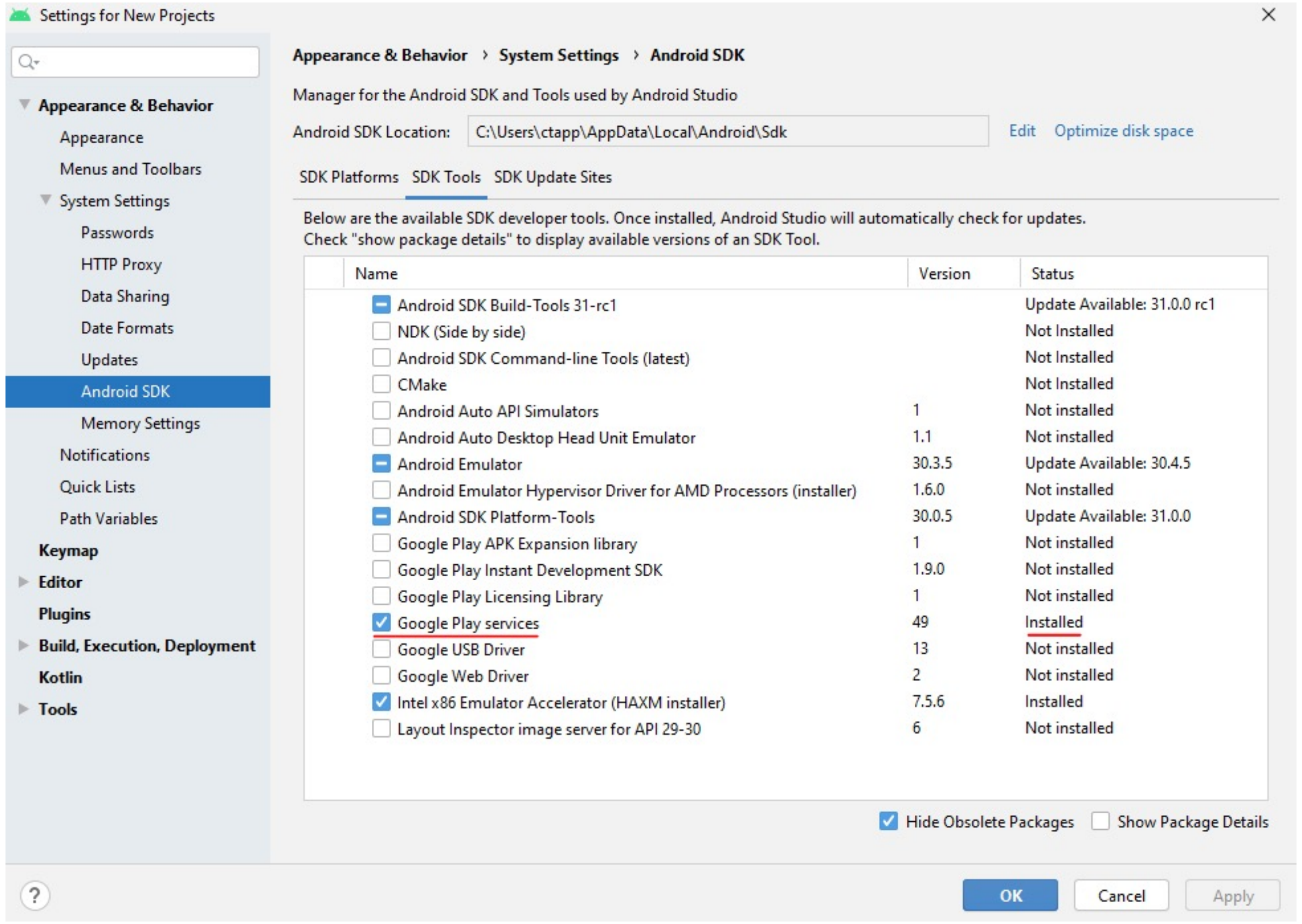
Creating a Firebase project (cont.)

- First thing is to make sure you have **Google play services installed**
  - Tools->SDK Manager



Manager for the Android SDK and Tools used by Android Studio

Android SDK Location: C:\Users\ctapp\AppData\Local\Android\Sdk

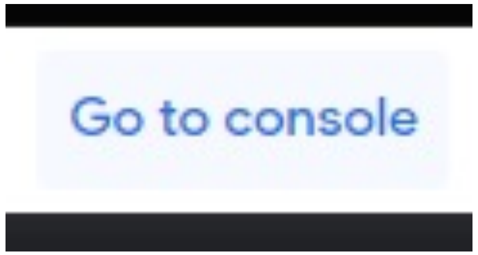
Below are the available SDK developer tools. Once installed, Android Studio will automatically check for updates.
Check "show package details" to display available versions of an SDK Tool.

	Name	Version	Status
–	Android SDK Build-Tools 31-rc1		Update Available: 31.0.0 rc1
☐	NDK (Side by side)		Not Installed
☐	Android SDK Command-line Tools (latest)		Not Installed
☐	CMake		Not Installed
☐	Android Auto API Simulators	1	Not installed
☐	Android Auto Desktop Head Unit Emulator	1.1	Not installed
–	Android Emulator	30.3.5	Update Available: 30.4.5
☐	Android Emulator Hypervisor Driver for AMD Processors (installer)	1.6.0	Not installed
–	Android SDK Platform-Tools	30.0.5	Update Available: 31.0.0
☐	Google Play APK Expansion library	1	Not installed
☐	Google Play Instant Development SDK	1.9.0	Not installed
☐	Google Play Licensing Library	1	Not installed
✓	Google Play services	49	Installed
☐	Google USB Driver	13	Not installed
☐	Google Web Driver	2	Not installed
✓	Intel x86 Emulator Accelerator (HAXM installer)	7.5.6	Installed
☐	Layout Inspector image server for API 29-30	6	Not installed

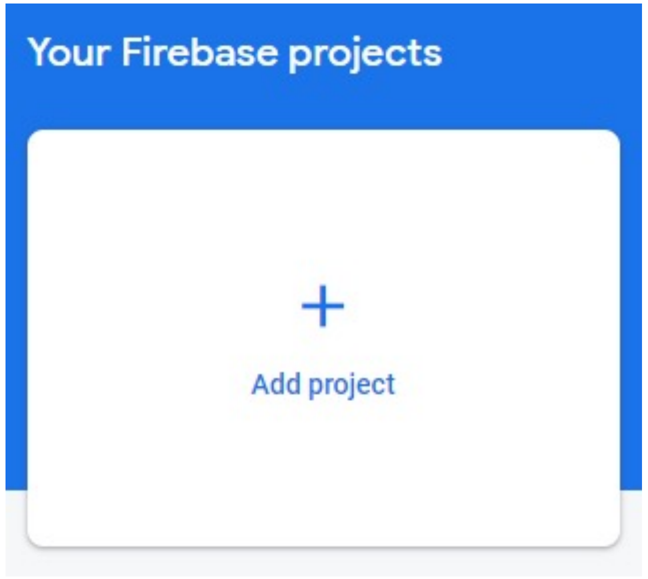
✓ Hide Obsolete Packages ☐ Show Package Details

Creating a Firebase project (cont.)

- Go to https://firebase.google.com
- Sign in then click 'Go to the console' on the top right

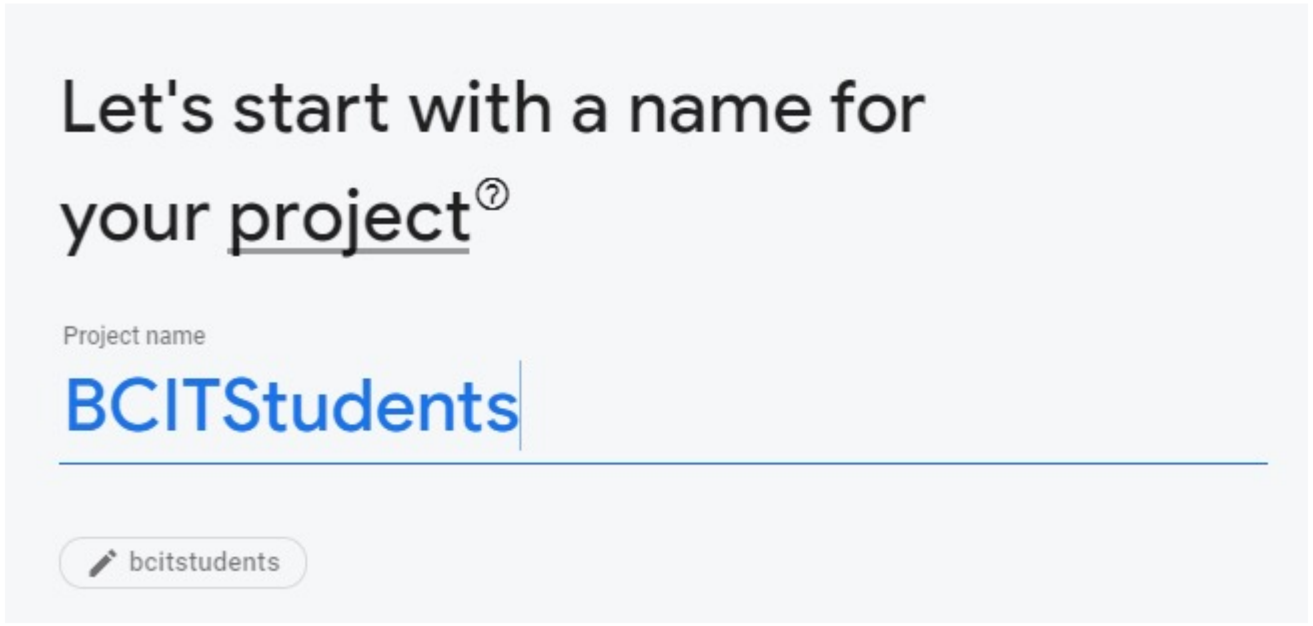


- Add a new project



Creating a Firebase project (cont.)

- Give your project a name such as *BCITStudents*

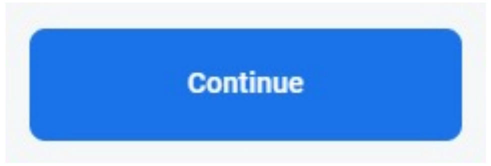

Let's start with a name for your project

Project name

BCITStudents

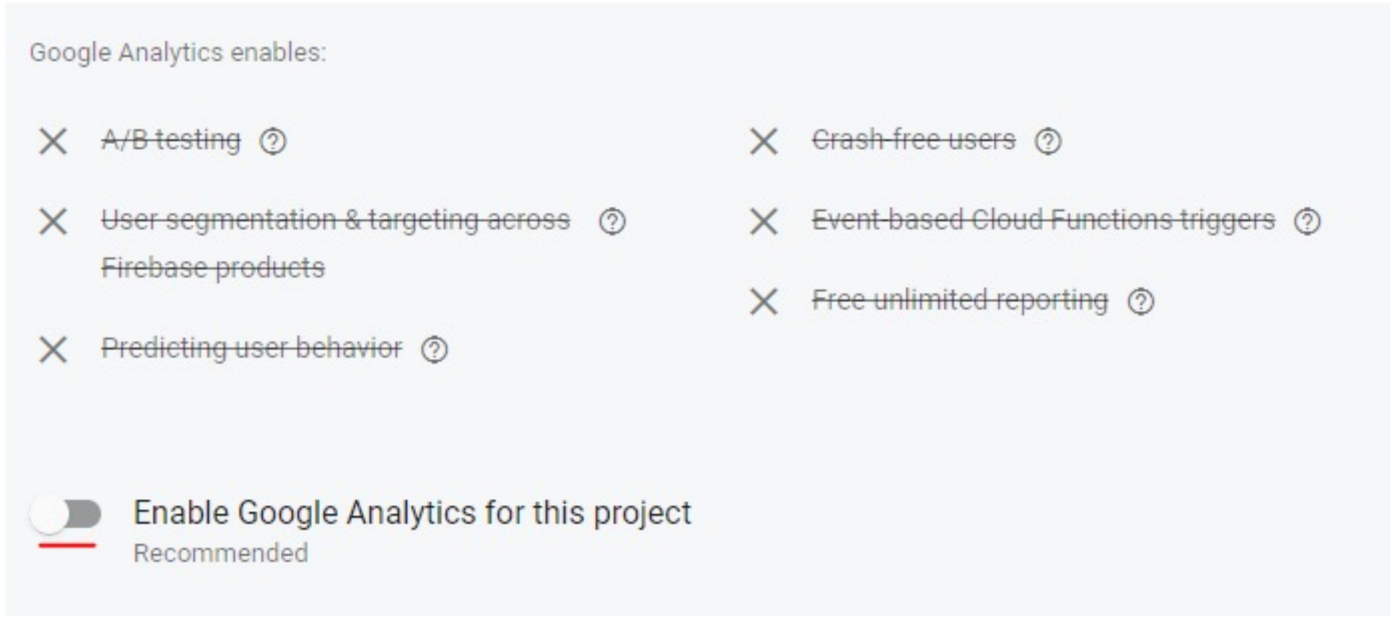
bcitstudents

Then hit continue



Creating a Firebase project (cont.)

