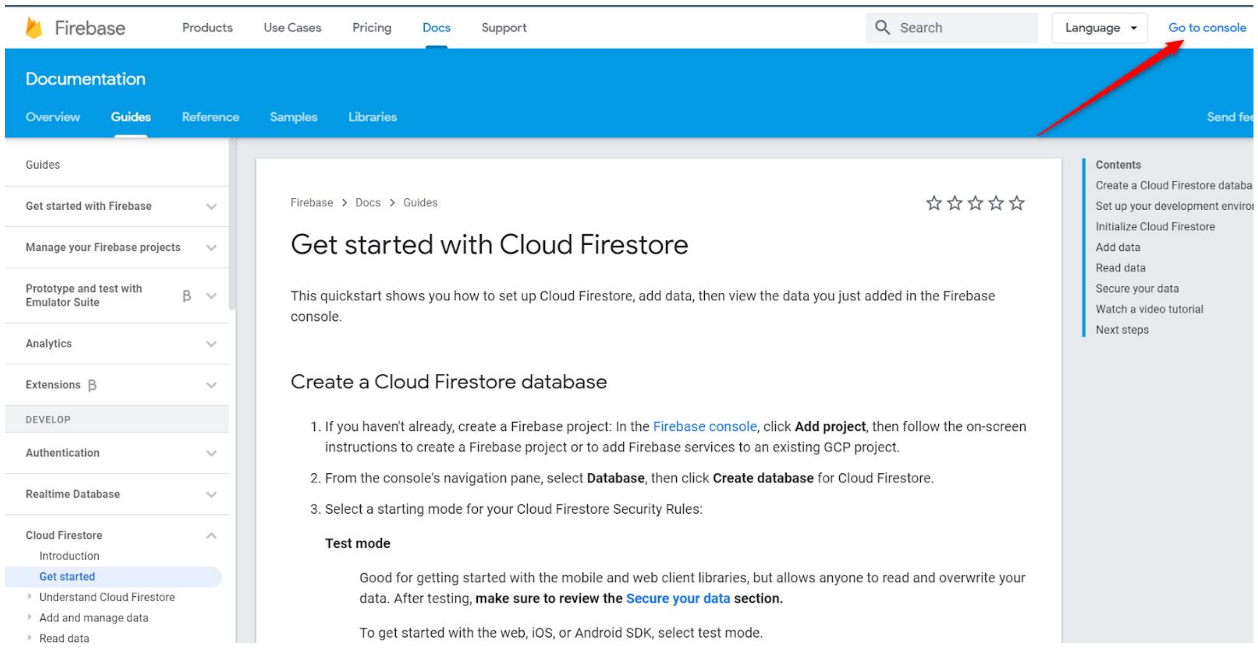
Firestore

1. Download the ListyCity with custom adapter code from the following link if you don't have it with you.
   Git clone the repo below using the git commands you learned during Lab 4.
   https://github.com/simpleParadox/CMPUT-301-CustomList.git
   Make sure that you use your ualberta email id to access the file.

2. Go to the following link and follow the instructions on how to setup Firestore for your application (The lab will walk you through this). NOTE: Only follow the instructions from the beginning and stop before the 'add Data' section. Follow the instructions carefully.
   https://firebase.google.com/docs/firestore/quickstart
   If you are having any trouble, please ask the TAs for help. This step is crucial for your app to work properly.

3. Now go to the firebase console and locate the application that you just registered with the local android project. (You register your app with firebase in step 2).



Get started with Cloud Firestore

This quickstart shows you how to set up Cloud Firestore, add data, then view the data you just added in the Firebase console.

