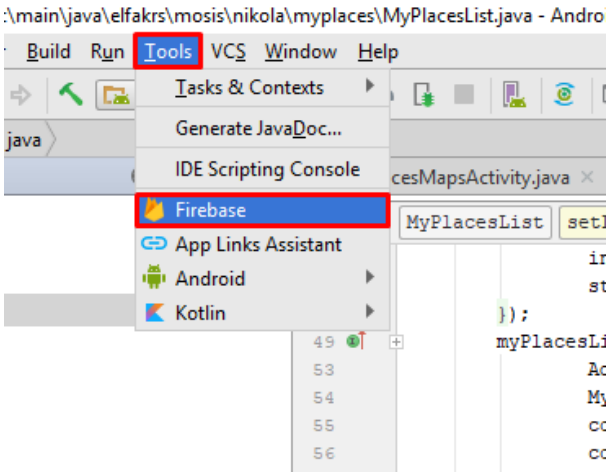
MOSIS - Laboratorijska vežba 5

Nakon što je na prošloj vežbi dodata podrška za prikaz na mapi, potrebno je dodati podršku za perzistenciju dodatih podataka. Za perzistenciju podataka je zgodno koristiti bazu podataka. Moguće je koristiti lokalnu relacionalnu bazu podataka SQLite, koja je ugrađena u Android operativni sistem. Alternativa je perzistiranje podataka na bazi podataka koja se nalazi na serveru i koju je moguće deliti između više instanci aplikacije. Ipak, da bi bilo moguće koristiti bazu podataka koja se nalazi na serveru, neophodno je podržati konekciju na takvu bazu ili napraviti odgovarajući interfejs prema bazi podataka u vidu posebne servis komponente na serveru. Aplikacija koja objedinjuje ovakvu fleksibilnost ali i omogućava asinhrone notifikacije o promenama podataka je Firebase. Firebase je platforma za razvoj mobilnih i Web aplikacija koja omogućava podršku za autentifikaciju, smeštanje podataka i fajlova, bazu podataka... U ovoj vežbi biće korišćena komponenta baza podataka za perzistenciju podataka o omiljenom mestu.

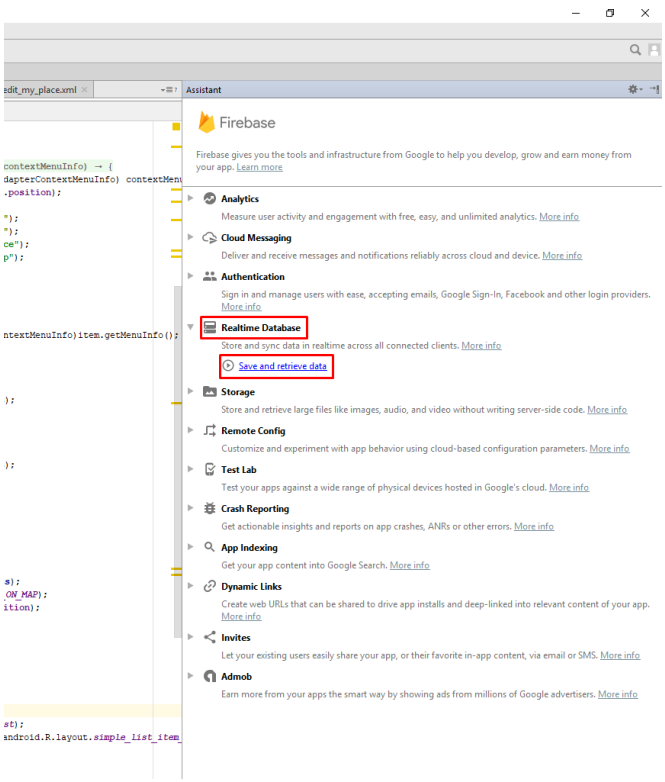
Ciljevi ove vežbe su:

1. Uključivanje podrške za perzistiranje omiljenih mesta na *Firebase* bazi podataka.
2. Upisivanje podataka u *Firebase*.
3. Učitavanje podataka iz *Firebase* baze podataka.