- Uncheck *Enable Google Analytics for this project*

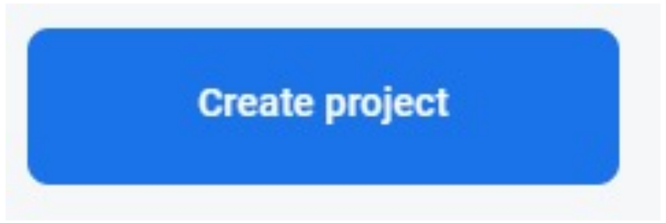


Google Analytics enables:

- ~~A/B testing~~
- ~~User segmentation & targeting across Firebase products~~
- ~~Predicting user behavior~~
- ~~Crash-free users~~
- ~~Event-based Cloud Functions triggers~~
- ~~Free unlimited reporting~~

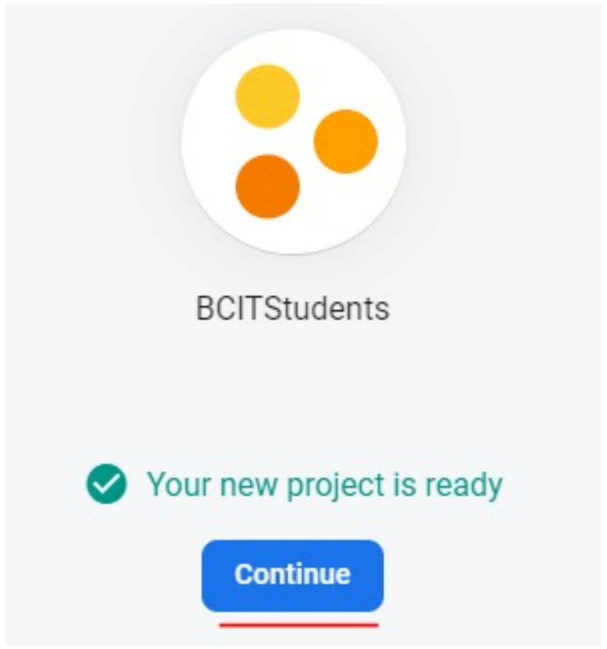
Enable Google Analytics for this project
Recommended

- Then click *Create project*



Creating a Firebase project (cont.)

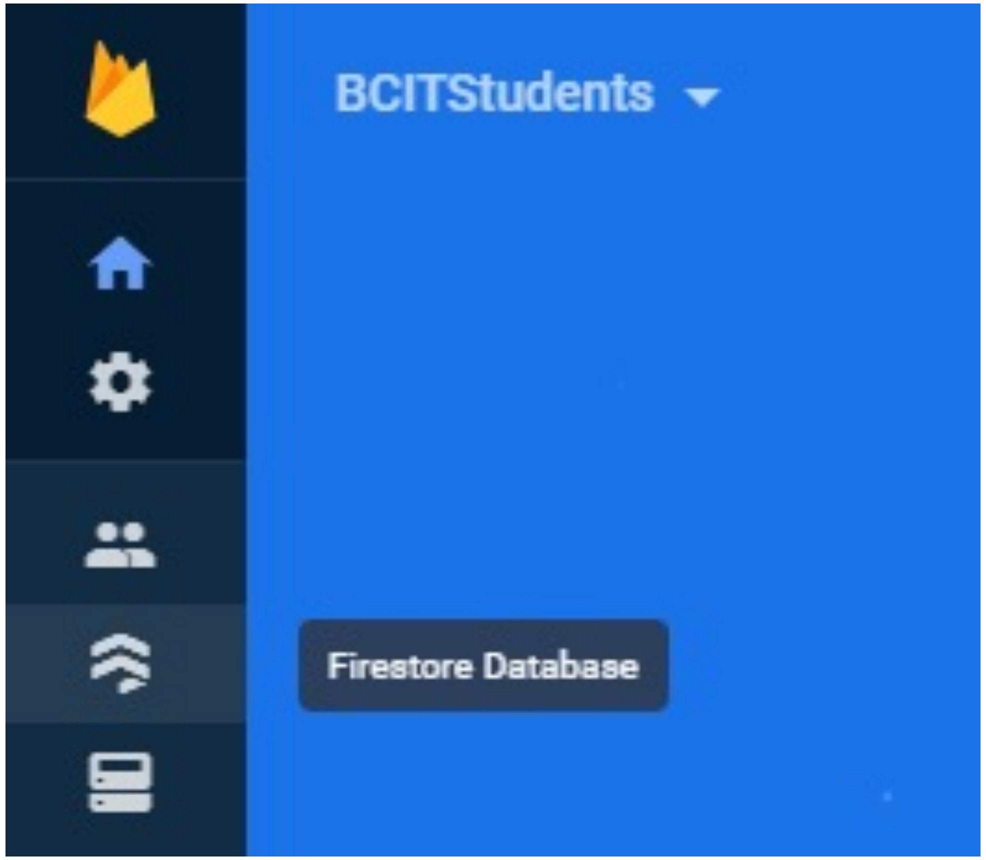
- When you project ready click continue



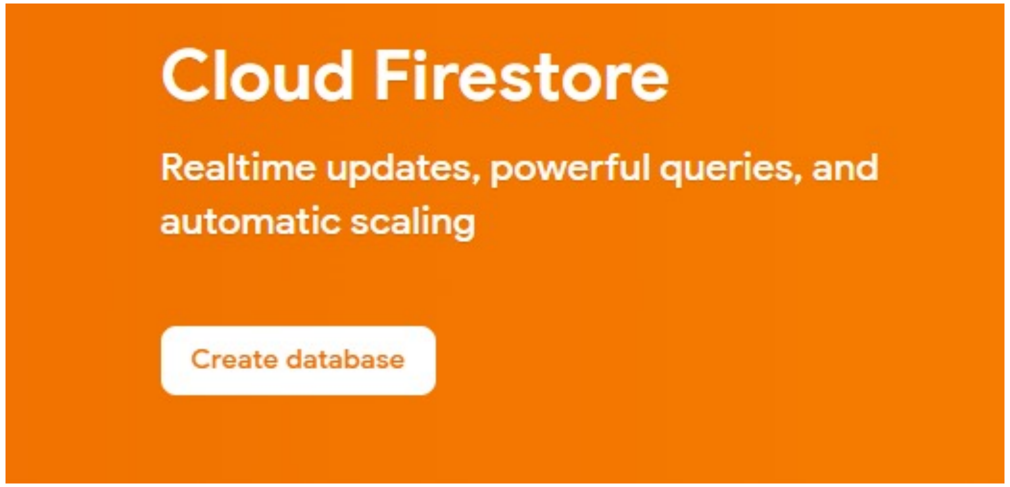
- You will then be taken to the *Project Overview* page. Here you can check out many of the services Firebase has to offer

Setting up Cloud Firestore

- Click on *Firestore Database* on the left

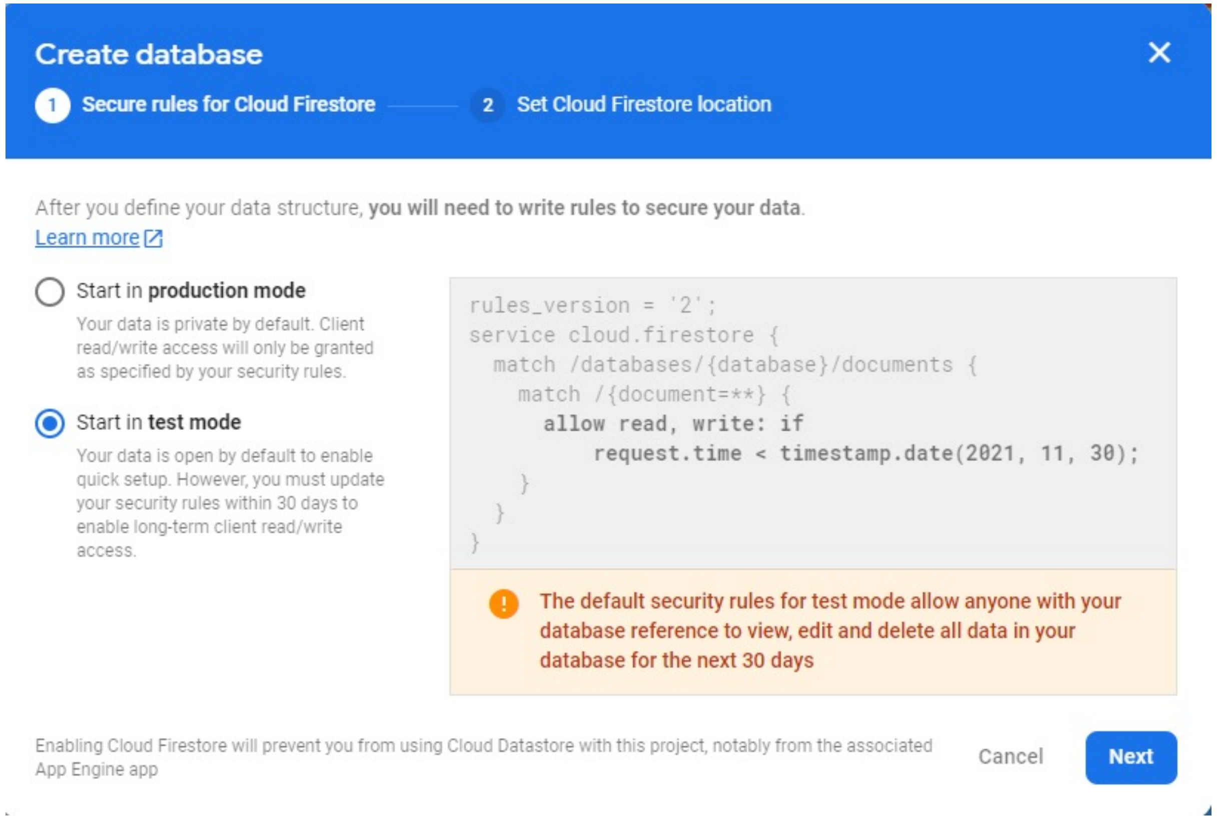


- Then click *Create Database*



Setting up Cloud Firestore (cont.)

- Choose *Start in test mode*
  - Here we are allowing read & write. This should be changed for production.



Create database
1 Secure rules for Cloud Firestore
2 Set Cloud Firestore location

After you define your data structure, **you will need to write rules to secure your data.**
Learn more

( ) Start in **production mode**
Your data is private by default. Client read/write access will only be granted as specified by your security rules.

(●) Start in **test mode**
Your data is open by default to enable quick setup. However, you must update your security rules within 30 days to enable long-term client read/write access.

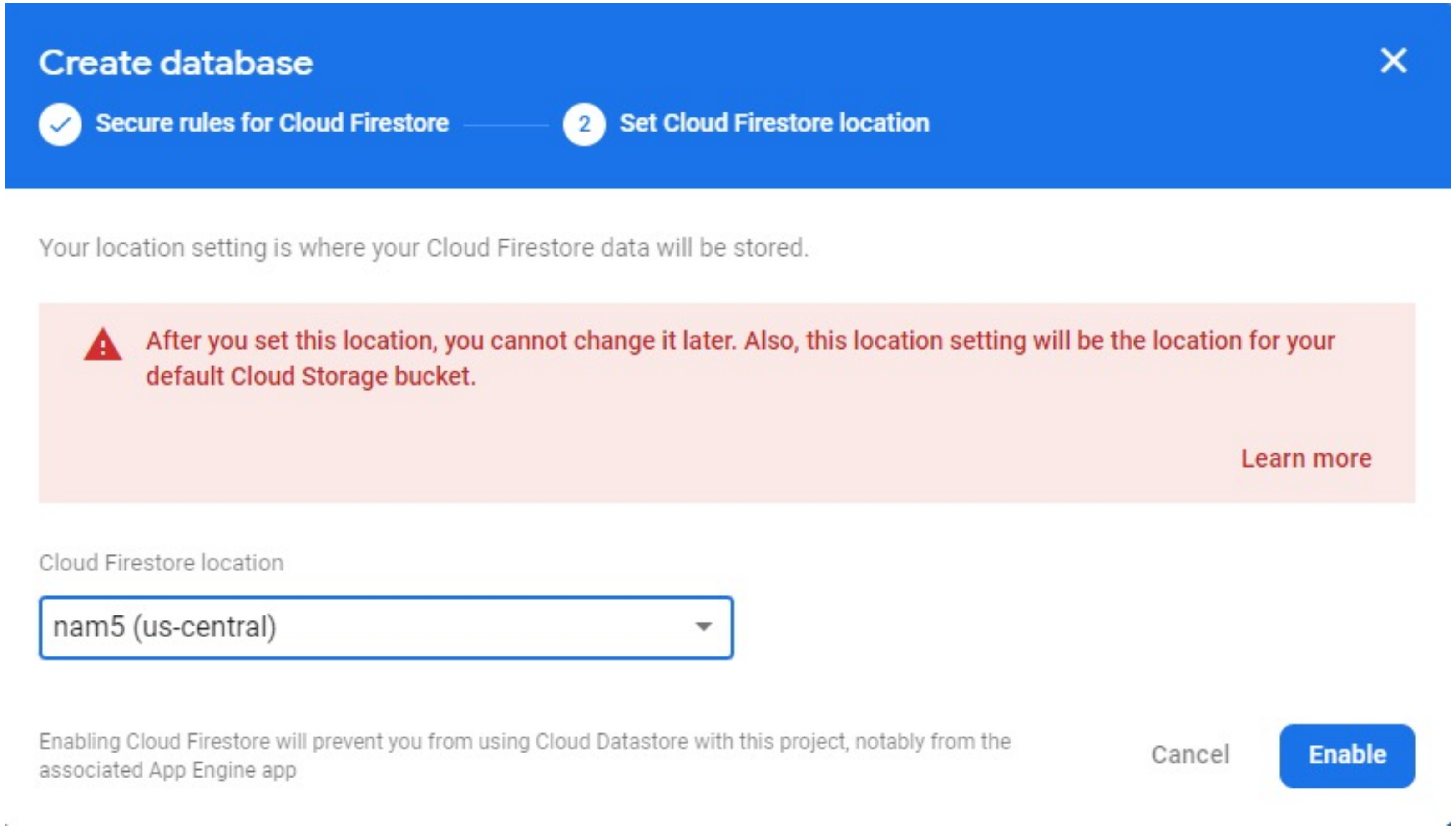
```
rules_version = '2';
service cloud.firestore {
  match /databases/{database}/documents {
    match /{document=**} {
      allow read, write: if
          request.time < timestamp.date(2021, 11, 30);
    }
  }
}
```

