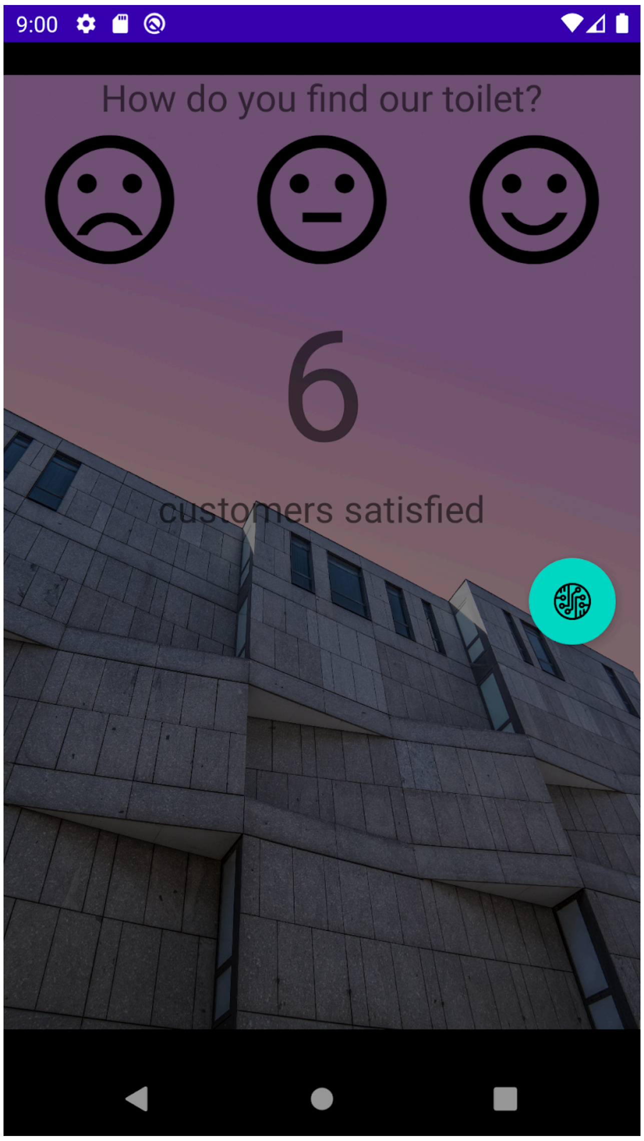
Lesson 5 - Firebase

Introduction

This is a simple app that introduces the code that you need to write to read/write data to firebase. It is a common app that you see in toilets and cashiers.

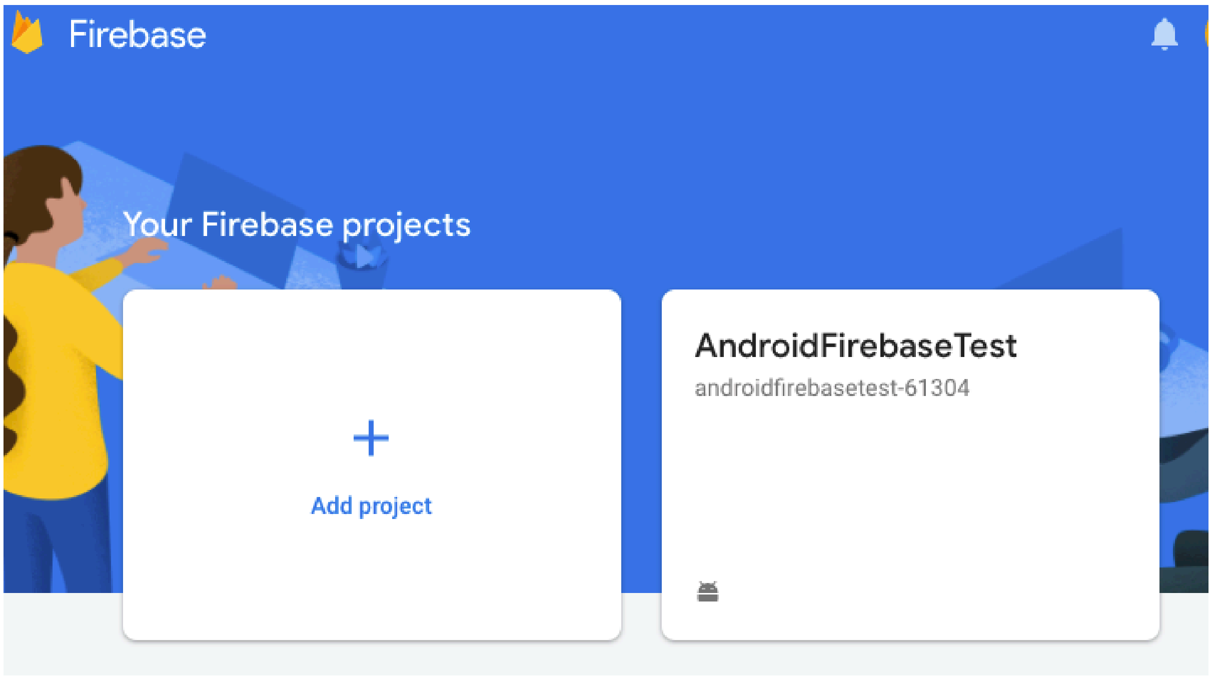
Credits: this Lesson has contributions by Kenny Lu, Norman Lee.



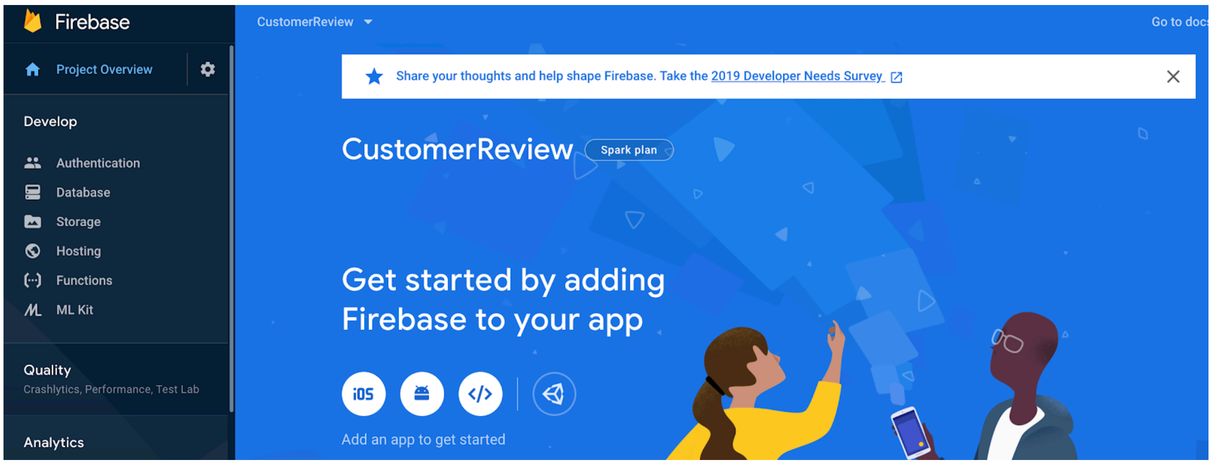
Set up your firebase project

1. Start a new project in Android studio, you will need it in Step 5. OR Alternatively, launch the starter project that you are provided with.

2. Go to console.firebase.google.com and create a new project by selecting Create a Project if you are the first time user of firebase, or add a project if you are an existing user of firebase. You would not need to add Google Analytics to your project at this point.