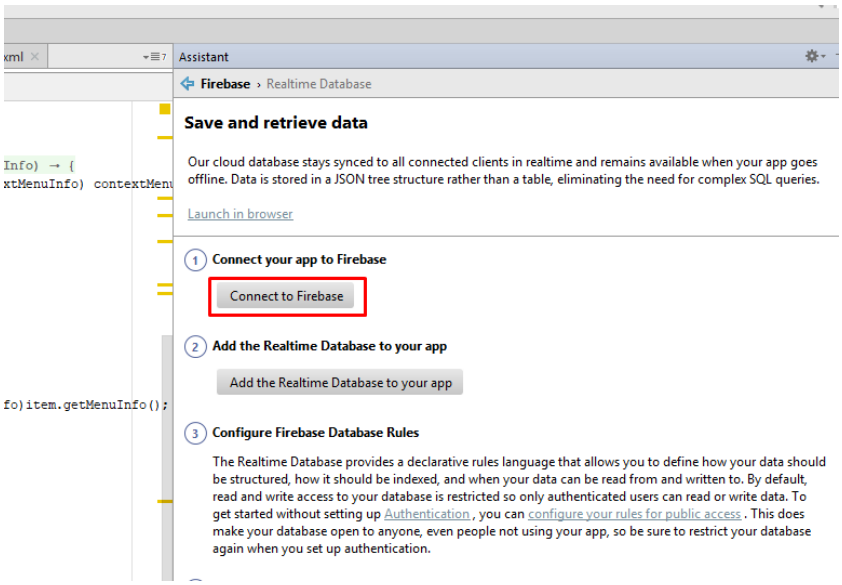
1. Dodavanje Firebase podrške
   a. Počevši od Android Studio razvojnog okruženja verzije 2.2, moguće je koristiti *Firebase Assistant* za lakše dodavanje podrške za *Firebase* u postojeći projekat. Alternativno, ručno povezivanje na Firebase je objašnjeno na ovoj stranici: https://firebase.google.com/docs/android/setup
   b. Kliknuti na *Tools->Firebase*



   c. Nakon što je *Firebase* asistent otvoren, potrebno je proširiti stavku *Realtime database* i kliknuti na dugme *Save and retrieve data*.

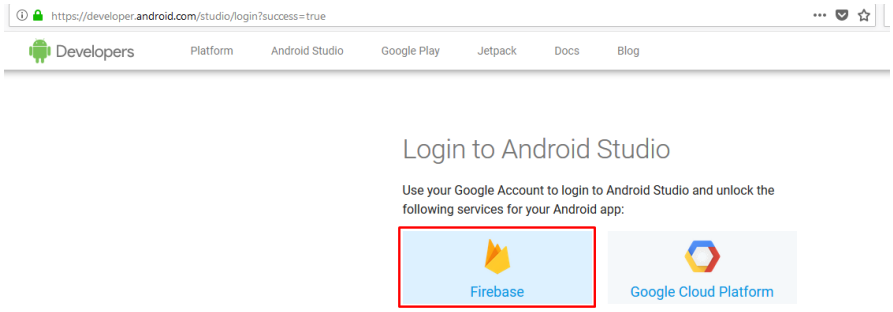


d. Prvi korak je povezivanje na bazu podataka. Kliknuti na dugme Connect to Firebase.

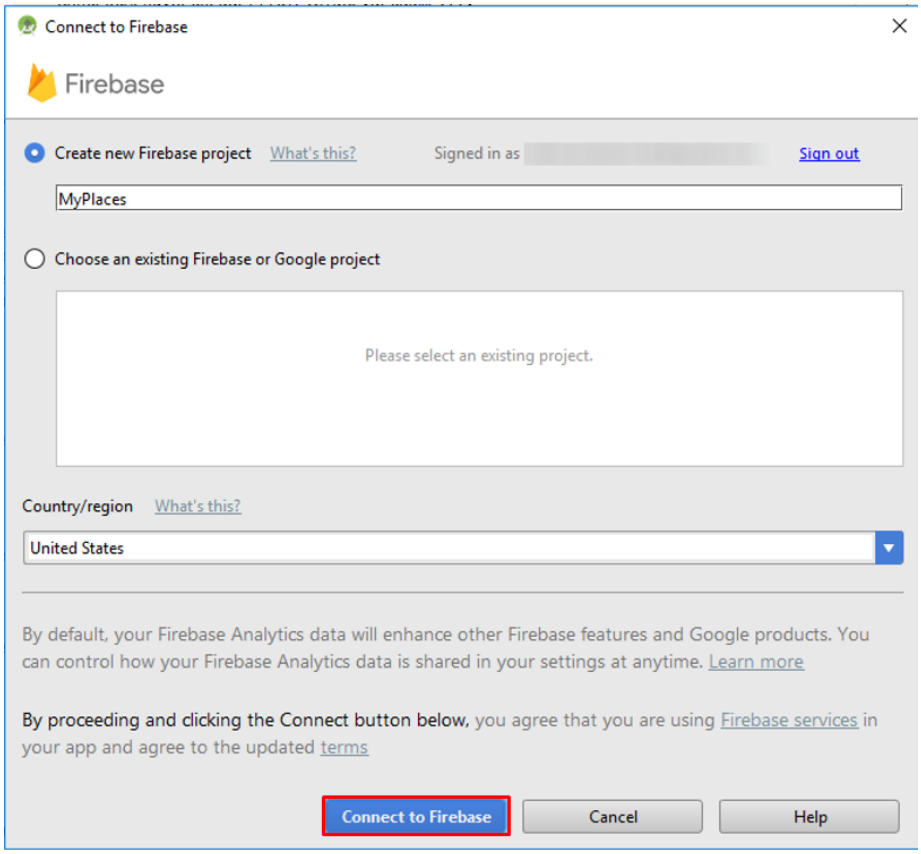


e. Nakon toga je potrebno saglasiti se da Android Studio pristupi korisnikovom google nalogu.