The default security rules for test mode allow anyone with your database reference to view, edit and delete all data in your database for the next 30 days

Enabling Cloud Firestore will prevent you from using Cloud Datastore with this project, notably from the associated App Engine app

Setting up Cloud Firestore (cont.)

- Choose a *us-central* location for your database



Create database

Secure rules for Cloud Firestore — 2 Set Cloud Firestore location

Your location setting is where your Cloud Firestore data will be stored.

After you set this location, you cannot change it later. Also, this location setting will be the location for your default Cloud Storage bucket.

Learn more

Cloud Firestore location

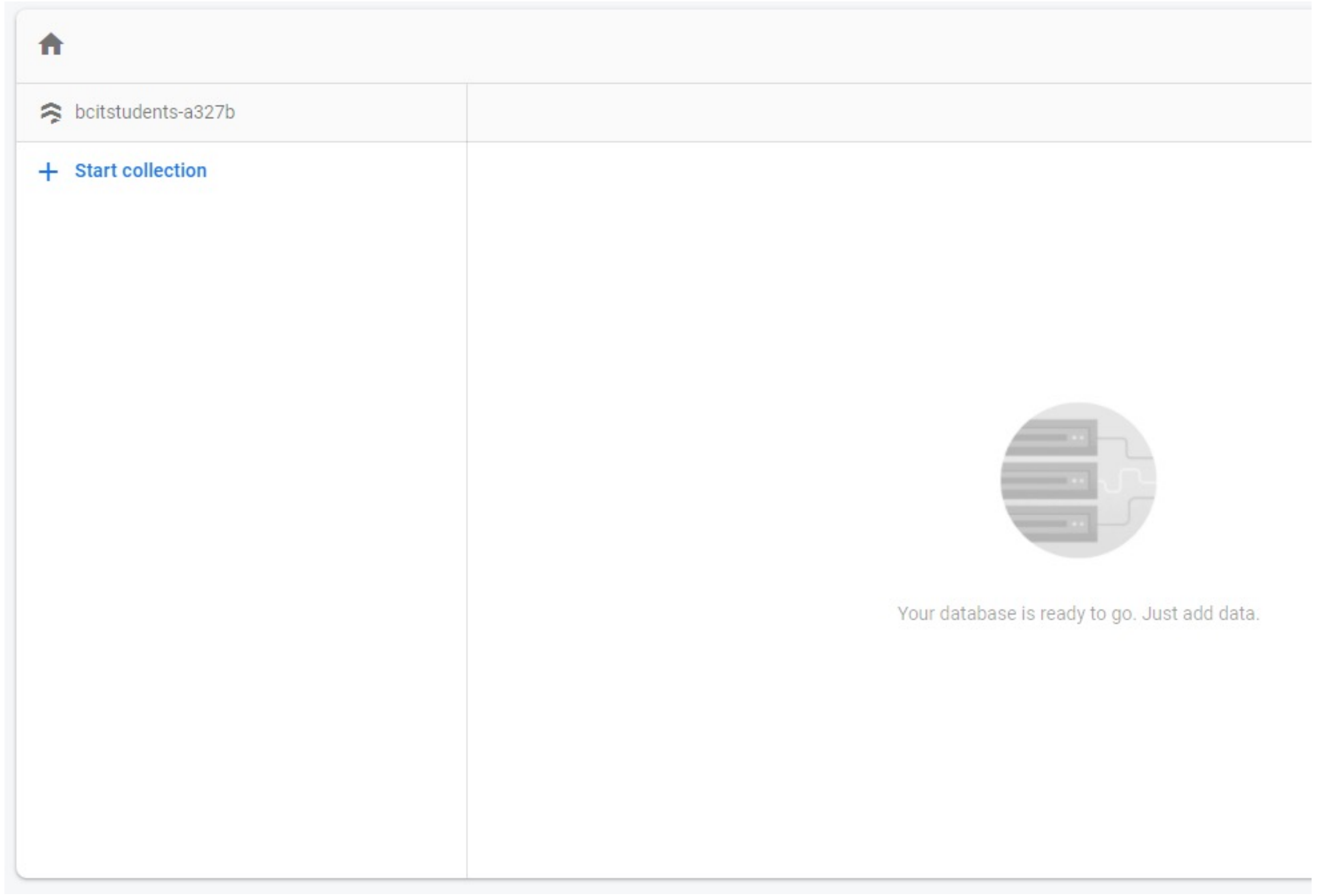
nam5 (us-central)

Enabling Cloud Firestore will prevent you from using Cloud Datastore with this project, notably from the associated App Engine app

Cancel Enable

Setting up a Cloud Firestore (cont.)

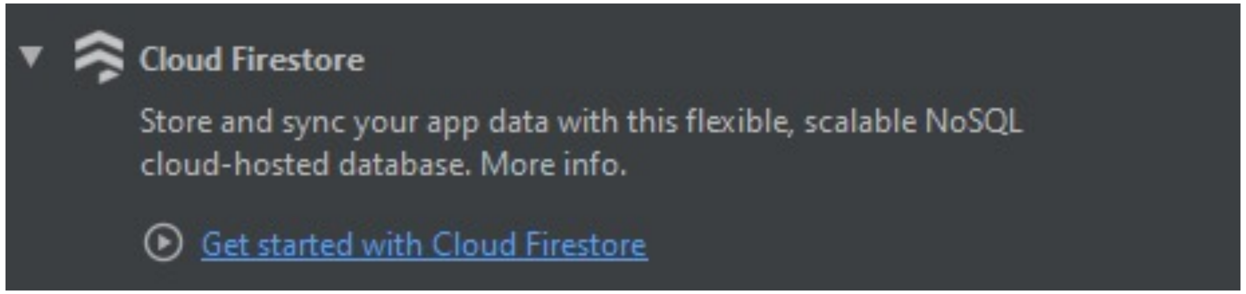
- That is it for the Firebase console for now, the data for you database is displayed here



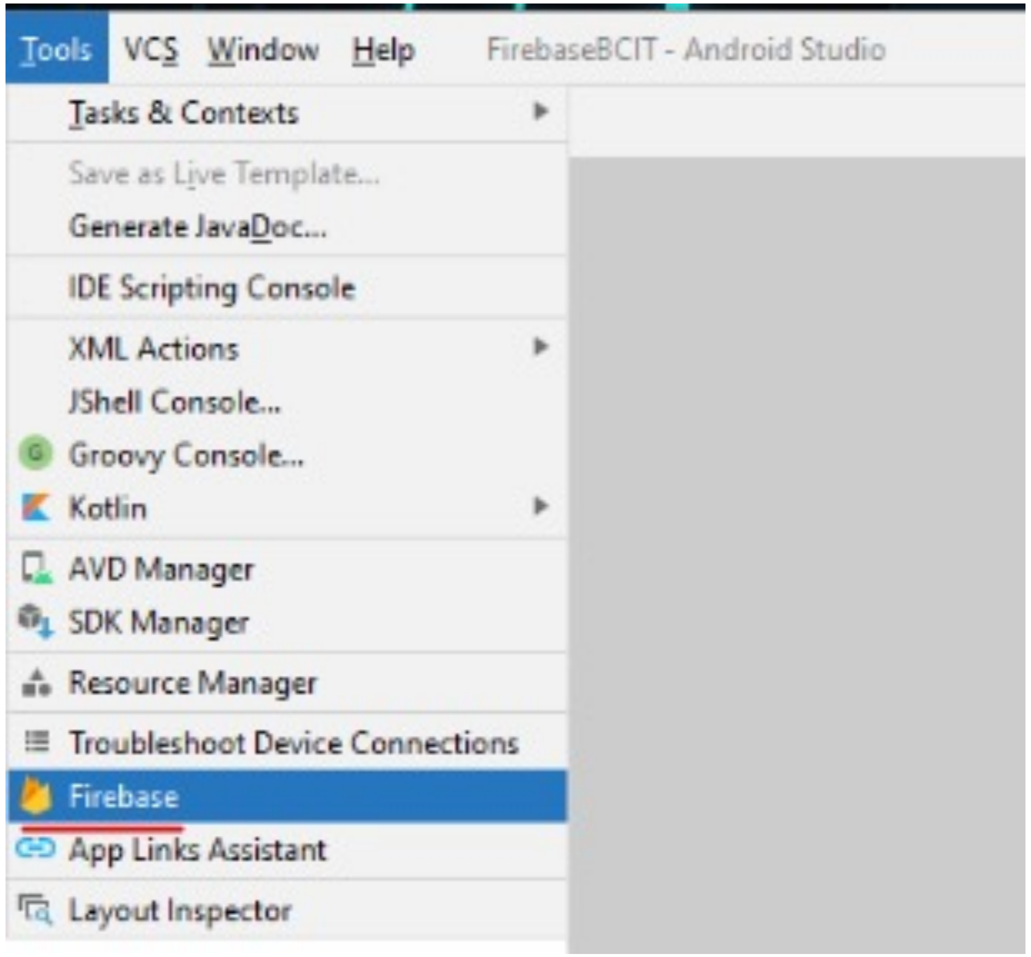
Connecting Firebase to app

- Next step is to connect the database to our app, go to *Tools->Firebase* in Android Studio

- Then click *Get Started with Cloud Firestore*

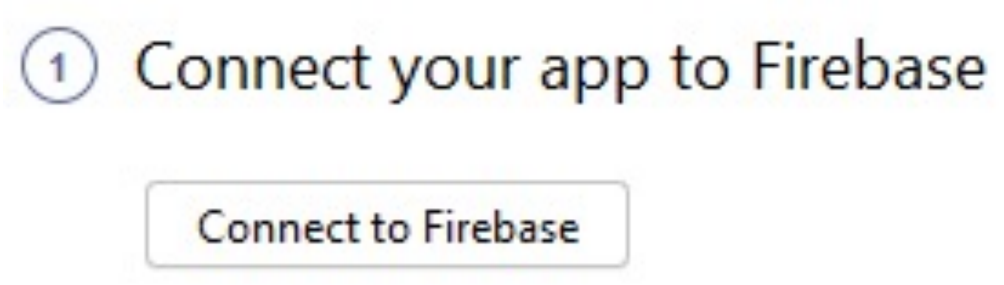


Store and sync your app data with this flexible, scalable NoSQL cloud-hosted database. More info.



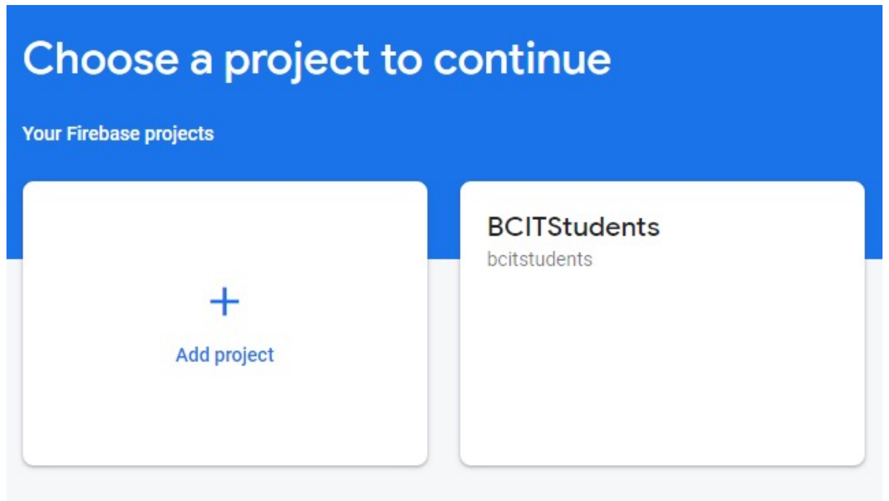
Connecting Firebase to app (cont.)

