna aplikacija koja rješava problem neefikasnog procesa rezervacije mjesta u restoranima i noćnim klubovima, što bi trebalo poboljšati ukupno iskustvo korisnika.
Ključne riječi: React, React Native, Figma, Firebase, Firestore NoSQL

Summary

The implementation of rental system using a hybrid framework

The paper is divided into a theoretical and practical part. The theoretical part describes the purpose and operation of React and React Native software frameworks, as well as the design tool Figma and Firebase and Firestore NoSQL hosting for manipulating over data. Additionally, the paper describes the development of an application in the React Native environment that facilitates the process of reserving spots in restaurants and nightclubs. The application allows users to browse the offerings of various restaurants and clubs, select a desired location and time, and easily make a reservation. The application also includes an option to view reviews from other users so that users can choose the best place for themselves. The paper describes the implementation details of the application. Through this project, a functional application has been created that solves the problem of inefficient seat reservation processes in restaurants and nightclubs, which should improve user experience.
Keywords: React, React Native, Figma, Firebase, Firestore NoSQL

1. Uvod

U novije vrijeme mobilne aplikacije postaju sve popularnije te sve više aplikacija koje su isprva bile pisane za preglednik također nude i svoju mobilnu verziju. S toga je izrazito važno poduzećima da što brže i jeftinije razviju i mobilne aplikacije svojih proizvoda.

Prateći trendove sve zastupljenijeg korištenja mobilnih aplikacija sve se više razvijaju tehnologije koje unaprjeđuju te moderniziraju aplikacije realizirane za mobilne uređaje. Glavna podjela operacijskih sustava dizajnirana za mobilne uređaje se dijeli na Android i IOS. Budući da su isti realizirani na različitim tehnologijama, svaki koristi svoje okvire za razvoj aplikacija na tim operativnim sustavima. Okvire pomoću kojih se razvija aplikacija za određeni operativni sustav naziva se izvorni. Oni su u pravilu idealniji za razvijanje aplikacija na pripadajućoj platformi. Međutim, problem nastaje dupliciranjem linija kôda što rezultira dugotrajnijim postupkom.

Kako bi se pisanje kôda učinilo efikasnijim realiziran je višeplatformski razvoj aplikacija. Isti nam omogućava razvoj na najčešće korištenim platformama kao Android i IOS, ali isto tako i manje poznate tehnologija kao Apple Watch, AndroidTV i slične. Kod odabira višeplatformskog okvira postoji mogućnost korištenja React Nativea, Fluttera, Xamarina itd.

Ovaj rad doprinosi upoznavanju s razvojem mobilne aplikacije koristeći React Native okvir. U teoretskom dijelu pobliže je opisan osnovni rad s React i React Nativeom te izuzetno bitnih dijelova Javascript programskog jezika potrebnih za razvoj React Native aplikacije. Također, opisan je rad s Firestore NoSQL spremnika podataka.

Kod praktičnog dijela bit će opisana bitna rješenja problema koristeći se Javascript programskim jezikom i tkz. *React Hookova*.

Temeljna svrha aplikacije je rješavanje problema neefikasnosti i pojednostavljenje samog procesa rezervacije mjesta u restoranima i noćnim klubovima.

2.Tehnologije

2.1 Figma

Slika 1: Figma logo

Figma, čiji je logo prikazan na slici 1, je alat koji se koristi za dizajniranje korisničkog sučelja kod mobilnih aplikacija i mrežnih stranica. Programska podrška omogućava dizajnerima da stvaraju, uređuju i dijele svoje projekte u stvarnom vremenu. Time se omogućuje uvid u promjene vezane uz projekt svim članovima tima dok se projekt razvija.

Figma nudi veliki broj značajki koje olakšavaju dizajniranje uključujući mogućnost stvaranja prototipa, komentiranja i pregledavanja projekta kao i mogućnost integracije s drugim alatima. Osim toga dostupna je i na mrežnom pregledniku čime nije potrebno preuzimanje programske podrške na računalo.

Figma se često koristi za dizajniranje mrežnih stranica, mobilnih aplikacija i drugih digitalnih proizvoda. Uz to, Figma ima i veliku zajednicu korisnika koja dijeli svoje resurse, kao što su gotove biblioteke elemenata, pa se može uštedjeti vrijeme u procesu dizajniranja.

2.2 Razvojno okruženje

2.2.1 Visual studio code

Slika 2: Visual studio code logo

Visual studio code, čiji je logo prikazan na slici 2, je besplatno integrirano razvojno okruženje koje se koristi za razvoj raznih programskih podrški. Razvijen je od strane Microsofta i dostupan na operativnim sustavima Windows, Linux i MacOS.

Neke od glavnih značajki VSCodea je podrška za razne programske jezike, refaktoriranje kôda, podrška za testiranje kôda, integracija s raznim alatima za upravljanje izvorima kao što je Git, proširenje putem dodataka te mogućnost prilagođavanja radnog okruženja po željama korisnika.

Prepoznat je među programerima zbog efikasnosti i jednostavnosti korištenja, te podrške za rad u timu.

2.2.2 Expo

Slika 3: Expo logo

Prilikom odabira korištenja React Nativea prvi korak bi bio izbor sučelja naredbenog retka odnosno „*command line interfacea*". Izbor pada na dvije mogućnosti- React Native CLI ili Expo.

Započeti razvoj pomoću Expo set alata, čiji je logo prikazan na slici 3, se čini idealnom opcijom, budući da uvelike olakšava razvoj, održavanje i implementaciju programskih podrški. Razvijen je isključivo za React Native, a najvažnije značajke su jednostavnost korištenja i brzina početka razvoja. Korištenjem Expoa nam nije nužan virtualni uređaj već je moguće fizički spojiti mobilni uređaj koji je spojen *USB*-om.

Prije inicijalizacije projekta također je potrebno instalirati NPM (Node package manager) koji omogućuje jednostavnu instalaciju Javascript paketa koje se koristiti u projektu. Najnužniji za rad s Expo je upravo Expo *CLI* koji se instalira na sljedeći način koristeći terminal ili naredbenu liniju.

```
npm install-g Expo-cli
```

Naredba `install` instalira, opcija `-g` označava da se paket namjerava koristiti globalno, a Expo-cli je ime paketa kojeg se preuzima.

Kada instalacija Expo-cli završi, generira se projekt na sljedeći način:

```
Expo init MyProject
```

Naredba Expo instalira sve potrebne pakete a naredba `init` generira predložak projekta koji je odmah po završetku naredbe moguće pokrenuti koristeći fizički uređaj.

2.2.3 Android studio

Slika 4: Android studio logo

Android studio, čiji je logo prikazan na slici 4, je besplatno integrirano razvojno okruženje razvijeno od strane Googlea koje nudi alate za razvoj, testiranje i profiliranje Android aplikacija.

Android emulator je alat u sklopu Android studia koji omogućuje prikaz i rad virtualnog uređaja na računalu. Korisnici stvaraju virtualni uređaj koji imitira hardverske karakteristike fizičkog Android mobilnog uređaja.

Korištenje Android studia zajedno s emulatorom programeru omogućuje jednostavno i brzo razvijanje Android aplikacija a da pritom ne treba posjedovati Android mobilni uređaj.

2.2.4 Yarn

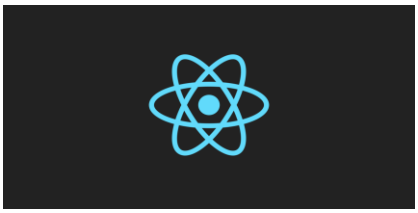
Slika 5: Yarn logo

Yarn, čiji je logo prikazan na slici 5, je alat za upravljanje ovisnostima razvijen od strane *Facebooka* koji se koristi u razvoju Javascript aplikacija. Osmišljen je kao alternativa NPM-u kako bi riješio neke od problema koje NPM sadržava Node.js upravitelj ovisnostima. Yarn koristi datoteku pod nazivom *package. json* za definiranje ovisnosti projekta i njihovih verzija. Yarn za razliku od *NPM*-a nudi značajke:

- *Workspaces* – Mogućnost upravljanja ovisnostima u višestrukim projektima jednom mjestu.
- *Peer Dependencies* – Mogućnost definiranja ovisnosti među više paketa u projektu
- *Offline mode* – Mogućnost rada bez internetske veze.

2.3 React i React Native

2.3.1 React



Slika 6: React logo

React, čiji je logo prikazan na slici 6, je popularna biblioteka Javascript programskog jezika koja služi za izgradnju mrežnih aplikacija. Razvijena je od strane Facebooka kao besplatan i programski okvir otvorenog kôda. Temelji se na konceptu komponenti, čime korisničko sučelje razbija na manje dijelove tzv. „komponente koji se koriste u različitim dijelovima aplikacije. Ovaj pristup olakšava održavanje i razvoj aplikacija budući da se promjene u pojedinoj komponenti odnose na više razina aplikacije kod kojih se određena komponenta koristi.

React koristi JSX sintaksu koja kombinira Javascript programski jezik i HTML za stvaranje korisničkog sučelja. Također React spada među najčešće korištene biblioteke zbog čega ima veliku zajednicu korisnika i dostupnih dodataka i alata koji olakšavaju razvoj i testiranje.