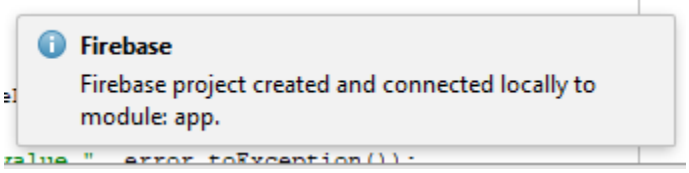
f. Potrebno je i povezati se na Firebase klikom na odgovarajuće dugme:



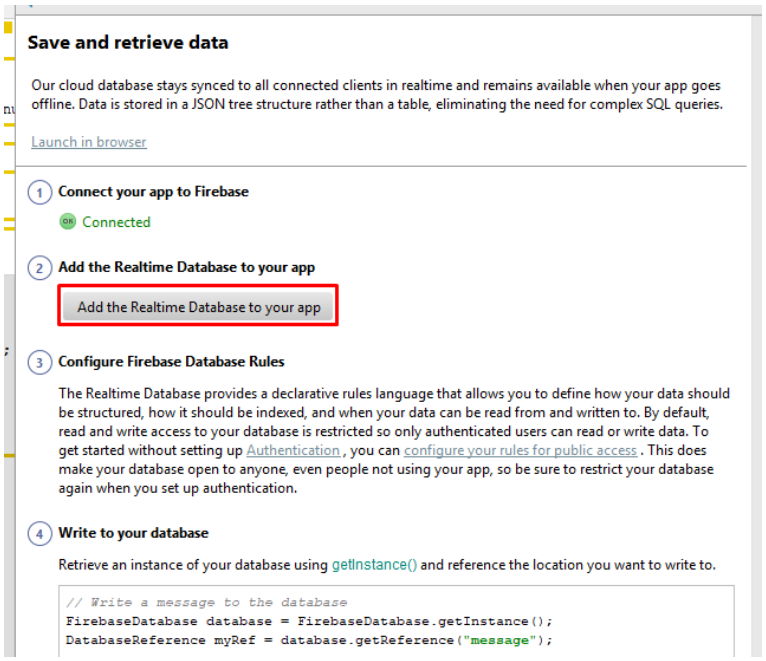
g. U razvojnom okruženju Android Studio, korisniku je omogućeno direktno konektovanje na *Firebase*. Potrebno je kliknuti na dugme *Connect to Firebase*.



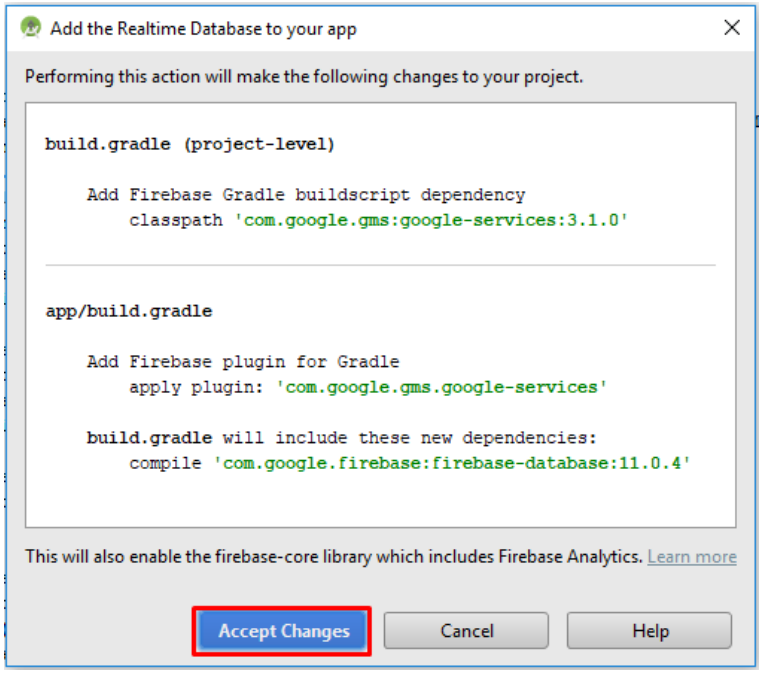
h. Nakon ovoga je Firebase povezan sa projektom.



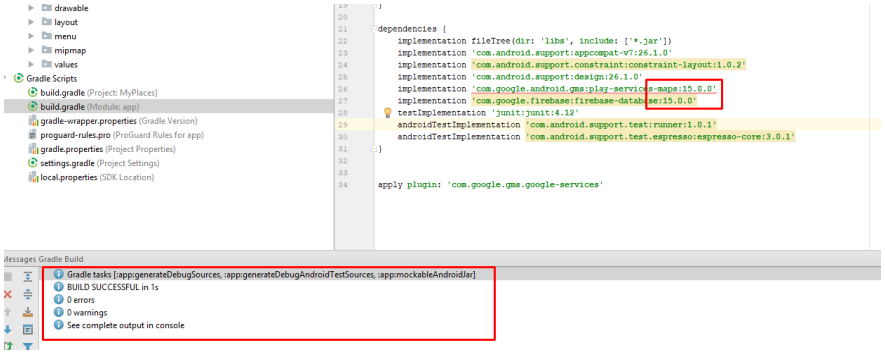
i. Potrebno je povezati *Firebase* bazu podataka sa projektom.