- Click *Connect to Firebase*



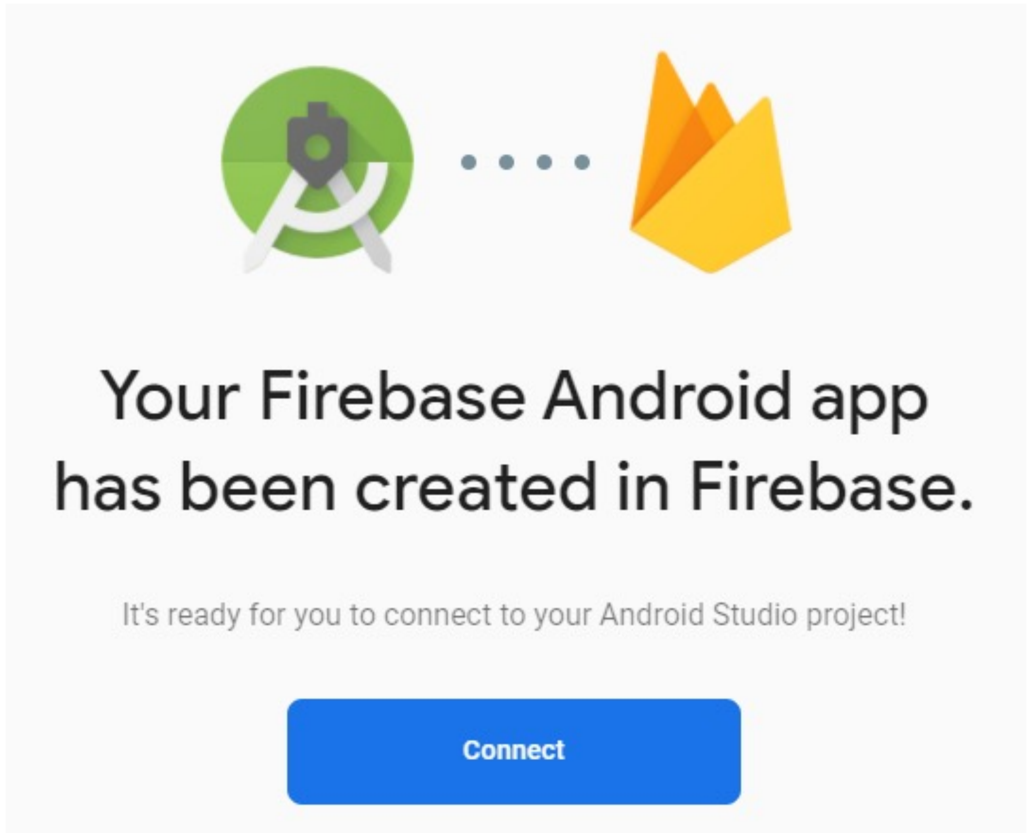
  (You may be prompted to build your project if you haven't yet. If so, click *Connect to Firebase* again once built)

- Your browser will open and you will be prompted to select a project to continue
  - Choose your project!



Connecting Firebase to app (cont.)

- Hit *Connect* when prompted

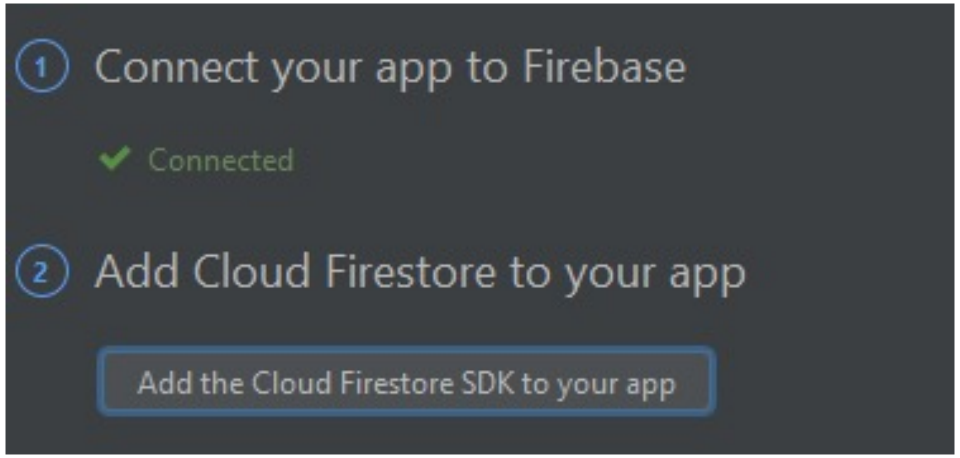


Your Firebase Android app has been created in Firebase.

It's ready for you to connect to your Android Studio project!

Adding Cloud Firestore to app

- Next click on *Add the Cloud Firestore SDK to your app*

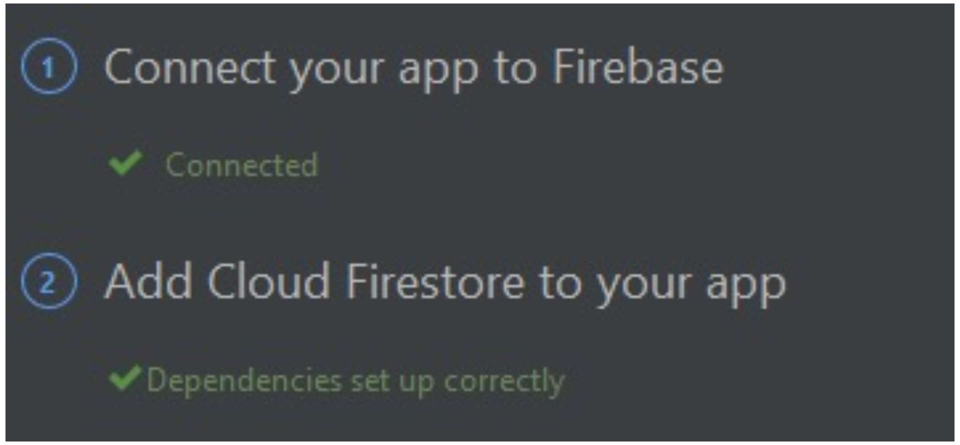


① Connect your app to Firebase

✓ Connected

② Add Cloud Firestore to your app

- Android Studio will prompt that is making some changes, click *Accept Changes*



① Connect your app to Firebase

✓ Connected

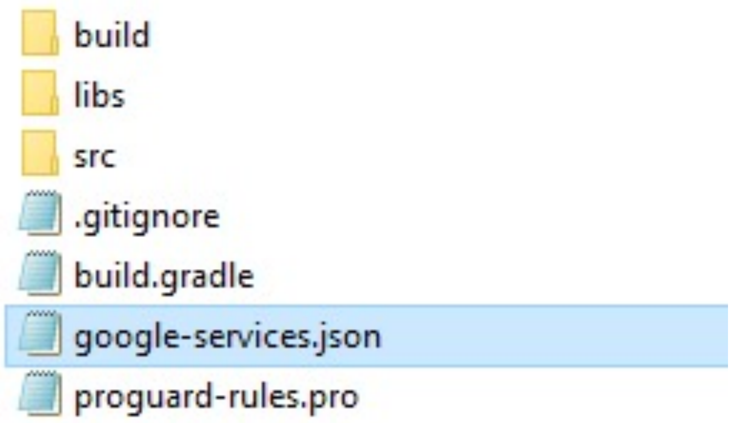
② Add Cloud Firestore to your app

✓ Dependencies set up correctly

- The app will then rebuild

Adding Cloud Firestore to app (cont.)

- Inside your *app* directory you will find a *google-services.json* file
    - It is not visible within android studio, only file system
- *google-services.json* contains the credentials needed for accessing your Firebase account
- When working with source control, you want to add this to your *.gitignore* file



Create a model class

- The model class should be *Person*
  - firstName (*String*)
  - lastName (*String*)
  - location (*String*)
- Create a constructor and pass through the variables above
- Firebase requires java objects to have *getters/setters*, so add those too

Google Maps

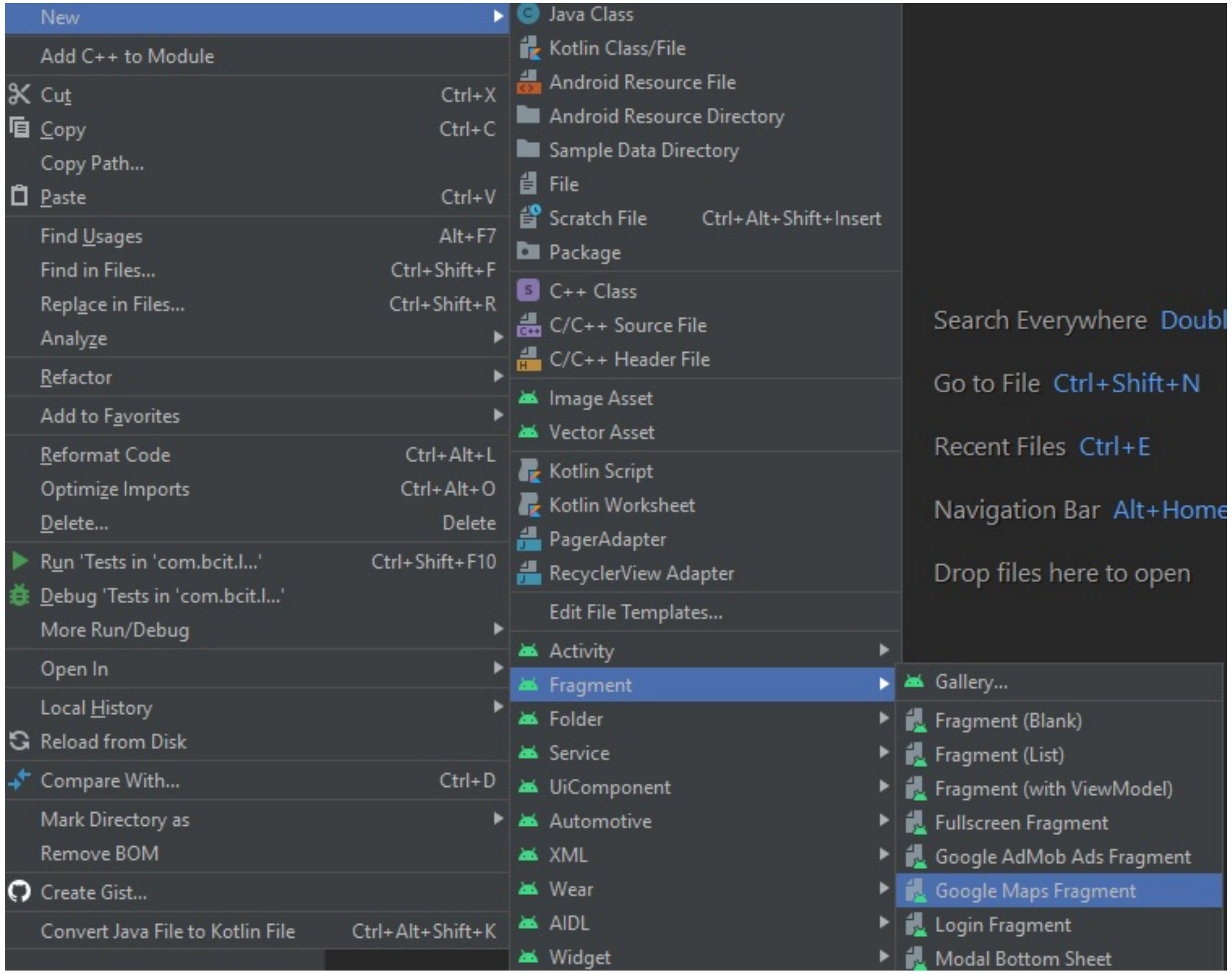
- There are different ways we can display a map in our app
  - *MapView*
  - *MapFragment*

- The framework provides us with a template for the simplest and most modern way to place a map in an application
  - ‘Google Maps Fragment’