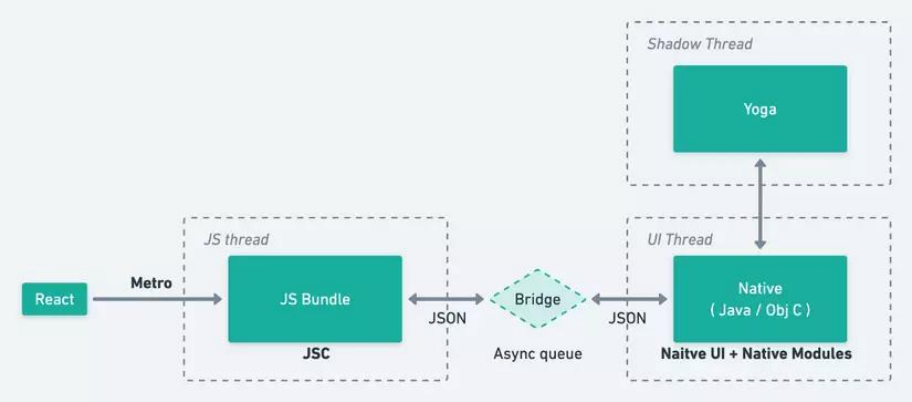
React se često koristi u kombinaciji s drugim tehnologijama poput Jave, ASP.NET-a i Node.js-a.

2.3.2 React Native

React Native je besplatan okvir za razvoj aplikacija na većem broju platformi koji se temelji na popularnoj biblioteci React. Okvir omogućuje razvoj aplikacije na IOS, Android i mrežnoj stranici koristeći jedinstveni kôd. React Native koristi slični princip kao kod React biblioteke tako što sučelje razbija na manje komponente. Razlika je u tome što React Native koristi vlastiti set komponenti umjesto *HTML*-a koji su optimizirani za mobilne uređaje.

Najznačajnija prednost korištenja React Native biblioteke je jedinstvenost kôda, čija je funkcionalnost prikazana na slici 7, kao i ponovno korištenje komponenti čime se brzo razvijaju aplikacije na više platformi. Također kao kod Reacta, React Native ima veliku zajednicu korisnika koji razvijaju dodatke i biblioteke čime se olakšava razvijanje aplikacija u istoj.

React Native također omogućava programerima korištenje izvornih funkcionalnosti poput korištenja kamere, senzora i *GPS*-a unutar svojih aplikacija.



Slika 7: React Native arhitektura

2.4 Javascript

Slika 8: Javascript logo

Javascript, čiji je logo prikazan na slici 8, je dinamički programski jezik najčešće korišten za razvijanje mrežnih stranica, ali se može koristiti i za razvijanje serverskih aplikacije te igara. Kako bi zavladali Reactom nužno je znati osnove Javascript programskog jezika budući da je React biblioteka iste.

Varijable se koriste za pohranjivanje podataka, a tipovi podataka uključuju brojeve, boolean vrijednosti, nizove, objekte i druge. Primjer navedenih je prikazan u ispisu 1.

```
let number = 5; // varijabla number pohranjuje broj 5

let string = „Ime”; // varijabla string pohranjuje Ime

 let isTrue = true; // varijabla isTrue pohranjuje boolean   vrijednost
true

let list = [1, 2]; // varijabla array pohranjuje listu s vrijednostima
1 i 2
```

Ispis 1: Tipovi podataka u Javascript

Vrijednosti liste u prošlom primjeru se u programskom jeziku Javascript mogu dohvatiti na sljedeći način prikazan pod ispisom 2:

```
let [ first, second]= list;
```

Ispis 2: Dohvaćanje vrijednosti liste

Vrijednosti liste koja sadrži podatke tipa objekt također je moguće dohvatiti na isti način opisan u ispisu 3, međutim u tom slučaju redoslijed nije bitan:

```
let list= {first: 1, second: 2};

  let [second, first]= object;
```

Ispis 3: Dohvaćanje vrijednosti liste tipa objekt

Popularnost Javascripta nalazi se u jednostavnosti i intuitivnoj strukturi koja odgovara početnicima. Uz to, budući da je najpopularniji programski jezik na svijetu, na internetu ima jako mnogo materijala za učenje i rješavanje različitih problema, te ima iznimno veliku internetsku zajednicu.

2.5 React zakačke

React zakačke su jedne od najbitnijih stavki React biblioteke. One omogućuju React komponentama da koriste stanje i druge značajke Reacta bez upotrebe klasa. Koriste se kao funkcionalne komponente kao alternativa klasnim komponentama.

2.5.1 UseState

`UseState` zakačka je najčešće korištena zakačka, a ujedno i najjednostavnija. Ona se koristi za definiranje stanja funkcionalnih komponenti Reacta. Stanje je dinamički objekt koji sadrži podatke koji mogu biti promijenjeni tijekom vremena.

```
Import {useState} from 'React';

const[ isUsed, setUsed]= useState('') ;
```

Ispis 4: Korištenje `UseState` zakačke u Javascript

Funkcija `useState` vraća niz s trenutnom vrijednosti stanja i funkcije za ažuriranje stanja.

U ispisu 4 je prikazano da `useState` funkcija prima samo jedan parametar, a taj je početna vrijednost varijable. `UseState` funkcija je izuzetno bitna iz razloga što promjenom stanja se ponovno generira DOM (engl. Document Object Model) . Ukoliko se promjena radi koristeći običnu varijablu promjena neće biti vidljiva sve dok se DOM ponovno ne generira.

2.5.2 UseEffect

`UseEffect` je zakačka koja omogućuje izvođenje nuspojava(engl. *side-effects*) u funkcionalnim komponentama. Nuspojave su bilo koji kôd koji utječe na promjenu izvan komponente.

Zakačka `useEffect` prima dva argumenta: funkciju i listu ovisnosti. Funkcija koja se šalje `useEffect`*u* će se izvršiti kod svakog prikazivanja komponente, osim ako se polje ovisnosti nije promijenilo u odnosu na prethodno prikazivanje.

Ako zakačka `useEffect` ne primi drugi argument, kao što je prikazano u ispisu 5, funkcija se izvršava nakon svakog prikazivanja.

```
Import {useEffect} from 'React';


useEffect (() = > {
 fetch(API). then((data) = > setData(data))
}, [ ])
```

Ispis 5: Korištenje `useEffect` zakačke u Javascriptu

2.5.3 UseContext

`UseContext` je zakačka koja omogućuje pristup kontekstu u funkcionalnim komponentama. Kontekstom se smatra način dijeljenja podataka između komponenata koje nisu u roditelj-dijete relaciji.

U primjeru navedenom u ispisu 6, kada se koristi samo jednu globalnu varijablu koja označava tamnu temu kontekst će izgledati ovako:

```
Import {useContext} from 'React';


const context = {isDark: true};
const themeContext = createContext(context);
```

Ispis 6: Korištenje `useContext`*a* u Javascript

U ovom slučaju kontekst se sprema u varijablu *themeContext*, koja sadrži jednu vrijednost, varijablu *isDark*. Za korištenje konteksta potrebno je „omotati“ sav kôd u kojem se želi koristiti kontekst u element zvan dobavljač (engl. *provider*) .

```
Import {useContext} from 'React';

const context = {isDark: true};

const themeContext = createContext(context) ;

function App() {

 return(

          <ThemeContext. Provider value= {context}>

             <Button />

       </ThemeContext. Provider>

       ) ;

}

function Button() {

 const {isDark} =  useContext(ThemeContext) ;

 const buttonColor: isDark ? 'white': 'black';

 return(

       <button style= {{background: theme. background}}>

                  Button

       </button>
```

Ispis 7: Definiranje i upotreba `useContext` zakačke

U ispisu 7, kontekst se definira u prve dvije linije. U funkciji App je omotana komponenta *Button* u *ThemeContext. Provider* te je time omogućena pristup varijabli *isDark* u funkciji Button pomoću koje se može mijenjati pozadinsku boju gumba.

2.6 Nasljeđivanje i kompozicija

Nasljeđivanje i kompozicija su dvije osnovne značajke u React Native razvojnom okviru koje se koriste za organizaciju i ponovno korištenje kôda. Nasljeđivanje je jedan od najvažnijih dijelova objektno orijentiranog programiranja. Javascript podržava klase, međutim više je orijentirana funkcionalnom programiranju gdje se češće spominje koncept kompozicije. Nasljeđivanje i kompozicija rješavaju isti problem na dva potpuno različita načina.

Nasljeđivanje je proces nasljeđivanja svojstava i funkcionalnosti jedne komponente iz druge komponente. Nasljeđivanje u razvojnom okviru React Native se ostvaruje korištenjem svojstava (engl. *props*) koji se prosljeđuju drugim komponentama.

Kompozicija je proces kreiranja novih komponenti iz postojećih. Komponente se kombiniraju kako bi se stvorila nova funkcionalnost, umjesto ponovne izgradnje. Kompoziciju se rješava obrnutim putem , time se problem rješava na bolji način. Budući da je React biblioteka Javascripta, prirodno preferira metode funkcijskog programiranja a time i prednost kompozicije nad nasljeđivanjem.

Primjer kompozicije koristeći *button* komponentu, prikazan je u ispisu 8*:*

```
    <Button label= 'Youtube' color= 'red' url= 'https: //youtube. com/'
textColor= 'white' />
```

Ispis 8: Kompozicija koristeći *button* komponentu

2.7 React Native biblioteke

Biblioteke olakšavaju rad s odgovarajućim tehnologijama. Budući da su React i React Native otvorenog kôda, postoji veći broj prikladnih biblioteka. S obzirom da je broj biblioteka velik, moguće ih je podijeliti u kategorije:

- Styled components – Popularna biblioteka u React i React Native razvojnim okvirima. Glavna funkcija mu je korištenje standardnog CSS-a za uređivanje komponenti.
- React Native *Paper* – Biblioteka izričito namijenjena za korištenje u razvojnom okviru React Native. Visoko kvalitetna biblioteka čiji je primarni cilj dizajnom što pobliže približiti hibridnu tehnologiju izvornoj.
- Lottie – Biblioteka otvorenog kôda osnovana od strane *AirBNB*a koja pruža animiranu grafičku biblioteku.
- React Native *Maps* – Korisna biblioteka koja koristi *Google* i IOS servise geografskih karti za dodavanje aplikaciji mogućnost korištenja vizualnog geografskog prikaza u izvornom izdanju.
- React Native *Async storage* – Ne kriptirana, asinkrona, ključ-vrijednost biblioteka za pohranu podataka korištena na globalnoj razini unutar aplikacije.
- React Native *Navigation* – Biblioteka navigacije sažeta od osnovnih funkcionalnosti navigacija kako bi odredila navigacijsku strukturu aplikacije.