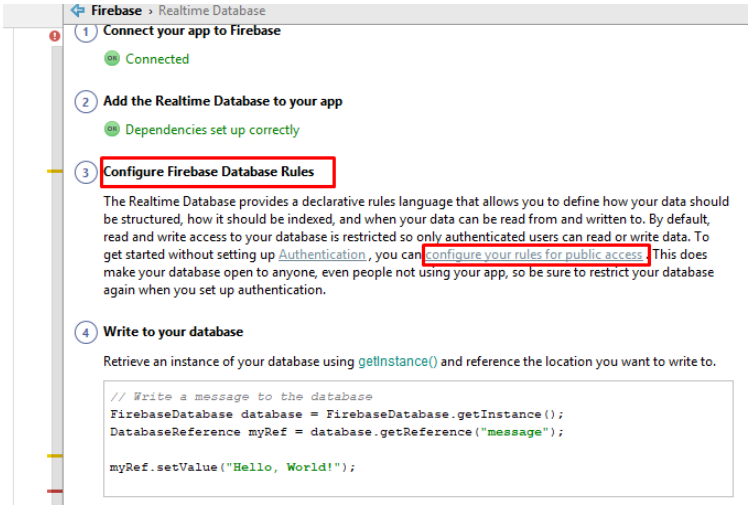
j. Potrebno je saglasiti se sa promenom u build.gradle fajlovima (uočiti da se na sličan način ovi fajlovi menjaju kod ručnog dodavanja *Firebase* baze podataka).



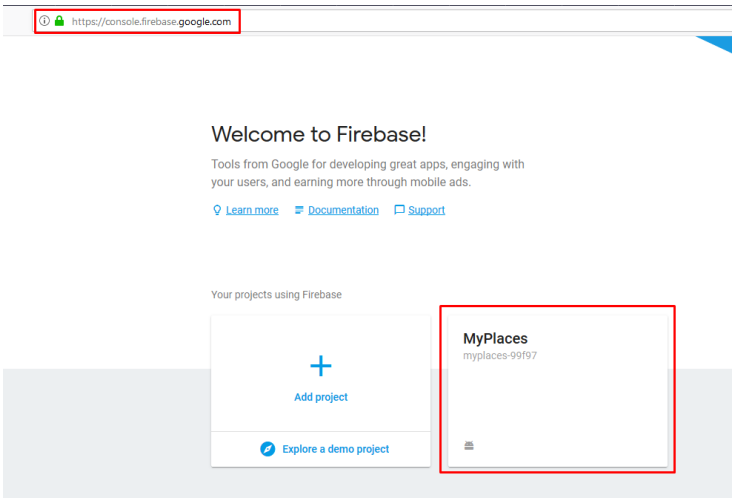
k. Izvršiti ažuriranje svih neophodnih biblioteka kako bi projekat mogao da se kompajlira. Čest problem je usaglašavanje verzije *firebase* biblioteke i verzije *gms*.



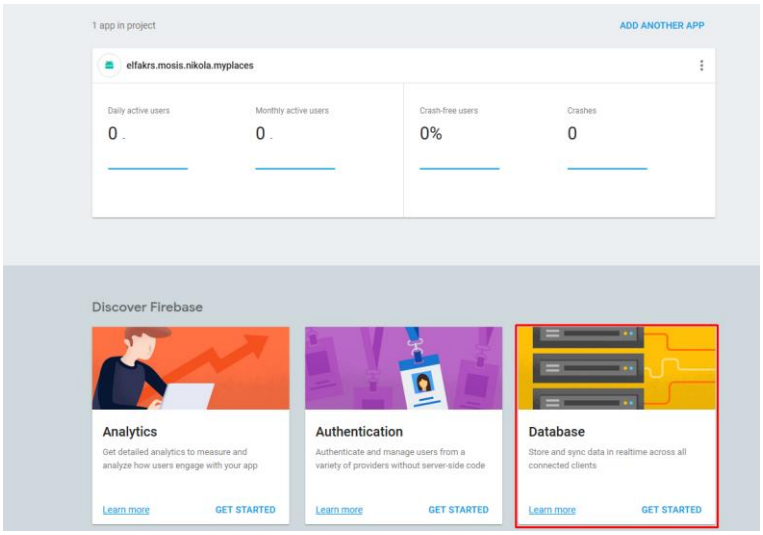
l. Inicijalno, pristup *Firebase* bazi podataka je ograničen samo autentifikovanim korisnicima. Potrebno je inicijalno ukloniti to ograničenje kako bi bilo lakše razviti aplikaciju. Klikom na *configure your rules for public access* link u *Firebase* asistentu potrebno je otvoriti uputstvo za konfigurisanje javnog pristupa.



