APLIKACIJA ZA INFORMIRANJE O ZNAMENITOSTIMA TEMELJENA NA GEOLOKACIJAMA

Sovilj, Ana

VISOKO UČILIŠTE ALGEBRA

ZAVRŠNI RAD

APLIKACIJA ZA INFORMIRANJE O ZNAMENITOSTIMA TEMELJENA NA GEOLOKACIJAMA

Ana Sovilj

Zagreb, veljača 2020.

„Pod punom odgovornošću pismeno potvrđujem da je ovo moj autorski rad čiji niti jedan dio nije nastao kopiranjem ili plagiranjem tuđeg sadržaja. Prilikom izrade rada koristio sam tuđe materijale navedene u popisu literature, ali nisam kopirao niti jedan njihov dio, osim citata za koje sam naveo autora i izvor, te ih jasno označio znakovima navodnika. U slučaju da se u bilo kojem trenutku dokaže suprotno, spreman sam snositi sve posljedice uključivo i poništenje javne isprave stečene dijelom i na temelju ovoga rada".

U Zagrebu, 18.02.2020.

Ana Sorilj

Predgovor

Zahvaljujem mentoru prof. Aleksanderu Radovanu na razumijevanju i savjetima kojima me usmjerio kroz proces izrade završnog rada.

Zahvaljujem svojoj obitelji i prijateljima na podršci tijekom izrade završnog rada i studiranja.

Sažetak

Aplikacija implementira funkcionalnost registracije, prijave korisnika, lokaciju i Google Maps za geolociranje korisnika i znamenitosti. Za potrebe aplikacije nužni su Wi-Fi ili mobilni podaci te uključena lokacija, odobrenja (engl. *permissions*) koja se definiraju u AndroidManifest XML-u. Podaci o korisniku spremljeni su u Firebase bazu podataka. Korisnik može vidjeti svoje podatke na profilu. Aplikacija omogućuje korisniku pretraživanje, kreiranje vlastite liste znamenitosti kao i funkcionalnost planiranja obilaženja znamenitosti unutar zadanog kruga korištenjem informacija o udaljenostima i Google Maps usluga. Cilj aplikacije je da korisnik pristupi znamenitostima u određenom opsegu s obzirom na ograničeno vrijeme.

Application implements the functionality of registration, user login, location and Google Maps to geolocate users and toursit attractions. The application requires WiFi or mobile data, location, permissions which are defined in AndroidManifest XML. Informations about users are saved to the Firebase database. User can see personal informations on profile. The application allows user to search, create own list of tourist attractions as functionality of planning sightseeing with a given radius based on distance information using Google Maps services. The goal is that user can access tourist attractions with a certain range considering limited time.

Ključne riječi: Firebase, Google Maps, AndroidManifest XML, odobrenja, vlastita lista, plan obilaska, informacije o znamenitostima, profil korisnika.

Sadržaj

1. Uvod

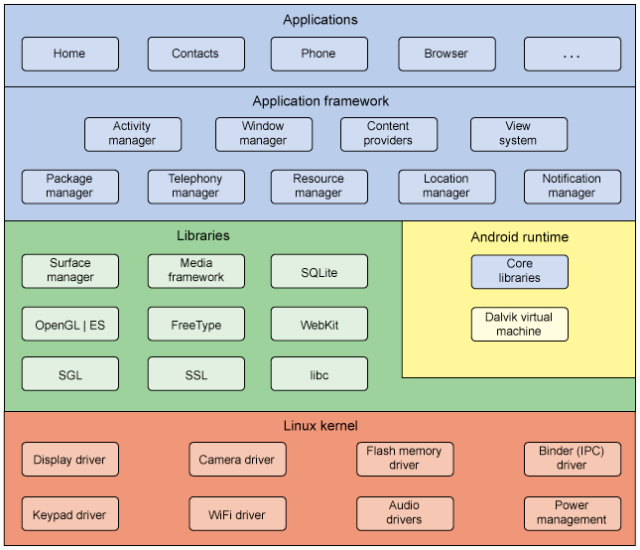
Aplikacija je namijenjena za mobilne uređaje koji podržavaju Android. Cijela implementacija aplikacije je odrađena u Android Studiu u programskom jeziku Java. Kada se aplikacija pokrene, korisnik se mora prijaviti. Ukoliko korisnik nema još korisnički račun, također ima ponuđenu registraciju. Podaci se spremaju u Firebase bazu podataka. Jedna od glavnih funkcionalnosti Firebasea je da osjetljive podatke kao npr. lozinke, stavlja u niz teksta pomoću algoritma te rezultat bude kriptirana vrijednost. Na taj način Firebase čini podatke sigurnim. Nakon registracije ili prijave prikazuje se karta. Na karti korisnik može odabrati znamenitosti koje želi vidjeti u blizini obzirom na trenutnu lokaciju unutar određenog radijusa. Na vrhu ekrana je prikazano polje za pretraživanje mjesta, znamenitosti itd. U podnožju je prikazana traka koja sadrži profil korisnika, listu željenih znamenitosti plan obilaska. Ukoliko korisnik odabere profil, može vidjeti svoje podatke, ako odabere listu željenih znamenitosti, prikazuje se popis znamenitosti koje je korisnik označio kao željene, ako odabere plan obilaska, prikazuju se znamenitosti koje obilazi u blizini. Kako bi aplikacija imala mogućnost rada nad kartom, potrebno je definirati pristupe i odobrenja za lokaciju, Internet u AndroidManifest XML-u, datoteci koju ima svaka aplikacija i sadrži informacije o aplikaciji. U sljedećem poglavlju opisan je Android koji obuhvaća arhitekturu i Android Studio unutar kojeg se razvija Android aplikacija definirana Android Manifestom i API razinama. U trećem poglavlju opisana je baza podataka Firebase koju je potrebno integrirati s aplikacijom kako bi se implementirale funkcionalnosti registracije i prijave korisnika. U četvrtom poglavlju opisane su funkcionalnosti Google Maps-a, plana obilaska, prikaz informacija određene znamenitosti te upute o dolasku do određene znamenitosti. U petom poglavlju opisani su sigurnosni aspekti aplikacije.

2. Android

Android je operacijski sustav za mobilne uređaje temeljen na Linux jezgri (engl. *kernel)*. Linux je odabran kao temelj arhitekture Androida zato što se njegov model pokazao uspješnim na različitim uređajima. Opisan je kao siguran sustav te pokretanjem aplikacije pokreće se proces na jezgri s dopuštenjima koja daje operacijski sustav. Raspolaže s korisnim značajkama kao npr. upravljanje memorijom, procesima itd. [1] Još jedna stavka koja čini Linux odabranim je otvorenost koda (engl. *open source*) te je Android pod licencom koja se zove Apache Licence koja omogućuje korisnicima slobodno preuzimanje, korištenje softvera u svrhu modifikacije, distribucije, distribucije modificiranih verzija pod uvjetima navedenih u licenci. [2]

2.1. Android arhitektura

Android se sastoji od nekoliko slojeva.



Slika 1: Android arhitektura podijeljena na Linux jezgru, biblioteke, aplikacijski okvir i aplikacije[3]

[1] J.Božić, T.Kadežabek, F.Tomić, T.Kaštelan; Izrada aplikacija za mobilne uređaje; Algebra(2013.), značajke Android sustava
[2] M.Karaga, M.Stojanović; Programiranje aplikacija za Android; Element(2018.), uvod u Android
[3] http://www.techplayon.com/wp-content/uploads/2017/06/architec.png, Android arhitektura u razinama

Najniži sloj je Linux jezgra opisana pod točkom 2. Android. Blok iznad Linux jezgre predstavlja ugrađene biblioteke (engl. *libraries*) koje su pisane u C ili C++ programskom jeziku i pružaju usluge aplikacijskom sloju. Do njih se nalazi segment arhitekture pod nazivom „Android runtime“ kojeg čine biblioteke koje omogućuju programerima pisanje aplikacija i Dalvik virtualni stroj. Dalvik je virtualni stroj koji je Google razvio za mobilne uređaje. Google se odlučio razviti svoj virtualni stroj zato što prije njega su koristili Javin virtualni stroj za mobilne uređaje i naišli su na probleme: baterija na mobilnim uređajima koja se brzo trošila, procesori su na mobilnim uređajima bili slabiji i Javin virtualni stroj nije besplatan. Kompajliranje Java koda odvija se tako da kod napisan u Javi prevodi se u Javin bajtkod (engl. *Java bytecode*) te Dex Compiler prevede Javin bajtkod u Dalvik bajtkod (engl. *Dalvik bytecode*) koji se izvršava na Dalvik virtualnom stroju. Zatim slijedi aplikacijski okvir koji nudi usluge koje programeri koriste za svoje aplikacije. Zadnji sloj su aplikacije koji će korisnici Androida koristiti. [4] ART (engl. *Android Run Time*) je nasljednik Dalvika. Izvršava .dex datoteke, stoga kod može raditi pod Dalvikom i pod ART-om. Međutim, postoje prednosti ART-a: AOT (engl. *Ahead Of Time compilation*) kako bi performanse bile ubrzane, poboljšani GC (engl. *Garbage Collector*) koji automatski upravlja memorijom te jasnije iskazivanje grešaka u aplikaciji. Android inačice 5.0 i 6.0 još koriste Dalvik, ali od inačice 7.0 dolazi ART.[5]

2.2. Android Studio

Android Studio je razvojno okruženje izgrađeno na JetBrainsovom Intellij IDEA softveru. Namijenjen je za razvoj Android aplikacija. Dostupan je na Windows, macOS i Linux operacijskim sustavima. Aplikacije se razvijaju ili u Javi ili u Kotlinu koje nudi Android Studio.

[4] J.Božić, T.Kadežabek, F.Tomić, T.Kaštelan; Izrada aplikacija za mobilne uređaje; Algebra (2013.), arhitektura Androida
[5] M.Karaga, M.Stojanović; Programiranje aplikacija za Android; Element(2018.), značajke ART-a

2.2.1. AndroidManifest XML

AndroidManifest.xml je datoteka koju ima svaka aplikacija. Prilikom kreiranja projekta tj. aplikacije, automatski se generira i ta datoteka. Sadrži važne informacije o aplikaciji.