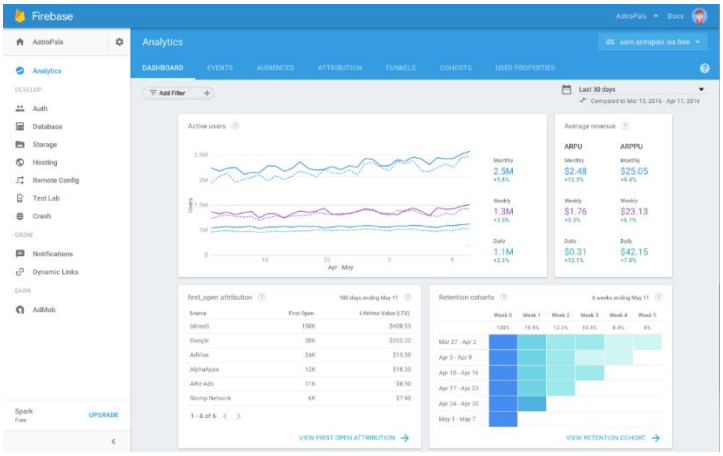
3. Firebase, Firestore i Async storage

3.1 Firebase

Slika 9: Firebase logo

Firebase, čiji je logo prikazan na slici 9, je platforma za razvoj, analitiku i praćenje mrežnih i mobilnih aplikacija. Osnovana je 2011. godine a preuzeta od strane *Googlea* 2014. godine. Zanimljiv je zbog toga što vrlo jednostavnom implementacijom uz mali broj linija kôda nudi usluge autorizacije i autentifikacije, pristupu i obradi podataka nad Firestore bazom podataka koristeći NoSQL jezik, a uz to praćenjem i analizom rada i popularnosti aplikacija i još mnoge druge, kao i moderno sučelje prikazano na slici 10.

Usluge Firebasea se nakon određenih granica naplaćuju, međutim ta granica je sasvim visoka. U razvojnoj fazi programske podrške gotovo je nemoguće doseći tu granicu. Firebase nudi dva sistema plaćanja – „No cost spark“ plan i „Pay as you go blaze“ plan.



Slika 10: Firebase sučelje

Usluge koje nudi Firebase :

- Testiranje
- Analitika
- Distribucija aplikacije
- Indeksiranje aplikacije
- Autentikacija i autorizacija
- Firestore NoSQL baza podataka
- Oblak funkcije
- Oblak razgovori
- Analitika pada aplikacije
- Dinamičke poveznice
- Usluge poslužitelja
- Nadgledanje performansi

3.1.1 Firebase autorizacija i autentifikacija

Autorizaciju i autentifikaciju mobilne ili mrežne aplikacije moguće je osposobiti pomoću Firebase Authentification usluge, čije su funkcionalnosti prikazane na slici 11. Većina modernih aplikacija ima potrebu sadržavati osnovne podatke korisnika kako bi privatizirali i personalizirali određene sadržaje aplikacije. Firebase autentifikacija pruža pozadinske servise i završene biblioteke korisničkog sučelja.

Firebase pruža mogućnost autentifikacije korištenjem platformi:

- Google
- Apple
- Facebook
- Twitter
- Github

Kao i korištenjem elektroničke pošte i mobilnog broja.

Slika 11: Firebase autentifikacija

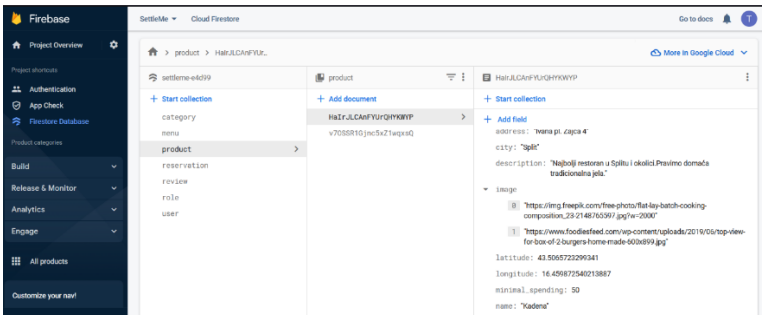
3.2 Firestore

Slika 12: Cloud Firestore logo

Firestore je automatski skalirana baza podataka realizirana u oblaku od strane *Googlea*, čiji je logo prikazan na slici 12. Često se koristi kod mobilnih i mrežnih aplikacija za pohranu i upravljanje podacima zbog toga što omogućava brz i pouzdan pristup podacima. Pristup i obrada Firestore bazi podataka se obavlja pomoću NoSQL jezika. Omogućava jednostavno spremanje i filtriranje podataka pomoću dokumenata koji se grupiraju u kolekcije.

Dokumenti su jednostavni objekti koji mogu sadržavati bilo koje podatke, kao što su nizovi, objekti, brojevi i tekstovi. Kolekcije su skupovi dokumenata gdje svaki dokument ima jedinstveni identifikator.

Firestore pruža napredne mogućnosti upita i filtriranja prema različitim kriterijima, te mogućnost prijenosa podataka između različitih aplikacija i sustava. Firestore također ima integrirane mehanizme za autentifikaciju korisnika, ograničenost pristupa i sigurnost podataka što ga čini sigurnim alatom za upravljanjem podataka. Primjer upita dohvaćanja specifičnog podatka putem identifikacijskog broja prikazan je na ispisu 9.



Slika 13: Firestore konzolno sučelje

Primjer rada s Firestoreom:

```
const db= Firebase. Firestore();


db. collection(collectionName). doc(documentId). get()
            . then((doc) = > {
                  if (doc. exists) {console. log (doc. data()) ;}
                  else {console. log(Document doesn't exists) ;}
                  }) ;
```

Ispis 9: Dohvaćanje vrijednosti iz Firestorea u Javascript

3.3 Async storage

Async storage je jedna od najčešćih React Native biblioteka za pohranu i upravljanje podacima. Biblioteka omogućuje asinkronu pohranu podataka lokalno na uređaj što znači da se podaci mogu spremiti na uređaju bez prekida rada aplikacije. Vrlo je korisna biblioteka ukoliko podatke treba pohraniti u aplikaciji.

Jednostavan je za upotrebu i sličan objektu *local storage* u mrežnim preglednicima. Podaci se pohranjuju kao parovi ključ-vrijednost. U praksi se ključevi i vrijednosti uglavnom pohranjuju kao stringovi, ali je isto tako moguća pohrana kao drugi objekti u Javascript jeziku. Prije rada s Async storageom potrebno je instalirati istoimenu biblioteku koristeći Yarn ili NPM.

Primjer korištenja *AsyncStoragea*, prikazan u ispisu 10:

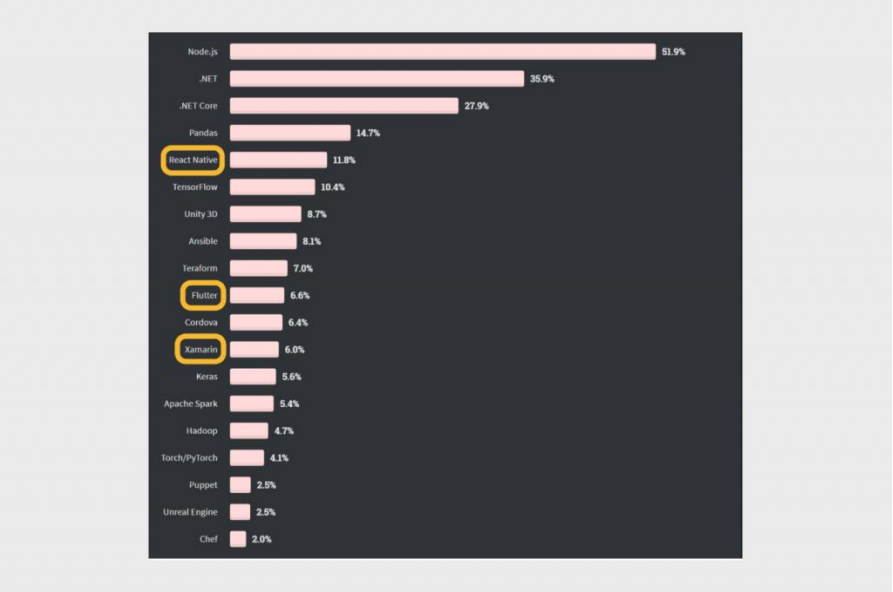
```
import AsyncStorage from @React-Native-async-storage/async-storage;


//dohvaćanje podataka
const getData =  async () = > {
 const token =  await AsyncStorage. getItem('token') ;
 };


//spremanje podataka


const saveData =  async () = > {
 const token =  await AsyncStorage. setItem('token', 'abcd1234') ;
 };
```

Ispis 10: Dohvaćanje i spremanje podataka korištenjem AsyncStoragea u Javascript

4. Alternativne tehnologije

Budući da višeplatformske tehnologije postaju sve popularnije, prikazano na slici 14, i priznatije kao moderne tehnologije, posebno kod startupova , zadnjih godina se nudi veći broj alternativnih rješenja za izradu višeplatformskih aplikacija. Naravno, kao i kod drugih grana razvoja programskih podrški , ne postoji idealno rješenje. Svaka problematika traži tehnologiju koja što jednostavnije i optimiziranije rješava problem. Najveća prednost React Nativea u tom smislu je relativno jednostavan prelazak s mrežne tehnologije React na mobilnu tehnologiju React Native koja sličnim sintaksama otvara prozor druge grane razvoja programske podrške. Slične tehnologije koje su vrijedne spomena su definitivno *Google*ov Flutter i Xamarin, koji dijele respektabilan status u razvoju višeplatformskih aplikacija. Dakako, najvažnija stavka istih su mogućnost rada na IOS i Android platformama ali tu su i druge platforme kao što su AndroidTV, TvOS, mrežna platforma i slično.