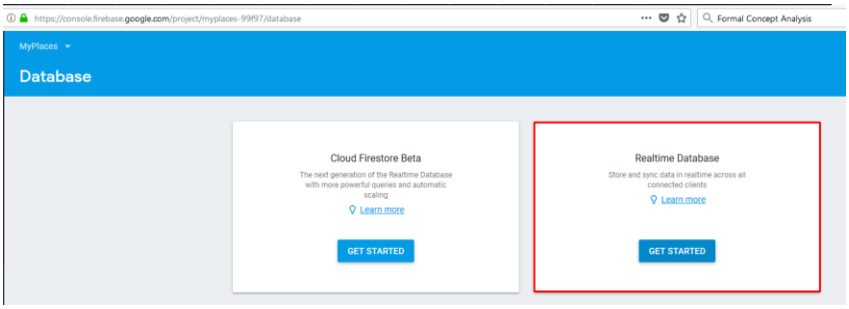
m. Kako bi pristup mogao biti konfigurisan, potrebno je otvoriti *Firebase* konzolu i u njoj izabrati projekat koji je potrebno konfigurisati.



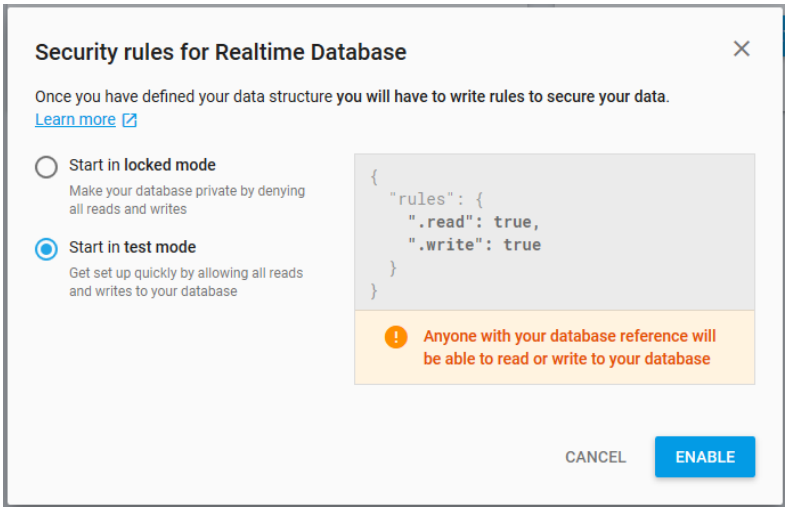
n. A zatim pristupiti bazi podataka.



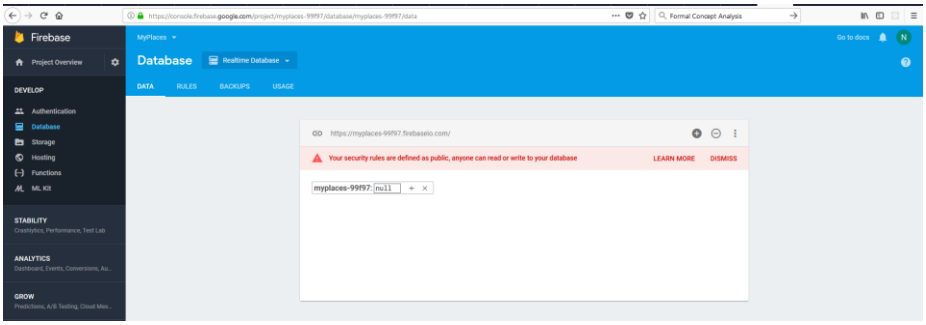
o. Izabrati *Realtime Database*



p. A zatim i *Start in test mode* čime će biti omogućen javni pristup bazi podataka od strane svakoga koi ma reference na bazu.



q. Nakon ovog koraka baza podataka je konfigurisana.



2. Upis i čitanje podataka iz *Firebase* baze podataka.
    a. Potrebno je prepraviti klasu MyPlace.java kako bi odgovarala modelu podataka koji će biti potrebno smestiti u *Firebase* bazu podataka. Prepraviti modifikatore pristupa svim atributima klase iz *private* u *public* i ukloniti odgovarajuće *get* i *set* metode. Kreirati konstruktor bez argumenata. Prepraviti atribut *int ID* u *String key*.

```
@IgnoreExtraProperties
public class MyPlace {
    public String name;
    public String description;
    public String longitude;
    public String latitude;
    @Exclude
    public String key;
    public MyPlace(){}

    public MyPlace(String nme, String desc)
    {
        this.name = nme;
        this.description = desc;
    }

    public MyPlace(String nme) { this(nme, desc: ""); }

    @Override
    public String toString() { return this.name; }
}
```