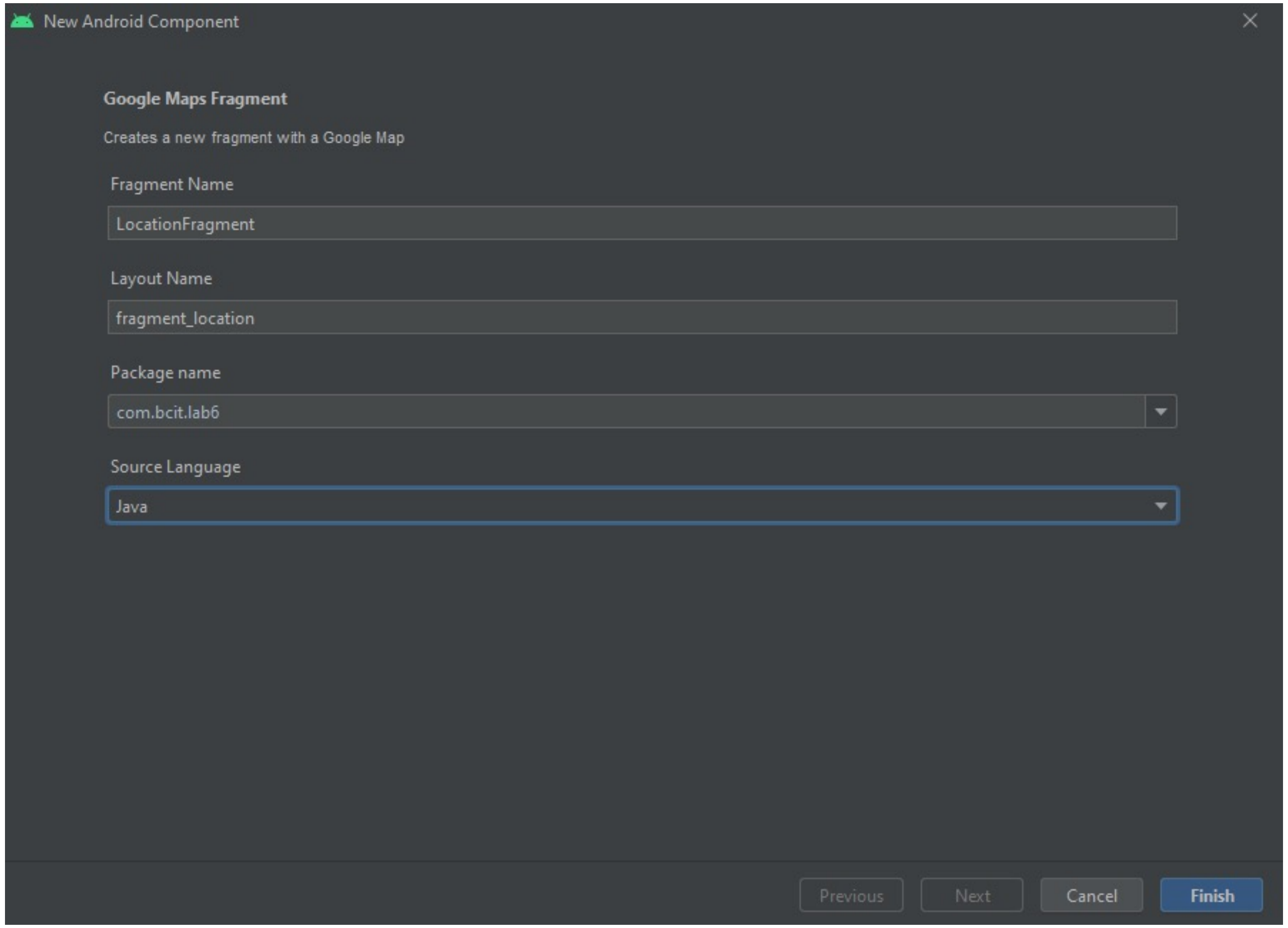
Create a Google Maps Fragment (cont.)

- The ‘Google Maps Fragment’ template provides us with a *SupportMapFragment* as a child *Fragment*
- A *MapFragment* (or *SupportMapFragment*) manages all the lifecycle events needed to use a map in a app
- If we used a *MapView* instead, we would need manage the lifecycle events

Create a Google Maps Fragment (cont.)



Create a Google Maps Fragment (cont.)



Google Maps Fragment

Creates a new fragment with a Google Map

Fragment Name: LocationFragment

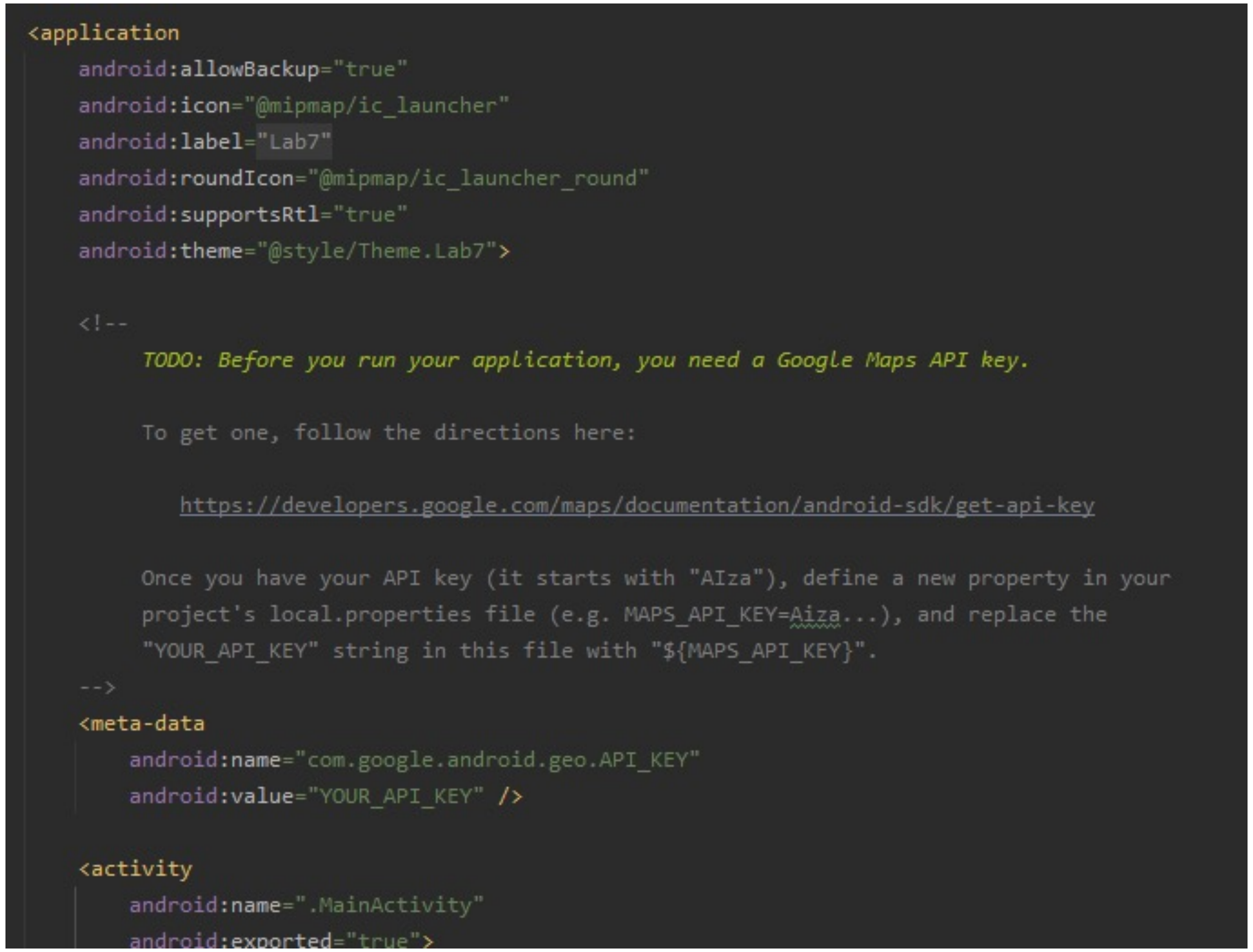
Layout Name: fragment_location

Package name: com.bcit.lab6

Source Language: Java

Create a Google Maps Fragment (cont.)

Look at your generated code in your manifest and follow the instructions

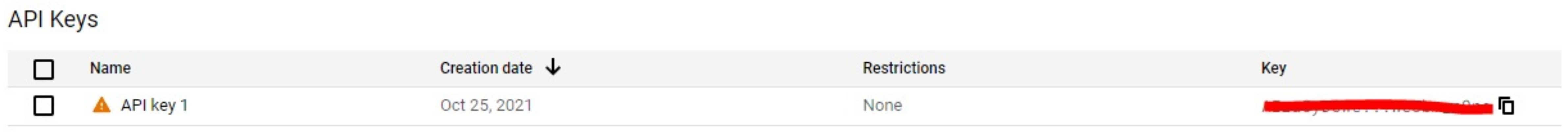


```
<application
    android:allowBackup="true"
    android:icon="@mipmap/ic_launcher"
    android:label="Lab7"
    android:roundIcon="@mipmap/ic_launcher_round"
    android:supportsRtl="true"
    android:theme="@style/Theme.Lab7">

    <!--
         TODO: Before you run your application, you need a Google Maps API key.

         To get one, follow the directions here:

            https://developers.google.com/maps/documentation/android-sdk/get-api-key

         Once you have your API key (it starts with "AIza"), define a new property in your
         project's local.properties file (e.g. MAPS_API_KEY=Aiza...), and replace the
         "YOUR_API_KEY" string in this file with "${MAPS_API_KEY}".
    -->
    <meta-data
        android:name="com.google.android.geo.API_KEY"
        android:value="YOUR_API_KEY" />

    <activity
        android:name=".MainActivity"
        android:exported="true">
```

Create a Google Maps Fragment (cont.)

- You will need to create a project in the Google Cloud Platform and enable the API Maps for Android

- Follow the instruction in the file and create an API Key if one wasn't generated for you

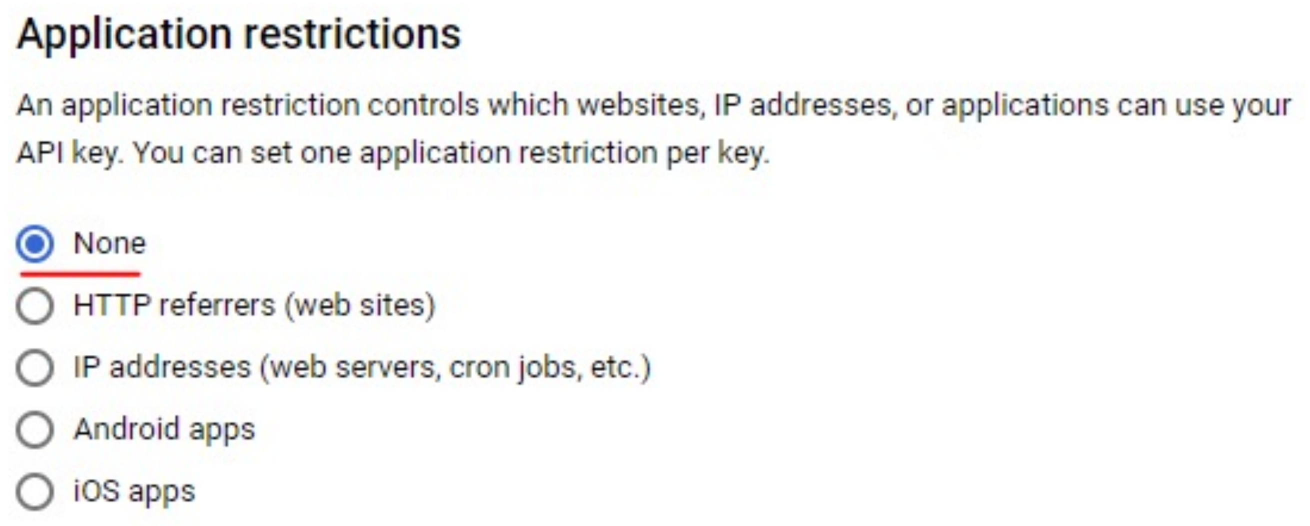


API Keys

	Name	Creation date ↓	Restrictions	Key
☐	API key 1	Oct 25, 2021	None	[redacted]

Create a Google Maps Fragment (cont.)