Najbolji pokazatelj uspješnosti i vrijednosti tehnologije je zajednica koja istu koristi. React Native trenutno na toj tablici zauzima prvo mjesto koje u tankoj liniji dijeli s Googleovim razvojnim okvirom Flutter, dok Xamarin naglo pada.



Slika 14: Popularnost višeplatformskih tehnologija

4.1 Flutter

Slika 15: Flutter logo

Razvijen je od strane Googlea kao besplatan i razvojni okvir otvorenog kôda za izradu mobilnih, mrežnih i desktop aplikacija pomoću programskog jezika Dart. Temelji se na konceptu „programčića“ koji predstavljaju grafičke elemente poput gumba, teksta, unosnog polja i slično. „Programčići“ se koriste za izradu grafičkog sučelja aplikacije a Flutter , čiji je logo prikazan na slici 15, nudi veliku kolekciju unaprijed definiranih „programčića“ kao i izrada posebnih animacija čime se postiže visoko kvalitetno i interaktivno sučelje.

Prednost Fluttera je brzo izvođenje kôda pomoću JIT (*Just in time*) i AOT (*Ahead of time*) kompilacija za konačnu izvedbu aplikacije. Također, Flutter nudi opciju vrućeg osvježenja (engl. *Hot reload*) čime trenutno prikazuje promjene u kôdu bez potrebe ponovno pokretanja aplikacije.

Flutter se može razvijati preko Visual studio codea , Android studia i IntelliJ Idea razvojnih okruženja.

4.2 Xamarin

Slika 16: Xamarin logo

Xamarin, čiji je logo prikazan na slici 16, je platforma koja koristi programski jezik *C#* za razvoj višeplatformskih aplikacija, uključujući IOS, Android i Windows platforme. Xamarin se sastoji tri ključne komponente – Xamarin. IOS , Xamarin. Android i Xamarin Forms. Xamarin IOS i Android su komponente specifične za razvoj na istoimenim platformama dok Xamarin forms omogućuje programerima razvoj aplikacija na različitim platformama.

Xamarin se također može razvijati pomoću Visual studio code*a*, Jetbrains ridera i Xamarin studia.

5. Aplikacija SettleMe

U ovom poglavlju slijedi opis funkcionalnosti te sama arhitektura aplikacije SettleMe.

5.1 Funkcionalnosti aplikacije

SettleMe je jednostavna i intuitivna aplikacija koja pruža korisnicima sveobuhvatan pregled noćnih klubova i restorana na području, omogućava jednostavno rezerviranje stolova, favoriziranje lokacija te upravljanje korisničkim računom, sve na jednom mjestu, što olakšava planiranje i uživanje u noćnom životu ili gastronomskim iskustvima.

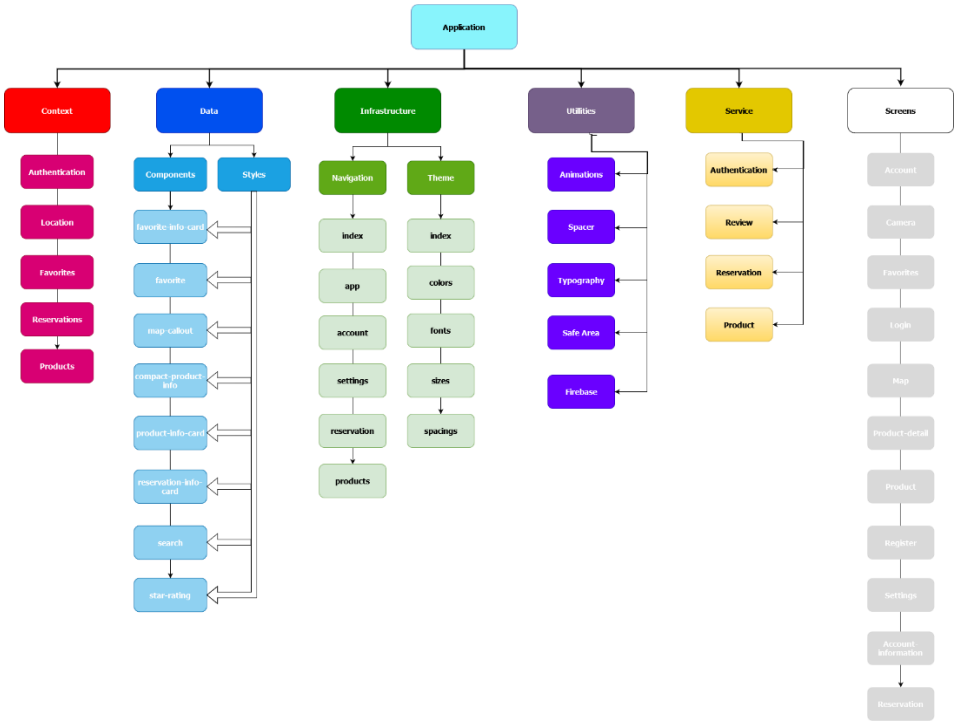
Jedna od ključnih funkcionalnosti SettleMe aplikacije je mogućnost pregleda noćnih klubova i restorana putem interaktivne mape. Korisnici mogu jednostavno pronaći klubove i restorane na karti, pregledati njihove lokacije i ocjene te dobiti informacije o ponudi, cijenama i radnom vremenu.

Aplikacija također omogućava korisnicima da rezerviraju stolove u noćnim klubovima i restoranima u nekoliko jednostavnih koraka. Korisnici mogu odabrati datum i vrijeme te jednostavno rezervirati u željenom restoranu putem aplikacije, čime se izbjegava potreba za telefonskim pozivima ili posjetama osobno.

SettleMe također nudi mogućnost označavanja noćnih klubova i restorana kao favorita. Korisnici mogu dodavati svoje omiljene lokacije na popis favorita, što im omogućava brz pristup i lakši pregled omiljenih mjesta za rezerviranje stolova ili za kasniju posjetu.

Osim toga, aplikacija nudi funkcionalnost uređivanja računa korisnika. Korisnici mogu stvoriti osobni račun unutar aplikacije, gdje mogu spremiti svoje preferencije, praćenje rezervacija, ocjene i recenzije, kao i uređivanje osobnih podataka.

5.2 Arhitektura aplikacije



Slika 17: Dijagram arhitekture aplikacije

Prateći moderne trendove i konvencije imenovanja React Native aplikacija, arhitektura aplikacije SettleMe , prikazana na slici 17, sadrži sljedeće mape:

Mapa `Context` (kontekst): Ova mapa sadrži kontekstualne podatke ili globalno stanje koje se koristi u cijeloj aplikaciji SettleMe. To su podaci koji se dijele između različitih komponenti ili zaslona aplikacije kako bi se omogućila komunikacija između njih i kako bi se očuvali određeni podaci na razini aplikacije. Podaci koji se pohranjuju u mapi *Context* su autorizacijski podaci, lokacijski podaci, favoriti, rezervacije te podaci proizvoda. Korištenje `Contexta` pomaže u jednostavnom i učinkovitom upravljanju globalnim podacima ili stanjem u aplikaciji, te omogućuju jednostavan pristup tim podacima iz različitih dijelova aplikacije, bez potrebe za prosljeđivanjem podataka kroz mnoge komponente ili zaslone. To može smanjiti nepotrebnu složenost i omogućiti bolje upravljanje podacima u aplikaciji SettleMe.

Mapa `Data` (komponente i stilovi): Ova mapa sadrži komponente i stilove koji se koriste za oblikovanje korisničkog sučelja aplikacije SettleMe. To su komponente kao što

su kartice, dugmad, forme, ikone, karte i drugi elementi koji se koriste za grafički dizajn aplikacije. Također, ova mapa sadrži stilove za oblikovanje komponenti kako bi se osigurala dosljednost izgleda i osjećaja aplikacije.

Mapa `Infrastructure` (tema, boje, razmaci, fontovi, veličine i navigacija aplikacije, korisnički račun, postavke, rezervacije i proizvodi): Ova mapa sadrži infrastrukturne elemente koji se koriste za konfiguriranje teme aplikacije SettleMe. To su postavke za boje, razmake, fontove, veličine i navigaciju unutar aplikacije.

Mapa `Utilities` (animacije, tipografija, priključak na Firebase): Ova mapa sadrži različite alate i resurse koji se koriste za dodatne funkcionalnosti aplikacije SettleMe. To su animacije za poboljšanje korisničkog iskustva, tipografija za odabir fontova koji se koriste u aplikaciji, te priključak na Firebase kao backend uslugu za upravljanje podacima i autentifikacijom korisnika.

Mapa `Service` (servisne funkcije): Ova mapa sadrži različite servisne funkcije koje se koriste za obradu poslovne logike aplikacije SettleMe. To su funkcije za autentifikaciju korisnika, funkcije za upravljanje korisničkim računom, funkcije za obradu rezervacija i proizvoda, te druge funkcije koje se koriste za izvršavanje specifičnih zadataka u aplikaciji.

Mapa `Screens` (Izgled zaslona mobilne aplikacije): Ova mapa sadrži sve komponente koje su odgovorne za prikaz korisničkog sučelja aplikacije SettleMe. U ovoj mapi se nalaze različiti zasloni ili ekrani koji se koriste u aplikaciji, kao što su početni zaslon, zaslon za prijavu korisnika, zaslon za registraciju, zaslon za upravljanje korisničkim računom, zaslon za postavke aplikacije, zasloni za rezervacije i proizvode, te druge komponente koje su vidljive korisniku prilikom korištenja aplikacije.