    b. Atribut *key* će biti dodeljen svakom smeštenom objektu od strane *Firebase* baze podataka. Zbog toga ga nije potrebno dodavati u bazu podataka prilikom snimanja. Isključivanje se vrši korišćenjem specijalnih modifikatora klase, *@IgnoreExtraProperties* i atributa, *@Exclude*.
    c. Da bi bila dodata podrška za rad sa *Firebase* bazom podataka, potrebno je izvršiti i mapiranje različitih domena ključeva. *Firebase* baza podataka svakom smeštenom objektu dodeljuje ključ, tipa string, koji zavisi od vremena dodavanja objekta. Model podataka na kome se zasniva *MyPlaces* aplikacija je lista čijim elementima se pristupa preko indeksa. Pošto se kao identifikatori instanci *MyPlace* klase u aplikaciji koriste celi

brojevi, njihovi indeksi u listi, a kako baza koristi stringove, potrebno je izvršiti i mapiranje ovih ključeva. Kako je prilikom promene vrednosti atributa instance *MyPlace* klase dostupan indeks a na osnovu njega i ključ (preko atributa klase *MyPlace*), problem predstavlja mapiranje ključa u indeks. Ovo mapiranje je potrebno kako bi moglo da se pristupi odgovarajućem elementu na osnovu indeksa u slučaju kada je poznat samo ključ. Pored toga, ovakvo mapiranje služi i za brzu proveru da li je podatak dostupan ili ne u aplikaciji.

d. Kako bi aplikacija mogla da prati promene nad podacima i da ih isčitava, potrebno je registrovati odgovarajuće *event listenere* koji se aktiviraju prilikom inicijalnog učitavanja iz baze podataka kao i prilikom promena koje su izvršene nad podacima. Pošto je osnovni model podataka aplikacije lista, u tom cilju je potrebno dodati *ChildEventListener* i *SingleValueListener* referenci baze podataka kako bi promene bile ispravno praćene.

```
public class MyPlacesData {

    private ArrayList<MyPlace> myPlaces;
    private HashMap<String, Integer> myPlacesKeyIndexMapping;
    private DatabaseReference database;
    private static final String FIREBASE_CHILD = "my-places";

    private MyPlacesData() {
        myPlaces = new ArrayList<>();
        myPlacesKeyIndexMapping = new HashMap<String, Integer>();
        database = FirebaseDatabase.getInstance().getReference();
        database.child(FIREBASE_CHILD).addChildEventListener(childEventListener);
        database.child(FIREBASE_CHILD).addListenerForSingleValueEvent(parentEventListener);
    }

ValueEventListener parentEventListener = new ValueEventListener() {
    @Override
    public void onDataChange(DataSnapshot dataSnapshot) {

    }

    @Override
    public void onCancelled(DatabaseError databaseError) {

    }
};

ChildEventListener childEventListener = new ChildEventListener() {
    @Override
    public void onChildAdded(DataSnapshot dataSnapshot, String previousChildName) {

    }

    @Override
    public void onChildChanged(DataSnapshot dataSnapshot, String previousChildName) {

    }

    @Override
    public void onChildRemoved(DataSnapshot dataSnapshot) {

    }

    @Override
    public void onChildMoved(DataSnapshot dataSnapshot, String previousChildName) {
        //kljucevi se ne menjaju, nije potrebno implementirati
    }

    @Override
    public void onCancelled(DatabaseError databaseError) {

    }
};
```

e. Prepraviti metodu *addNewPlace* klase *MyPlacesData.java* kako bi dodato omiljeno mesto bilo upisano u bazu podataka. Voditi računa da se prvo od Firebase baze podataka pribavi

ključ, koji će biti dodeljen elementu, postaviti odgovarajući atribut key i dodati par ključ-indeks u *HashMap* a zatim dodati objekat u bazu.

```
public void addNewPlace(MyPlace place){
    String key = database.push().getKey();
    myPlaces.add(place);
    myPlacesKeyIndexMapping.put(key,myPlaces.size()-1);
    database.child(FIREBASE_CHILD).child(key).setValue(place);
    place.key = key;
}
```