3. After the project set-up is completed, you should see this screen. Click on the android icon.

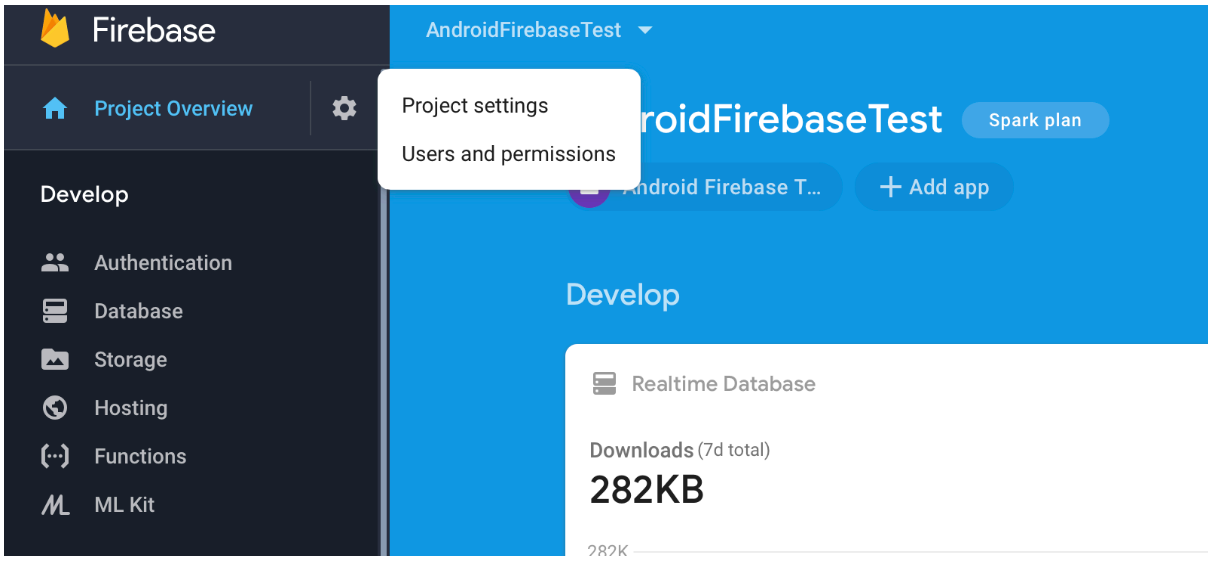


Share your thoughts and help shape Firebase. Take the 2019 Developer Needs Survey

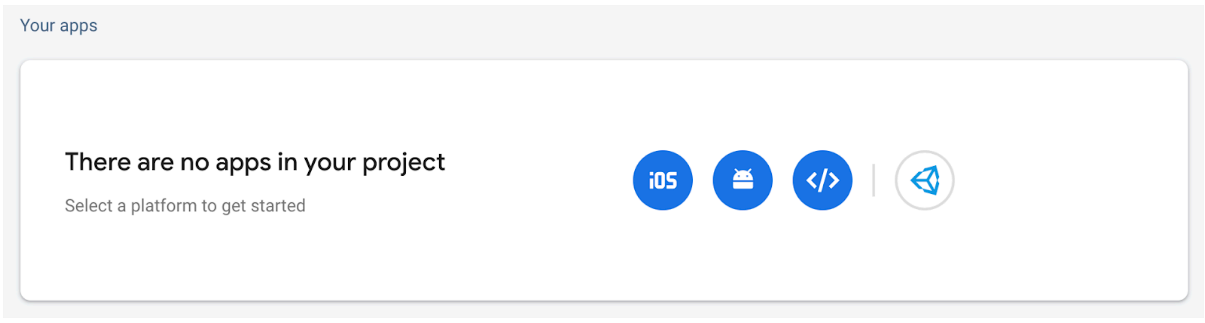
Get started by adding Firebase to your app

Updated Mar 2024

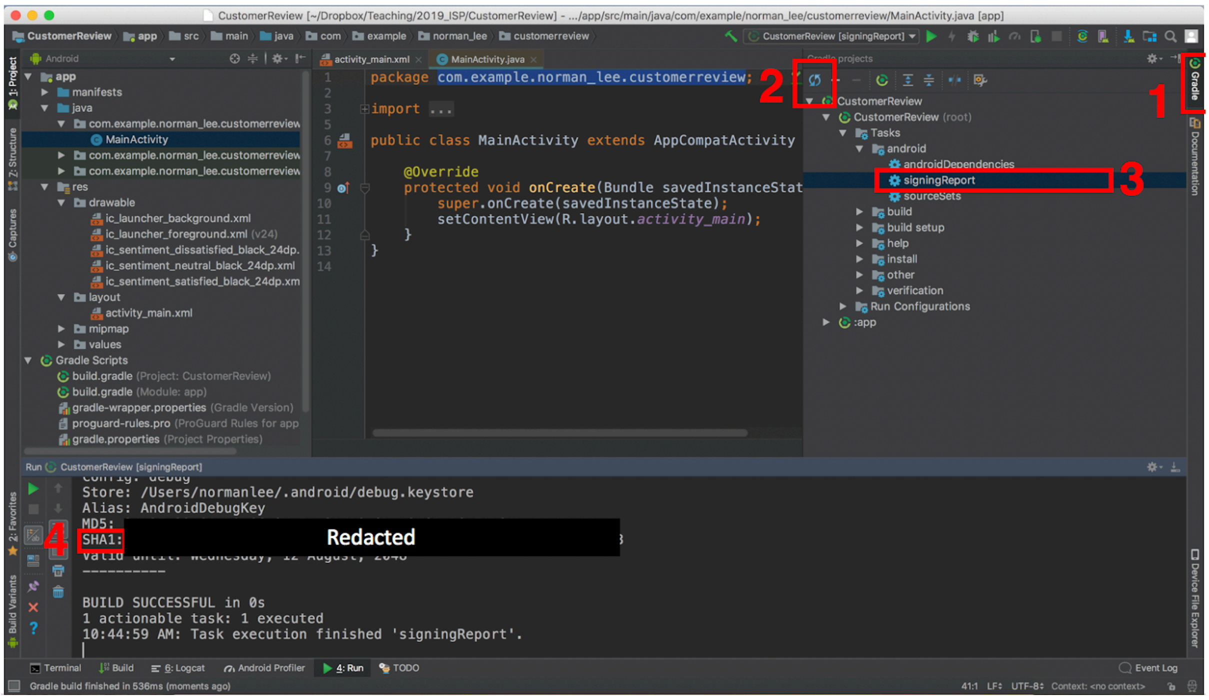
a. Alternatively, Find **Project Overview** and click on the icon beside it, then select **Project Settings**.



b. Select the **General Tab**, and scroll down and look for the following screen, and select the Android icon.


Your apps
There are no apps in your project
Select a platform to get started
iOS

4. You are required to enter some information:
    a. The package name. Go to your android studio project and get the package name.
       As an example, my package name is
       `com.example.kenny_lu.androidfirebasetest.` Use your own.
    b. SHA1 fingerprint: please see instructions here
       https://stackoverflow.com/questions/15727912/sha-1-fingerprint-of-keystore-certificate
       Here's a summary of the steps:

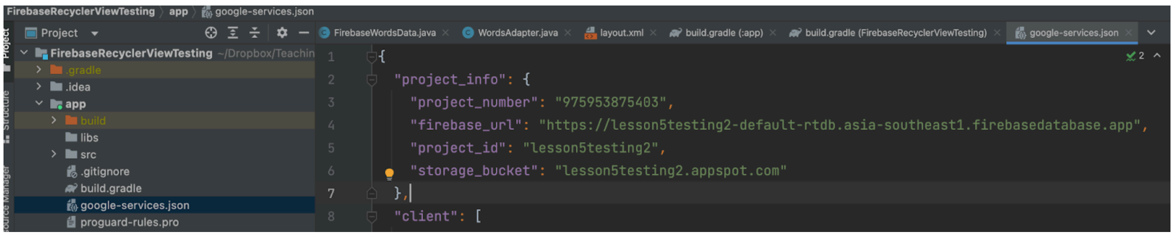


```
package com.example.norman_lee.customerreview;

import ...

public class MainActivity extends AppCompatActivity

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