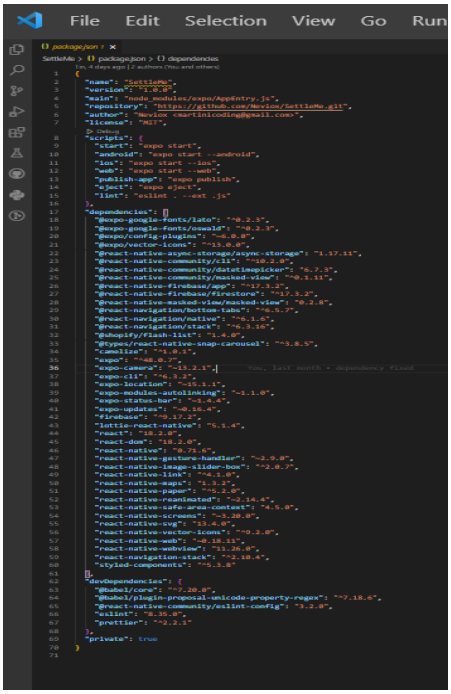
5.3 Popis vanjskih paketa

Vanjski paketi su gotovi skupovi koda ili biblioteke koje se mogu koristiti u aplikacijama kako bi se proširile funkcionalnosti ili ubrzao proces razvoja. Oni se obično instaliraju putem upravitelja paketima, kao što je NPM (Node Package Manager) u slučaju

Javascript projekata. Služe za brži razvoj aplikacija, poboljšanje funkcionalnosti, stabilnost i pouzdanost, rješavanje specifičnih problema itd. Popis vanjskih paketa nalazi se u datoteci `package.json` koja upravlja realizacijom i verzioniranjem istih.

U nastavku su kratki opisi najznačajnijih paketa,prikazani na slici 18, koji su korišteni u aplikaciji SettleMe:

- Expo, Expo-cli, Expo-camera, Expo-location, Expo-status-bar- paketi Expo eko-sustava koji nam služe za pristup i rad kamere, lokacije, prilagođavanja statusne trake te pokretanje u Expo okruženju.
- React-Native async-storage – paket koji omogućava spremanje i upotrebu podataka na lokalnom uređaju.
- React-navigation – paket za određivanje i upravljanje različitim ekranima aplikacije. Sadrži nekoliko podvrsta navigacije- *stack,tab i drawer*.
- React-Native-maps – paket koji omogućava prikaz i rad standardne mape uređaja (Google Maps , Apple Maps)
- React-Native-svg, React-Native-vector-icons- Omogućava korištenje SVG i vektoriziranih ikona u aplikaciji.
- Shopify-Flashlist – Napredna verzija React-Native Flatliste.
- Firebase - paket koji omogućava korištenje Firebase i Firestore funkcionalnosti
- ESLint- paket za statičku analizu, identifikaciju i ispravljanje potencijalnih problema u React Native aplikaciji.



Slika 18: Popis vanjskih paketa

Korištenje vanjskih paketa u React Nativeu može ubrzati razvojni proces, pružiti veću funkcionalnost, stabilnost i fleksibilnost aplikacije te iskoristiti podršku i resurse iz društvene zajednice razvijača.

5.4 Razvoj aplikacije

5.4.1 Postavljanje temelja

Temeljni postulati uređivanja korisničkog sučelja i korisničkog iskustva se definiraju kao savršeni balans konstante i simetrije. Prateći unaprijed definirane stilove najvjerojatnije se razvija oku ugodna aplikacija koja će omogućiti korisniku jednostavno i uglađeno korištenje funkcionalnosti aplikacije. Prateći taj trend, prvi korak razvijanja aplikacije je upravo definiranje vizualnih svojstava kao što su boje, razmaci, fontovi i veličine.

Kada se definiraju željena svojstva, uobičajena praksa je definiranje `index.js` datoteke, prikazane na slici 19, koja nam omogućava pristup svim svojstvima.

```
import { colors } from "./colors";
import { space, lineHeights } from "./spacings";
import { sizes } from "./sizes";
import { fonts, fontWeights, fontSizes } from "./fonts";

export const theme = {
  colors,
  space,
  lineHeights,
  sizes,
  fonts,
  fontSizes,
  fontWeights,
};
```

Slika 19: `Index.js` datoteka

Pristup određenim svojstvima je moguće napraviti na više načina. Nekome tko je već koristio standardni *CSS*, dodavanjem vanjskog paketa *styled-components* može se realizirati uređivanje stilova, prikazano na slici 20.

```
export const AccountContainer = styled.View`
  padding: ${(props) => props.theme.space[4]};
  margin-top: ${(props) => props.theme.space[2]};
  allign-items: center;
  width: 90%;
`;
```

Slika 20: Korištenje CSS-a pomoću paketa *styled-components*

5.4.2 Priključak na Firebase

Kao što je već navedeno, Firebase se koristi za lakše upravljanje korisnika, njihovih prava, manipuliranje podacima pomoću Firestore NOSQL baze podataka te analitike rada aplikacije. Korištenje Firebasea pomaže razumijevanju korištenja aplikacije te kako se može poboljšati. Ukratko, korištenje priključka na Firebase štedi vrijeme i resurse u razvoju i upravljanju aplikacijom, dok istovremeno omogućuje korištenje širokog spektra funkcionalnosti pomažu realizaciji kvalitetne i uspješne aplikacije.

Kako bi se moglo koristiti navedene prednosti potrebno je definirati jedinstvene vrijednosti unutar aplikacije, prikazane ispisom 11, koje određuju spajanje na točno specificiranu Firebase konzolu.

```
// Import the functions you need from the SDKs you need
import { initializeApp } from "Firebase/app";
import { getFirestore } from "Firebase/Firestore";
import { getAuth } from "Firebase/auth";

// TODO: Add SDKs for Firebase products that you want to use
// https://Firebase.Google.com/docs/web/setup#available-libraries

// Your web app's Firebase configuration
const FirebaseConfig = {
  apiKey: "AIzaSyAqBlxP35uEtc6Vb-GxPIqDe4HMo0MAAuc",
  authDomain: "settleme-e4d99.Firebaseapp.com",
  projectId: "settleme-e4d99",
  storageBucket: "settleme-e4d99.appspot.com",
  messagingSenderId: "219423936459",
  appId: "1:219423936459:web:3e1147bf7bc55f5268f04e",
};

// Initialize Firebase
const app = initializeApp(FirebaseConfig);
Export const db = getFirestore(app);
Export const auth = getAuth(app);
Export default app;p.com",
  projectId: "settleme-e4d99",
  storageBucket: "settleme-e4d99.appspot.com",
  messagingSenderId: "219423936459",
  appId: "1:219423936459:web:3e1147bf7bc55f5268f04e",
};
```

Ispis 11: Definiranje priključka na Firebase usluge

Za svaki pokušaj pristupa i obradi podataka koji su definirani u Firebase konzoli potrebno je koristiti definirane metode iz Firebase knjižnice. Međutim, moguće je pridodati objekt lokalnoj varijabli koju se kasnije dijeli i poziva po potrebi. Objekt baze podataka je spremljen u lokalnu varijablu `db`, vrijednost objekta autorizacije u `auth` te aplikacije u `app`.

5.4.3 Korištenje konteksta

U React Nativeu, `context` se koristi za dijeljenje podataka između komponenti koje se nalaze na različitim razinama hijerarhije komponenti.

`Context` je objekt koji sadrži vrijednosti koje se mogu prosljeđivati od roditeljske do dječje komponente. Uobičajeno je da se kontekst koristi kada se prosljeđuju podaci koji su potrebni u više komponenti, a nije poželjno trošiti performanse na prijenos svojstava u svakoj od njih.

U aplikaciji SettleMe, podaci se dohvaćaju iz Firestore spremnika podataka, te se nad istima poziva `createContext` kako bi ih spremili u kontekst, prikazano ispisom 12, te im se lakše pristupilo jednom kada su potrebni koristeći `useContext` zakačku, prikazano ispisom 13.

```
export const ProductsContext = createContext();

export const ProductsContextProvider = ({ children }) => {
  const [products, setProducts] = useState([]);
  const [isLoading, setIsLoading] = useState(false);
  const [error, setError] = useState(null);

  useEffect(() => {
    setIsLoading(true);
    getAllProducts()
      .then((result) => {
        setError(null);
        setIsLoading(false);
        setProducts(result);
      })
      .catch((err) => {
        setIsLoading(false);
        setError(err);
      });
  }, []);

  return (
    <ProductsContext.Provider
      value={{
        products,
        isLoading,
        error,
      }}
    >
      {children}
    </ProductsContext.Provider>
  );
};
```

Ispis 12: Korištenje `createContext` zakačke u aplikaciji SettleMe

```
export const ProductsScreen = ({ navigation }) => {
  const { isLoading, products, error } = useContext(ProductsContext);
  const { error: locationError, locationProducts } =
    useContext(LocationContext);
  const hasError = !!error || !!locationError;

  return (
    <SafeArea>
      {isLoading && (
        <LoadingContainer>
          <Loading size={50} animating={true} color={MD2Colors.yellow500}
/>
        </LoadingContainer>
      )}
      <Search />
      {hasError && (
        <Spacer position="left" size="large">
          <Text variant="error">Something went wrong retrieving the
data</Text>
        </Spacer>
```

```
      )}
      {!hasError && (
        <ProductList
          data={!locationProducts ? products : locationProducts}
          keyExtractor={(item, index) => (item.id ? item.id :
index.toString())}
          renderItem={({ item, index }) => (
            <TouchableOpacity
              key={item.id || index.toString()}
              onPress={() =>
                navigation.navigate("ProductDetail", {
                  product: item,
                  key: item.id,
                })
              }
            >
              <Spacer position="bottom" size="large">
                <FadeInView>
                  <ProductInfoCard product={item} />
                </FadeInView>
              </Spacer>
            </TouchableOpacity>
          )}
        />
      )}
    </SafeArea>
  );
};
```

Ispis 13: Korištenje `useContext` zakačke u aplikaciji SettleMe

5.4.4 Početni zaslon

Početni zaslon je datoteka koja sadrži React komponentu za ekran proizvoda u aplikaciji, prikazan na slici 21. Komponenta koristi mnoge React funkcionalnosti, uključujući React, `useContext` i druge komponente iz React Native i drugih biblioteka.