Create a Cloud Firestore database

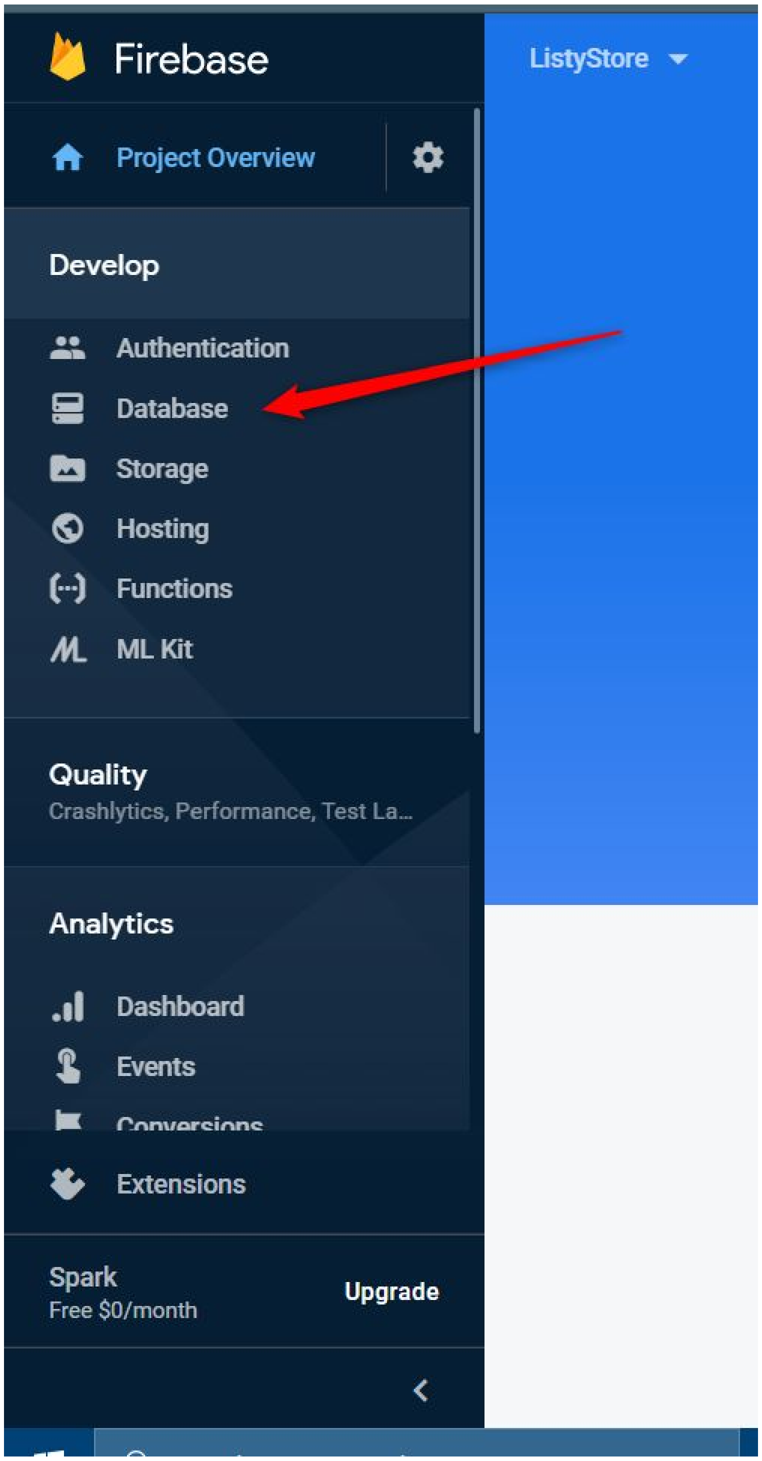
1. If you haven't already, create a Firebase project: In the Firebase console, click **Add project**, then follow the on-screen instructions to create a Firebase project or to add Firebase services to an existing GCP project.
2. From the console's navigation pane, select **Database**, then click **Create database** for Cloud Firestore.
3. Select a starting mode for your Cloud Firestore Security Rules:

Test mode

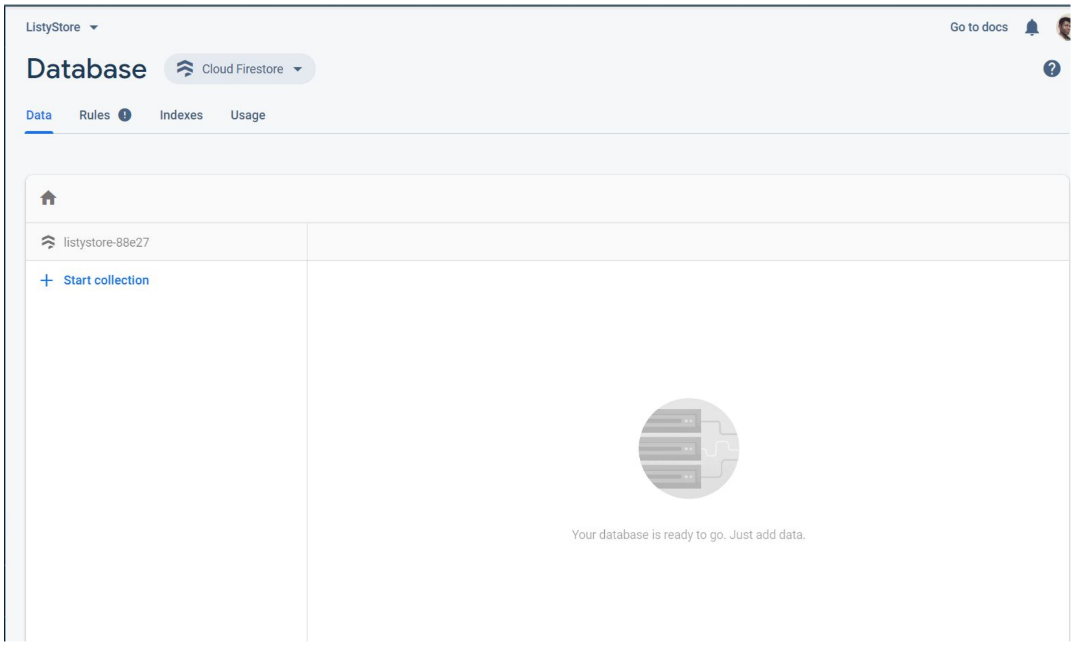
Good for getting started with the mobile and web client libraries, but allows anyone to read and overwrite your data. After testing, **make sure to review the Secure your data section.**

To get started with the web, iOS, or Android SDK, select test mode.