```
Config: debug
Store: /Users/normanlee/.android/debug.keystore
Alias: AndroidDebugKey
MD5:
SHA1:  Redacted
Valid until: Wednesday, 12 August, 2048
----------

BUILD SUCCESSFUL in 0s
1 actionable task: 1 executed
10:44:59 AM: Task execution finished 'signingReport'.
```

5. The website is user friendly and will tell you what to do next. First, you are given a **google-services.json** file to be put in your project. Make sure you put it in the right location under the project view.

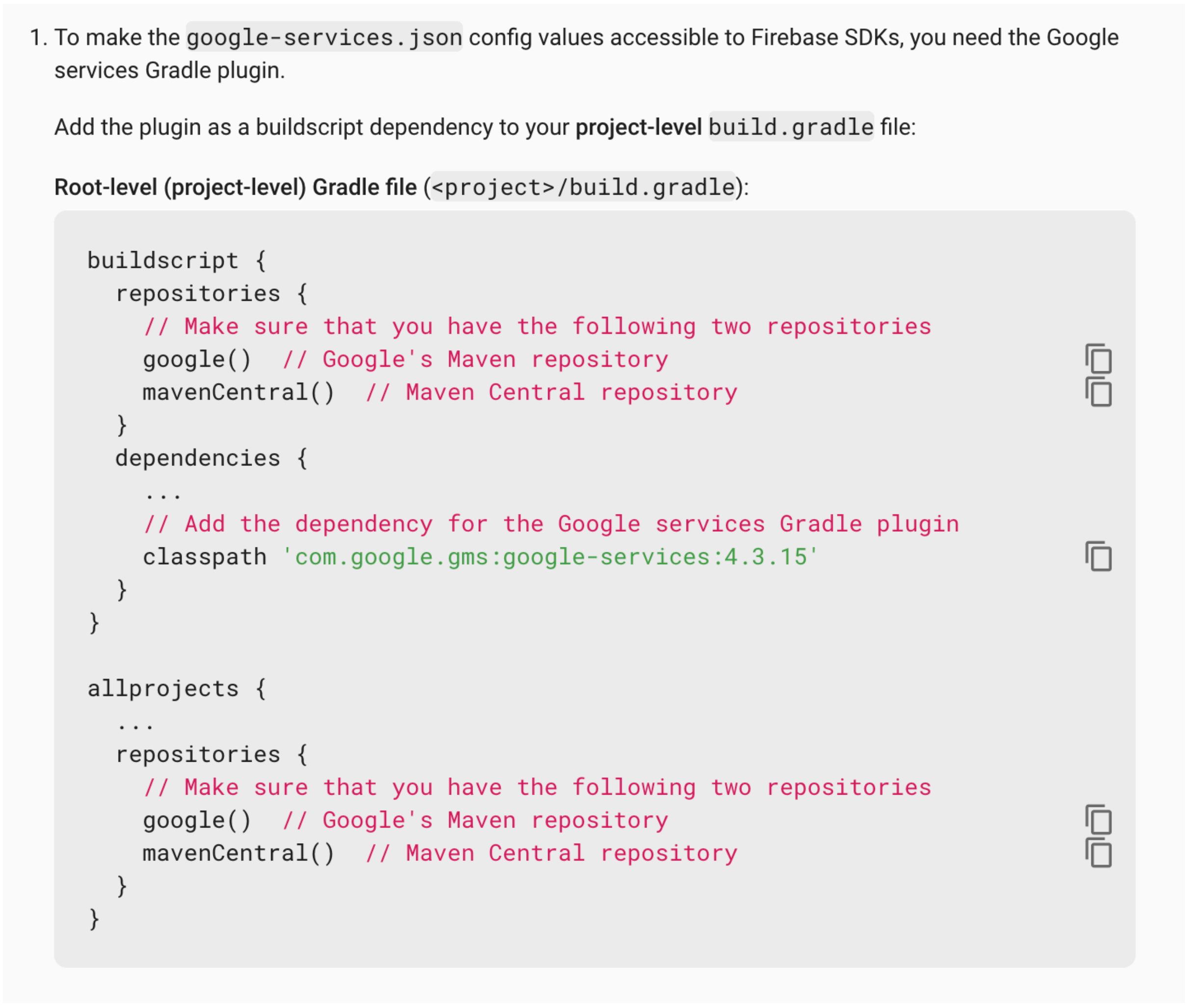
   Open it. Right at the top you should see a `firebase_url` key. Here's mine.



```
{
  "project_info": {
    "project_number": "975953875403",
    "firebase_url": "https://lesson5testing2-default-rtdb.asia-southeast1.firebasedatabase.app",
    "project_id": "lesson5testing2",
    "storage_bucket": "lesson5testing2.appspot.com"
  },
  "client": [
```

   If you don't have this `firebase_url` key, you'll have to generate the google-services.json again later. You can move on to the next step.

6. You are also told how to modify the gradle files. For this step, change back to the Android view. As of March 2023, I see this:



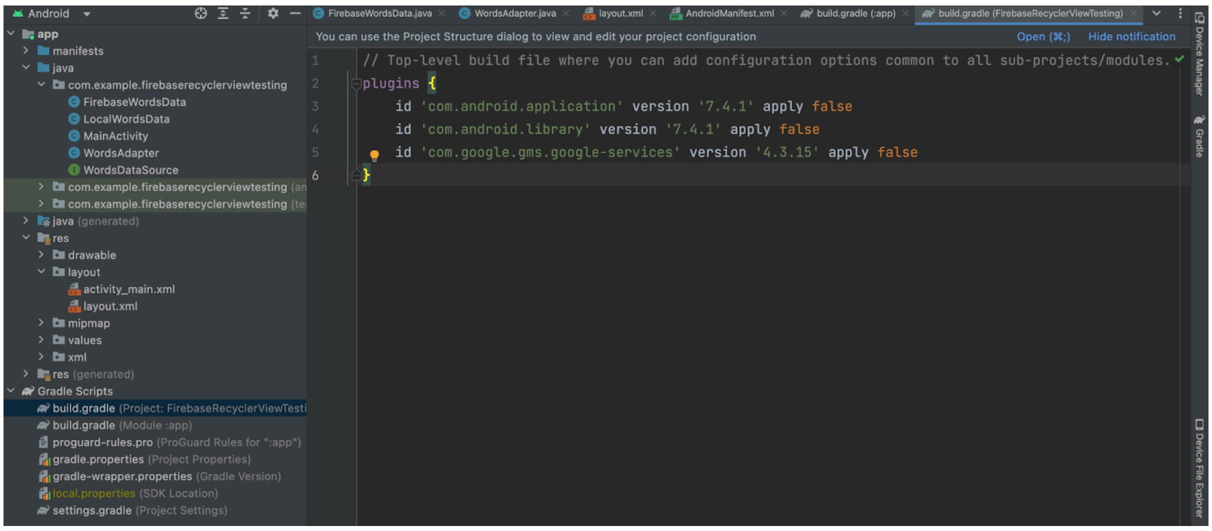
1. To make the `google-services.json` config values accessible to Firebase SDKs, you need the Google services Gradle plugin.

   Add the plugin as a buildscript dependency to your **project-level** `build.gradle` file:

   Root-level (project-level) Gradle file (`<project>/build.gradle`):