Komponenta koristi `product` i *`location-product`* kontekst za pristup podacima o proizvodima i proizvodima na lokacijama koji su dostupni putem *contexta*. Kroz `useContext` zakačku, komponenta dobiva pristup *`isLoading`*, *`products`*, i `error` vrijednostima iz `ProductsContext`, kao i `locationError` i `locationProducts` vrijednostima iz `LocationContext`. Ove vrijednosti se koriste kako bi se prikazala

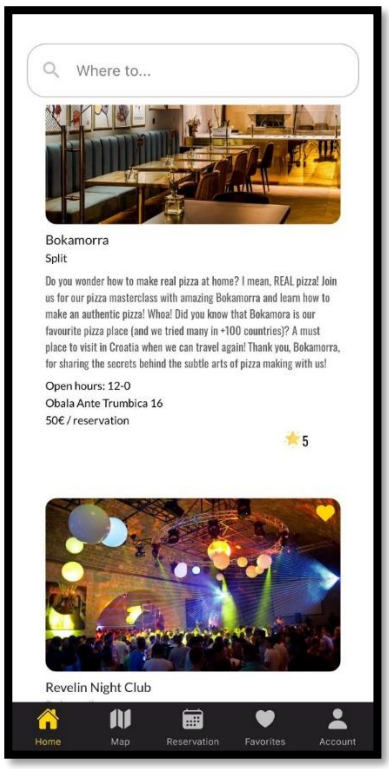
animacija učitavanja ako je `isLoading` *true,* ili prikazala greška ako postoji `error` ili `locationError`.

Komponenta također koristi *Search* komponentu za pretragu proizvoda, koja je također uvezena iz druge datoteke. Također se koriste `SafeArea, Spacer` i `ProductInfoCard` komponente za pravilno prikazivanje i stiliziranje elemenata na ekranu.

`ProductList` je *FlatList* komponenta koja prikazuje listu proizvoda na ekranu. Koristi se *data prop* za prikazivanje proizvoda, *keyExtractor prop* za generiranje jedinstvenog ključa za svaki proizvod, i *renderItem* svojstvo za definiranje izgleda i ponašanja svakog elementa u listi.

Klikom na proizvod u listi, korisnik se preusmjerava na ekran za prikaz detalja proizvoda putem *navigation*. *navigate* funkcije, gdje se prosljeđuje odabrani proizvod kao parametar.

Ukoliko ne postoji greška, proizvodi se prikazuju u listi koristeći `ProductInfoCard` komponentu, koja se također animira koristeći `FadeInView` komponentu za ulaznu animaciju.



Slika 21: Početni zaslon aplikacije SettleMe

Korištenjem trake za pretragu, ukoliko je unesena postojana vrijednost radi se upit na bazu koja generira sve proizvode koji pod vrijednosti atributa *city* sadrže unesenu vrijednost po kojoj se pretražuje. Ukoliko vrijednost ne odgovara nijednom upitu u bazi, funkcija generira sve raspoložive proizvode.

5.4.5 Zaslon informacije proizvoda

Ovaj zaslon definira komponentu `ProductDetailScreen` koja se koristi za prikazivanje detalja o određenom proizvodu. Kôd je prikazan ispisom 14. Komponenta prima objekt ruta (engl. *route*) kao svoj parametar, koji sadrži podatke o trenutnom rutiranju.

U komponenti se koriste različite zakačke za upravljanje stanjem, uključujući `useState` za praćenje rezervacijskog datuma, moda prikaza (*date* ili *time*) , prikazivanja

`DateTimePicker` vanjskog paketa i ocjene proizvoda. Također se koristi `ScrollView` komponenta za omogućavanje klizanja sadržaja ako je potrebno.

Komponenta prikazuje listu slika proizvoda koristeći `SliderBox` komponentu, a zatim prikazuje detalje proizvoda kao što su naziv, ocjena, kategorija, radno vrijeme, adresa i opis koristeći različite tekstualne komponente. Također se koristi `SectionSeparator` komponenta za dodavanje razdjelnika između odjeljaka.

Korisnik može odabrati datum i vrijeme rezervacije pomoću `DateTimePicker` komponente koja se prikazuje kad se pritisne dugme *Select Date* ili *Select Time*. Odabrani datum i vrijeme se prikazuju ispod gumba kao tekstualni podaci.

Nakon odabira datuma i vremena, korisnik može izvršiti rezervaciju pritiskom na dugme *Reserve* koji pokreće funkciju `handleSubmit`. Ako rezervacija bude uspješna, prikazuje se obavijest „Successfully reserved!“. Također, ukoliko su uneseni podaci vjerodostojni poziva se funkcija `setReservationByUser` koja pohranjuje vrijednosti – trenutno vrijeme, odabrano vrijeme, korisnik, proizvod i status rezervacije.

Također, komponenta omogućava korisniku da ocijeni proizvod koristeći `StarRating` komponentu koja se prikazuje ako vrijednost rating varijable nije istinita. Ako korisnik ocijeni proizvod, prikazuje se odgovarajući broj zvjezdica na temelju ocjene proizvoda, koristeći `SvgXml` komponentu te mijenja naslovnu poruku iz „Tap to rate“ u „Thanks for the rate“.

```
export const ProductDetailScreen = ({ route }) => {
  const { product } = route.params;

  const [reservationDate, setDate] = useState(new Date());
  const [mode, setMode] = useState("date");
  const [show, setShow] = useState(false);
  const [rating, setRating] = useState(false);

  const ratingArray = Array.from(new Array(Math.floor(product.rate)));

  const onChange = (event, selectedDate) => {
    const currentDate = selectedDate || reservationDate;
    setShow(Platform.OS === "ios");
    setDate(currentDate);
  };

  const showMode = (currentMode) => {
    setShow(true);
    setMode(currentMode);
  };

  const showDatepicker = () => {
    showMode("date");
  };

  const showTimepicker = () => {
    showMode("time");
  };

  const handleSubmit = () => {
    try {
      setReservationByUser(reservationDate, product);
      Alert.alert("Successfully reserved!");
    } catch (e) {
      Alert.alert(e);
    }
  };

  function handleRatingChange() {
    setRating(rating);
  }

  return (
    <View>
      <ScrollView>
        <SliderBox
          images={product.image}
          dotColor="gold"
          inactiveDotColor="black"
          imageLoadingColor="gold"
        />
        <Spacer size="large" />
        <ProductDetailView>
          <ProductHeader>
            <Title>{product.name}</Title>
            <RatingSection>
              <SvgXml xml={star} width={30} height={30} />
              <Rating>{product.rate}</Rating>
            </RatingSection>
          </ProductHeader>
```

```
        <Spacer size="medium" />
        <Category>{product.category}</Category>
        <Spacer size="small" />
        <WorkingHoursSection>
          <Category>
            {product.opening_at}-{product.closing_at}
          </Category>
        </WorkingHoursSection>
        <Spacer size="small" />
        <Category>{product.address}</Category>
        <Spacer size="small" />
        <Description>{product.description}</Description>
        <Spacer size="large" />
        <Spacer size="large" />
      </ProductDetailView>
      <SectionSeparator />
      <Spacer size="large" />
      <Spacer size="large" />
      <ReservationSection>
        <DateTimeSection>
          <DateTimeButton
            onPress={showDatepicker}
            mode="contained"
            textColor="black"
            buttonColor="gold"
          >
            Select Date
          </DateTimeButton>
          <Spacer size="medium" />
          <DateTimeButton
            onPress={showTimepicker}
            mode="contained"
            textColor="black"
            buttonColor="gold"
          >
            Select Time
          </DateTimeButton>
        </DateTimeSection>
        {show && (
          <DateTimePicker
            testID="dateTimePicker"
            value={reservationDate}
            mode={mode}
            is24Hour={true}
            display="default"
            onChange={onChange}
          />
        )}
        <Spacer size="medium" />
        <DateTimeText>{reservationDate.toLocaleString()}</DateTimeText>
        <Spacer size="large" />
        <Spacer size="large" />
        <SubmitButton
          onPress={handleSubmit}
          mode="contained"
          textColor="black"
          buttonColor="gold"
        >
          Reserve
        </SubmitButton>
```

```
        </ReservationSection>
        <Spacer size="large" />
        <Spacer size="large" />
        <SectionSeparator />
        <Spacer size="large" />
        <Spacer size="large" />
        <ReviewSection>
          {rating ? (
            <StarsSection>
              {ratingArray.map((_, i) => (
                <SvgXml
                  key={`star-${product.id}-${i}`}
                  xml={star}
                  width={30}
                  height={30}
                />
              ))}
            </StarsSection>
          ) : (
            <StarRating
              product_name={product.name}
              onRatingChange={handleRatingChange}
            />
          )}
          <Spacer size="large" />
          <Spacer size="large" />
          <SectionSeparator />
        </ReviewSection>
      </ScrollView>
    </View>
  );
};
```

Ispis 14: `ProductDetail`.*Screen* kôd

5.4.6 Zaslon rezervacija

Zaslon rezervacija predstavlja React komponentu nazvanu `ReservationScreen` koja se koristi za prikazivanje ekrana s rezervacijama, čiji je kôd prikazan ispisom 15. Komponenta koristi `useContext`, `useState` i `useEffect` zakačke iz React biblioteke za upravljanje stanjem i efektima.

Komponenta također koristi sljedeće komponente i stilove iz drugih modula:

- *FlatList*: Komponenta za prikazivanje liste podataka u listi.
- *View*: Osnovna komponenta za definiranje struktura korisničkog sučelja.