4. After the console opens up, click on Database on the left pane.



5. You should see an empty database

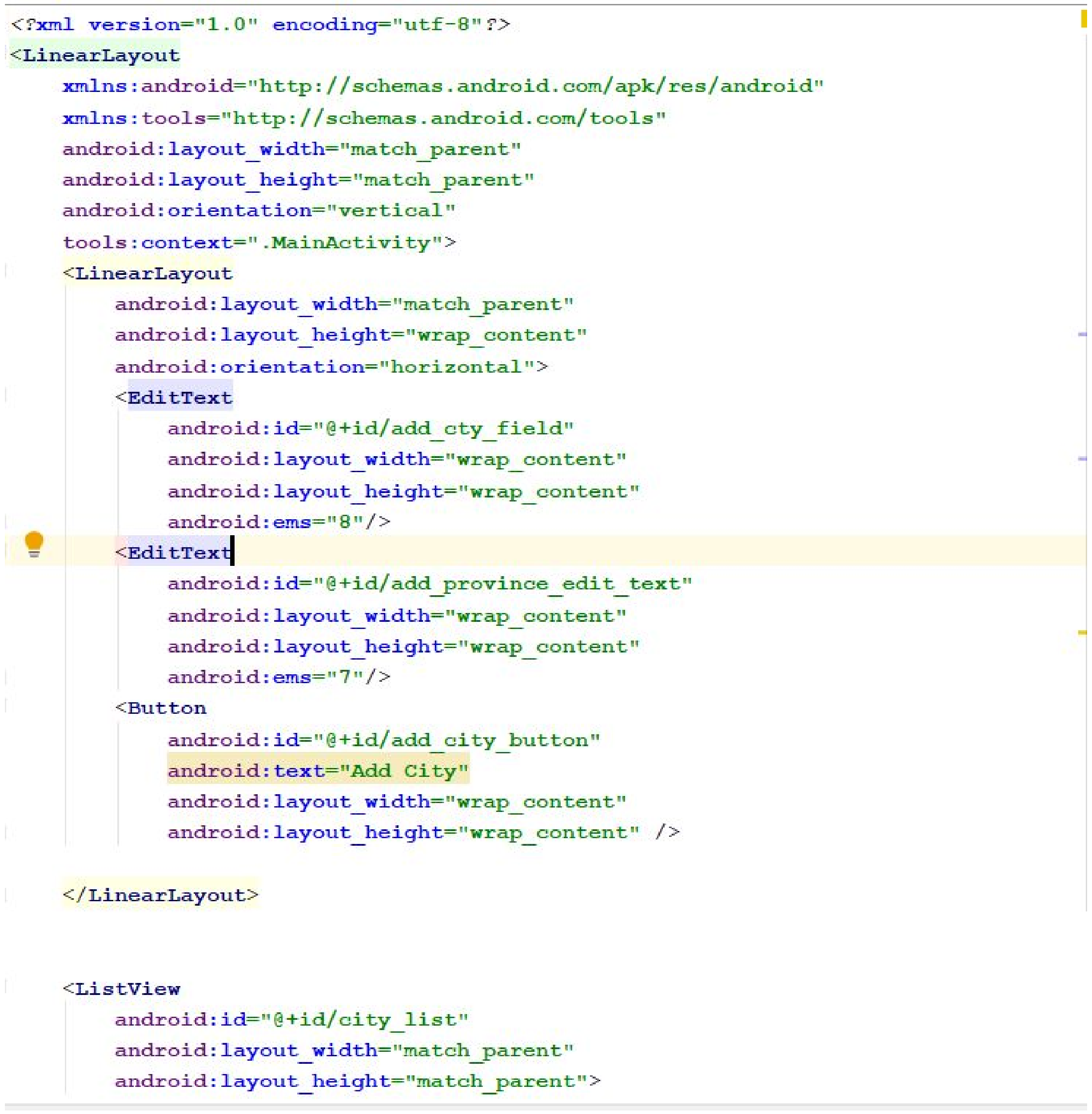


Your database is ready to go. Just add data.

Congrats, if you can now access the database for your application.