```
buildscript {
  repositories {
    // Make sure that you have the following two repositories
    google()  // Google's Maven repository
    mavenCentral()  // Maven Central repository
  }
  dependencies {
    ...
    // Add the dependency for the Google services Gradle plugin
    classpath 'com.google.gms:google-services:4.3.15'
  }
}

allprojects {
  ...
  repositories {
    // Make sure that you have the following two repositories
    google()  // Google's Maven repository
    mavenCentral()  // Maven Central repository
  }
}
```

Go to the **Project-level gradle file** and you'll realise that things have changed. Here's how to edit it.

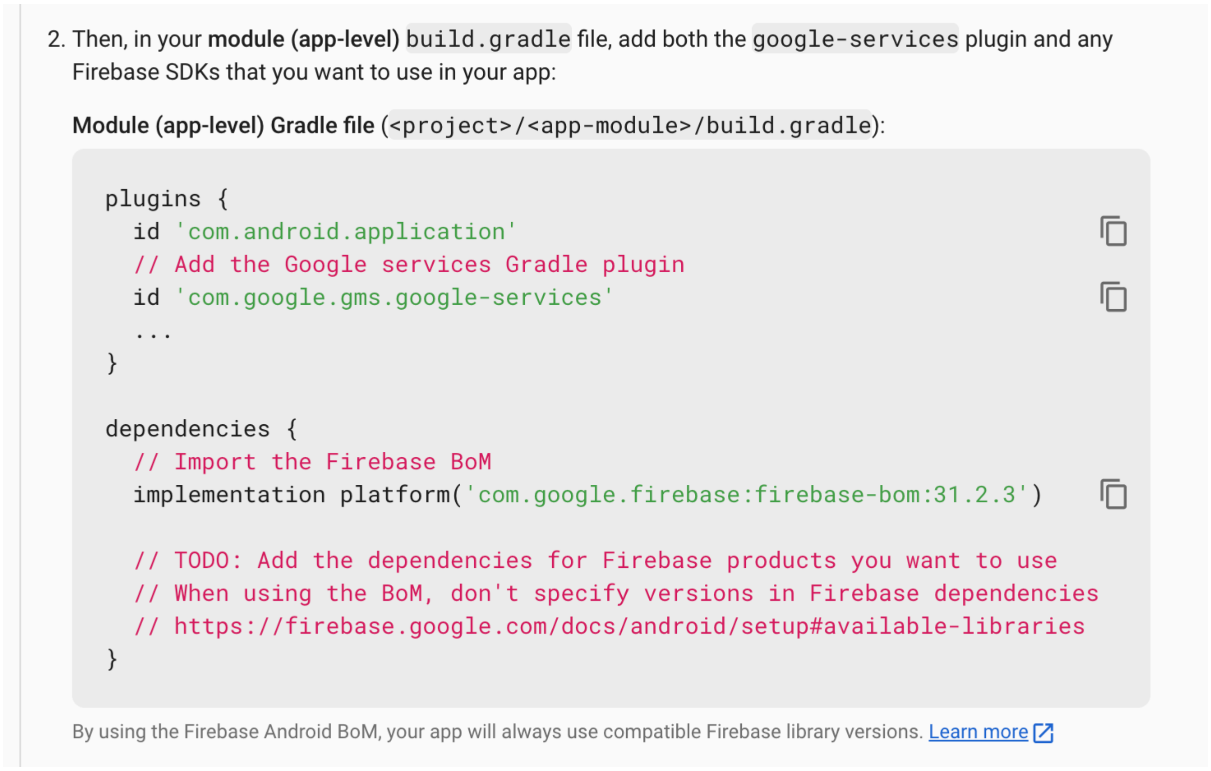


Android
app
manifests
java
com.example.firebaserecyclerviewtesting
FirebaseWordsData
LocalWordsData
MainActivity
WordsAdapter
WordsDataSource
com.example.firebaserecyclerviewtesting (an
com.example.firebaserecyclerviewtesting (te
java (generated)
res
drawable
layout
activity_main.xml
layout.xml
mipmap
values
xml
res (generated)
Gradle Scripts
build.gradle (Project: FirebaseRecyclerViewTesti
build.gradle (Module :app)
proguard-rules.pro (ProGuard Rules for ":app")
gradle.properties (Project Properties)
gradle-wrapper.properties (Gradle Version)
local.properties (SDK Location)
settings.gradle (Project Settings)
FirebaseWordsData.java
WordsAdapter.java
layout.xml
AndroidManifest.xml
build.gradle (:app)
build.gradle (FirebaseRecyclerViewTesting)
Open (⌘;)
Hide notification
Device Manager
Gradle
Device File Explorer

You can use the Project Structure dialog to view and edit your project configuration

```
// Top-level build file where you can add configuration options common to all sub-projects/modules.
plugins {
    id 'com.android.application' version '7.4.1' apply false
    id 'com.android.library' version '7.4.1' apply false
    id 'com.google.gms.google-services' version '4.3.15' apply false
}
```

7. On the same page, you'll also see this. Go to the **App-level gradle file** and add in the lines that are given to you.



2. Then, in your **module (app-level)** `build.gradle` file, add both the `google-services` plugin and any Firebase SDKs that you want to use in your app:

Module (app-level) Gradle file (`<project>/<app-module>/build.gradle`):

```
plugins {
  id 'com.android.application'
  // Add the Google services Gradle plugin
  id 'com.google.gms.google-services'
  ...
}

dependencies {
  // Import the Firebase BoM
  implementation platform('com.google.firebase:firebase-bom:31.2.3')

  // TODO: Add the dependencies for Firebase products you want to use
  // When using the BoM, don't specify versions in Firebase dependencies
  // https://firebase.google.com/docs/android/setup#available-libraries
}
```

By using the Firebase Android BoM, your app will always use compatible Firebase library versions. Learn more

Under the dependencies section, you can also add the following two lines.