- *ActivityIndicator*: Komponenta za prikazivanje indikatora u obliku aktivnosti kao što je učitavanje podataka.
- *styled-components*: Biblioteka za stiliziranje komponenti u React Native aplikacijama.

Komponenta `ReservationScreen` koristi `ReservationContext` za pristup stanju rezervacija. Stanje uključuje `isLoading` (indikator učitavaju li se još podaci), `reservations` (lista rezervacija) i `error` (indikator je li došlo do greške pri učitavanju podataka).

Ako postoji greška pri učitavanju podataka, prikazuje se poruka o grešci. Ako se podaci još uvijek učitavaju, prikazuje se indikator aktivnosti. Ako nema rezervacija, prikazuje se poruka "No Reservations yet*"*. Ako postoje rezervacije, prikazuje se naslov "Reservations" i lista rezervacija koristeći *FlatList* komponentu. Svaka rezervacija se prikazuje kao `ReservationInfoCard` komponenta koja prikazuje informacije o rezervaciji, uz primjenu animacije *FadeInView* (izbljeđivanje pri prikazu) na svaku rezervaciju.

```
export const ReservationScreen = () => {
  const      {      isLoading,      reservations,      error      }      =
useContext(ReservationContext);
  const [isEmpty, setIsEmpty] = useState(false);

  useEffect(() => {
    setIsEmpty(reservations.length < 1);
  }, [reservations]);

  return (
    <>
      {error ? (
        <SafeArea>
          <ErrorContainer>
            <ErrorText>
              An error occurred while fetching reservations.
            </ErrorText>
          </ErrorContainer>
        </SafeArea>
      ) : (
        <SafeArea>
          {isLoading && (
            <LoadingContainer>
              <Loading              size={50}              animating={true}
color={MD2Colors.yellow500} />
            </LoadingContainer>
          )}
          {isEmpty ? (
            <EmptyContainer>
              <Empty>No Reservations yet</Empty>
            </EmptyContainer>
          ) : (
            <>
              <TitleContainer>
                <TitleText>Reservations</TitleText>
              </TitleContainer>
              <FlatList
                data={reservations}
                keyExtractor={(item) => item.id}
                renderItem={({ item, index }) => (
                  <FadeInView>
                    <View style={{ flex: 1 }}>
                      <ReservationInfoCard
                        reservation={item}
                        key={`${item.name}_${index}`}
                      />
                    </View>
                  </FadeInView>
                )}
              />
            </>
          )}
        </SafeArea>
      )}
    </>
  );
};
```

Ispis 15: `ReservationScreen` kôd

5.4.7 Zaslon omiljenih

Zaslon omiljenih je dio React Native aplikacije i sastoji se od nekoliko komponenti koje se koriste za prikazivanje zaslona omiljenih proizvoda, prikazano na slici 22.

`FavoritesList` je komponenta koja je stvorena pomoću *styled-components* biblioteke i nasljeđuje *FlatList* komponentu iz React Nativea. `contentContainerStyle` svojstvu je dodijeljen stil s definiranim paddingom od 16 piksela.

`Loading` je komponenta koja je također stvorena pomoću *styled-components* biblioteke i nasljeđuje *ActivityIndicator* komponentu. Dodijeljen je stil koji pomiče indikator za učitavanje 25 piksela ulijevo.

`LoadingContainer` je komponenta s definiranim stilom za pozicioniranje. Postavlja se na sredinu ekrana (vertikalno i horizontalno) koristeći *position: absolute, top: 50%* i *left: 50%.*

`FavoritesScreen` je funkcionalna komponenta koja prikazuje listu favorita proizvoda. Koristi zakačke `useContext` za pristup stanju favorita (`isLoading` i `favorites`) iz `FavoritesContext`. Ako je `isLoading` istinit, prikazuje se `Loading` komponenta, inače se prikazuje naslov `Favorites` i lista favorita koristeći `FavoritesList` komponentu. Svaki objekt omiljenih se renderira kao *TouchableOpacity* koji omogućuje navigaciju na detaljni ekran proizvoda. Također, koristi se *keyExtractor* svojstvo za izvlačenje ključa iz svakog objekta.

Također se koriste i druge komponente poput *SafeArea* (koja pruža zaštitu od izlaska iz granica ekrana na uređajima s izrezom) , `TitleContainer` i `TitleText` komponente (koje služe za prikazivanje naslova `Favorites`) , `Spacer` komponente (koja dodaje prazan prostor između favorita) , te `FavoriteInfoCard` komponente (koja prikazuje informacije o omiljenom proizvodu), čiji je kôd prikazan ispisom 16 .

Slika 22: Zaslon omiljenih

```
export const FavoritesScreen = ({ navigation }) => {
  const { isLoading, favorites } = useContext(FavoritesContext);
  return (
    <SafeArea>
      {isLoading && (
        <LoadingContainer>
          <Loading size={50} animating={true} color={MD2Colors.yellow500}
/>
        </LoadingContainer>
      )}
      <TitleContainer>
        <TitleText>Favorites</TitleText>
      </TitleContainer>
      <FavoritesList
        data={favorites}
        renderItem={({ item }) => {
          return (
            <TouchableOpacity
              onPress={() =>
                navigation.navigate("ProductDetail", {
                  product: item,
                })
              }
            >
              <Spacer position="bottom" size="large">
                <FavoriteInfoCard favorite={item} />
              </Spacer>
            </TouchableOpacity>
          );
        }}
        keyExtractor={(item) => item.name}
      />
    </SafeArea>
  );
};
```

Ispis 16: Kôd zaslona omiljenih

5.4.8 Zaslon postavki

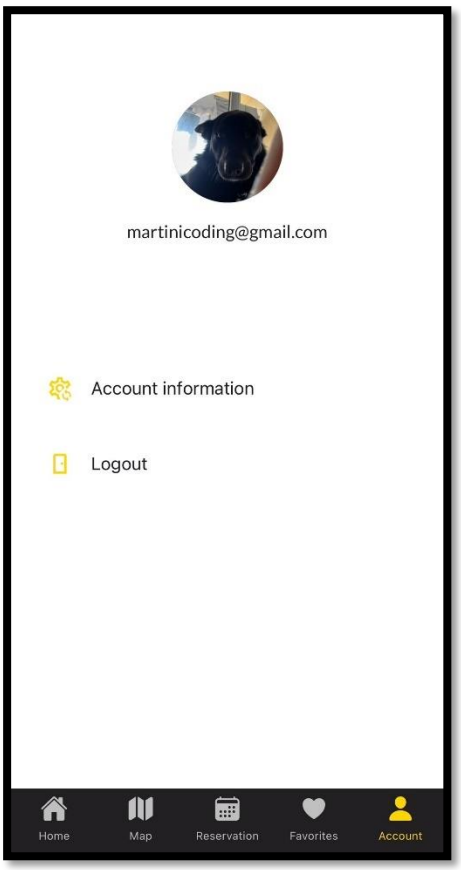
`SettingsScreen` komponenta u React Native aplikaciji prikazuje postavke korisničkog računa, uključujući prikaz korisničke slike, e-mail adrese, te opcije za promjenu korisničkih podataka ili odjavu, prikazane na slici 23.

Komponenta koristi `useContext` zakačku za pristup stanju autentikacije (*onLogout* i *user*) iz `AuthenticationContext`. Također koristi `useState` zakačku za praćenje trenutne slike korisničkog profila (*photo*) .

Komponenta također koristi `useFocusEffect` zakačku koja se izvršava kad se fokus prebacuje na `SettingsScreen` komponentu. Ova zakačka koristi `useCallback` kako bi se osiguralo da se funkcija `getProfilePicture` izvodi samo kad se promijeni *user* objekt. `getProfilePicture` funkcija asinkrono dohvaća URI slike korisničkog profila iz AsyncStorage pomoću *currentUser. uid* kao ključa, te ažurira stanje *photo* s dohvaćenim URI-jem slike.

Kod prikaza, koristi se *SafeArea* komponenta koja osigurava gornji prostor za prikazivanje sadržaja na ekranu zavisno od vrste operativnog sustava. Također se koriste `BlankSpace` komponenta koja dodaje prazan prostor između elemenata, *AvatarContainer* komponenta koja prikazuje korisničku sliku i e-mail adresu, te `List.Section` komponenta koja grupira opcije postavki u sekcije.

Unutar `List.Section` komponente, koriste se *TouchableOpacity* komponente kako bi se omogućila reakcija na dodir korisnika. `SettingsItem` komponenta se koristi za prikazivanje pojedine opcije postavki. Klikom na *Account information* opciju, korisnik se preusmjerava na `AccountInformation` zaslon koji omogućuje korisniku mijenjanje vrijednosti njegovih podataka koji su automatski ispunjeni s trenutnim vrijednostima istih, profilne slike i lozinke.



Slika 21: Zaslon postavki

Ukoliko se vrijednost informacija ne promijeni ništa se ne događa, međutim, promjenom bilo koje vrijednosti osobnih podataka korisnika te pritiskom na Change Account Info dugme, radi se upit na bazu, u kojoj se ažurira korisnikov dokument s unesenim vrijednostima.

Također, postoji opcija promjene trenutne lozinke, gdje se također radi upit, međutim ovaj put na Firebase autorizaciju, u kojoj se mijenja vrijednost lozinke trenutnog korisnika.

5.4.9 Ocjenjivanje proizvoda

Komponenta se izvozi kao funkcija nazvana *StarRating* i prima dva svojstva: `product_name` (naziv proizvoda) i `onRatingChange` (funkcija koja se poziva kada se promijeni ocjena) .

Unutar komponente koriste se zakačke `useState` i `useEffect` za praćenje i ažuriranje stanja ocjene proizvoda (`starRating`) te animacije skaliranja gumba (*animatedButtonScale*).