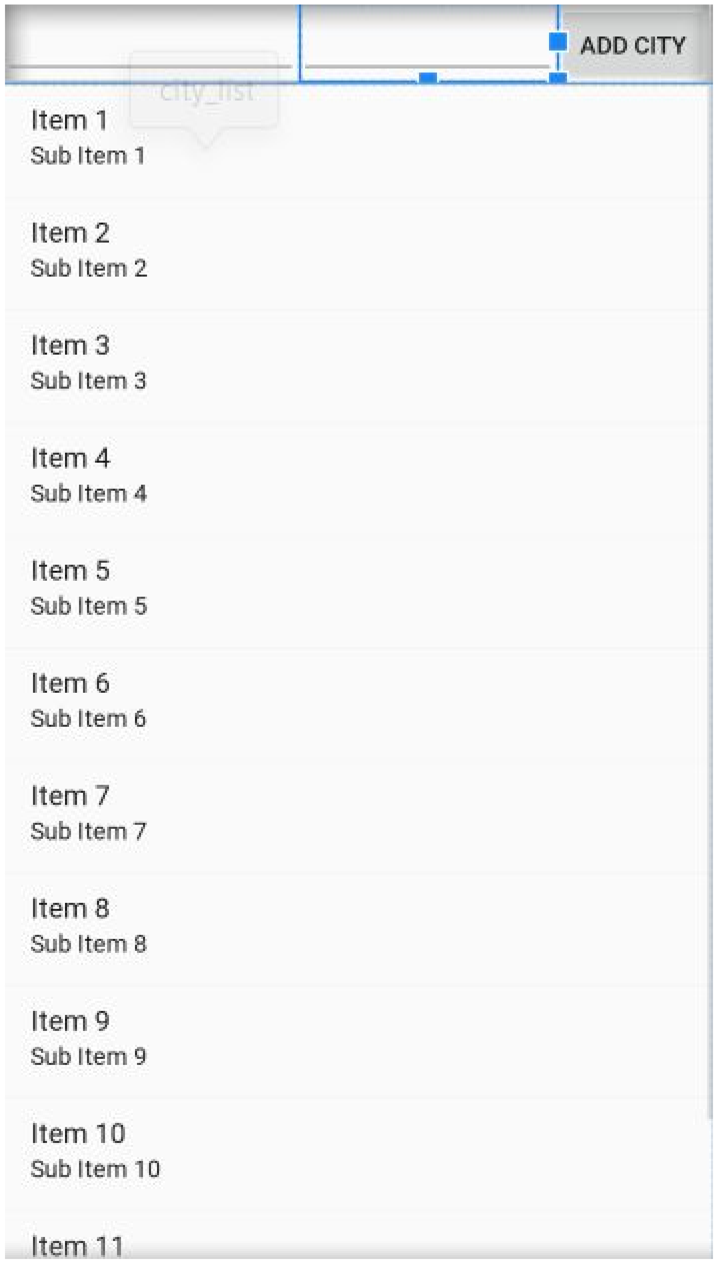
6. Now let’s write some code to store data remotely.

7. Go to your android application and navigate to activity_main.xml. We will add two EditTexts and a Button to add a new city and province to the list and concurrently store the city in the Firestore. We create an addition LinearLayout to hold the three new views. Note that the orientation is horizontal for the nested LinearLayout. This makes sure that the views are placed beside each other.



```xml
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    tools:context=".MainActivity">
    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:orientation="horizontal">
        <EditText
            android:id="@+id/add_cty_field"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:ems="8"/>
        <EditText
            android:id="@+id/add_province_edit_text"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:ems="7"/>
        <Button
            android:id="@+id/add_city_button"
            android:text="Add City"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content" />

    </LinearLayout>


    <ListView
        android:id="@+id/city_list"
        android:layout_width="match_parent"
        android:layout_height="match_parent">
```

After updating the code in activity_main.xml, you will get something as follows.
NOTE: You can customize the layouts in any way you want.



As you can see, there are two new EditText views and a button to add a new city to the list.

We have completed updating the layout. Let's write up the logic for the two EditTexts and the Button.

8. There is a lot of functionality that needs to be implemented for sending data to Firestore and also listening for any changes required. Follow along carefully.
   We will do everything inside the 'onCreate' method.

9. First of all, declare the following as class variables.

```
String TAG = "Sample";
Button addCityButton;
EditText addCityEditText;
EditText addProvinceEditText;
FirebaseFirestore db;
```

   And get the reference to the corresponding views from the xml file. From this part of the tutorial, everything will be inside the 'onCreate' method unless specified otherwise.

```
addCityButton = findViewById(R.id.add_city_button);
addCityEditText = findViewById(R.id.add_cty_field);
addProvinceEditText = findViewById(R.id.add_province_edit_text);
```

10. Make sure you have the cityDataList and the adapter set to their correct values as follows:

```
// Get a reference to the ListView and create an object for the city list.
cityList = findViewById(R.id.city_list);
cityDataList = new ArrayList<>();

// Set the adapter for the listView to the CustomAdapter that we created in Lab 3.
cityAdapter = new CustomList( context: MainActivity.this, cityDataList);
cityList.setAdapter(cityAdapter);
```

11. Now we create an instance of the Firestore as follows:

```
//          Access a Cloud Firestore instance from your Activity
        db = FirebaseFirestore.getInstance();
```

12. The next step is to get a top-level reference to the collection in the database.

```
// Get a top-level reference to the collection.
final CollectionReference collectionReference = db.collection( collectionPath: "Cities");
```

   The 'collectionReference' object is declared final for a reason. Can you guess why is it so?