- Make sure there are no restrictions with you API key



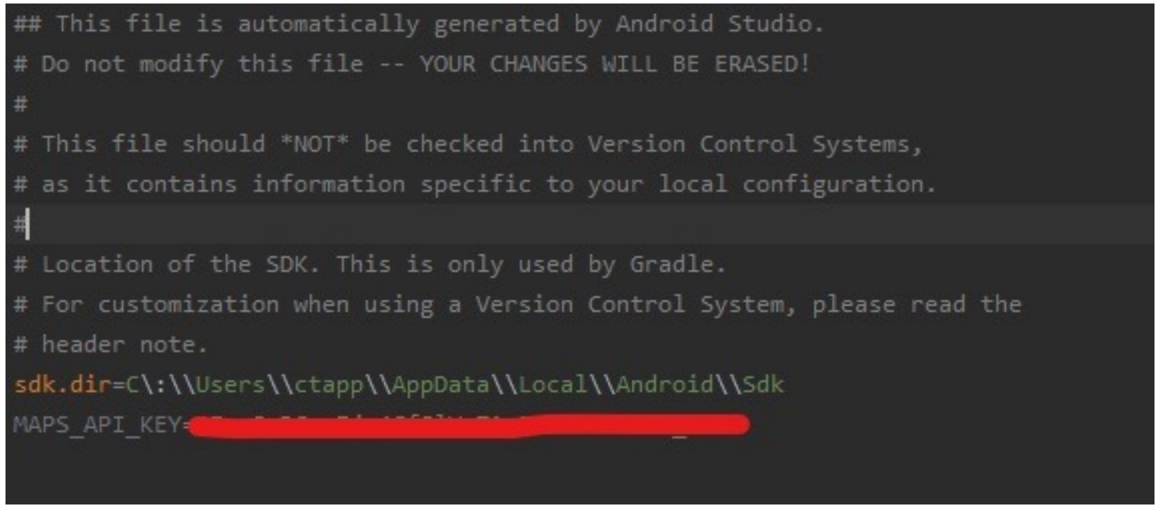
Application restrictions

An application restriction controls which websites, IP addresses, or applications can use your API key. You can set one application restriction per key.

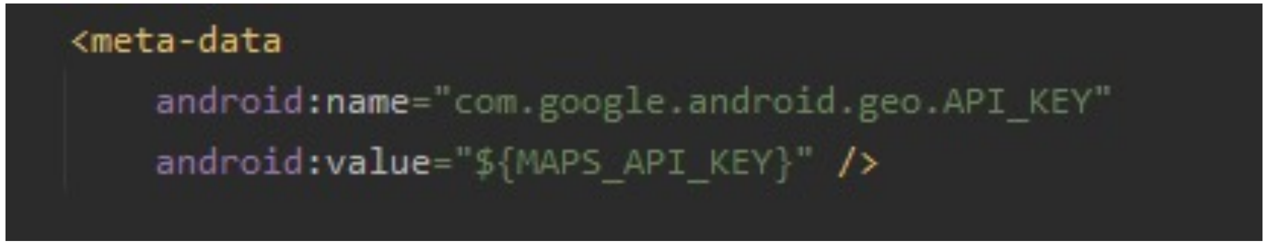
[✓ None]
HTTP referrers (web sites)
IP addresses (web servers, cron jobs, etc.)
Android apps
iOS apps

Create a Google Maps Fragment (cont.)

- Update your local.properties and manifest



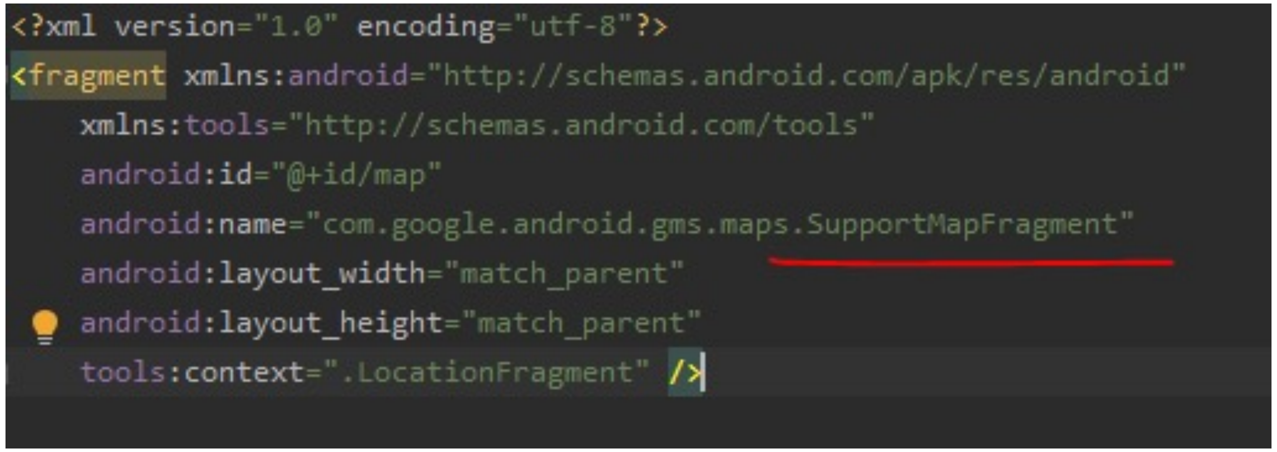
```
## This file is automatically generated by Android Studio.
# Do not modify this file -- YOUR CHANGES WILL BE ERASED!
#
# This file should *NOT* be checked into Version Control Systems,
# as it contains information specific to your local configuration.
#
# Location of the SDK. This is only used by Gradle.
# For customization when using a Version Control System, please read the
# header note.
sdk.dir=C\:\\Users\\ctapp\\AppData\\Local\\Android\\Sdk
MAPS_API_KEY=[redacted]
```



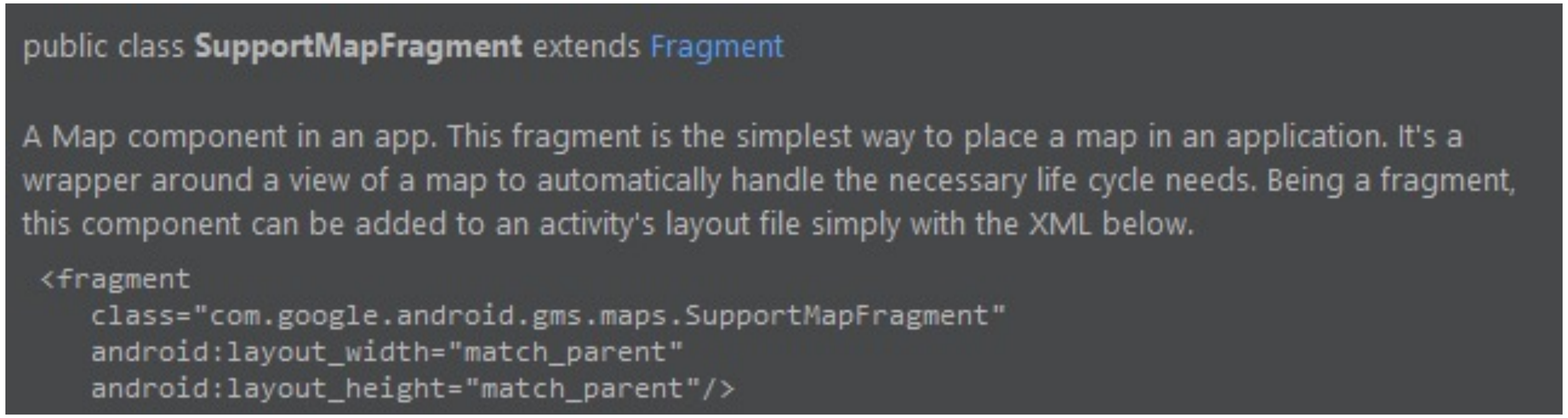
```
<meta-data
    android:name="com.google.android.geo.API_KEY"
    android:value="${MAPS_API_KEY}" />
```

Create a Google Maps Fragment (cont.)

- Look at the code of generated layout file and check out *SupportMapFragment*



```
<?xml version="1.0" encoding="utf-8"?>
<fragment xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/map"
    android:name="com.google.android.gms.maps.SupportMapFragment"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".LocationFragment" />
```



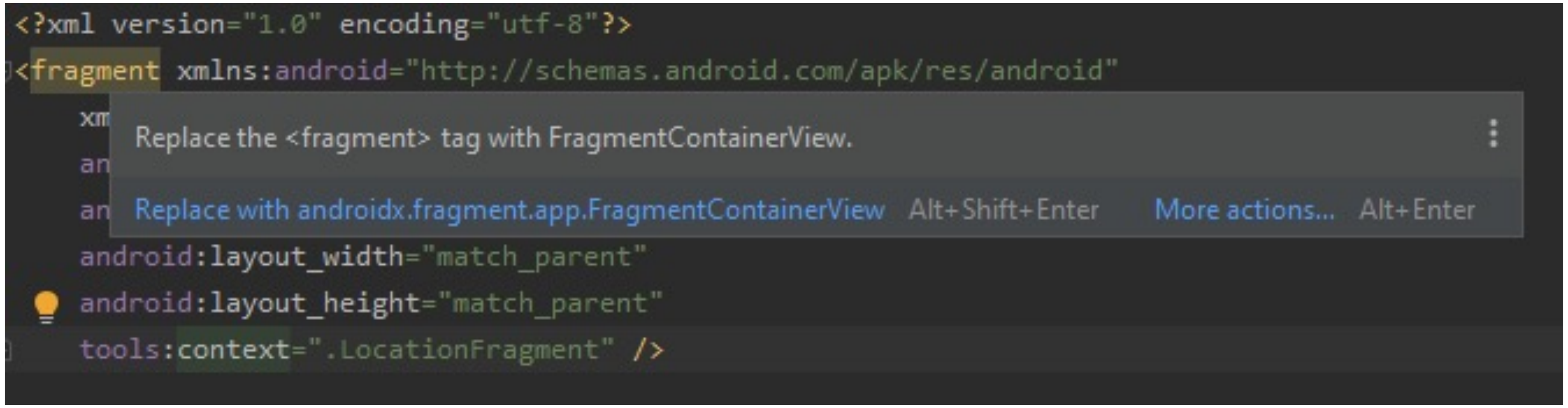
public class **SupportMapFragment** extends Fragment

A Map component in an app. This fragment is the simplest way to place a map in an application. It's a wrapper around a view of a map to automatically handle the necessary life cycle needs. Being a fragment, this component can be added to an activity's layout file simply with the XML below.

```
<fragment
   class="com.google.android.gms.maps.SupportMapFragment"
   android:layout_width="match_parent"
   android:layout_height="match_parent"/>
```

Create a Google Maps Fragment (cont.)

- Also, if you are curious what the *<fragment>* tag is here, it is basically what *FragmentContainerView* used to be in older versions
- You can replace it



```
<?xml version="1.0" encoding="utf-8"?>
<fragment xmlns:android="http://schemas.android.com/apk/res/android"
    xm
    an
    an
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".LocationFragment" />
```

Replace the <fragment> tag with FragmentContainerView.

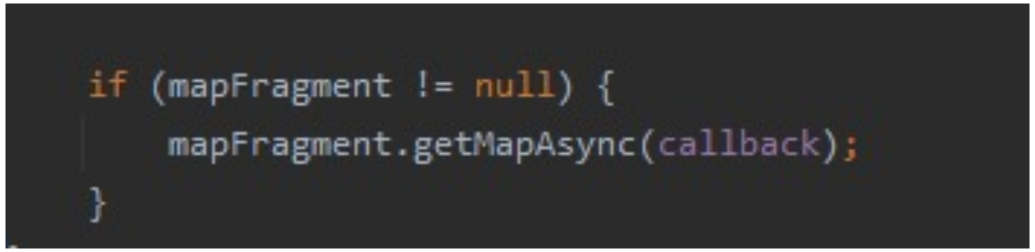
Replace with androidx.fragment.app.FragmentContainerView Alt+Shift+Enter More actions... Alt+Enter

Create a Google Maps Fragment (cont.)

- Look at *onViewCreated* in your *LocationFragment*
- *LocationFragment* uses a *<fragment>* (or *FragmentContainerView*) as it's parent layout
  - A fragment within a fragment!
- Therefore, we use the helper function *getChildFragmentManager* to get a reference to the *SupportMapFragment*
- Take some time to make sure you understand what is happening here ☺

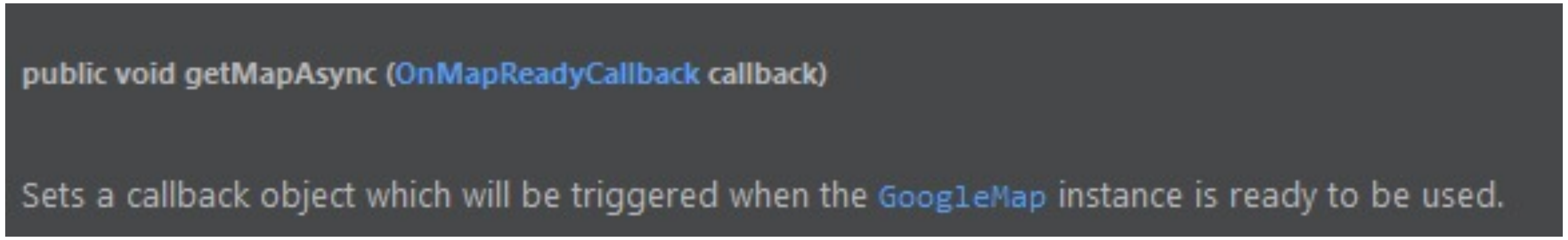
Create a Google Maps Fragment (cont.)