Svaka datoteka sadržava atribut koji predstavlja identifikator aplikacije (engl. *package*). Sve izvorne programske datoteke sustav izgradnje Gradle (engl. *build sustav Gradle)* uzima te na njih primjenjuje alate povezivanja i prevođenja kako bi ih grupirao u .apk datoteku s kojom Android zna raditi. U datoteci build.gradle identifikator se dodjeljuje ID-u aplikacije koji je naveden u datoteci. Identifikator mora biti univerzalan tj. jedinstven zato što je pokazatelj čija je aplikacija na Google Play-u. [6]

Primjer koda Android Manifesta

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.as.landmarkapplication">

<uses-permission android:name="android.permission.INTERNET" />
<uses-permission
   android:name="android.permission.ACCESS_COARSE_LOCATION" />
<uses-permission
 android:name="android.permission.ACCESS_FINE_LOCATION" />

<application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">

    <meta-data
         android:name="com.google.android.geo.API_KEY"
         android:value="@string/google_maps_key" />
```

[6] M.Karaga, M.Stojanović; Programiranje aplikacija za Android; Element(2018.), značajke Gradle-a

```
      <activity
          android:name=".MapsActivity"
          android:label="@string/title_activity_maps" />
        <activity android:name=".SignUpActivity" />
        <activity android:name=".MainActivity">
           <intent-filter>
             <action android:name="android.intent.action.MAIN" />

             <category android:name="android.intent.category.LAUNCHER" />
           </intent-filter>
        </activity>
    </application>
</manifest>
```

Kôd 1. Android Manifest

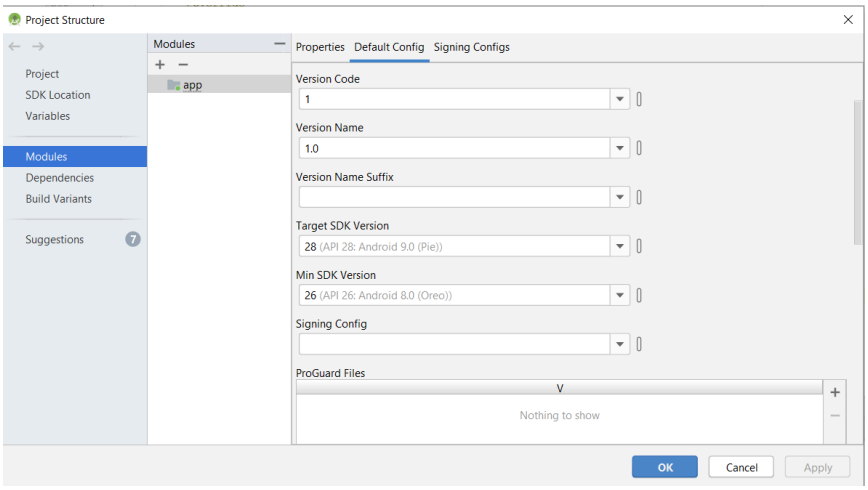
U kodu 1 unutar AndroidManifest.xml datoteke definiraju se odobrenja (engl. *permissions*) koja aplikacija zahtjeva kako bi pristupila određenim značajkama sustava kao npr. lokacija, Internet itd. Lokacije mogu biti orijentirane na mrežne usluge (engl. *coarse location)* ili na usluge GPS-a i mrežne usluge (engl. *fine location)*. Unutar datoteke nalazi se API ključ koji povezuje aplikaciju s Google Mapsom. Element aktivnost (engl. *activity)* predstavlja aktivnost koja implementira korisničko sučelje. Svaka novokreirana aktivnost mora biti navedena u datoteci, ako nije, neće se nikada pokrenuti. Element namjere (engl. *intent filter)* predstavlja mogućnosti koje aktivnosti odrađuju. [7]

2.2.2. API razine u Androidu

Prilikom kreiranja aplikacije treba uzeti u obzir minimalnu i ciljanu API razinu koju će aplikacija podržavati. Raspon API razina s kojima raspolaže Android Studio jest od 21 do 29. Svakom novom verzijom Android Studio uzima novu API razinu i samim time i novu verziju Androida. API razine aplikacije nalaze se unutar Android Studia, tako da se klikne na File > Project Structure. Unutar strukture projekta prikazana je verzija koda, verzija aplikacije, minimalna API razina i ciljana API razina. Raspon razina govori o tome koliki

[7] https://developer.android.com/, definiranje odobrenja, značajke lokacija, značajke AndroidManifest XML-a

postotak tržišta aplikacija pokriva. API razine trebale bi se prilagoditi i alatima koje aplikacija koristi te mogu biti prilagođene u bilo kojem trenutku unutar Android Studia.



Slika 2: API razine u aplikaciji

Aplikacija za informiranje o znamenitostima temeljena na geolokacijama podržava API razinu 26 koja je minimalna i ciljanu API razinu 28. To znači da aplikacija je namijenjena za Android uređaje verzije 8.0 i 9.0. Verzije izvan tog raspona ne podržavaju aplikaciju. API razine programer bira prema potrebama aplikacije.

Jedan od ključnih faktora kod odabira API razine aplikacije je i korištenje drugih alata ili okruženja koji su povezani s aplikacijom te tako pružaju aplikaciji dodatne mogućnosti.

Budući da aplikacija implementira funkcionalnost registracije i prijave korisnika te sprema podatke u Firebase bazu podataka, prema dokumentaciji Firebase baze podataka, ciljana razina aplikacije mora biti 16 ili veća od 16. [8]

[8] https://firebase.google.com/docs/android/setup, preduvjeti integriranja Firebasea s Android aplikacijom

3. Firebase

Firebase je baza podataka u kojoj su podaci pohranjeni u JSON formatu. Baza je strukturirana kao stablo s mnoštvo čvorova, stoga kada se izrađuje baza treba imati na umu da JSON format mora biti pripremljen tako da su podaci lako dostupni izbjegavajući ugniježđene čvorove. [9]

Preduvjeti potrebni za integraciju Firebasea s Android aplikacijom su:

- Zadnja verzija Android Studia
- API razina 16 (verzija Androida: Jelly Bean) ili kasnije razine
- Gradle 4.1. ili kasnija verzija
- Postaviti uređaj ili emulator za pokretanje aplikacije
- Prijava u Firebase pomoću Google korisničkog računa

Navedeni preduvjeti definirani su u dokumentaciji Firebasea. [10]

3.1. Integracija Firebasea s Android aplikacijom

Potrebno je prijaviti se putem Google korisničkog računa u Firebase. Nakon toga, odabere se kreiranje novog projekta. Za novi projekt potrebno je definirati ime projekta koji sukladno tome prikaže ID. Sljedeći korak je opcionalan te nudi da se uključe dodatne mogućnosti koje nudi Firebase i nakon toga kliknuti na gumb Završi.

Nakon što je kreiran Firebase projekt, aplikacija može biti integrirana tako da se odabere mogućnost povezivanja Firebasea s Androidom u Firebaseu. Kako bi došlo do integracije potrebno je napraviti projekt u Android Studiu te omogućiti strukturu na razini projekta kako bi se mogli uočiti direktoriji. Unutar aplikacijskog modula potrebno je staviti google-services.json datoteku koju traži servis (engl. *google-services-plugin*) te uz nju traži i druge konfigurirane postavke. Sljedeći korak je dodavanje potrebnih pravila kako bi se mogle koristiti biblioteke koje traže servisi. Budući da Firebase raspolaže različitim uslugama

[9] https://www.androidhive.info/2016/10/android-working-with-firebase-realtime-database/, značajke Firebase baze podataka
[10] https://firebase.google.com/docs/android/setup, preduvjeti integriranja Firebasea s Android aplikacijom

potrebno je unutar Gradle datoteke definirati s kojim alatima će se raspolagati tijekom projekta.

Drugi način je, unutar Android Studia pod alatima (engl. *tools*), odabrati Firebase te jednu od usluga koju nudi i spojiti se na Firebase.

3.2. Funkcionalnosti baze

Firebase raspolaže s mnogo API-ja za Android, iOS, Web, Unity, C++ itd.

Za izgradnju aplikacije nudi Cloud Firestore, ML Kit, Cloud Functions, Authentication, Hosting, Cloud Storage, Realtime Database. Cloud Firestore je fleksibilna baza podataka te omogućava sinkronizaciju podataka i izvanmrežnu podršku tako da aplikacije mogu biti responzivne bez obzira na internetsku povezanost. Omogućava integraciju i s drugim Firebase i Google Cloud Platform proizvodima. ML Kit je mobilni SDK (engl. *software development kit*). SDK je skup softverskih alata koji omogućavaju stvaranje aplikacija s naprednim funkcijama. ML Kit nudi API-je koji implementiraju funkcionalnosti strojnog učenja u nekoliko redaka koda bez obzira je li se radi o novom ili iskusnom programeru. Firebase Authentication pruža usluge, SDK, korisnička sučelja za provjeru autentičnosti korisnika pomoću lozinki ili pružatelja identiteta kao što su Google, Facebook, Twitter itd. Provjera autentičnosti integrira se i s ostalim Firebase uslugama i koristi druge standarde npr. OAuth 2.0 i OpenID Connect. Cloud Storage je usluga skladištenja objekata te se SDK-ovi mogu koristiti za pohranu slika, videa itd. Firebase Realtime Database je baza podataka u kojoj se podaci pohranjuju u obliku JSON formata i sinkronizirani su u stvarnom vremenu na svim klijentima i ostaju dostupni kada aplikacija bude izvan mreže. Za unaprjeđenje aplikacije Firebase nudi Crashlytics, App Distribution, Performance Monitoring i Test Lab. Crashlytics je usluga izvještaja o padu tj. problemu sa stabilnošću koji narušava kvalitetu aplikacije. Mjeri stabilnost te tako se može dobiti uvid kada je aplikacija spremna za isporuku. App Distribution je usluga postavljanja aplikacija na uređaje testera tako da se povratne informacije mogu dobiti vrlo rano. Performance Monitoring je usluga koja omogućuje uvid u karakteristike performansi aplikacija. Potrebno je koristiti SDK za praćenje performansi da bi se prikupili podaci o izvedbi iz aplikacije te kasnije mogli pregledati i analizirati. Nadgledavanje performansi pomaže da se dobije uvid kada i gdje se

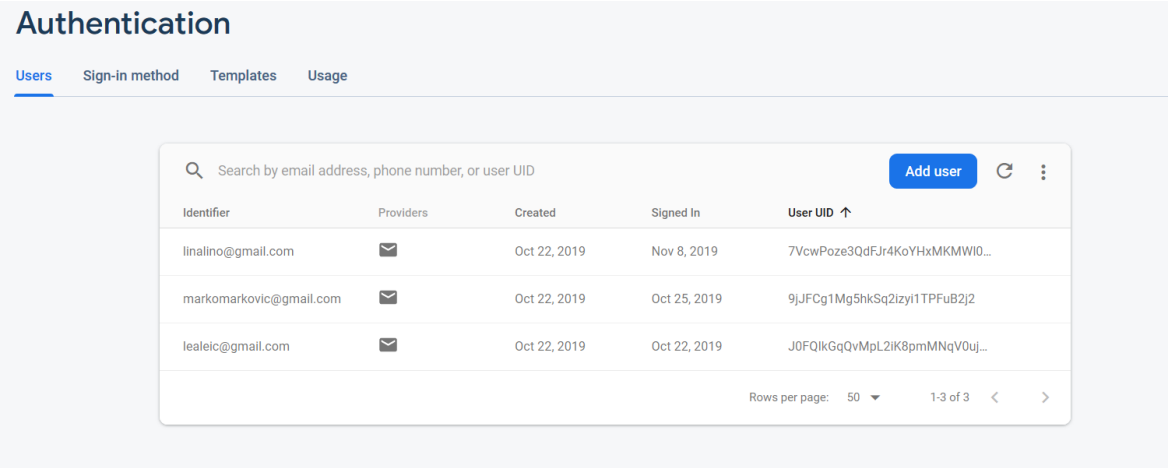
može poboljšati učinkovitost aplikacije. Test Lab je infrastruktura za testiranje aplikacija na širokom rasponu uređaja te kasnije se dobije uvid u rezultate. Za razvijanje aplikacije Firebase nudi Analytics, Predictions, Firebase A/B Testing, Cloud Messaging, Remote Config itd. Google Analytics je usluga za mjerenje aplikacija koja pruža uvid u upotrebu aplikacije i ponašanje korisnika kako bi se donijele odluke o optimizaciji performansi aplikacija. Predictions je usluga koja primjenjuje strojno učenje na analitičke podatke na temelju predviđenog ponašanja korisnika. Firebase A/B Testing je usluga koja pomaže optimizirati aplikaciju olakšavajući pokretanje, analizu i skaliranje proizvoda i marketinških eksperimenata. Cloud Messaging je usluga koja omogućuje pouzdanu razmjenu poruka i slanje obavijesti korisnicima. Za slučaj trenutnih poruka, može se prenijeti do 4KB na aplikaciju klijenta. Remote Config je usluga koja omogućuje promjenu ponašanja i izgleda aplikacije bez da korisnici zahtijevaju preuzimanja ažuriranja aplikacije. [11]

3.3. Registracija i prijava korisnika

Za implementaciju registracije i prijave korisnika, u Aplikaciji o znamenitostima temeljenoj na geolokacijama, odabrana je usluga autentikacije (engl. *authentication).* Usluga omogućava kreiranje korisničkog računa sa zadanom email adresom i lozinkom te daljnje prijavljivanje pomoću korisničkog računa. Firebase ima zasebni dio za autentikaciju gdje se nalazi tablica registriranih korisnika. Unutar tablice definirani su: email korisnika, datum kreiranja računa, datum prijave korisnika pomoću kreiranog računa i identifikator korisnika koji je jedinstven za svakoga.

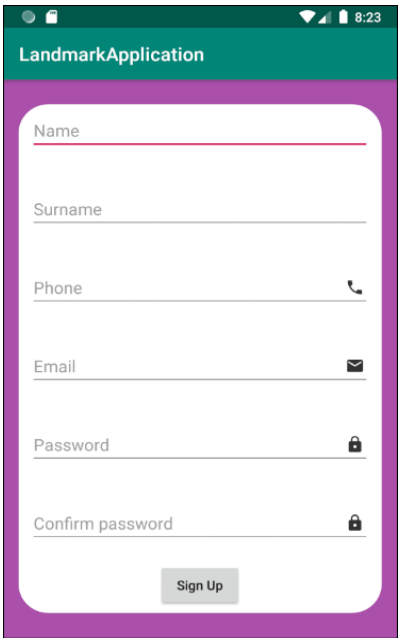
Firebase autentikacija nudi i načine prijave korisnika u aplikaciju koje mogu biti putem emaila i lozinke, telefona, Google korisničkog računa, Facebooka, Twittera, GitHuba itd. [12]

[11] https://firebase.google.com/docs, navedene usluge Firebasea
[12] https://firebase.google.com/docs/auth, navedena usluga autentikacije



Slika 3: Tablica registriranih korisnika pomoću usluge autentikacije

3.3.1. Registracija korisnika



Slika 4: Ekran za registraciju korisnika

Prilikom registriranja korisnik mora unijeti navedene podatke (ime, prezime itd.). Na svakom polju za unos teksta određen je tip podatka tako da korisnik ne može unijeti krivi tip podatka. Ako je polje za lozinku tipa lozinka, onda lozinka mora imati bar šest znakova. Sva polja za unos moraju biti ispunjena kako bi se korisnik registrirao.

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_sign_up);

    init();

    firebaseAuth = FirebaseAuth.getInstance();

    signUp();
}

private void signUp() {
  btnSignUp.setOnClickListener(new View.OnClickListener() {
  @Override
  public void onClick(View v) {

  final String name = etName.getText().toString().trim();
  final String surname = etSurname.getText().toString().trim();
  final String phone = etPhone.getText().toString().trim();
  final String email = etEmail.getText().toString().trim();
  final String password = etPassword.getText().toString().trim();
  final String confirmPassword = etConfirmPassword.getText()
                                 .toString().trim();

if (!(name.isEmpty() && surname.isEmpty() && phone.isEmpty() &&
      email.isEmpty() && password.isEmpty() &&
      confirmPassword.isEmpty()) && confirmPassword.equals(password)) {
      firebaseAuth.createUserWithEmailAndPassword(email, password)
     .addOnCompleteListener(new OnCompleteListener<AuthResult>() {
      @Override
      public void onComplete(@NonNull Task<AuthResult> task) {
      if (task.isSuccessful()) {

      String user_id = firebaseAuth.getCurrentUser().getUid();
      DatabaseReference userDb = FirebaseDatabase
     .getInstance().getReference().child("Users").child(user_id);

      Map newData = new HashMap();
      newData.put("name", name);
      newData.put("surname", surname);
```