13. Let's write the logic for the 'addCityButton' view which adds a new city to the list and also stores the data to firestore. Do you remember how to assign a listener to a view? Or more specifically, a button?

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

    }
});
```

14. The next step is to actually write something inside the overriden ‘onClick’ method so that we have some functionality.
    First of all, let’s get the city name and province name from the EditText fields.

```
// Retrieving the city name and the province name from the EditText fields.
final String cityName = addCityEditText.getText().toString();
final String provinceName = addProvinceEditText.getText().toString();
```

Then we will create a HashMap to store the data in the form of Key-value pairs.

```
HashMap<String, String> data = new HashMap<>();
```

Can you guess why we are using a key-value model to store the data?

The next step is to check whether the user has entered something in the field or not. (We don’t want an empty city or a city with no province. That will be undesirable).

```
if(cityName.length()>0 && provinceName.length()>0){
```

If everything is okay, we add the data into the HashMap

```
// If there is some data in the EditText field, then we create a new key-value pair.
data.put("province_name",provinceName);
```

Note that the key is named as ‘province_name’. This is because of the collection ID to be the same as that of the city name. Taking a closer look at the database will give you a good idea.

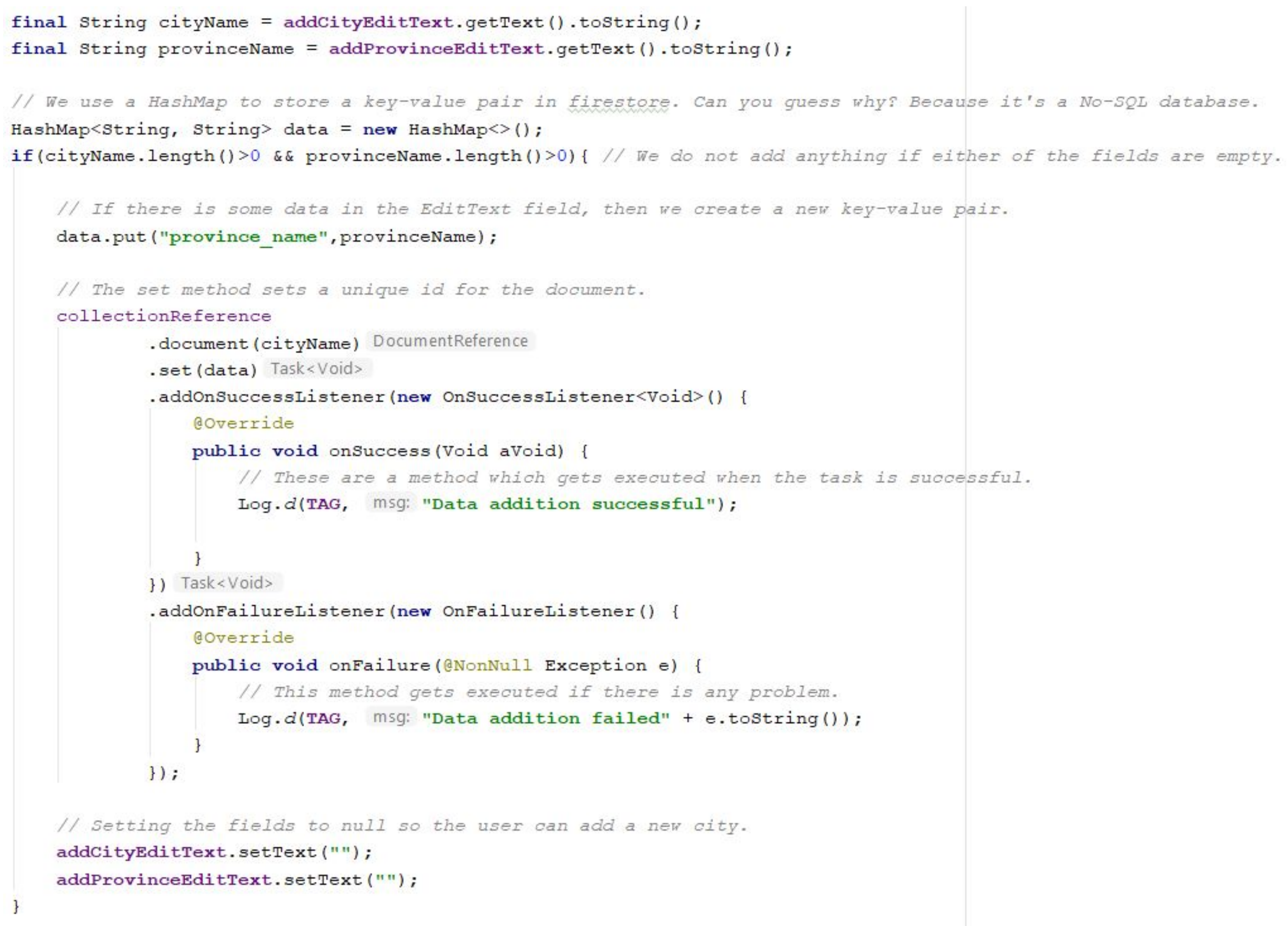
Now we add the data to Firestore using the following code snippet.