Funkcija `handleRating(number)` se poziva kada korisnik ocijeni proizvod i ažurira stanje ocjene proizvoda (`starRating`) na temelju broja (1-5) koji je korisnik odabrao. Također se pozivaju dvije asinkrone funkcije `setStars` i `setProductAverageRating` koje postavljaju ocjenu za proizvod u pozadini aplikacije i ažuriraju prosječnu ocjenu proizvoda, korištenjem naredbe kod upita na bazu te se poziva funkcija `onRatingChange()` koja šalje novu ocjenu roditeljskoj komponenti.

Funkcije `handlePressIn()` i `handlePressOut()` koriste se za animiranje gumba prilikom pritiska na njega, koristeći *Animated API* iz React Native biblioteke. Dugme se skalira na veću veličinu kada je pritisnut te se vraća na normalnu veličinu (1) kada korisnik pusti pritisak.

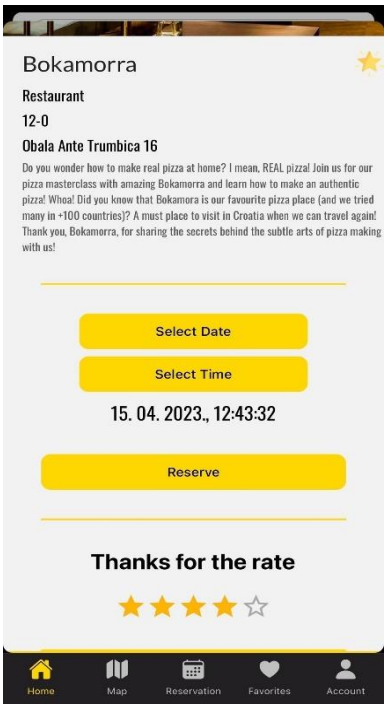
Koristi se poseban stil koji se primjenjuje na *View* oko svake zvjezdice, koji koristi transformaciju skaliranja na temelju *animatedButtonScale* vrijednosti, što omogućuje animiranje skaliranja gumba.

Komponenta koristi *View* za prikazivanje ocjenjivanja proizvoda, *Text* za prikazivanje naslova (koji se mijenja ovisno o ocjeni) , te TouchableWithoutFeedback za svaku zvjezdicu kako bi se omogućio interakcija s korisnikom prilikom ocjenjivanja. Korištena je MaterialIcons ikona za prikaz zvjezdica, gdje se stil zvjezdica mijenja na temelju ocjene proizvoda.

Prilikom svake nove ocjene, vrijednost unesene ocjene se pridodaje ili mijenja, ukoliko je isti korisnik promijenio vrijednost svoje ocjene te se automatski radi upit na bazu

gdje se generira nova prosječna vrijednost svih ocjena proizvoda te posljedično mijenja prosječna ocjena spomenutog proizvoda.

Slikom 24 prikazan je zaslon komponente koja rješava funcionalnost ocjenjivanja, a kôd prikazan ispisom 17.



Slika 24: Ocjenjivanje komponenta-zaslon

```
    export default function StarRating({ product_name, onRatingChange })
{
      const [starRating, setStarRating] = useState(null);

      const animatedButtonScale = new Animated.Value(1);

      async function handleRating(number) {
        setStarRating(number);
        try {
          await setStars(number, product_name);
          await setProductAverageRating(product_name);
          onRatingChange(number);
        } catch (error) {
          Alert.alert(error.message);
        }
      }

      const handlePressIn = () => {
        Animated.spring(animatedButtonScale, {
```

```
          toValue: 1.5,
          useNativeDriver: true,
          speed: 50,
          bounciness: 4,
        }).start();
      };

      const handlePressOut = () => {
        Animated.spring(animatedButtonScale, {
          toValue: 1,
          useNativeDriver: true,
          speed: 50,
          bounciness: 4,
        }).start();
      };

      const animatedScaleStyle = {
        transform: [{ scale: animatedButtonScale }],
      };

      return (
        <SafeAreaView>
          <View style={styles.container}>
            <Text style={styles.heading}>
              {starRating ? "Thanks for the rate" : "Tap to rate"}
            </Text>
            <View style={styles.stars}>
              <TouchableWithoutFeedback
                onPressIn={handlePressIn}
                onPressOut={handlePressOut}
                onPress={() => handleRating(1)}
              >
                <Animated.View style={animatedScaleStyle}>
                  <MaterialIcons
                    name={starRating >= 1 ? "star" : "star-border"}
                    size={32}
                    style={
                      starRating >= 1 ? styles.starSelected :
styles.starUnselected
                    }
                  />
                </Animated.View>
              </TouchableWithoutFeedback>
              <TouchableWithoutFeedback
                onPressIn={handlePressIn}
                onPressOut={handlePressOut}
                onPress={() => handleRating(2)}
              >
                <Animated.View style={animatedScaleStyle}>
                  <MaterialIcons
                    name={starRating >= 2 ? "star" : "star-border"}
                    size={32}
                    style={
                      starRating >= 2 ? styles.starSelected :
styles.starUnselected
                    }
                  />
                </Animated.View>
              </TouchableWithoutFeedback>
              <TouchableWithoutFeedback
```

```
                onPressIn={handlePressIn}
                onPressOut={handlePressOut}
                onPress={() => handleRating(3)}
              >
                <Animated.View style={animatedScaleStyle}>
                  <MaterialIcons
                    name={starRating >= 3 ? "star" : "star-border"}
                    size={32}
                    style={
                      starRating >= 3 ? styles.starSelected :
styles.starUnselected
                    }
                  />
                </Animated.View>
              </TouchableWithoutFeedback>
              <TouchableWithoutFeedback
                onPressIn={handlePressIn}
                onPressOut={handlePressOut}
                onPress={() => handleRating(4)}
              >
                <Animated.View style={animatedScaleStyle}>
                  <MaterialIcons
                    name={starRating >= 4 ? "star" : "star-border"}
                    size={32}
                    style={
                      starRating >= 4 ? styles.starSelected :
styles.starUnselected
                    }
                  />
                </Animated.View>
              </TouchableWithoutFeedback>
              <TouchableWithoutFeedback
                onPressIn={handlePressIn}
                onPressOut={handlePressOut}
                onPress={() => handleRating(5)}
              >
                <Animated.View style={animatedScaleStyle}>
                  <MaterialIcons
                    name={starRating >= 5 ? "star" : "star-border"}
                    size={32}
                    style={
                      starRating >= 5 ? styles.starSelected :
styles.starUnselected
                    }
                  />
                </Animated.View>
              </TouchableWithoutFeedback>
            </View>
          </View>
        </SafeAreaView>
      );
    }
```

Ispis 17: Ocijenjivanje komponenta- kôd

6. Zaključak

U svijetu mrežnog i mobilnog razvoja, React i React Native su popularni alati koji omogućavaju razvoj suvremenih korisničkih sučelja. Razvijena je mobilna aplikaciju koja se bavi rezerviranjem stolova u noćnim klubovima i restoranima, uz mogućnost ocjenjivanja određenih proizvoda.

Prilikom razvoja ove aplikacije, korišten je React Native, koji omogućuje razvoj mobilnih aplikacija koristeći React, popularnu Javascript biblioteku za izgradnju korisničkih sučelja. React Native omogućuje ponovno korištenje sintakse i logike iz React aplikacije za mrežu, što također znatno pomaže u razumijevanju razvijanja mrežnih stranica.

Jedan od ključnih dijelova aplikacije je bio sustav za rezervaciju stolova. Koristeći React Native komponente, izgrađeno je intuitivno korisničko sučelje koje omogućuje korisnicima jednostavno rezerviranje stolova u noćnim klubovima i restoranima. Implementirana je interakcija s Firebase sustavom za rezervaciju i upravljanje podacima o stolovima, korisnicima i rezervacijama, koristeći asinkrone pozive.

Još jedan važan dio aplikacije bio je sustav za ocjenjivanje proizvoda. Koristeći React Native komponente za izradu ocjene pomoću zvjezdica, implementiran je sustav koji omogućuje korisnicima da ocijene proizvode na temelju njihovog iskustva. Korišten je React Native animacije za animiranje zvjezdica prilikom ocjenjivanja i za dodavanje vizualnog efekta korisničkom iskustvu.

Razvoj aplikacije zahtjeva izazove specifične za mobilni razvoj, kao što su prilagođavanje sučelja za različite uređaje i rukovanje gestama na mobilnim uređajima. Korištenje React Nativea i njegovih ugrađenih komponenti pomaže uvelike u prevladavanju izazova i razvijanju mobilne aplikacije koja pruža dosljedno korisničko iskustvo na različitim uređajima.

Zaključno, razvoj mobilne aplikacije pomoću React i React Nativea za rezervaciju stolova u noćnim klubovima i restoranima s mogućnošću ocjenjivanja proizvoda je zahtjevno, ali veoma korisno iskustvo.

Literatura

[1] https: //reactnative.dev/docs/ (zadnje posjećeno 22. 01. 2023.)

[2] https: //reactrativepaper.com/ (zadnje posjećeno 03. 02. 2023.)

[3] https: //docs.expo.dev/ (zadnje posjećeno 28. 01. 2023.)

[4] https: //classic.yarnpkg.com/lang/en/docs/getting-started/ (zadnje posjećeno 04. 03. 2023.)

[5] https: //firebase.google.com/docs (zadnje posjećeno 05. 02. 2023.)

[6] https: //firebase.google.com/docs/firestore (zadnje posjećeno 12. 03. 2023.)

[7] https://medium.com/@imchathu87/react-native-and-best-practices-9c9866fdabb7 (zadnje posjećeno 06. 03. 2023.)

[8] https: //zerotomastery.io/community/developer-community-discord/ (zadnje posjećeno 24. 01. 2023.)

[9] https: //github. com/react-native-datetimepicker/datetimepicker/ (zadnje posjećeno 05. 04. 2023.)