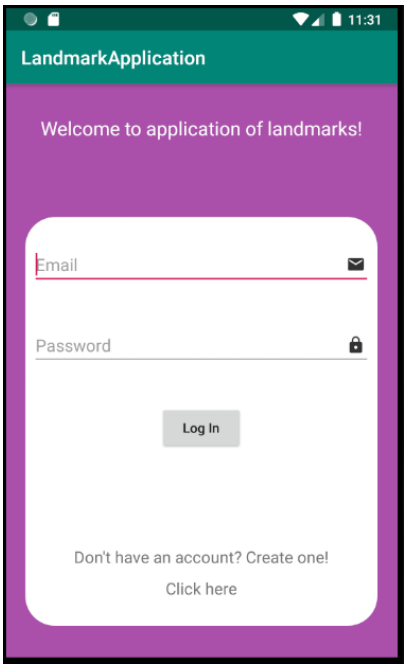
```
        newData.put("phone", phone);
        newData.put("email", email);

        userDb.setValue(newData);

        Toast.makeText(SignUpActivity.this, "User is registered",
        Toast.LENGTH_LONG).show();
        Intent intent = new Intent(SignUpActivity.this,
        MapsActivity.class);
        startActivity(intent);
        }
else {
        Toast.makeText(SignUpActivity.this, task.getException()
                        .getMessage(), Toast.LENGTH_LONG).show();
      }
                        }
                    });
                }
            }
        });
}
```

Kôd 2. Kod za registraciju korisnika

U kodu 2 kada korisnik ispuni sva polja za unos, pritisne gumb za registraciju (engl. *Sign Up*). Pritiskom gumba kreira se korisnički račun tako da se proslijedi email i lozinka od korisnika u metodu `createUserWithEmailAndPassword()`. Da bi metoda bila pozvana, potrebno je unutar aktivnosti koja je namijenjena za registraciju u `onCreate()` metodi dohvatiti instancu `FirebaseAuth` objekta. Ako su podaci uspješno proslijeđeni, podaci korisnika smještaju se u mapu pod ključem i vrijednosti i spremaju se u bazu pod jedinstvenim ključem unutar korijena korisnika (engl. *Users*). Nakon što korisnik pritisne gumb, prikazuje se karta.

Slika 5: Korisnici spremljeni u bazu u JSON formatu

3.3.2. Prijava korisnika



Slika 6: Ekran za prijavu korisnika

Prilikom prijave korisnik mora unijeti navedene podatke. Sva polja za unos moraju biti ispunjena kako bi se korisnik prijavio. Polje za lozinku zahtijeva minimalno šest znakova. Kada korisnik unese email i lozinku, pritisne na gumb za prijavu (engl. *Log In*).

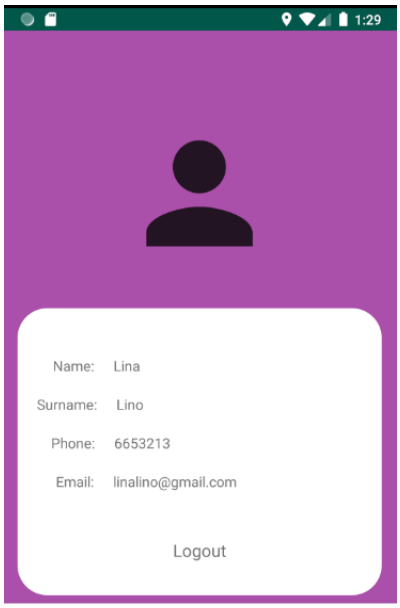
```
private void logIn() {
btnLogIn.setOnClickListener(new View.OnClickListener() {
@Override
public void onClick(View v) {

final String email = etEmail.getText().toString().trim();
final String password = etPassword.getText().toString().trim();

if (!(email.isEmpty() && password.isEmpty())) {

     firebaseAuth.fetchSignInMethodsForEmail
    (etEmail.getText().toString()).addOnCompleteListener(new
     OnCompleteListener<SignInMethodQueryResult>() {
     @Override
     public void onComplete(@NonNull Task<SignInMethodQueryResult> task)
     {

          boolean check = !task.getResult().getSignInMethods().isEmpty();

          if (check) {
              firebaseAuth.signInWithEmailAndPassword(email, password)
              .addOnCompleteListener(MainActivity.this, new
              OnCompleteListener<AuthResult>() {
              @Override
              public void onComplete(@NonNull Task<AuthResult> task) {

                 if (task.isSuccessful()) {

                     Intent intent = new Intent(MainActivity.this,
                                                MapsActivity.class);
                     startActivity(intent);

                  }
                  else {

                     Toast.makeText(MainActivity.this, "Unable to
                     login",Toast.LENGTH_LONG).show();
                  }
            }
       });
}
```

```
 else {
     Toast.makeText(MainActivity.this, "Something went wrong, please try
      again", Toast.LENGTH_LONG).show();
 }
 }});
}}});
```

Kôd 3. Kod za prijavu korisnika

U kodu 3 pritiskom na gumb provjerava se email s kojim se korisnik registrirao te se poziva metoda `signInWithEmailAndPassword()` kojoj se prosljeđuje email i lozinka kao parametri. Ako su podaci valjani, korisniku se prikazuje karta.