```
// The set method sets a unique id for the document.
collectionReference
        .document(cityName) DocumentReference
        .set(data) Task<Void>
        .addOnSuccessListener(new OnSuccessListener<Void>() {
            @Override
            public void onSuccess(Void aVoid) {
                // These are a method which gets executed when the task is succ
                Log.d(TAG, msg: "Data addition successful");

            }
        }) Task<Void>
        .addOnFailureListener(new OnFailureListener() {
            @Override
            public void onFailure(@NonNull Exception e) {
                // This method gets executed if there is any problem.
                Log.d(TAG, msg: "Data addition failed" + e.toString());
            }
        });
```

Finally, we set the EditText fields to an empty string for any new input.

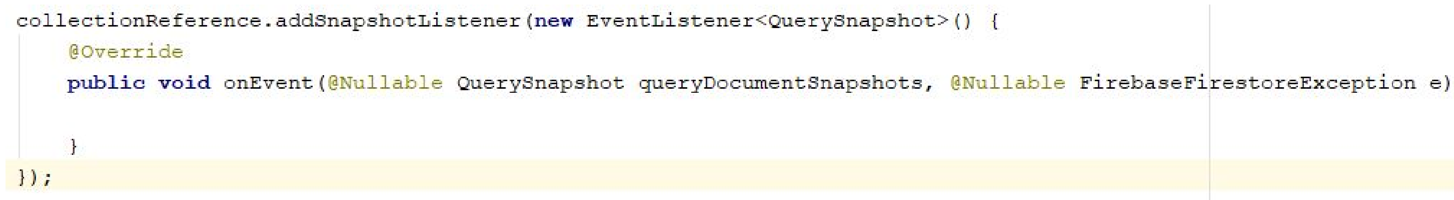
```
// Setting the fields to null so the user can add a new city.
addCityEditText.setText("");
addProvinceEditText.setText("");
```

In the end, your ‘onClick’ method should look like this. Make sure you understand each statement carefully and not just complete the tutorial for the sake of doing so.



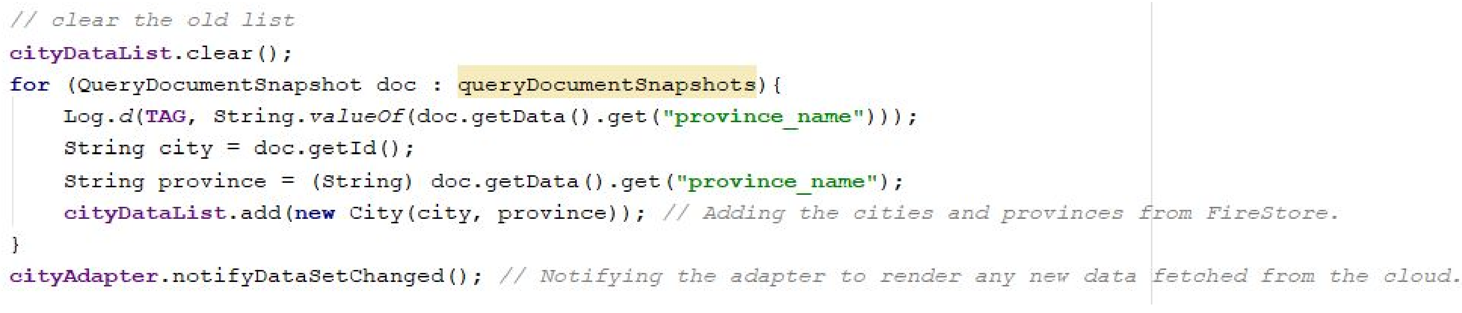
```
final String cityName = addCityEditText.getText().toString();
final String provinceName = addProvinceEditText.getText().toString();

// We use a HashMap to store a key-value pair in firestore. Can you guess why? Because it's a No-SQL database.
HashMap<String, String> data = new HashMap<>();
if(cityName.length()>0 && provinceName.length()>0){ // We do not add anything if either of the fields are empty.