- The second part of *onViewCreated*



```
if (mapFragment != null) {
    mapFragment.getMapAsync(callback);
}
```

- Sets a callback to be used for the *MapFragment*

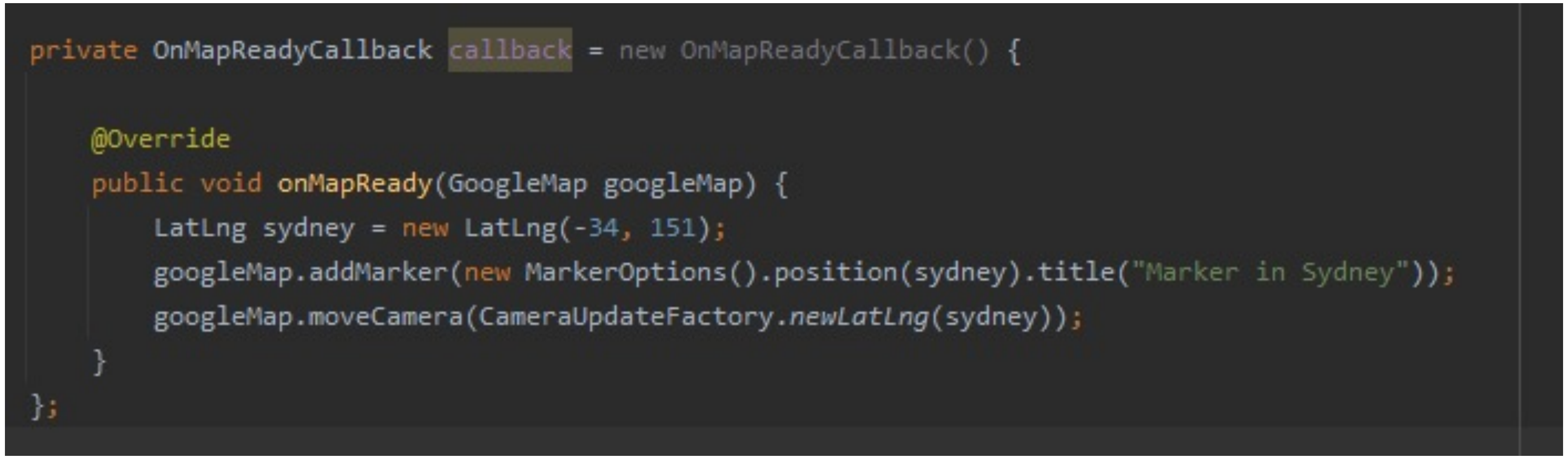


public void getMapAsync (OnMapReadyCallback callback)

Sets a callback object which will be triggered when the GoogleMap instance is ready to be used.

Create a Google Maps Fragment (cont.)

- The callback itself, is created for us inside the our *LocationFragment*
- *Inside onMapReady* is where we can add functionality to update our *Map*



```
private OnMapReadyCallback callback = new OnMapReadyCallback() {

    @Override
    public void onMapReady(GoogleMap googleMap) {
        LatLng sydney = new LatLng(-34, 151);
        googleMap.addMarker(new MarkerOptions().position(sydney).title("Marker in Sydney"));
        googleMap.moveCamera(CameraUpdateFactory.newLatLng(sydney));
    }
};
```

Create a Google Maps Fragment (cont.)

- You can see the default location is set to Sydney, Australia
- A marker is added on the map to that location

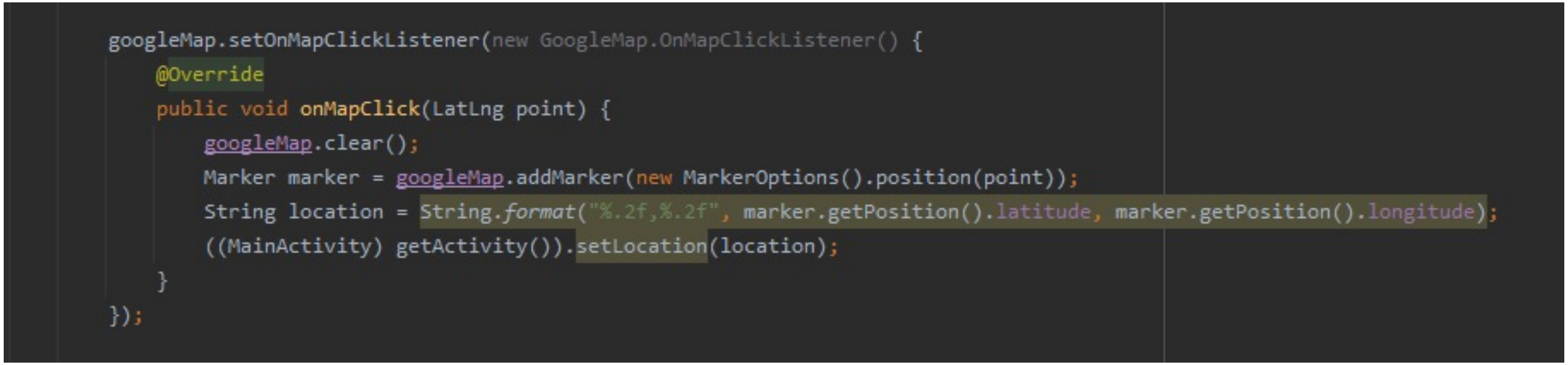
  Pro Tip: To zoom in/out on a map with your emulator:

  Double click with your cursor and hold the second click. Now pan either up/down

- For this lab, we want to be able to click on the map, and retrieve that location

Create a Google Maps Fragment (cont.)

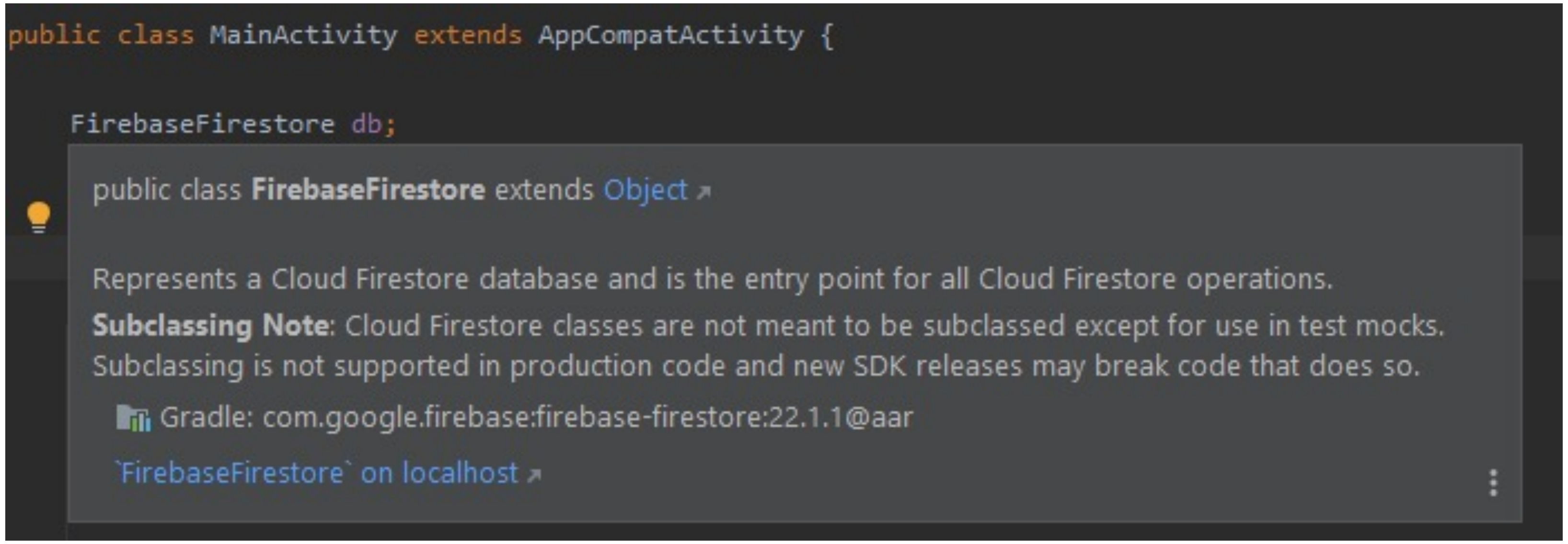
- See if you can piece together what this code is doing below
- This also assumes you have a function in MainActivity (*setLocation*)



```
googleMap.setOnMapClickListener(new GoogleMap.OnMapClickListener() {
    @Override
    public void onMapClick(LatLng point) {
        googleMap.clear();
        Marker marker = googleMap.addMarker(new MarkerOptions().position(point));
        String location = String.format("%.2f,%.2f", marker.getPosition().latitude, marker.getPosition().longitude);
        ((MainActivity) getActivity()).setLocation(location);
    }
});
```

Writing to our database

- Inside your *MainActivity.java* add an instance variable *FirebaseFirestore*
  - The entry point for all Cloud Firestore operations



```
public class MainActivity extends AppCompatActivity {

    FirebaseFirestore db;
```

public class **FirebaseFirestore** extends Object ↗

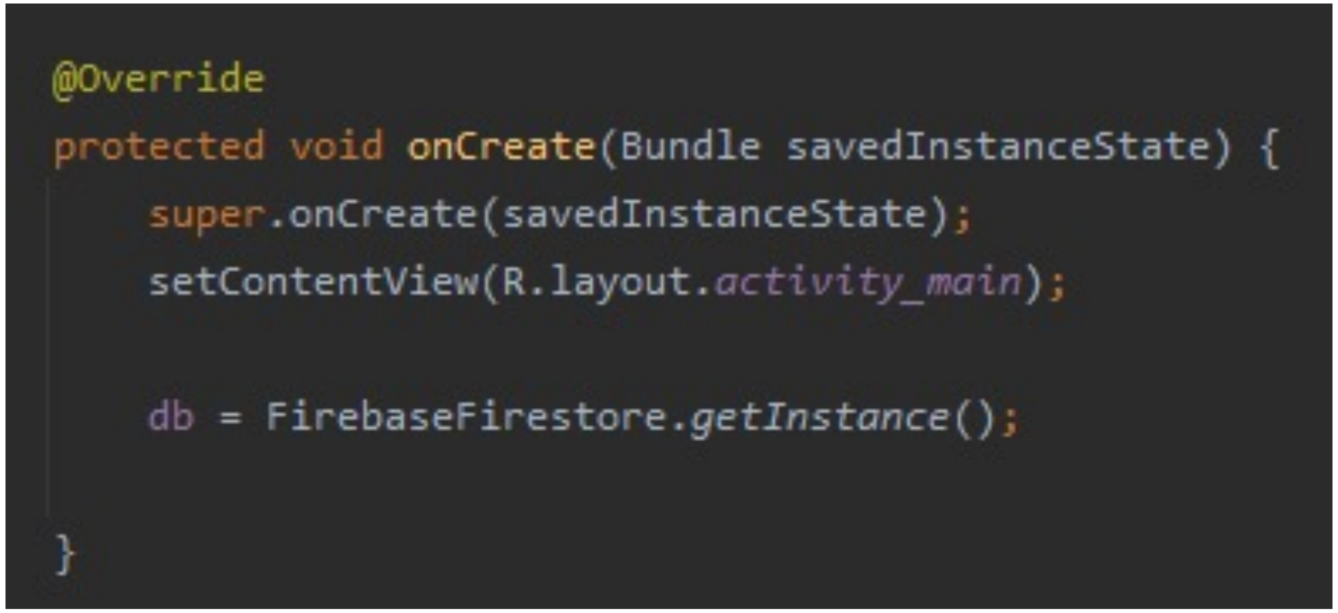
Represents a Cloud Firestore database and is the entry point for all Cloud Firestore operations.
Subclassing Note: Cloud Firestore classes are not meant to be subclassed except for use in test mocks. Subclassing is not supported in production code and new SDK releases may break code that does so.

Gradle: com.google.firebase:firebase-firestore:22.1.1@aar

`FirebaseFirestore` on localhost ↗

Writing to our database (cont.)