3.4. Profil korisnika



Slika 7: Ekran za profil korisnika

Unutar aplikacije korisnik ima mogućnost vidjeti svoj profil na kojem su informacije o korisniku. Ispod informacija prikazana je odjava korisnika.

```
private void showUserInfo() {
FirebaseUser user = FirebaseAuth.getInstance().getCurrentUser();
String user_id = user.getUid();

DatabaseReference reference =
FirebaseDatabase.getInstance().getReference("Users");

reference.child(user_id).addListenerForSingleValueEvent(new
ValueEventListener() {
    @Override
    public void onDataChange(@NonNull DataSnapshot dataSnapshot) {

      if (dataSnapshot.exists()) {
          String name = dataSnapshot.child("name").getValue().toString();
          String surname = dataSnapshot.child("surname").getValue()
                                                        .toString();
          String phone = dataSnapshot.child("phone").getValue()
                                                    .toString();
          String email = dataSnapshot.child("email").getValue()
                                                    .toString();

          tvName.setText(name);
          tvSurname.setText(surname);
          tvPhone.setText(phone);
          tvEmail.setText(email);

      }
    }

    @Override
    public void onCancelled(@NonNull DatabaseError databaseError) {
     Log.d(ProfileFragment.class.getName(), databaseError.getMessage());
    }
 });
```

Kôd 4. Kod za profil korisnika

U kodu 4 prvo se dohvaća trenutni korisnik metodom `getCurrentUser()` kako bi se došlo do informacija o korisniku. ID korisnika je dohvaćen metodom `getUid()`. Zatim je potrebna referenca na korijen korisnika (engl. *Users)* preko koje se dohvaća dijete (engl.

child) i proslijedi se ID. Kako bi podaci bili učitani od djeteta, metoda `addListenerForSingleValueEvent()` mora biti pozvana. Unutar te metode treba provjeriti da li podaci postoje za to dijete. Ako postoje, vrijednosti se dohvaćaju i spremaju u varijable te se ispisuju vrijednosti unutar pregleda teksta (engl. *TextView*) koji je smješten na ekranu.

3.4.1. Odjava korisnika

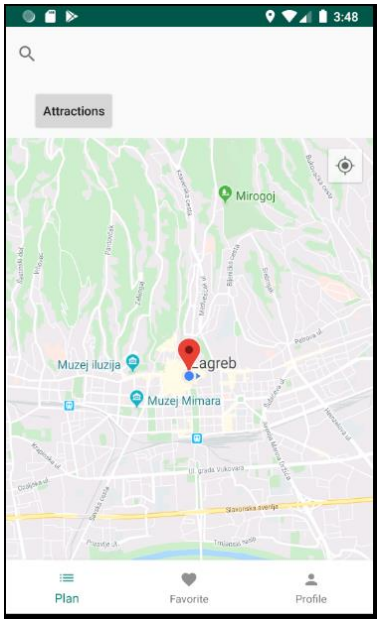
Ispod informacija o korisniku, prikazana je odjava. Klikom na nju korisnik se odjavljuje iz aplikacije.

```
private void logoutUser() {
    tvLogout.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
        FirebaseAuth.getInstance().signOut();
        Intent intent = new Intent(getActivity(), MainActivity.class);
        startActivity(intent);
        }
    });
}
```

Kôd 5. Kod odjave korisnika

U kodu 5 `FirebaseAuth` objekt zove metodu `getInstance()` preko koje je pozvana metoda `signOut()` kojom se odjavljuje trenutni korisnik i briše ga se iz priručne ili predmemorije (engl. *cache)* diska. Klikom na odjavu, korisnika se usmjeri na ekran prijave tj. početni ekran aplikacije.

4. Google Maps



Slika 8: Prikaz karte, pretraživača, gumba znamenitosti, donje trake s podacima

Kako bi aplikacija mogla prikazati trenutnu lokaciju i obližnje znamenitosti, potrebne informacije o dolasku do određene znamenitosti, potrebno je koristiti Google Maps. Za potrebe ove aplikacije unutar Android Studia kreira se nova aktivnost (engl. *activity*) pod nazivom „Google Maps Activity“. Kada se aktivnost kreira, uz nju dolazi Activity Maps XML i Google Maps API XML. Unutar Google Maps API XML-a potrebno je smjestiti API ključ, koji povezuje Google Maps s aplikacijom, unutar *String* elementa pri dnu XML-a. Do API ključa dolazi se tako da se kopira prvi link unutar Google Maps API XML-a i zalijepi u jedan od preglednika. Link vodi na stranicu na kojoj se kreira ključ. Google Maps nudi mnoge API-je putem Google Console gdje je potrebno prijaviti se Google korisničkim računom.

Na slici 8 prikazan je glavni ekran aplikacije. Na vrhu je prikazano povećalo za pretraživanje mjesta tj. znamenitosti, gumb na koji pritiskom prikazuje obližnje znamenitosti, karta i donji dio koji čini traku s karticama na kojima su plan obilaska, lista željenih znamenitosti i profil korisnika.

Za korištenje Google Maps usluga, potrebno je otići na Google Cloud Platform i odabrati Google Console. Zatim se prijaviti putem Google korisničkog računa na Google Console.

Nakon uspješne prijave, moguće je dobiti uvid u usluge koje su korištene za razvijanje aplikacije te pretražiti API-je i aktivirati ih kako bi aplikacija mogla pristupiti uslugama. Također je omogućen uvid u ključeve koji su potrebni za pojedini API i grafički prikaz korištenja API-ja. Aplikacija koristi Places API i Directions API. Za Directions API, potreban je račun naplate (engl. *Billing Account*) kako bi pristup uslugama u potpunosti bio omogućen bez prekida.

4.1. Trenutna lokacija

Za prikaz trenutne lokacije potrebno je zatražiti dozvolu lokacije koja se definira unutar Android Manifest XML-a i u Java kodu. Zatim da se omogući trenutna lokacija potrebno je koristiti pružatelja lokacije (engl. *fused location provider*). [13]

```
private void getUserLocation() {

   if (ActivityCompat.checkSelfPermission(this.getApplicationContext(),
       android.Manifest.permission.ACCESS_FINE_LOCATION)
       != PackageManager.PERMISSION_GRANTED) {
       ActivityCompat.requestPermissions(this, new String[]
       {Manifest.permission.ACCESS_FINE_LOCATION}, REQUEST_CODE);
            return;
      }

    Task<Location> task = fusedLocationProviderClient.getLastLocation();

    task.addOnSuccessListener(new OnSuccessListener<Location>() {
        @Override
        public void onSuccess(Location location) {
            if (location != null){
                currentLocation = location;
                SupportMapFragment mapFragment = (SupportMapFragment)
```

[13] https://developers.google.com/maps/documentation/android-sdk/current-place-tutorial, dokumentacija u kojoj je opisan pronalazak trenutne lokacije

```
                    getSupportFragmentManager().findFragmentById(R.id.map);

                    mapFragment.getMapAsync(MapsActivity.this);
                }
                }
            });
}

@Override
public void onRequestPermissionsResult(int requestCode, @NonNull String[]
permissions, @NonNull int[] grantResults) {
    switch (requestCode){
        case REQUEST_CODE:
        if (grantResults.length > 0 && grantResults[0] ==
            PackageManager.PERMISSION_GRANTED) {
                        getUserLocation();
            }
      }
}
```

Kôd 6. Kod za trenutnu lokaciju

U kodu 6 prvo se traži odobrenje za lokaciju, a rezultat koji je dan na zahtjev obrađuje se metodom `onRequestPermissionsResult()`. Ta metoda daje odgovor korisniku, koji je zatražio lokacijske usluge, da li je pristup odobren ili nije. Preko pružatelja lokacije dohvaća se zadnja lokacija korisnika. Unutar metode `onSuccess()` provjerava se je li lokacija uspješno dohvaćena, ako je prikazuje se mapa s trenutnom lokacijom korisnika.

4.2. Pretraživanje

```
@Override
public boolean onQueryTextSubmit(String query) {
       String location = svSearchLocation.getQuery().toString();
       List<Address> addressList = null;

       if(location != null || !location.equals("")){
           Geocoder geocoder = new Geocoder(MapsActivity.this);
```

```
            try {
                addressList = geocoder.getFromLocationName(location, 5);
            } catch (IOException e) {
                e.printStackTrace();
            }

            if(marker != null){
               marker.remove();
               marker = null;
            }

           for (int i = 0; i < addressList.size(); i++) {
                Address address = addressList.get(i);
                LatLng latLng = new LatLng(address.getLatitude(),
                                address.getLongitude());
                MarkerOptions markerOptions = new MarkerOptions()
                                .position(latLng).title(location);
                mMap.animateCamera(CameraUpdateFactory
                                .newLatLngZoom(latLng, 15));
                marker =  mMap.addMarker(markerOptions);
            }
        }
        return false;
}
```

Kôd 7. Pretraživanje

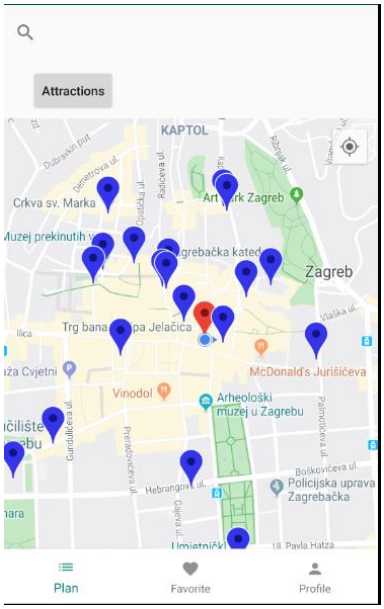
U kodu 7 opisano je pretraživanje mjesta, znamenitosti itd., pomoću klase `Geocoder` koja upravlja procesom pretvaranja adrese u koordinate i dobivanja adrese iz koordinata. Metoda `getFromLocationName()` vraća polje adresa koje opisuju zadanu lokaciju s maksimalnim brojem rezultata. Za rezultat predlažu se brojevi od 1 do 5.[14]

Unutar `for` petlje dohvaća se zadana adresa, pridružuju joj se zadane koordinate te se postavlja na mapu. Unutar uvjeta provjerava se da li postoji marker obilježen na mapi kod

[14] https://developer.android.com/reference/android/location/Geocoder.html#getFromLocationName(java.lang.String,%20int), značajke klase `Geocoder`

pretraživanja, ako ne postoji doda se novi, u suprotnom uklanja se, tako da se izbjegne puno obilježenih markera na mapi.