```
implementation 'com.google.firebase:firebase-database:20.1.0'
implementation 'com.google.firebase:firebase-storage:20.1.0'
```

8. Sync your gradle files. (look for the elephant icon button to the right of the Run and Debug buttons in Android Studio).

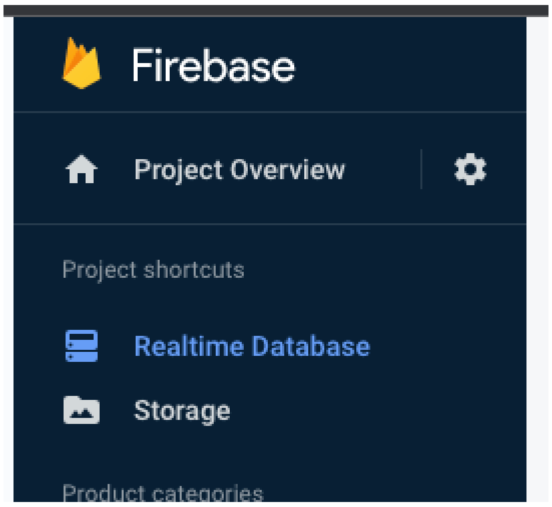
In Android Studio, run your project on your emulator or a phone. If you have followed all the instructions, the website will congratulate you.

9. If you did not see the **firebase_url** key in the google-services.json earlier, one likely reason is that the Firebase Realtime Database and Storage has not been created by you. Follow the instructions on the webpage to create these services.
   Go to the **Initialize your databases** section (a few pages after this section) for hints, then come back here.

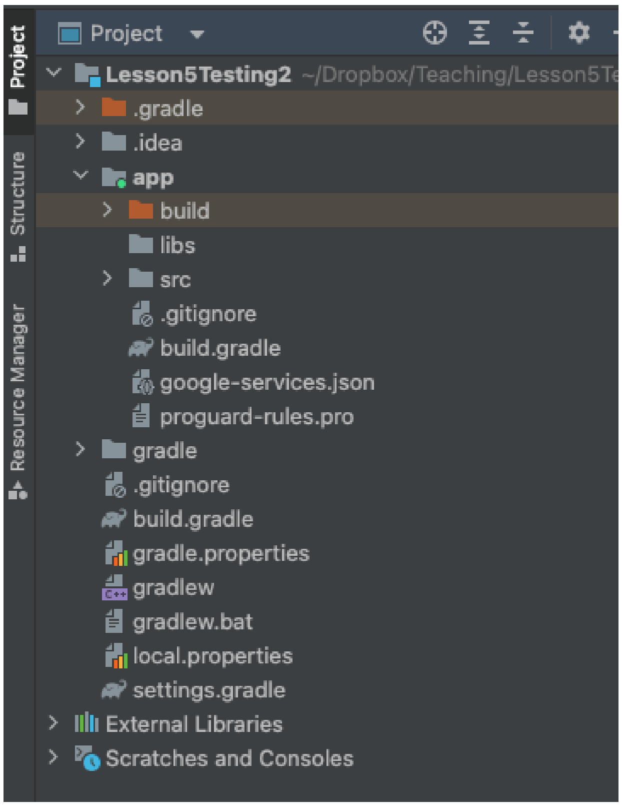
   Once you have created them, you should see this on the left-hand corner.

   Go to the Settings icon beside Project Overview, and click Project settings.

   You'll get to a page where you can then download an updated google-services.json.



10. Change to the Project view of your project, and put the google-services.json file in the correct location. Right-click on the app icon, and select Open In Explore / Finder.



FireStore vs Realtime Database

In Oct 2020 when the first version of these notes were written, Google Firebase team launched a new product named FireStore.

When shall we use FireStore

- Large Scale Database (>100GB)
- Data are organized into collections
- Real Time sorting, transaction, aggregation query and batch processing
- One single database

When we shall use Realtime Database

- Small Scale Database that changes frequently
- Key-value query which retrieve a small amount of data
- JSON like data
- Multiple databases for an app

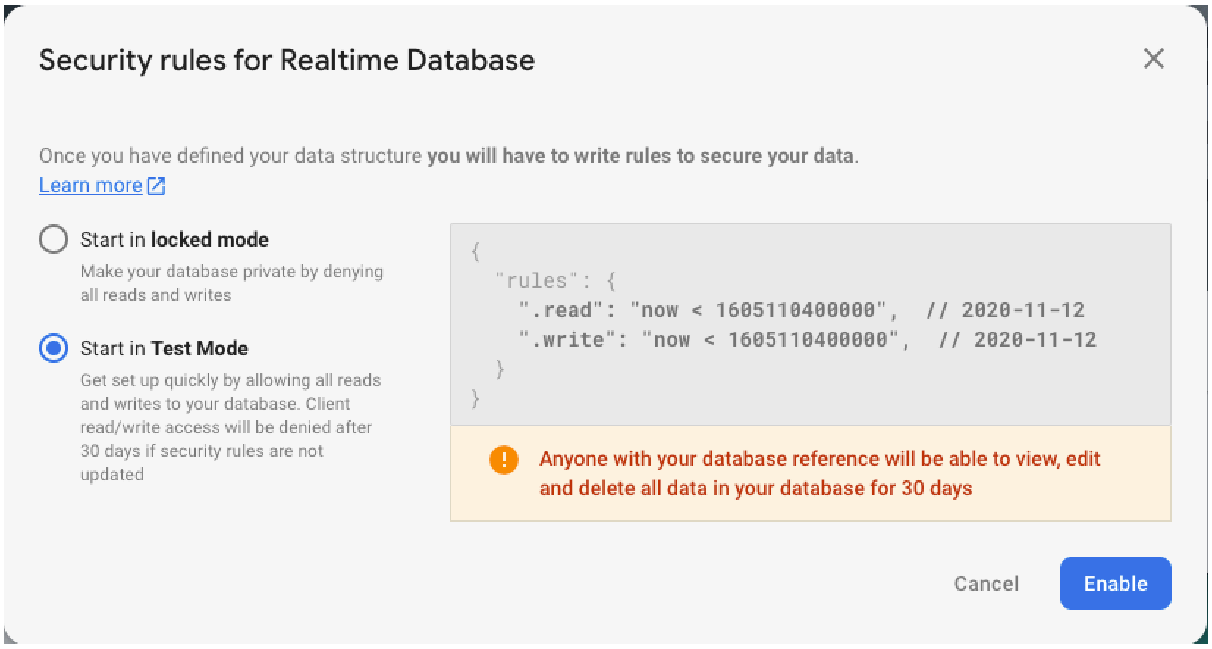
For this lesson, we use Realtime database. You are encouraged to explore the FireStore.

Initialise your databases

Set up the Firebase Realtime Database.

This is the **Realtime Database** item on the left-hand menu in your firebase console.
https://console.firebase.google.com/u/0/project/<*your project name*>/overview

It will prompt you to set a security rule. Let's go with the **Test Mode** to eliminate the need of setting up the authentication for the app. We can update the rules later in the **rule** tab.



Security rules for Realtime Database

Once you have defined your data structure **you will have to write rules to secure your data**.
Learn more

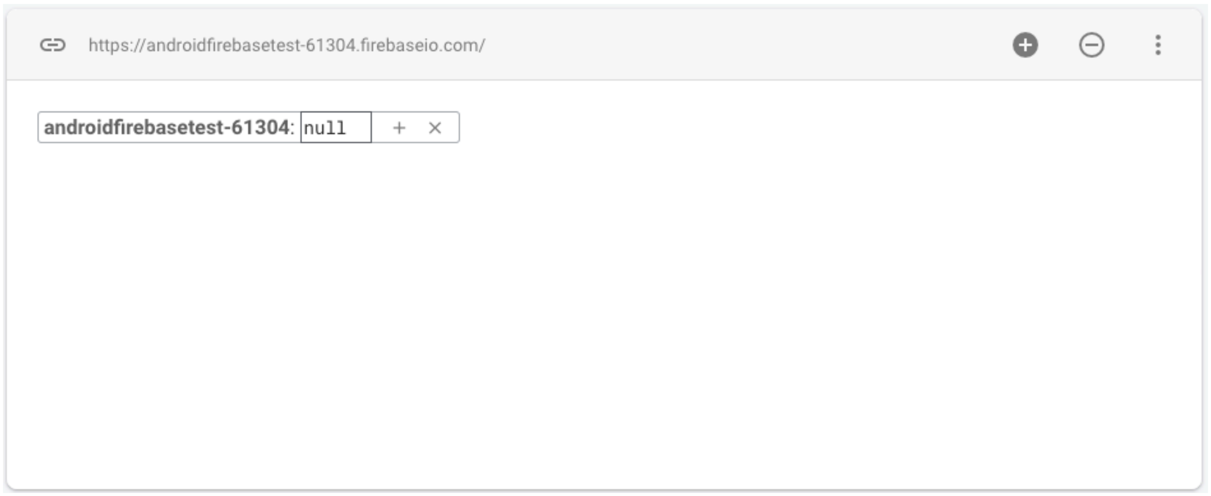
Start in **locked mode**
Make your database private by denying all reads and writes

[✓ Start in **Test Mode**]
Get set up quickly by allowing all reads and writes to your database. Client read/write access will be denied after 30 days if security rules are not updated