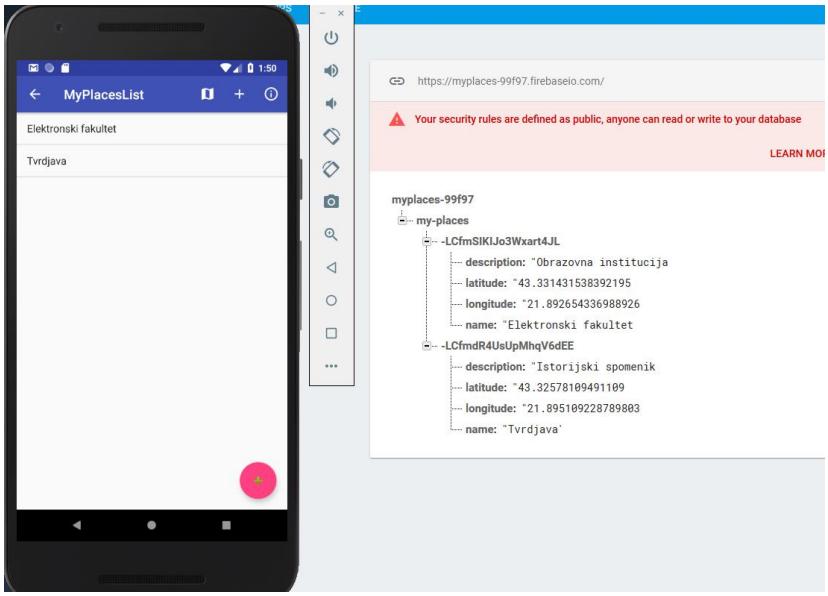
f. Na sličan način, vodeći računa o redosledu operacija, prepraviti metodu *deletePlace* i implementirati metodu *updatePlace*. Svaki put kada je element izbačen iz liste, zbog promene indeksa instanci koje su unutar liste, potrebno je rekreirati mapiranje ključeva.

```
public void deletePlace(int index){
    database.child(FIREBASE_CHILD).child(myPlaces.get(index).key).removeValue();
    myPlaces.remove(index);
    recreateKeyIndexMapping();
}

public void updatePlace(int index, String nme, String desc, String lng, String lat){
    MyPlace myPlace = myPlaces.get(index);
    myPlace.name = nme;
    myPlace.description = desc;
    myPlace.latitude = lat;
    myPlace.longitude = lng;
    database.child(FIREBASE_CHILD).child(myPlace.key).setValue(myPlace);
}

private void recreateKeyIndexMapping(){
    myPlacesKeyIndexMapping.clear();
    for (int i = 0;i<myPlaces.size();i++) {
        myPlacesKeyIndexMapping.put(myPlaces.get(i).key, i);
    }
}
```

g. Zameniti pozivom *updatePlace* metode u EditMyPlaceActivity kod koji ažurira instance MyPlace klase.

```
@Override
public void onClick(View view) {
    switch(view.getId()) {
        case R.id.editmyplace_finished_button: {
            EditText etName = (EditText)findViewById(R.id.editmyplace_name_edit);
            String nme = etName.getText().toString();
            EditText etDesc = (EditText)findViewById(R.id.editmyplace_desc_edit);
            String desc = etDesc.getText().toString();
            EditText latEdit = (EditText)findViewById(R.id.editmyplace_lat_edit);
            String lat = latEdit.getText().toString();
            EditText lonEdit = (EditText)findViewById(R.id.editmyplace_lon_edit);
            String lon = lonEdit.getText().toString();
            if(!editMode)
            {
                MyPlace place = new MyPlace(nme, desc);
                place.latitude = lat;
                place.longitude = lon;
                MyPlacesData.getInstance().addNewPlace(place);
            }
            else
            {
                MyPlacesData.getInstance().updatePlace(position, nme, desc, lon, lat);
            }
            setResult(Activity.RESULT_OK);
            finish();
            break;
        }

        case R.id.editmyplace_cancel_button: {
            setResult(Activity.RESULT_CANCELED);
            finish();
            break;
        }
        case R.id.editmyplace_location_button: {
            Intent i = new Intent( packageContext: this, MyPlacesMapsActivity.class);
            i.putExtra( name: "state", MyPlacesMapsActivity.SELECT_COORDINATES);
            startActivityForResult(i, requestCode: 1);
        }
    }
}
```

h. Probati aplikaciju. Proveravati da li se u Firebase konsoli nešto vidi.



3. Čitanje podataka iz baze podataka i reagovanje na promene u podacima.
   a. Trenutno aplikacija ne učitava podatke koji su upisani u bazu. Da bi aplikacija učitavala podatke i reagovala na njihovu promenu, potrebno je da osluškuje događaje koje emituje *Firebase*. Prilikom svake promene u podacima, odgovarajuće funkcije implementacije *listener* klasa se pozivaju. Potrebno je implementirati ove funkcije tako da ne reaguju ako je promenu nad podacima inicirala sama aplikacija. To je najlakše uraditi proverom da li *HashMap* ključeva sadrži upravo ključ podataka na čiju promenu ili dodavanje aplikacija treba da reaguje.

```
ChildEventListener childEventListener = new ChildEventListener() {
    @Override
    public void onChildAdded(DataSnapshot dataSnapshot, String previousChildName) {
        String myPlaceKey = dataSnapshot.getKey();
        if(!myPlacesKeyIndexMapping.containsKey(myPlaceKey)){
            MyPlace myPlace = dataSnapshot.getValue(MyPlace.class);
            myPlace.key = myPlaceKey;
            myPlaces.add(myPlace);
            myPlacesKeyIndexMapping.put(myPlaceKey,myPlaces.size()-1);
        }
    }

    @Override
    public void onChildChanged(DataSnapshot dataSnapshot, String previousChildName) {
        String myPlaceKey = dataSnapshot.getKey();
        MyPlace myPlace = dataSnapshot.getValue(MyPlace.class);
        myPlace.key = myPlaceKey;
        if(myPlacesKeyIndexMapping.containsKey(myPlaceKey)){
            int index = myPlacesKeyIndexMapping.get(myPlaceKey);
            myPlaces.set(index, myPlace);
        } else {
            myPlaces.add(myPlace);
            myPlacesKeyIndexMapping.put(myPlaceKey,myPlaces.size()-1);
        }
    }

    @Override
    public void onChildRemoved(DataSnapshot dataSnapshot) {
        String myPlaceKey = dataSnapshot.getKey();
        if(myPlacesKeyIndexMapping.containsKey(myPlaceKey)){
            int index = myPlacesKeyIndexMapping.get(myPlaceKey);
            myPlaces.remove(index);
            recreateKeyIndexMapping();
        }

    }
    @Override
    public void onChildMoved(DataSnapshot dataSnapshot, String previousChildName) {
        //kljucevi se ne menjaju, nije potrebno implementirati
    }
    @Override
    public void onCancelled(DatabaseError databaseError) {
    }
};
```