4.3. Prikaz obližnjih znamenitosti



Slika 9: Ekran s prikazom obližnjih znamenitosti

Za prikaz obližnjih znamenitosti potrebne su tri klase: `PlacesUrl`, `DataParser` i `NearbyPlaces`. Unutar klase `PlacesUrl` dohvaćaju se podaci s URL-a pomoću apstraktne klase `HttpURLConnection`.

```
public String readUrl(String stringUrl) throws IOException {

    String data = "";
    InputStream inputStream = null;
    HttpURLConnection httpURLConnection = null;

    try {
        URL url = new URL(stringUrl);

        httpURLConnection = (HttpURLConnection) url.openConnection();

        httpURLConnection.connect();
```

```
            inputStream = httpURLConnection.getInputStream();

            BufferedReader br = new BufferedReader(new
            InputStreamReader(inputStream));

            StringBuffer sb = new StringBuffer();

            String line = "";

            while ((line = br.readLine()) != null){
                sb.append(line);
            }

            data = sb.toString();
            br.close();
        }
        catch (Exception e){
            e.printStackTrace();
        }
        finally {
            inputStream.close();
            httpURLConnection.disconnect();
        }
        return data;
}
```

Kôd 8. Kod za dohvaćanje podataka s URL-a

U kodu 8 opisana je klasa `PlacesUrl`. Prvo je potrebno doći do konekcije metodom `openConnection()`. Metodom `getInputStream()` vraća odgovor tj. podatke koji su dostupni za čitanje s otvorene konekcije. Podaci koji su vraćeni s konekcije dostupni su u JSON formatu.

```
"geometry" : {
            "location" : {
               "lat" : 45.8093983,
               "lng" : 15.9700411
            },
            "viewport" : {
```

```
      "northeast" : {
         "lat" : 45.8109509302915,
         "lng" : 15.97173845
      },
      "southwest" : {
         "lat" : 45.8082529697085,
         "lng" : 15.96833824999999
      }
   }
},
"icon" : "https://maps.gstatic.com/mapfiles/
         place_api/icons/generic_business-71.png",
"id" : "df32be6c1691fa853fbf20fd0b756d5a8a15ac36",
"name" : "Croatian National Theatre in Zagreb",
"photos" : [
   {
      "height" : 3840,
      "html_attributions" : [
         "\u003ca href=\"https://maps.google.com/maps/
             contrib/103349552118426391594\"\u003eAlbert
              Bredenhann\u003c/a\u003e"
      ],
      "photo_reference" : "CmRaAAAAAxYFpVp1A
        b6Bl69Oydz_yOSeiQVrcZMCcrbiYaCAvewVUQCAsjlwJCmQNae
        b029kt-65DYQL9pHshkXC4FpTroy1Yzamb14rn
        4QGLgic7BTmmPzQP5rfulSzLVVHJEhDayVx1dE_7
        jwRGP7tMEOPsGhTPe2eAka4MTOXRhhkuBGt8fXGOZQ",
      "width" : 5120
   }
],
"place_id" : "ChIJp5AjU_vWZUcRIronALJH5p4",
"plus_code" : {
   "compound_code" : "RX5C+Q2 Zagreb, Croatia",
   "global_code" : "8FQQRX5C+Q2"
},
"rating" : 4.7,
"reference" : "ChIJp5AjU_vWZUcRIronALJH5p4",
"scope" : "GOOGLE",
"types" : [ "tourist_attraction", "point_of_interest",
"establishment" ],
"user_ratings_total" : 5193,
```

```
         "vicinity" : "Trg Republike Hrvatske 15, Zagreb"
      }
```

Kôd 9. Podaci obližnjih znamenitosti u JSON formatu

Vraćeni podaci parsiraju se pomoću klase `DataParser`. `DataParser` sadrži metode `getPlace()`, `getPlaces()` i `parse()`. Metodom `getPlace()` dohvaća se znamenitost pomoću JSON objekta (engl. *JSONObject*). Metodi `getPlaces()` se proslijedi JSON polje (engl. *JSONArray*) i unutar `for` petlje koja ide od 0 do ukupne veličine JSON polja dodaje se znamenitost na listu. Metodom `parse()` dohvaćaju se znamenitosti unutar JSON polja od čvora rezultati (engl. *results*).

```
try {

   if (!googlePlaceJson.isNull("name")){
       placeName = googlePlaceJson.getString("name");
   }
   if (!googlePlaceJson.isNull("vicinity")){
        vicinity = googlePlaceJson.getString("vicinity");
   }

        latitude = googlePlaceJson.getJSONObject("geometry")
                   .getJSONObject("location").getString("lat")
        longitude = googlePlaceJson.getJSONObject("geometry")
                   .getJSONObject("location").getString("lng");
        reference = googlePlaceJson.getString("reference");

        googlePlaceMap.put("place_name", placeName);
        googlePlaceMap.put("vicinity", vicinity);
        googlePlaceMap.put("lat", latitude);
        googlePlaceMap.put("lng", longitude);
        googlePlaceMap.put("reference", reference);

} catch (JSONException e) {
        e.printStackTrace();
}
```

Kôd 10. Kod za dohvaćanje pojedine znamenitosti

U kodu 10 dohvaća se znamenitost s nazivom, bliskošću druge znamenitosti, koordinatama i referencom znamenitosti na mapi koja predstavlja i ID znamenitosti. Pojedinosti o znamenitosti dohvaćaju se pomoću JSON objekta kojem se proslijedi naziv pod kojim se dobije željena vrijednost. Zatim se znamenitost sprema u mapu pod ključem i vrijednosti (engl. *HashMap*).

```
private List<HashMap<String, String>> getPlaces(JSONArray jsonArray){

    int numberOfPlaces = jsonArray.length();

    List<HashMap<String, String>> listOfPlaces = new ArrayList<>();
    HashMap<String, String> place = null;

    for (int i = 0; i < numberOfPlaces; i++) {
        try {
            place = getPlace((JSONObject) jsonArray.get(i));
            listOfPlaces.add(place);
        } catch (JSONException e) {
            e.printStackTrace();
        }
    }
    return listOfPlaces;
 }
```

Kôd 11. Kod za dohvaćanje liste znamenitosti

U kodu 11 definira se broj znamenitosti pomoću duljine JSON polja. Zatim se definira lista koja sadrži mapu ključeva i vrijednosti pod kojima su znamenitosti spremljene. Unutar `for` petlje dohvaća se znamenitost metodom `getPlace()` i dodaje se na listu.

```
public List<HashMap<String, String>> parse(String data){

   JSONArray jsonArray = null;
   JSONObject jsonObject;

    try {
        jsonObject = new JSONObject(data);
```

```
        jsonArray = jsonObject.getJSONArray("results");
    } catch (JSONException e) {
        e.printStackTrace();
    }
    return getPlaces(jsonArray);
}
```

Kôd 12. Kod za dohvaćanje rezultata iz JSON formata

U kodu 12 dohvaća se JSON polje pod nazivom rezultati (engl. *results*) koji sadrži indexe rezultata počevši od 0.

Klasa `NearbyPlaces` nasljeđuje klasu `AsyncTask` i unutar nje definira se tri tipa objekta. S tim objektima određuje se koju vrstu parametara pozadinski zadatak prima na početku izvršavanja, zatim kod ažuriranja stanja i nakon izvršavanja zadatka. Klasa prilikom nasljeđivanja implementira dvije metode `doInBackground()` i `onPostExecute()`.

```
@Override
protected String doInBackground(Object... objects) {

    try {
        mMap = (GoogleMap) objects[0];
        url = (String) objects[1];

        PlacesUrl placesUrl = new PlacesUrl();
        googlePlaces = placesUrl.readUrl(url);
    } catch (IOException e) {
        e.printStackTrace();
    }
    return googlePlaces;
}
```

Kôd 13. Kod za dohvaćanje URL-a kako bi se pristupilo podacima

U kodu 13 metoda `doInBackground()` prima polje parametara koji je prvi naveden kod definicije klase. Polje sadrži 2 indeksa, na mjestu 0 je mapa, a na mjestu 1 je proslijeđeni

URL. Ova metoda će se pokrenuti u trenutku kada je kreiran objekt pozadinskog zadatka i na njemu pozvana metoda `execute()` u Maps Activityu. Za zadatke čije izvršavanje dugo traje koristi se `AsyncTask`. Omogućuje izvršavanje nekog zadatka u pozadini bez manipuliranja dretvama. Da nema `AsyncTaska`, došlo bi do zastoja na sučelju i zadatak ne bi mogao biti obavljen.

```
@Override
protected void onPostExecute(String s) {
    List<HashMap<String, String>> nearPlacesList = null;
    DataParser dataParser = new DataParser();
    nearPlacesList = dataParser.parse(s);
    showNearbyPlaces(nearPlacesList);
}
```

Kôd 14. Kod za izvršeni zadatak

U kodu 14 na objektu klase `DataParser` pozove se metoda `parse()` koja vraća listu objekata iz JSON formata pozivanjem metode `getJSONArray()`. Zatim poziva se metoda `showNearbyPlaces()` kojoj se kao parametar prosljeđuje lista znamenitosti. Metoda `onPostExecute()` pokreće se kada se izvrši metoda `doInBackground()`. Obavještava kada je izvršavanje zadatka završeno.

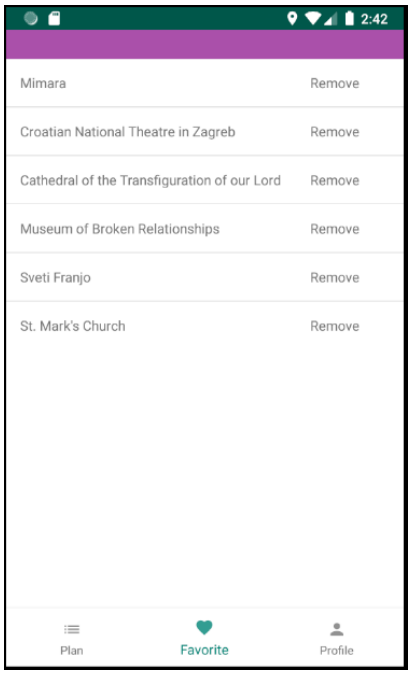
```
private String getUrl(double latitude, double longitude, String place){

  StringBuilder placeUrl = new StringBuilder("https://maps.googleapis.com
                           /maps/api/place/nearbysearch/json?");

    placeUrl.append("location="+latitude+","+longitude);
    placeUrl.append("&radius=" + radius);
    placeUrl.append("&type=" + place);
    placeUrl.append("&sensor=true");
    placeUrl.append("&key=" + "AIzaSyDLQp38T0bLfsrqmxipJDo-BFbvJiWHek0");

    return placeUrl.toString();
}
```

```
private void getTouristAttractions() {
    btnAttractions.setOnClickListener(new View.OnClickListener() {
        String tourist_attraction = "tourist_attraction";
        @Override
        public void onClick(View v) {

            Object dataPlaces[] = new Object[2];
            NearbyPlaces nearbyPlaces = new NearbyPlaces();


            String url = getUrl(currentLocation.getLatitude(),
            currentLocation.getLongitude(), tourist_attraction);

            dataPlaces[0] = mMap;
            dataPlaces[1] = url;

            nearbyPlaces.execute(dataPlaces);

            Toast.makeText(MapsActivity.this,"Tourist attractions
            displayed", Toast.LENGTH_LONG).show();

              }
          });
      }
```

Kôd 15. Kod za prikaz obližnjih znamenitosti pritiskom na gumb

U kodu 15 pritiskom na gumb za prikaz obližnjih znamenitosti, definira se polje objekata koje je definirano i metodom `doInBackground()`, URL koji metodom `getUrl()` prima trenutne koordinate i tip obližnjih mjesta koji se traži, u ovom slučaju znamenitosti (engl. *tourist_attraction*). Na kreiranom objektu pozadinskog zadataka, `NearbyPlaces` klase, pozove se metoda `execute()` kojom se pokreće metoda `doInBackground()` i preda joj se polje objekata.