    // If there is some data in the EditText field, then we create a new key-value pair.
    data.put("province_name",provinceName);

    // The set method sets a unique id for the document.
    collectionReference
            .document(cityName) DocumentReference
            .set(data) Task<Void>
            .addOnSuccessListener(new OnSuccessListener<Void>() {
                @Override
                public void onSuccess(Void aVoid) {
                    // These are a method which gets executed when the task is successful.
                    Log.d(TAG,  msg: "Data addition successful");

                }
            }) Task<Void>
            .addOnFailureListener(new OnFailureListener() {
                @Override
                public void onFailure(@NonNull Exception e) {
                    // This method gets executed if there is any problem.
                    Log.d(TAG,  msg: "Data addition failed" + e.toString());
                }
            });

    // Setting the fields to null so the user can add a new city.
    addCityEditText.setText("");
    addProvinceEditText.setText("");
}
```

15. Now we implement the most important feature Firestore provides, realtime updates.
    We add a snapshot listener to the 'collectionReference'. A snapshot is the state of the database at any given point of time.



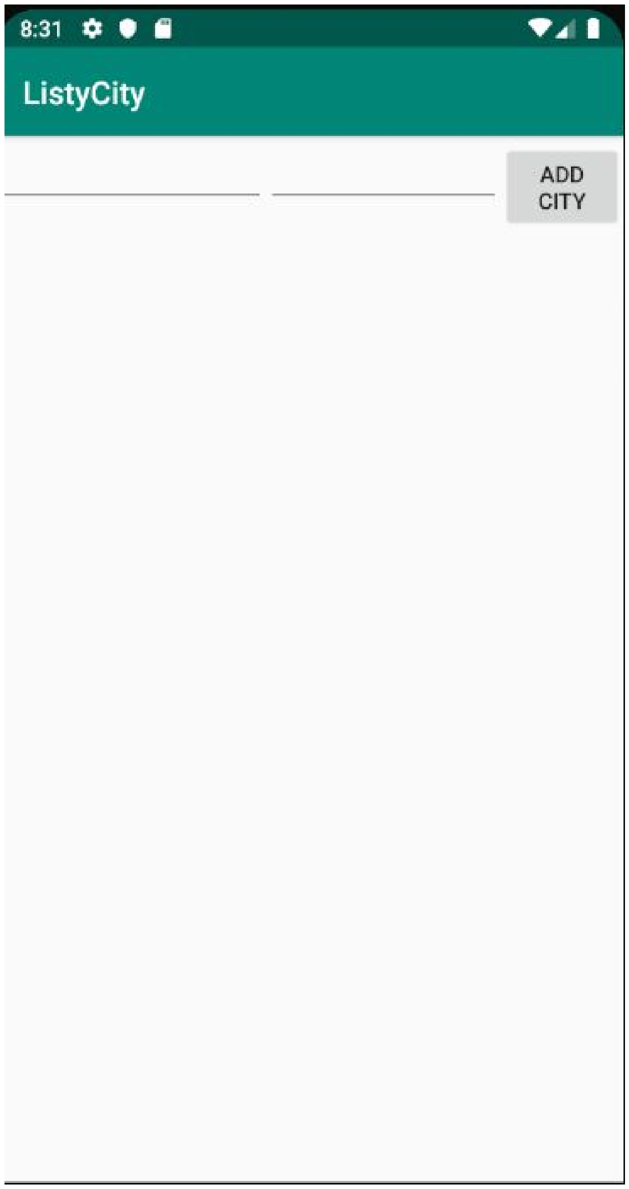
```
collectionReference.addSnapshotListener(new EventListener<QuerySnapshot>() {
    @Override
    public void onEvent(@Nullable QuerySnapshot queryDocumentSnapshots, @Nullable FirebaseFirestoreException e)

    }
});
```

Now we override the 'onEvent' method. This method is executed whenever any new event occurs in the remote database.



```
// clear the old list
cityDataList.clear();
for (QueryDocumentSnapshot doc : queryDocumentSnapshots){
    Log.d(TAG, String.valueOf(doc.getData().get("province_name")));
    String city = doc.getId();
    String province = (String) doc.getData().get("province_name");
    cityDataList.add(new City(city, province)); // Adding the cities and provinces from FireStore.
}
cityAdapter.notifyDataSetChanged(); // Notifying the adapter to render any new data fetched from the cloud.
```

16. Go ahead and run the application. You should see something like this initially. If you see an empty list, then it means that your Firestore Storage does not have any data. Otherwise, you will get some data.



17. Now try to add a city along with a province.
    Check the Firestore database whether the data new data has been added. Try to refresh the webpage if you do not see any data. If you still do not see the data, check whether your code.

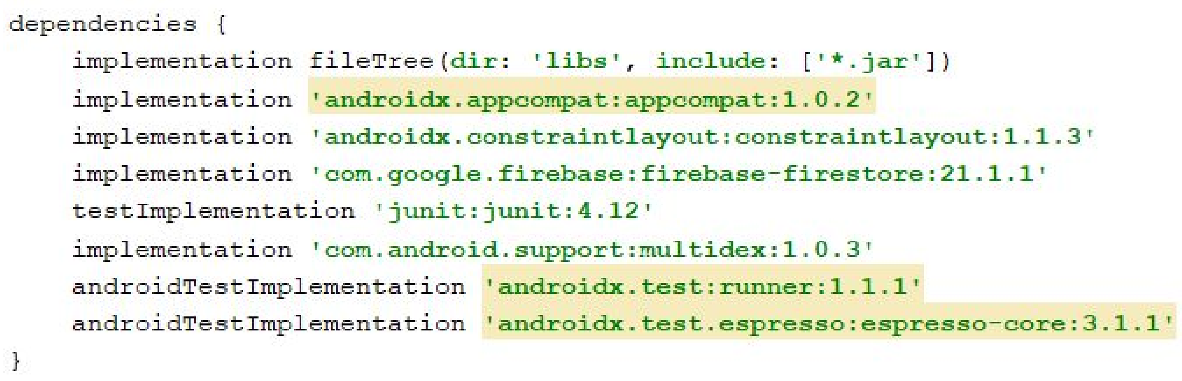
KNOWN ISSUES

Android Studio sometimes gives an error regarding .dex files and how it cannot fit the requested class in a single 'dex' file. To resolve this, include the following in your app level gradle file.

Under dependencies,