b. Probati aplikaciju. Dodavati elemente i brisati. Proveriti da li osnovne metode dobro funkcionišu i da li se inicijalno učitavanje izvršava.

c. Učitavanje liste nije trenutno tako da će najčešće prilikom prvog učitavanja lista biti prazna. Pored toga, ukoliko neko drugi doda omiljeno mesto u istu listu ili je omiljeno mesto dodato iz konzole, lista neće biti osvežena. Glavni problem je, prilikom učitavanje vreme koje je potrebno da se lista asinhrono napuni. Kao i mehanizam notifikacije korsnika liste o tome da se lista promenila.

d. Prvi problem je moguće rešiti korišćenjem *ValueEventListener*-a koji će osluškivati promene na samoj listi podataka. Garantovano je da će ovaj listener biti iniciran samo nakon što su sve promene (inicjalna iščitavanja) članova liste već završile svoje pozive.

e. Da bi bio rešen i drugi problem, potrebno je dodati mehanizam notifikacije korisnika liste o tome da je lista promenjena. U klasu *MyPlacesData* dodati implementaciju listener mehanizma koji će obaveštavati korisnike liste koje će oslučkivati ovaj listener.

```
    ListUpdatedEventListener updateListener;
    public void setEventListener(ListUpdatedEventListener listener) {
        updateListener = listener;
    }

    public interface ListUpdatedEventListener {
        void onListUpdated();
    }

}
```

f. Dodati i pozive koji aktiviraju ovaj *listener*.

```
ChildEventListener childEventListener = new ChildEventListener() {
    @Override
    public void onChildAdded(DataSnapshot dataSnapshot, String previousChildName) {
        String myPlaceKey = dataSnapshot.getKey();
        if(!myPlacesKeyIndexMapping.containsKey(myPlaceKey)){
            MyPlace myPlace = dataSnapshot.getValue(MyPlace.class);
            myPlace.key = myPlaceKey;
            myPlaces.add(myPlace);
            myPlacesKeyIndexMapping.put(myPlaceKey,myPlaces.size()-1);
            if (updateListener != null)
                updateListener.onListUpdated();
        }
    }

    @Override
    public void onChildChanged(DataSnapshot dataSnapshot, String previousChildName) {
        String myPlaceKey = dataSnapshot.getKey();
        MyPlace myPlace = dataSnapshot.getValue(MyPlace.class);
        myPlace.key = myPlaceKey;
        if(myPlacesKeyIndexMapping.containsKey(myPlaceKey)){
            int index = myPlacesKeyIndexMapping.get(myPlaceKey);
            myPlaces.set(index, myPlace);
        } else {
            myPlaces.add(myPlace);
            myPlacesKeyIndexMapping.put(myPlaceKey,myPlaces.size()-1);
        }
        if (updateListener != null)
            updateListener.onListUpdated();
    }

    @Override
    public void onChildRemoved(DataSnapshot dataSnapshot) {
        String myPlaceKey = dataSnapshot.getKey();
        if(myPlacesKeyIndexMapping.containsKey(myPlaceKey)){
            int index = myPlacesKeyIndexMapping.get(myPlaceKey);
            myPlaces.remove(index);
            recreateKeyIndexMapping();
            if (updateListener != null)
                updateListener.onListUpdated();
        }
    }
};

ValueEventListener parentEventListener = new ValueEventListener() {
    @Override
    public void onDataChange(DataSnapshot dataSnapshot) {
        if (updateListener != null)
            updateListener.onListUpdated();
    }

    @Override
    public void onCancelled(DatabaseError databaseError) {
    }
};
```

g. Probati aplikaciju. Probati simultano dodavanje omiljenih lokacija na dva uređaja.
h. Za domaći:
    i. Registrovati mapu na promene u listi
    ii. Dodati autentifikaciju u aplikaciju preko *Firebase*-a.