4.4. Lista željenih znamenitosti



Slika 10: Ekran za prikaz željenih znamenitosti

Da bi korisnik mogao dodati znamenitost na listu željenih, potrebno je pritisnuti plavu oznaku kojom je obilježena znamenitost i tada se dodaje znamenitost na listu. Znamenitosti su prikazane pomoću liste stavaka (engl. *ListItem*). Budući da su liste željenih znamenitosti trajno pohranjene, onda se spremaju u SQLite bazu podataka pod identifikacijskom oznakom korisnika. Svaki korisnik ima pristup svojoj željenoj listi. Lista stavaka u Androidu funkcionira preko adaptera. Ona postavlja adapter. Da bi adapter bio postavljen potrebno je:

- definirati XML koji prikazuje kako bi lista izgledala, a adapter ga kasnije prikazuje (engl. *inflate*)
- definirati klasu `FavoriteItem`
- definirati klasu `FavoriteItemAdapter` koja nasljeđuje `ArrayAdapter` tipa `FavoriteItem`
- postavljanje adaptera

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
```

```
    android:layout_height="match_parent"
    android:padding="15dp">

    <TextView
        android:id="@+id/tvItem"
        android:layout_width="300dp"
        android:layout_height="wrap_content"/>

    <TextView
        android:id="@+id/tvRemove"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Remove"/>
</LinearLayout>
```

Kôd 16. Kod za definiranje XML-a

U kodu 16 opisano je kako izgleda prikaz znamenitosti na listi kada korisnik stisne na oznaku za dodavanje na listu željenih znamenitosti. Prikazan je naziv znamenitosti i oznaka za uklanjanje znamenitosti ukoliko korisnik ne želi više tu znamenitost na listi.

```
public class FavoriteItem {

    private String title;

    public FavoriteItem(String title) {
        this.title = title;
    }

    public String getTitle() {
        return title;
    }

    public void setTitle(String title) {
        this.title = title;
    }
}
```

Kôd 17. Kod klase `FavoriteItem`

U kodu 17 opisana je klasa `FavoriteItem`. Potrebno je definirati naziv znamenitosti, kreirati konstruktor i postaviti `get()` i `set()` metode.

```
@NonNull
@Override
public View getView(final int position, @Nullable View convertView,
                    @NonNull ViewGroup parent) {

    if(convertView == null){

        convertView = layoutInflater.inflate(R.layout.favoriteitem,
                                             parent, false);
    }

    initDatabase();

    final FavoriteItem favoriteItem = getItem(position);

    TextView tvItem = convertView.findViewById(R.id.tvItem);

    TextView tvRemove = convertView.findViewById(R.id.tvRemove);

    tvItem.setText(favoriteItem.getTitle());

    tvRemove.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {

            FavoriteItem item = items.get(position);

            items.remove(item);

            deleteFavoritePlace();

            notifyDataSetChanged();

            Toast.makeText(context, "Favorite place deleted",
                           Toast.LENGTH_LONG).show();
        }
    });
```

```
    return convertView;
}
```

Kôd 18. Kod klase `FavoriteItemAdapter`

U kodu 18 opisano je nasljeđivanje `ArrayAdaptera`, postavljanje naziva znamenitosti i mogućnosti uklanjanja s liste. Nasljeđivanjem klasa mora implementirati konstruktor i metodu `getView()` koja napuhuje (engl. *inflate*) XML. Zatim pomoću metode `findViewById()` pronalazi se ID naziva znamenitosti i ID za uklanjanje. Svi ID-evi pospremljeni su u `R` klasu. Ako korisnik odabere ukloniti znamenitost s liste, lista se ažurira i znamenitost je obrisana u bazi pod identifikacijskom oznakom korisnika. Korisnika se obavijesti kada je znamenitost obrisana.

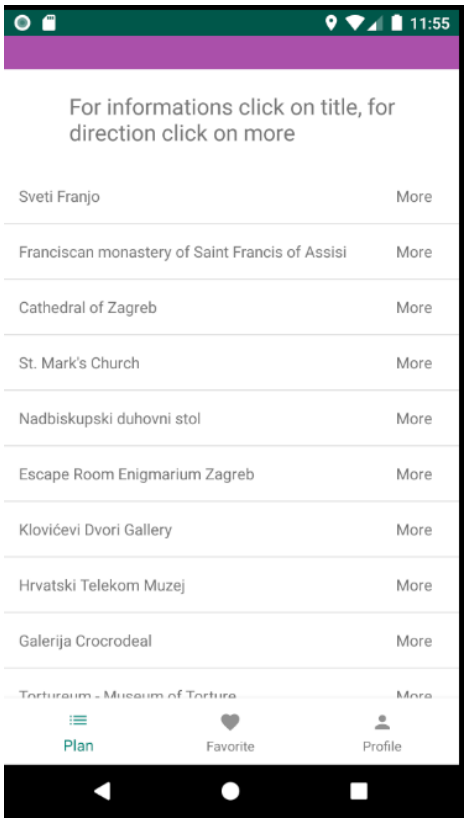
```
FirebaseUser user = FirebaseAuth.getInstance().getCurrentUser();

String userID = user.getUid();

SQLiteDatabase db = favoritesDatabaseHelper.getReadableDatabase();

String[] columns = {FavoritesDatabaseHelper.COLUMN_ID,
                    FavoritesDatabaseHelper.COLUMN_USERID,
                    FavoritesDatabaseHelper.COLUMN_NAME};

String selection = FavoritesDatabaseHelper.COLUMN_USERID + "=?";

String[] selectionArgs = {userID};

Cursor cursor = db.query(FavoritesDatabaseHelper.TABLE_NAME, columns,
                         selection, selectionArgs, null, null, null);

List<String> titles = new ArrayList<>();

while (cursor.moveToNext()) {

    title = cursor.getString(cursor.getColumnIndexOrThrow
                             (FavoritesDatabaseHelper.COLUMN_NAME));
```

```
    titles.add(title);
}

adapter = new FavoriteItemAdapter(getActivity(),
android.R.layout.simple_list_item_1, new ArrayList<FavoriteItem>());

for (int i = 0; i < titles.size(); i++) {

    favoriteItem = new FavoriteItem(titles.get(i));

    adapter.add(favoriteItem);
}

lvItems.setAdapter(adapter);
```

Kôd 19. Kod za postavljanje adaptera

U kodu 19 opisano je dohvaćanje identifikacijske oznake korisnika, čitanje naziva željenih znamenitosti iz baze podataka, dodavanje na listu, prosljeđivanje naziva u konstruktor, dodavanje objekta na adapter unutar `for` petlje i postavljanje adaptera na listu stavaka. Čitanje znamenitosti odvija se pomoću klase `Cursor` koja preko `SqliteDatabase` klase izvršava upit kojem se prosljeđuje naziv tablice, kolone i eventualno uvjeti ako su potrebni. U `while` petlji čita se naziv znamenitosti na koju se pritisnulo te se naziv dodaje na listu. Zatim kreira se adapter i unutar `for` petlje koja ide po listi prosljeđuje se u konstruktor naziv i dodaje se objekt na adapter. Na kraju adapter je postavljen na listu stavaka tj. znamenitosti.

4.5. Plan obilaska znamenitosti



Slika 11: Ekran za prikaz plana obilaska znamenitosti

Prikaz plana obilaska znamenitosti realizira se na sličan način kao i prikaz željenih znamenitosti. Potreban je adapter. Proces kreiranja adaptera je isti kao i za željene znamenitosti. Potreban je XML, klasa koja nasljeđuje `ArrayAdapter` te je potrebno implementirati metode koje su potrebne prilikom nasljeđivanja i prikaz postavljanja adaptera.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:padding="15dp">

    <TextView
        android:id="@+id/tvTitlePlace"
        android:layout_width="match_parent"
```

```
        android:layout_height="wrap_content"/>
</LinearLayout>
```

Kôd 20. plan.xml

U kodu 20 opisano je definiranje pojedine znamenitosti na listi. Kao što je navedeno u XML-u na listi je prikazan naziv znamenitosti.

Kada je kreiran XML potrebno je kreirati klasu koja će nasljeđivati `ArrayAdapter` koji definira sami adapter. Ta klasa je zadužena za prikaz XML-a.

```
@NonNull
    @Override
    public View getView(int position, @Nullable View convertView,@NonNull
                        ViewGroup parent) {

        if(convertView == null){
            convertView = layoutInflater.inflate(R.layout.plan, parent,
                                                 false);
        }

        final NearbyPlace nearbyPlace = getItem(position);

        TextView tvTitlePlace = convertView.findViewById
                                    (R.id.tvTitlePlace);

        tvTitlePlace.setText(nearbyPlace.getName());

        tvTitlePlace.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {

                Fragment fragment = new PlaceInformationFragment();
                Bundle bundle = new Bundle();
                bundle.putString("title", nearbyPlace.getName());
                fragment.setArguments(bundle);

                ((FragmentActivity)v.getContext())
                  .getSupportFragmentManager()
                  .beginTransaction()
```

```
                .replace(R.id.frameContainer,fragment)
                .commit();
        }
    });

    return convertView;
  }
}
```

Kôd 21. Adapter za prikaz plana obilaska znamenitosti

U kodu 21 opisana je metoda `getView()` koja prikazuje dani XML. Zatim se uzima pozicija svake znamenitosti i za svaku se pridodaje odgovarajući naziv. Pritiskom na naziv znamenitosti prikazuje se novi dio ekrana gdje su ponuđene informacije kojima se može pristupiti. Da bi se prikazale informacije za određenu znamenitost, potrebno je proslijediti naziv znamenitosti drugom fragmentu pomoću klase `Bundle` koja prima ključ pod kojim će biti postavljena vrijednost. Detaljnije o prikazu informacija u sljedećem poglavlju.

```
    Cursor cursor = db.query(NearbyDatabaseHelper.TABLE_NAME,
                             columns,selection, selectionArgs, null,
                             null, null);
    List<NearbyPlace> nearbyPlaces = new ArrayList<>();
    List<String> sortTitles = new ArrayList<>();
    List<LatLng> coordinates = new ArrayList<>();


    while (cursor.moveToNext()){

        String name  = cursor.getString(cursor.getColumnIndexOrThrow
                         (NearbyDatabaseHelper.COLUMN_NAME));
        String latitude = cursor.getString(cursor
                         .getColumnIndexOrThrow
                         (NearbyDatabaseHelper.COLUMN_LATITUDE));
        String longitude = cursor.getString(cursor
                         .getColumnIndexOrThrow
                         (NearbyDatabaseHelper.COLUMN_LONGITUDE));

        NearbyPlace nearbyPlace = new NearbyPlace(name,
```