```
implementation 'com.android.support:multidex:1.0.3'
```

Something like this:

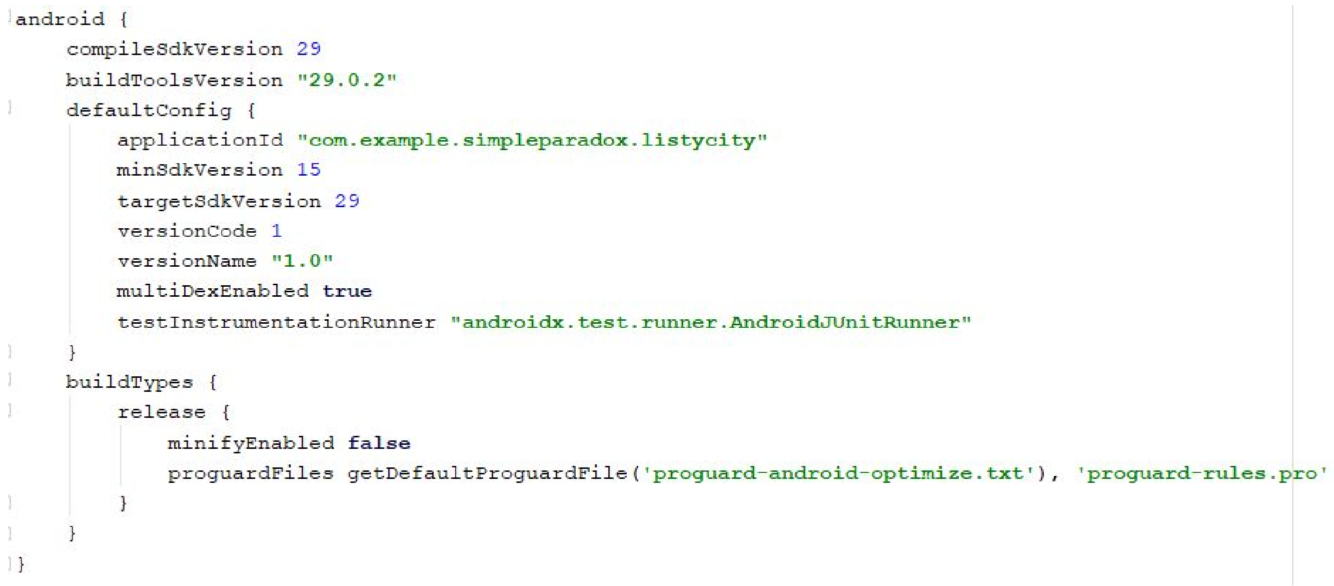


```
dependencies {
    implementation fileTree(dir: 'libs', include: ['*.jar'])
    implementation 'androidx.appcompat:appcompat:1.0.2'
    implementation 'androidx.constraintlayout:constraintlayout:1.1.3'
    implementation 'com.google.firebase:firebase-firestore:21.1.1'
    testImplementation 'junit:junit:4.12'
    implementation 'com.android.support:multidex:1.0.3'
    androidTestImplementation 'androidx.test:runner:1.1.1'
    androidTestImplementation 'androidx.test.espresso:espresso-core:3.1.1'
}
```

And under the 'defaultConfig' section under the 'android' section(in the same app level gradle file):
Add the following line:

```
multiDexEnabled true
```

Something like this:



```
android {
    compileSdkVersion 29
    buildToolsVersion "29.0.2"
    defaultConfig {
        applicationId "com.example.simpleparadox.listycity"
        minSdkVersion 15
        targetSdkVersion 29
        versionCode 1
        versionName "1.0"
        multiDexEnabled true
        testInstrumentationRunner "androidx.test.runner.AndroidJUnitRunner"
    }
    buildTypes {
        release {
            minifyEnabled false
            proguardFiles getDefaultProguardFile('proguard-android-optimize.txt'), 'proguard-rules.pro'
        }
    }
}
```