```
{
  "rules": {
    ".read": "now < 1605110400000",  // 2020-11-12
    ".write": "now < 1605110400000",  // 2020-11-12
  }
}
```

Anyone with your database reference will be able to view, edit and delete all data in your database for 30 days

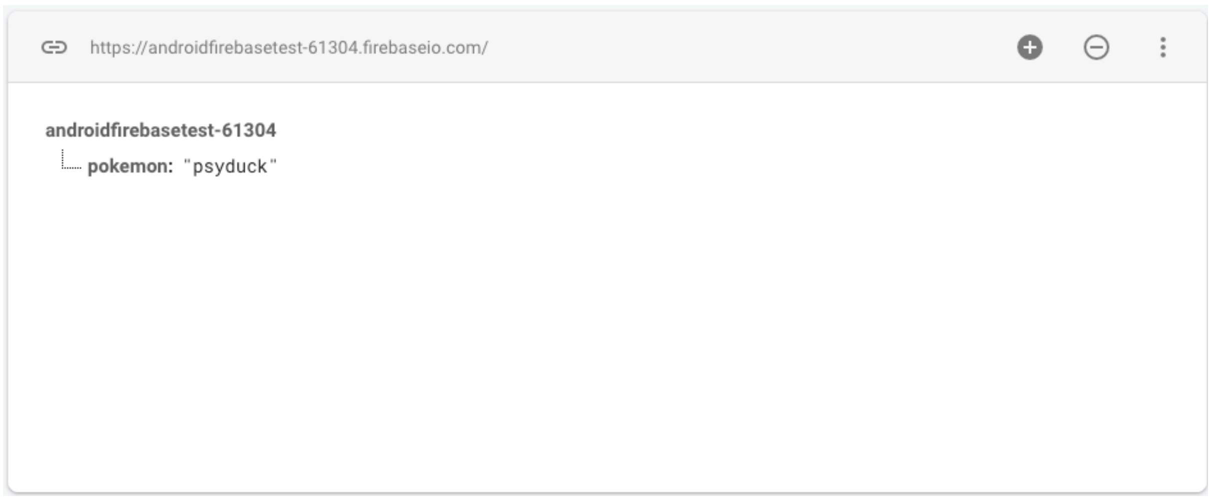
Initially your database will be empty.



https://androidfirebasetest-61304.firebaseio.com/

androidfirebasetest-61304: null

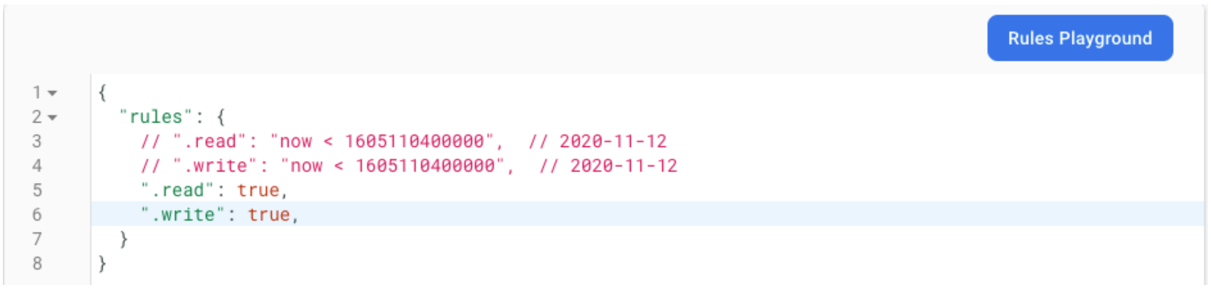
Set it up so that it will contain one key-value pair.



https://androidfirebasetest-61304.firebaseio.com/

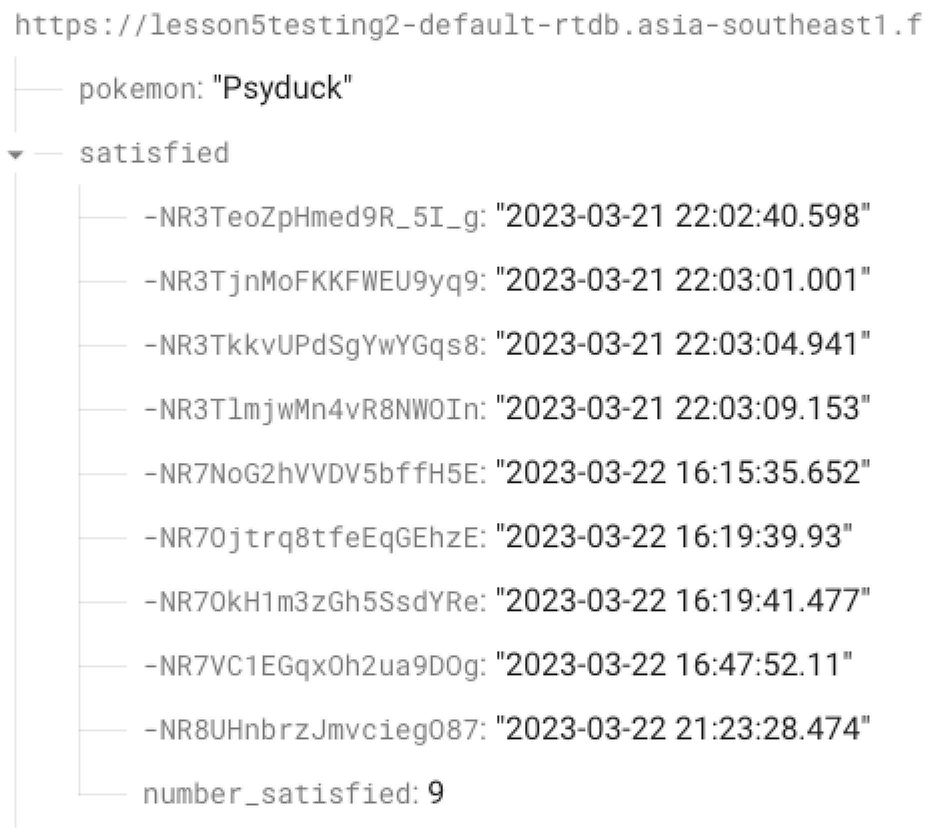
```
androidfirebasetest-61304
└── pokemon: "psyduck"
```

If you would like, you can go to the Rules tab to change the security rules as follows,



```
{
  "rules": {
    // ".read": "now < 1605110400000",  // 2020-11-12
    // ".write": "now < 1605110400000",  // 2020-11-12
    ".read": true,
    ".write": true,
  }
}
```

How my firebase realtime database looks like after being in use for some time

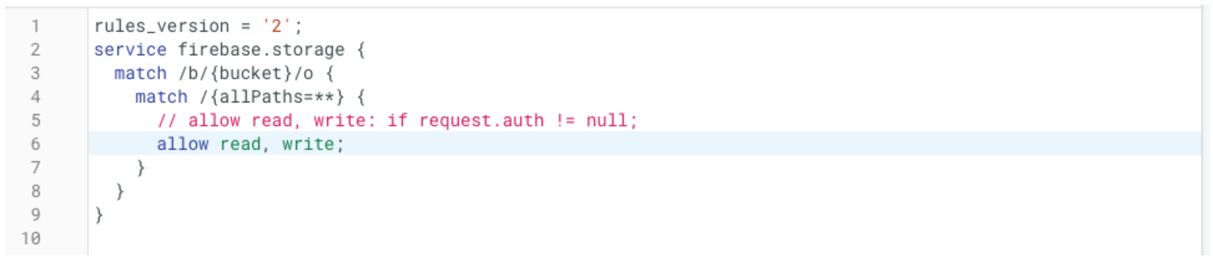