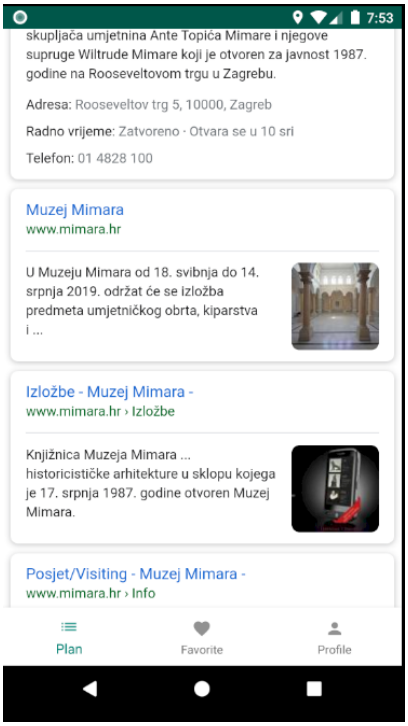
```
    Double.parseDouble(latitude), Double.parseDouble(longitude));
    nearbyPlaces.add(nearbyPlace);
}

adapter = new NearbyPlaceAdapter(getActivity(),
          android.R.layout.simple_list_item_1,
          new ArrayList<NearbyPlace>());

Collections.sort(nearbyPlaces, new Comparator<NearbyPlace>() {
    @Override
    public int compare(NearbyPlace o1, NearbyPlace o2) {
        Double d1 = Math.hypot(o1.getLatitude(),
                               o1.getLongitude());
        Double d2 = Math.hypot(o2.getLatitude(),
                               o2.getLongitude());
        return -d1.compareTo(d2);
    }
});

for (int i = 0; i < nearbyPlaces.size(); i++) {

    String namePlace = nearbyPlaces.get(i).getName();
    sortTitles.add(namePlace);
}

for (int i = 0; i < sortTitles.size(); i++) {

   place = new NearbyPlace(sortTitles.get(i));
   adapter.add(place);
}

lvPlan.setAdapter(adapter);
```

Kôd 22. Prikaz plana obilaska znamenitosti

U kodu 22 opisano je dohvaćanje podataka o znamenitostima iz SQLite baze podataka pod određenom identifikacijskom oznakom korisnika. Pomoću klase `Cursor` šalje se upit na određenu tablicu, s definiranim kolonama i uvjetima. U `while` petlji slijedi čitanje naziva znamenitosti i njihovih pripadajućih koordinata. Kreira se u međuvremenu novi objekt tj. nova znamenitost s pripadajućim nazivom i koordinatama. Nakon toga definira se adapter.

Zatim potrebno je sortirati listu znamenitosti od najbliže do najdalje koordinate pomoću metode `compare()` koja vraća razliku dviju koordinata. Kada je lista sortirana, u `for` petlji dodaju se sortirani nazivi na listu i u sortiranoj listi kreiranom objektu se proslijedi naziv. Objekt se dodaje na adapter, a na listu stavaka postavlja se adapter.

4.5.1. Informacije o znamenitosti



Slika 12: Ekran za prikaz informacija o znamenitosti

Za prikaz informacija o određenoj znamenitosti potrebno je iščitati vrijednost koja je proslijeđena pomoću klase `Bundle`. Zatim vrijednost je proslijeđena u URL koji se učitava pomoću klase `WebView`.

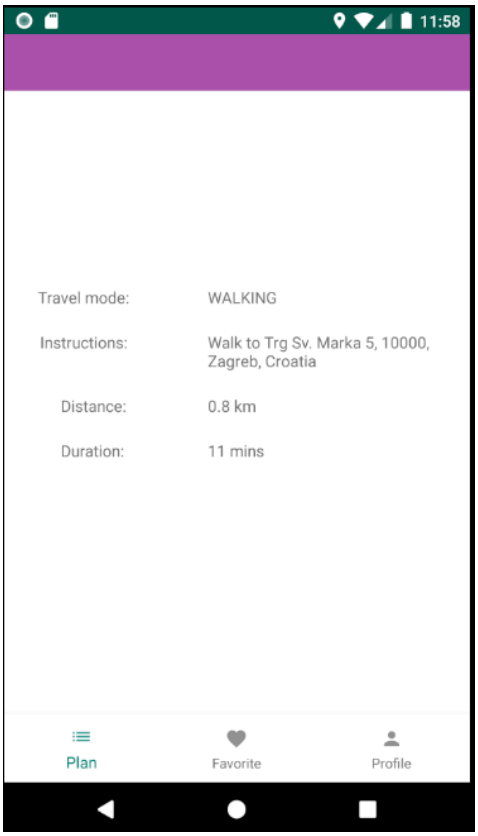
```
@Override
    public View onCreateView(LayoutInflater inflater, ViewGroup
    container, Bundle savedInstanceState) {

       View view = inflater.inflate(R.layout.fragment_place_information,
                                    container, false);
```

```
        String value = getArguments().getString("title");

        webView = view.findViewById(R.id.webView);
        webView.setWebViewClient(new WebViewClient());


        try {
            String data = "https://www.google.com/search?q=" +
                          URLEncoder.encode(value, "UTF-8");
            webView.loadUrl(data);
        } catch (UnsupportedEncodingException e) {
            e.printStackTrace();
        }

        return view;
    }
```

Kôd 23. Prikaz informacija o određenoj znamenitosti

U kodu 23 opisano je dohvaćanje vrijednosti pomoću metode `getArguments()` koja dohvaća vrijednost preko ključa. Zatim potrebno je omogućiti komunikaciju za slanje zahtjeva i dobivanje odgovora metodom `setWebViewClient()`. Unutar bloka definira se varijabla tekstualnog tipa kojoj se prosljeđuje upit i klasa `URLEncoder` koja prima pročitanu vrijednost preko ključa i shemu kodiranja UTF-8 ukoliko postoje nesigurni znakovi.

4.5.2. Upute do određene znamenitosti



Slika 13: Ekran za prikaz dodatnih informacija o dolasku do određene znamenitosti

Kada je riječ o načinu obilaska znamenitosti potreban je Directions API. Za ovu aplikaciju potrebna je klasa `DirectionsTask` koja nasljeđuje već spomenuti `AsyncTask`.

```
@Override
protected String doInBackground(Object... objects) {

try {

      url = (String)objects[0];

      PlacesUrl placesUrl = new PlacesUrl();
      data = placesUrl.readUrl(url);
 }
 catch (Exception e){
      e.printStackTrace();
 }
```

```
        return data;
}
```

Kôd 24. Dohvaćanje podataka za dani URL

U kodu 24 opisana je metoda `doInBackground()` koja je već prije opisana. Način na koji se poziva je isti kao i za prikaz znamenitosti, tako da se na kreiranom objektu klase pozove metoda `execute()` koja zove metodu `doInBackground()`. Metoda vraća podatke u JSON formatu.

```
@Override
protected void onPostExecute(String s) {

    String travel = "";
    String instructions = "";
    String distance = "";
    String duration = "";
    String startLatitude = "";
    String startLongitude = "";
    String endLatitude = "";
    String endLongitude = "";

    try {
        JSONObject jsonObject = new JSONObject(s);
        JSONArray jsonArray = jsonObject.getJSONArray("routes")
                              .getJSONObject(0).getJSONArray("legs")
                              .getJSONObject(0).getJSONArray("steps");

        int arraySize = jsonArray.length();

        for (int i = 0; i < arraySize; i++) {
            JSONObject jsonObjectFor = jsonArray.getJSONObject(0);

            travel = jsonObjectFor.getString("travel_mode");
            instructions = jsonObjectFor.getString("html_instructions");
            distance = jsonObjectFor.getJSONObject("distance")
                       .getString("text");
            duration = jsonObjectFor.getJSONObject("duration")
```

```
                    .getString("text");
            startLatitude = jsonObjectFor.getJSONObject("start_location")
                            .getString("lat");
            startLongitude = jsonObjectFor.getJSONObject
                            ("start_location").getString("lng");
            endLatitude = jsonObjectFor.getJSONObject("end_location")
                           .getString("lat");
            endLongitude = jsonObjectFor.getJSONObject("end_location")
                            .getString("lng");

            MyUtils.saveTransport(startLatitude, startLongitude,
                                  endLatitude, endLongitude, travel,
                                  instructions, distance, duration);

           iDirections.update();
        }

    } catch (JSONException e) {
            e.printStackTrace();
    }
```

Kôd 25. Dohvaćanje vrijednosti iz JSON-a

U kodu 25 opisana je metoda `onPostExecute()` koja kao rezultat uspješno obavljenog zadatka tj. vraćanja podataka, dohvaća vrijednosti. U ovom slučaju potreban je način dolaska tj. potrebna su prijevozna sredstva. U navedenom kodu postoji nekoliko čvorova tj. polja preko kojih se dolazi s nultim indeksom. Dobiveni JSON sadrži podatke sumarno od jedne do druge točke ili analizira pojedine segmente. Unutar `for` petlje uzimaju se podaci sumarno od jedne do druge točke. Unutar JSON-a navedeni su podaci o vremenu polaska i dolaska, trajanju u minutama, udaljenosti, instrukcijama do pojedine točke, predloženom načinu dolaska do određene točke itd. Unutar metode pozvana je metoda `saveTransport()` statičke klase `MyUtils`. Budući da treba vremena dok Directions API dohvati potrebne podatke od jedne do druge točke, da bi podaci mogli biti dohvaćeni potreban je *interface*. *Interface* sadrži metodu `update()` čija je zadaća dohvatiti listu pospremljenih podataka tako da se *interface* implementira na fragment i unutar metode napravi lista i pozove statička metoda `getDirections()` koja vraća listu.

```
for (int i = 0; i < coordinates.size(); i++) {

        Object data[] = new Object[1];

     if (i + 1 < coordinates.size()) {

        String url = getUrl(coordinates.get(i), coordinates.get(i + 1));

        data[0] = url;

        DirectionsTask directionsTask = new DirectionsTask(this);
        directionsTask.execute(data);
}
      else{
         break;
      }
}

private String getUrl(LatLng origin, LatLng destination) {

       StringBuilder sb = new StringBuilder("https://maps.googleapis.com/
                              maps/api/directions/json?");
       sb.append("origin="+origin.latitude+","+origin.longitude);
                sb.append("&destination=" + destination.latitude + ","
                           + destination.longitude);
       sb.append("&mode=transit");
       sb.append("&key=" + "AIzaSyDLQp38T0bLfsrqmxipJDo-BFbvJiWHek0");

       return sb.toString();
 }
```