- Let's access a *FirebaseFirestore* instance by assigning one to our variable

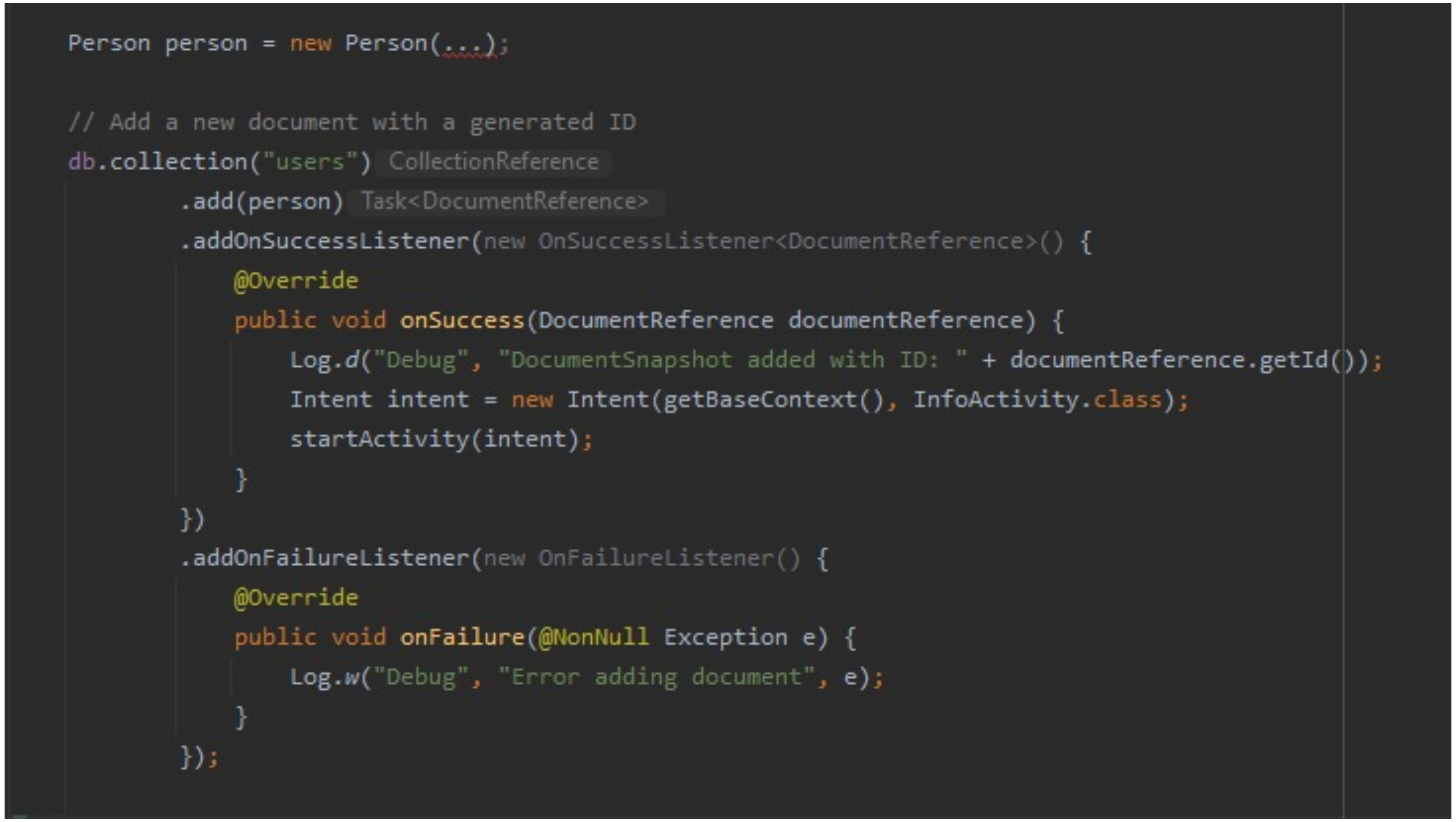


```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    db = FirebaseFirestore.getInstance();

}
```

Writing to our database (cont.)

- Next, create a function called *AddPerson,* this is what will get called when we click the *Button*
- I will provide the code needed to add new data to the database, but you need to provide the data
- From what we have so far, how can we get the firstname, lastname and location and fill it into our *Person* object?
  - code snippet on next slide

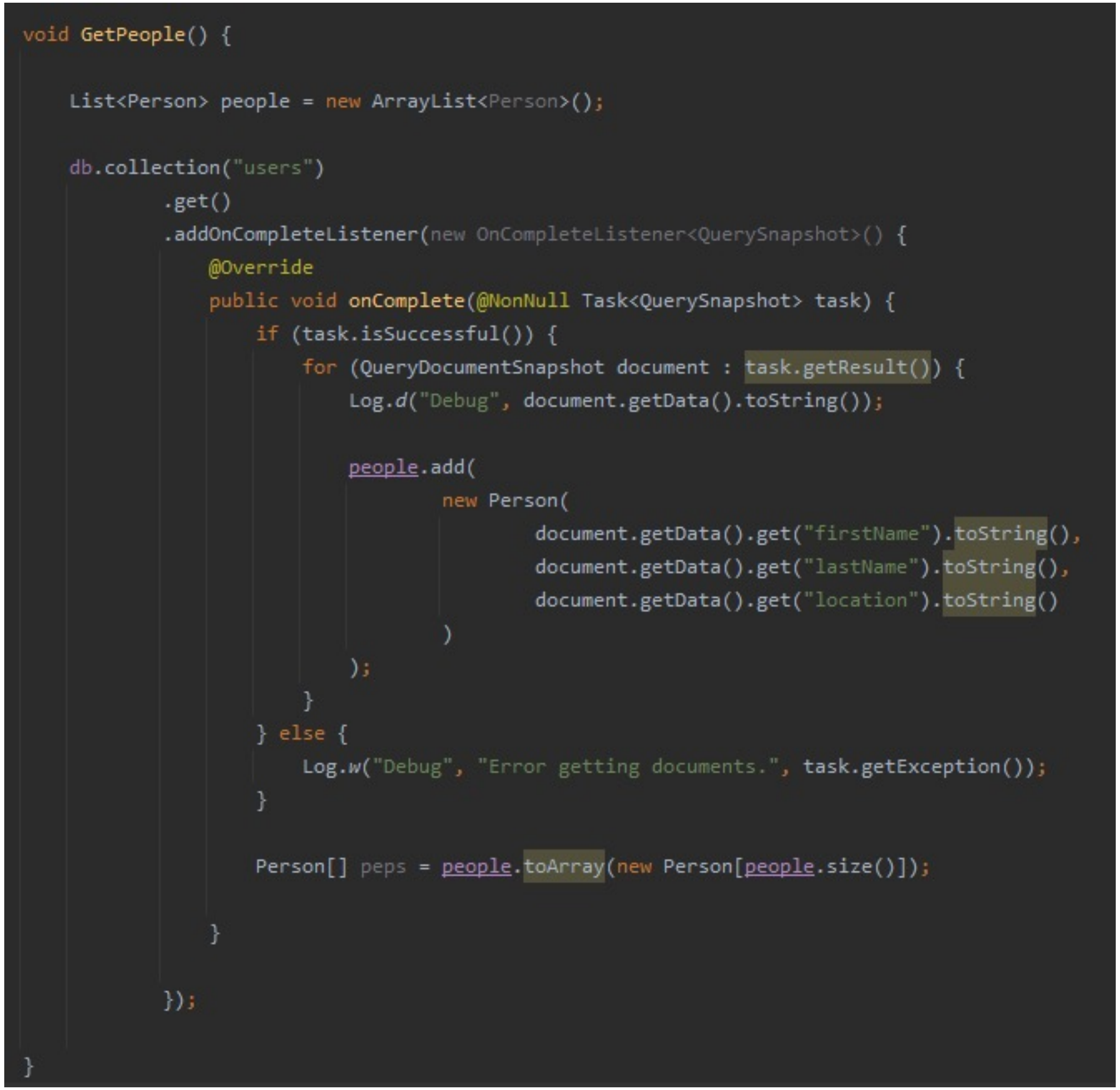
Writing to our database (cont.)



```java
Person person = new Person(...);

// Add a new document with a generated ID
db.collection("users") CollectionReference
        .add(person) Task<DocumentReference>
        .addOnSuccessListener(new OnSuccessListener<DocumentReference>() {
            @Override
            public void onSuccess(DocumentReference documentReference) {
                Log.d("Debug", "DocumentSnapshot added with ID: " + documentReference.getId());
                Intent intent = new Intent(getBaseContext(), InfoActivity.class);
                startActivity(intent);
            }
        })
        .addOnFailureListener(new OnFailureListener() {
            @Override
            public void onFailure(@NonNull Exception e) {
                Log.w("Debug", "Error adding document", e);
            }
        });
```

Reading from our database

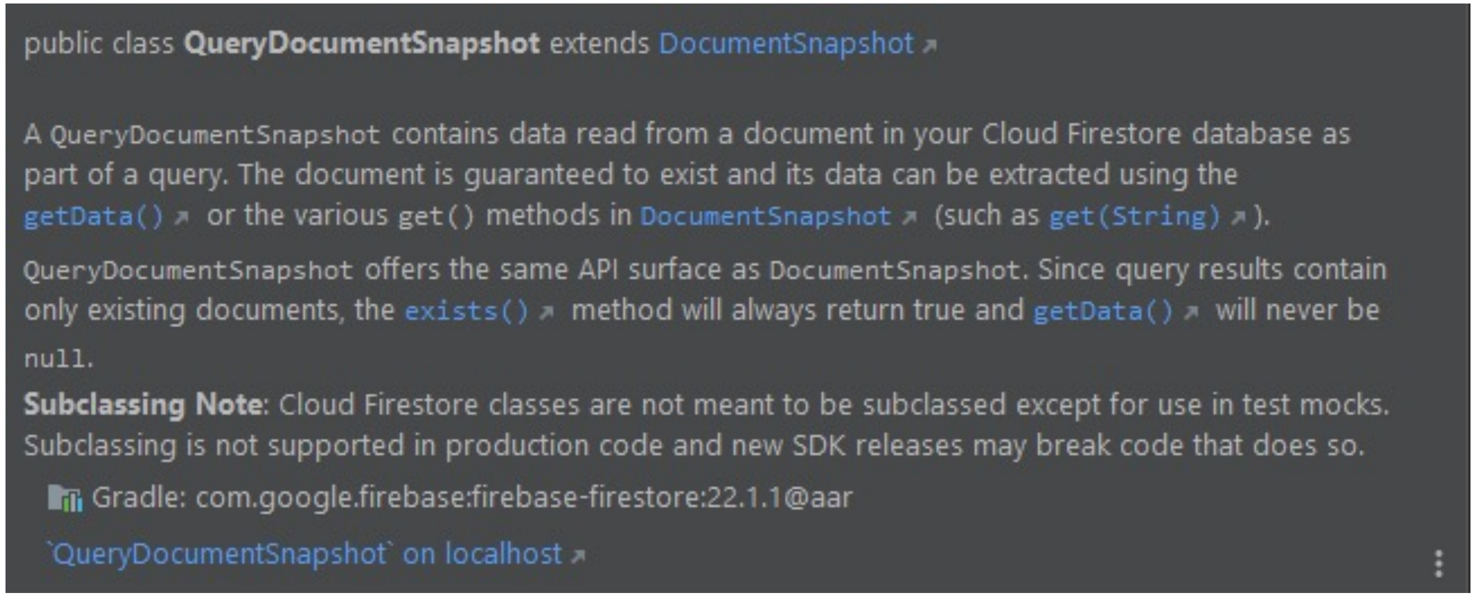
- Now that we can *add* people to our database, we need to *get* them in our *InfoActivity,* and update our *RecyclerView.*
- I will provide you with most the code for retrieving the data, you just need to update your *RecyclerView* with it
- Also make sure to create another instance for your *FirebaseFirestore* in *InfoActivity*



```java
void GetPeople() {

    List<Person> people = new ArrayList<Person>();

    db.collection("users")
            .get()
            .addOnCompleteListener(new OnCompleteListener<QuerySnapshot>() {
                @Override
                public void onComplete(@NonNull Task<QuerySnapshot> task) {
                    if (task.isSuccessful()) {
                        for (QueryDocumentSnapshot document : task.getResult()) {
                            Log.d("Debug", document.getData().toString());

                            people.add(
                                    new Person(
                                            document.getData().get("firstName").toString(),
                                            document.getData().get("lastName").toString(),
                                            document.getData().get("location").toString()
                                    )
                            );
                        }
                    } else {
                        Log.w("Debug", "Error getting documents.", task.getException());
                    }

                    Person[] peps = people.toArray(new Person[people.size()]);

                }

            });

}
```

Reading from our database (cont.)

- I assume most of you all have used Firebase databases before.
- Make sure you understand the difference between a *collection* and a *document*
- We read the data from our database by querying *snapshots*



public class **QueryDocumentSnapshot** extends DocumentSnapshot ↗

A `QueryDocumentSnapshot` contains data read from a document in your Cloud Firestore database as part of a query. The document is guaranteed to exist and its data can be extracted using the `getData()` ↗ or the various `get()` methods in `DocumentSnapshot` ↗ (such as `get(String)` ↗).

`QueryDocumentSnapshot` offers the same API surface as `DocumentSnapshot`. Since query results contain only existing documents, the `exists()` ↗ method will always return true and `getData()` ↗ will never be `null`.

Subclassing Note: Cloud Firestore classes are not meant to be subclassed except for use in test mocks. Subclassing is not supported in production code and new SDK releases may break code that does so.

Gradle: com.google.firebase:firebase-firestore:22.1.1@aar

`QueryDocumentSnapshot` on localhost ↗

Async

- Notice that when we *add* and *get* data to our Firestore database it is done through a *Task*
- Task: *Represents an asynchronous operation*
- You will notice when you click the *Button* there is a delay before going to the *InfoActivity*
- This is because we are waiting for the database to update, then let us know before we start the *Intent*