```
https://lesson5testing2-default-rtdb.asia-southeast1.f
├── pokemon: "Psyduck"
▾── satisfied
    ├── -NR3TeoZpHmed9R_5I_g: "2023-03-21 22:02:40.598"
    ├── -NR3TjnMoFKKFWEU9yq9: "2023-03-21 22:03:01.001"
    ├── -NR3TkkvUPdSgYwYGqs8: "2023-03-21 22:03:04.941"
    ├── -NR3TlmjwMn4vR8NWOIn: "2023-03-21 22:03:09.153"
    ├── -NR7NoG2hVVDV5bffH5E: "2023-03-22 16:15:35.652"
    ├── -NR7Ojtrq8tfeEqGEhzE: "2023-03-22 16:19:39.93"
    ├── -NR7OkH1m3zGh5SsdYRe: "2023-03-22 16:19:41.477"
    ├── -NR7VC1EGqxOh2ua9DOg: "2023-03-22 16:47:52.11"
    ├── -NR8UHnbrzJmvcieg087: "2023-03-22 21:23:28.474"
    └── number_satisfied: 9
```

If you need to set up The Storage Database

This is the **Storage** item on the left-hand menu and is meant for blob data such as images. Follow the instructions on the screen.

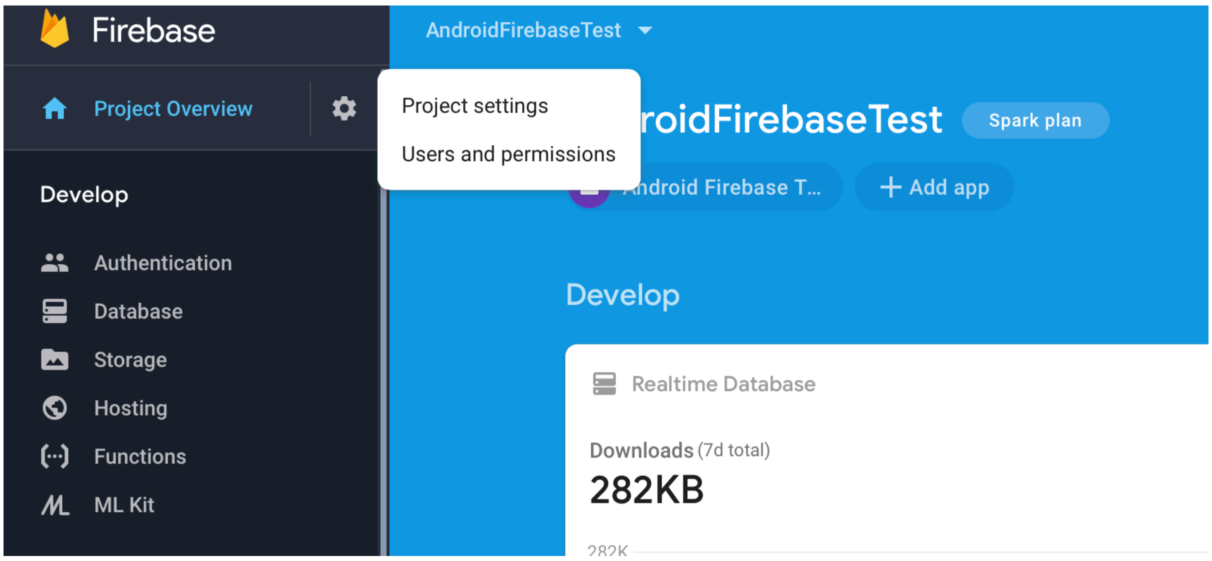
Click on the **Rules** tab and ensure the code within looks like this.
Again, this is to remove the need for authentication.



```
rules_version = '2';
service firebase.storage {
  match /b/{bucket}/o {
    match /{allPaths=**} {
      // allow read, write: if request.auth != null;
      allow read, write;
    }
  }
}
```

More than one app can use the same database

Just go to your firebase database and select **Add App**, and follow the steps.



Using Firebase realtime database in an Android app

First, Initialise the connection instance

Before performing any operation to the realtime database, we need to establish the connection. Recall that Firebase realtime database is a NoSQL database whose data are stored in a JSON-like structure.

By executing the code below, mRootDatabaseRef is a variable that points to the root node of your Realtime Database instance.

```
DatabaseReference mRootDatabaseRef;

mRootDatabaseRef = FirebaseDatabase.getInstance().getReference();
```

Think of how you want to design your database

Though database design is not the focus in this module, we still have to think about how to organise the data. For your database, plan out the keys and the corresponding values that they will store.

Do not have many levels of nested nodes (e.g. nodes that are a child of a node, which is a child of another node and so on). Even though Firebase realtime database supports nested data structure with max height (or depth) of 32, it is recommended to flatten your data to increase the performance of the query. For more details, please refer to https://firebase.google.com/docs/database/android/structure-data

Reading / Writing key-value data to the database

Writing to a node. Given a reference to a data node in the Firebase realtime database, we set the value of of a child node with the following

```
final String SAMPLE_NODE = "pokemon";
mRootDatabaseRef.child(SAMPLE_NODE).setValue("Psyduck");
```

{ pokemon : “Psyduck” }

Reading a node. We can add an event listener to a node to “listen” out for changes in a node, and perform changes accordingly.
In the following code, an event listener is added to pokemon node.
When the value of the node changes, **onDataChange()** is invoked

- data is retrieved from a **`DataSnapshot`** object
- A **TextView** object is updated with that data.

Recall in Lesson 3, such downloading tasks take time and (1) must be executed on a background thread first to avoid freezing the UI, before (2) communicating with the UI thread with the result. You had to write code to specify which tasks go on which thread.

The Firebase library simplifies it for you. You specify these tasks in the ValueEventListener() object, and the handling of the tasks in the threads is abstracted from you i.e. handled by the library.

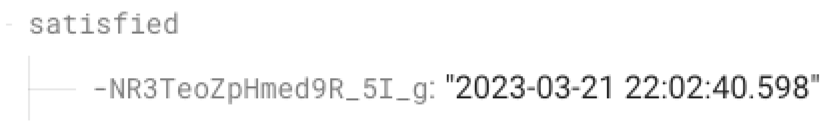
```
mRootDatabaseRef
     .child(SAMPLE_NODE)
     .addValueEventListener(new ValueEventListener() {
     @Override
     public void onDataChange(DataSnapshot snapshot) {
         String change = snapshot.getValue(String.class);
         someTextView.setText(change); // action
     }
     @Override
     public void onCancelled(DatabaseError error) {
         Log.i("Pokemon","" + error.toString());
     }
 });
```

Working with List data

If you call **`push()`** on a node reference, firebase creates a child node with a unique key and a value.

```
final String SATISFIED = "satisfied";
mNodeRefSatisfied = mRootDatabaseRef.child(SATISFIED);
mNodeRefSatisfied.push().setValue(timestamp.toString());
```

Example of how it looks like in the database.



```
- satisfied
  └── -NR3TeoZpHmed9R_5I_g: "2023-03-21 22:02:40.598"
```

Calling push() repeatedly will generate a series of child nodes which are very much like a list of data. See page 266 of how it looks like.

Reading from changes <u>within a list</u>. We can add a `ChildEventListener` object to detect any changes in the list elements.

```
mNodeRefSatisfied
.addChildEventListener(new ChildEventListener() {
   @Override
   public void onChildAdded(DataSnapshot dataSnapshot
                            , String previousChildName) {
       Log.d(TAG, "onChildAdded:" + dataSnapshot.getKey());
       String ts = dataSnapshot.getValue(String.class);
       // ...
   }
   @Override
   public void onChildChanged(DataSnapshot dataSnapshot
                             , String previousChildName) {
       Log.d(TAG, "onChildChanged:" + dataSnapshot.getKey());
       // ...
   }
   @Override
   public void onChildRemoved(DataSnapshot dataSnapshot) {
       Log.d(TAG, "onChildRemoved:" + dataSnapshot.getKey());
       // ...
   }
   @Override
   public void onChildMoved(DataSnapshot dataSnapshot
                            , String previousChildName) {
       Log.d(TAG, "onChildMoved:" + dataSnapshot.getKey());
       // ...
   }
   @Override
   public void onCancelled(DatabaseError databaseError) {
       Log.w(TAG, "onCancelled",
               databaseError.toException());
   }
};);
```

Reading the entire list. There are situations in which we need to get a copy of the entire stored list, in cases like this, use a **`ValueEventListener`** instead.