Kôd 26. Prosljeđivanje parametara URL-u i izvršavanje poziva metode

U kodu 26 opisano je prosljeđivanje parametara unutar URL-a. Potrebni parametri su: početna koordinata, krajnja koordinata, parametar koji definira način dolaska i ključ koji se odnosi na Directions API. Budući da su znamenitosti unutar liste i da se način dolaska temelji od jedne do druge točke, poziv se odrađuje unutar `for` petlje gdje između svake znamenitosti postoji određena udaljenost, trajanje, informacije o kretanju do druge tj. krajnje

točke itd. Na kreiranom objektu pozove se metoda `execute()` koja pokreće metodu `doInBackground()`.

```
private void readDirections() {

        for (int i = 0; i < directions.size(); i++) {

            tvTravelMode.setText(directions.get(i + position)
                        .getTransport());
            tvHtmlInstructions.setText(directions.get(i + position)
                              .getInstructions());
            tvDistance.setText(directions.get(i + position)
                      .getDistance());
            tvDuration.setText(directions.get(i + position)
                      .getDuration());
            break;
        }
}
```

Kôd 27. Prikaz informacija o dolasku do određene znamenitosti

U kodu 27 opisana je metoda `readDirections()` koja sadrži `for` petlju s kojom se prolazi po statičkoj listi. Unutar pregleda teksta (engl. *TextView*) postavlja se vrijednost na indeksu od i, uvećana za poziciju pojedine znamenitosti na listi. Da bi informacije odgovarale za određenu znamenitost, fragmentu je potrebno proslijediti poziciju (engl. *position*) pojedine znamenitosti pritiskom na pregled teksta „Više“ (engl. „More“) plana obilaska.

5. Sigurnosni aspekti aplikacije

Firebase je baza podataka izravno dostupna s bilo kojeg povezanog klijenta. Budući da bilo koji klijent može biti spojen s Firebaseom, potrebno je napisati sigurnosna pravila (engl. *security rules*) kako bi podaci bili osigurani. Sigurnosna pravila moraju biti ispravno napisana, svaki neuspjeh potencira opasnost.

```
{
 "rules":
  {
    ".read": true,
    ".write": false
  }
}
```

Kôd 28. Definiranje sigurnosnih pravila

U kodu 28 opisan je postupak definiranja sigurnosnih pravila. Korijen JSON objekta nazvan je „pravila“ (engl. *„rules“*). Primjer iznad omogućava čitanje (engl. *read*) i onemogućava pisanje (engl. *write*). U tom primjeru pravila se odnose na sve čvorove djece (engl. *child nodes*) te svatko izvana ima pristup čitanju i pisanju. Pravila bi se trebala odnositi na pojedinog korisnika temeljeno na ID-u korisnika (engl. *uid*). [15]

```
{
  "rules": {
    "some_path": {
      "$uid": {
        ".read": "request.auth.uid == uid"
        ".write": "request.auth.uid == uid"
      }
    }
  }
}
```

Kôd 29. Sigurnosna pravila prema ID-u korisnika[16]

[15] https://firebase.google.com/docs/rules/basics, dokumentacija unutar koje su definirana sigurnosna pravila
[16] https://firebase.google.com/docs/rules/basics, sigurnosna pravila temeljena na ID-u korisnika

U kodu 29 opisano je definiranje pravila za pojedinog korisnika prema ID-u. Čitanje i pisanje obično su definirani uvjetom. U ovom primjeru čitanje i pisanje je uvjetovano ID-em korisnika. Znači, kada korisnik bude provjeren autentikacijom Firebasea, korisniku se pridružuje ID uz pomoć kojeg se provjerava je li jednak onome koji je pospremljen. Ako se ID preklapa, korisniku se dodjeljuje pravilo čitanja i pisanja. Također čitanje i pisanje može biti uvjetovano AND, OR operatorima.

```
{
  "rules": {
    "some_path": {
      "${subpath}": {
        ".write":"root.child('users').child(auth.uid)
                      .child('role').val() == 'admin'",
        ".read": true
      }
    }
  }
}
```

Kôd 30. Pridruživanje pravila pisanja adminu[17]

U kodu 30 opisano je pridruživanje pravila pisanja adminu. Nekome od korisnika može biti pridružena uloga admina te samo adminu dati dopuštenje čitanja ili pisanja. Da bi admin imao mogućnost čitanja ili pisanja, potrebno je na korijenu pozvati čvor korisnika preko kojeg pomoću djeteta dolazi se do ID-a, zatim do uloge admin na kojemu se pozove metoda `val()` kako bi se pročitala vrijednost.

Svaki korisnik na Firebaseu ima svoj korisnički račun. Firebase Autentikacija (engl. *Firebase Authentication*) koristi posebnu verziju skripte za sažete (engl. *hash*) lozinke korisničkih računa. Svaka lozinka pospremljena je u zaseban format slučajno odabranih znakova. Firebase generira jedinstvene parametre za postavljanje lozinki u zaseban format. Da bi se pristupilo parametrima, potrebno je otići na karticu „Korisnici“ (engl. *„Users“*) u odjeljku „Autentikacija“ (engl. *„Authentication“*) na Firebase konzoli odabrati „Parametre hash lozinke“ s padajućeg izbornika u gornjem desnom kutu tablice. Skripta je interpretirana

[17] https://firebase.google.com/docs/rules/basics, sigurnosna pravila temeljena na ulozi admina

kao alat za šifriranje na temelju lozinke. Skripta sadržava: ključ (engl. *key*), dodatni slučajni znakovi (engl. *salt*), parametar zaokruživanja (engl. *rounds*), parametar memcost (engl. *memcost*), „parametar -P“. Funkcionalnost skripte temelji se na tome da zatraži običnu tekstualnu lozinku i iznese u drugom nizu slučajnih znakova (engl. *hash*). Dobiveni niz kodira se pomoću Base64 algoritma. [18]

Base64 naziv je za skupinu sličnih principa kodiranja binarnog teksta u tekst koji predstavljaju binarne podatke u ASCII formatu pretvarajući ih u zaseban prikaz. Algoritam je reverzibilan što omogućuje kodiranje, ali i dekodiranje podataka tj. vraćanje podataka u originalan format. [19]

[18] https://firebaseopensource.com/projects/firebase/scrypt/, pretvaranje lozinke u niz slučajnih znakova
[19] https://developer.mozilla.org/en-US/docs/Web/API/WindowBase64/Base64_encoding_and_decoding, značajke Base64 i opis principa kodiranja

Zaključak

Korisnik prijavom u aplikaciju pristupa mapi. Uz mapu su prisutne i druge mogućnosti: pretraživač, gumb za prikaz znamenitosti uz određeni radijus te traka koja sadrži kartice plana obilaska, željene znamenitosti i profil korisnika. Odabirom plana korisnik ima uvid u obilazak od najbliže znamenitosti do najdalje znamenitosti te pritiskom na naziv pojedine znamenitosti korisniku su dostupne informacije o toj znamenitosti. Pritiskom na "Više" korisnik ima dostupan način dolaska do znamenitosti prikazom prijevoza, udaljenosti, trajanja i instrukcija.

Popis kratica

AOT	*Ahead of Time compilation*	kompajliranje u izvorni strojni kod
API	*Application programming interface*	specifikacije koje omogućuju usluge
ART	*Android Run Time*	okružje za izvršavanje .dex datoteka
ASCII	*American Standard Code for Information Interchange*	kodiranje znakova temeljeno na engleskoj abecedi
GC	*Garbage Collector*	automatsko upravljanje memorijom
GPS	*Global Positioning System*	sustav za određivanje lokacije
ID	*Identification*	identifikacija
JSON	*JavaScript Object Notation*	datoteka, format podataka koji je čitljiv
SDK	*Software Development Kit*	oprema za stvaranje izvršnog softvera
URL	*Uniform Resource Locator*	putanja do sadržaja na Internetu
XML	*Extensible Markup Language*	jezik koji sadržaj prikazuje oznakama

Popis slika

Popis kôdova

Literatura

[1] J.Božić, T.Kadežabek, F.Tomić, T.Kaštelan, Izrada aplikacija za mobilne uređaje, Zagreb: Algebra, 2013., osnove Android operacijskog sustava, arhitektura Androida

[2] M.Karaga, M.Stojanović, Programiranje aplikacija za Android, Zagreb: Element, 2018., uvod u Android operacijski sustav, značajke Dalvika, ART-a i Gradlea

[3] http://www.techplayon.com/wp-content/uploads/2017/06/architec.png , slika arhitekture Androida u razinama, 15.11. 2019.

[4] https://developer.android.com/, dokumentacija s opisom trenutne lokacije, 15.11. 2019.

[5] https://firebase.google.com/docs/android/setup, preduvjeti potrebni za integraciju Firebasea s Android aplikacijom, 21.11. 2019.

[6] https://www.androidhive.info/2016/10/android-working-with-firebase-realtime-database/, uvod u Firebase, 21.11. 2019.

[7] https://firebase.google.com/docs, dokumentacija unutar koje su opisane usluge Firebasea, 22.11. 2019.

[8] https://firebase.google.com/docs/auth, dokumentacija s opisanom uslugom autentikacije, 22.11. 2019.

[9] https://developer.android.com/reference/android/location/Geocoder.html#getFromLocationName(java.lang.String,%20int), dokumentacija unutar koje su navedene značajke klase `Geocoder`, 23.11. 2019.

[10] https://developers.google.com/places/web-service/intro, dokumentacija s opisom Places API-a, 24.11. 2019., 01.12. 2019.

[11] https://developers.google.com/maps/documentation/directions/intro, dokumentacija s opisom Directions API-a, 15.12. 2019. , 04.01. 2020.

[12] https://developers.google.com/maps/billing-credits, otvaranje korisničkog računa za korištenje Directions API-a, 07.01. 2020.

[13] https://firebase.google.com/docs/rules/basics, dokumentacija unutar koje su opisana sigurnosna pravila, 20.01. 2020.

[14] https://firebaseopensource.com/projects/firebase/scrypt/, opisana skripta s postupkom pretvaranja lozinke u niz slučajnih znakova, 20.01. 2020.

[15] https://developer.mozilla.org/en-US/docs/Web/API/WindowBase64/Base64_encoding_and_decoding, značajke *Base64*, opis principa kodiranja, 21.01. 2020.

ALGEBRA
VISOKO UČILIŠTE

APLIKACIJA ZA INFORMIRANJE O ZNAMENITOSTIMA TEMELJENA NA GEOLOKACIJAMA

Pristupnik: Ana Sovilj, 0321006735

Mentor: prof. Aleksander Radovan