```
mNodeRefSatisfied
   .addValueEventListener(new ValueEventListener() {
   @Override
   public void onDataChange(DataSnapshot dataSnapshot) {
       for (DataSnapshot postSnapshot:
                dataSnapshot.getChildren()) {
           // TODO: handle the post e.g. store to an ArrayList
       }
   }

   @Override
   public void onCancelled(DatabaseError databaseError) {
       // Getting Post failed, log a message
       Log.w(TAG, "onCancelled",
               databaseError.toException());
       // ...
   }
});
```

Multithreading revisited: Executor vs Task

Google Firebase Java API is mostly defined using `Task` objects. Recall that multithreading programming with Executor, Looper and Runnable we learned and used in the previous lessons.

Executor, Looper and Runnable

1. We instantiate an executor (ExecutorService), call executor.execute(new Runnable(){ … }) to start a child thread (background task).
2. We instantiate a Handler object (with a Looper object) which captures the message queue of the main thread (UI thread).
3. We then call handler.post(new Runnable(){ … }) to schedule a UI task which will be executed upon the completion of the background task.

Task (Google GMS Task)

The Google GMS Task is a more general (higher level) abstract data type for computation that takes place asynchronously. This section is retained for your information.

To create a Task, we use a static method Tasks.call(callable) where callable is an object implementing the java.util.concurrent.Callable<V>. The statement returns a Task object that returns V when it is done. The task will be run in some thread. We can (busy) wait for the computation result of the Task using Tasks.await(task);

```java
Callable<Integer> callable = new Callable<Integer>() {
      public Integer call() { return 1 + 1; }
};
Task<Integer> task = Tasks.call(callable);
try {
   Integer result = Tasks.await(task);
   …
} catch (ExecutionException e) {
   e.printStackTrace();
} catch (InterruptedException e) {
   e.printStackTrace();
}
```

Similar to the handler.post() in the Executor approach, we can add event listeners to a Task.

```
task.addOnSuccessListener(new OnSuccessListener<Integer>() {
   @Override
   public void onSuccess(Integer integer) {
       // do something with the integer
   }
});
task.addOnFailureListener(new OnFailureListener() {
   @Override
   public void onFailure(@NonNull Exception e) {
       // handle the exception
   }
});
```

These are definitely better than the busy blocking await method.
One of the advantages of Task is that we can compose a bigger Task from smaller Tasks by chaining them up as continuation. Of course, you need not use this and you can still write code to execute individual tasks in sequence.

```
Task<Boolean> task2 = task.continueWithTask(new Continuation<Integer,
Task<Boolean>>() {
   @Override
   public Task<Boolean> then(@NonNull final Task<Integer> task) {
       return Tasks.call(new Callable<Boolean>() {
           @Override
           public Boolean call() {
               Integer intResult = task.getResult();
               return intResult % 2 == 0;
           }
       });
   }
});
```

References for GMS Tasks

https://developers.google.com/android/reference/com/google/android/gms/tasks/package-summary

Part 1 - Pushing and Pulling Text, Dynamic Layout

Applying the Single Responsibility Principle

Code to write to / read from Firebase Realtime Database is encapsulated in a **FirebaseDatabaseOperations** class, whose methods are then executed by **MainActivity**.

TODO 13.0 Setup the Firebase realtime database

- Create an android app and register it with Firebase
- Download the starter code, which consists of MainActivity.java, FirebaseUtils.java, the layout files and some background image files.
- Copy and paste these codes and files over the android studio-generated code in the project.
- Make sure you put the google-service.json file in the app folder after configuring your firebase instance for your app
- You need to add the following dependency into your app level build.gradle file. The build.gradle files in the starter code can also be a good reference for you.

```
dependencies {
   implementation platform('com.google.firebase:firebase-bom:31.2.3')
   implementation 'com.google.firebase:firebase-database:20.1.0'
   implementation 'com.google.firebase:firebase-storage:20.1.0'
...
```

Refer to the image on page 266 to see how the database is structured.

TODO 13.1 [MainActivity] Get references to the widgets

```
textViewSampleNodeValue =findViewById(R.id.textViewSampleNodeValue);
textViewTally = findViewById(R.id.textViewTally);
imageViewSatisfied = findViewById(R.id.imageViewSatisfied);
```

TODO 13.2 Get references to the nodes in the database

We implement the o

- **TODO 13.2a [MainActivity]** We instantiate a `FirebaseDbOpsSubject` object
- **TODO 13.2b [FirebaseDbOpsSubject]**
  We write a class **`FirebaseDatabaseOperations`** and put some of these statements in the constructor. The variables SATISFIED and NO_SATISFIED are String constants that contain the names of the nodes.

```
mRootDatabaseRef = FirebaseDatabase.getInstance().getReference();
mNodeRefSatisfied = mRootDatabaseRef.child(SATISFIED);
mNodeRefSatisfied = mRootDatabaseRef.child(NO_SATISFIED);
```

TODO 13.3 When the satisfied button is clicked, push the info to the database.

- **TODO 13.3a [MainActivity]**
    - Create OnClickcListener and set it to imageViewSatisfied
    - In the onClick() method in the listener, execute pushTimestampToSatisfied() of FirebaseDatabaseOperations.
- **TODO 13.3b [FirebaseDbOpsSubject]** in pushTimestampToSatisfied()
    - Create a Timestamp object using System.currentTimeMillis()
    - Push the timestamp as one of the entry to the list items under mNodeRefSatisfied
    - Increase the int value associated with the child node NO_SATISFIED of mNodeRefTally

Some points to take note

executing mNodeRefSatisfied.push() creates child nodes with random ID

- mNoteRefTally.child("data") creates a child node if it didn't exist
- mNoteRefTally.child("data").setValue( ) assigns a value to the node
- explore what happens if you did this subsequently:
    - mNoteRefTally.child("data").child("data1").setValue()

TODO 13.4 Listen out for changes in the "pokemon" node and update the Textview textViewSampleNodeValue

This part is to show you how ValueEventListener works and code is retained in MainActivity.java for your easy reference.

Recall that we can add a ValueEventListener to a data node reference, we need to override the onDataChange() and onCancelled() method.

The code given here listens out for changes in the Pokemon node, and displays a Toast.

Try it yourself. Change the data in the database directly e.g. to "snorlax" and see the Toast appear.

TODO 13.5 Listen out for changes in the “satisfied” node and update the TextViews satisfiedTallyView and textViewTally

To do this, we show you how the observer design pattern can be used.

1. We have the following interface for Observer objects
   and have MainActivity implement it. In the observer pattern, the observer’s role is to make use of the data passed to it.
   This will ensure that MainActivity has a **void update()** method, which will have code to make use of the data.

```
public interface Observer {
   void update(DataSnapshot dataSnapshot);
}
```

2. We have the following interface for Subject objects and have **FirebaseDbOps** implement it. In the observer pattern, the subject role is to download the data and notify the observer with the data.

```
public interface Subject {

   void registerActivity(Observer observer);
   void downloadToObserver();
}
```

3. **registerActivity()** is just a setter and is implemented like any other setter method. **downloadToObserve()** contains code that contacts the node you wish to download the data.
   a. Add a ValueEventListener to **mNodeRefSatisfiedNumber**
   b. The data is contained in the **snapshot** variable. Pass it to **observer.update().**

4. In MainActivity, complete **void update()** by extracting the data from the snapshot variable and use it to update the UI.

- **TODO 13.5a [MainActivity]**
  - Execute **registerActivity(this)** to pass an instance of MainActivity to the FirebaseDbOpsSubject object.
  - Complete **void update()** with code to update the UI with the number of child nodes

TODO 13.6 Complete the onStart() and onPause() methods to load and save the last updated data from the sharePreferences.

This is left to you as an exercise.

Part 2 - Pushing and Pulling Images

Expected result

We will upgrade the toilet feedback app built in Part 1 with the following enhancement.

1. When the background image is clicked/touched, we make an API call to firebase storage to retrieve an image and set it as the new background
2. When the FloatingActionButton is clicked, we redirect the user to the Android Phone Gallery to select an image to upload to the Firebase Storage as one of the background image options.

We had to apply this fix to get the images to load. Stackoverflow.

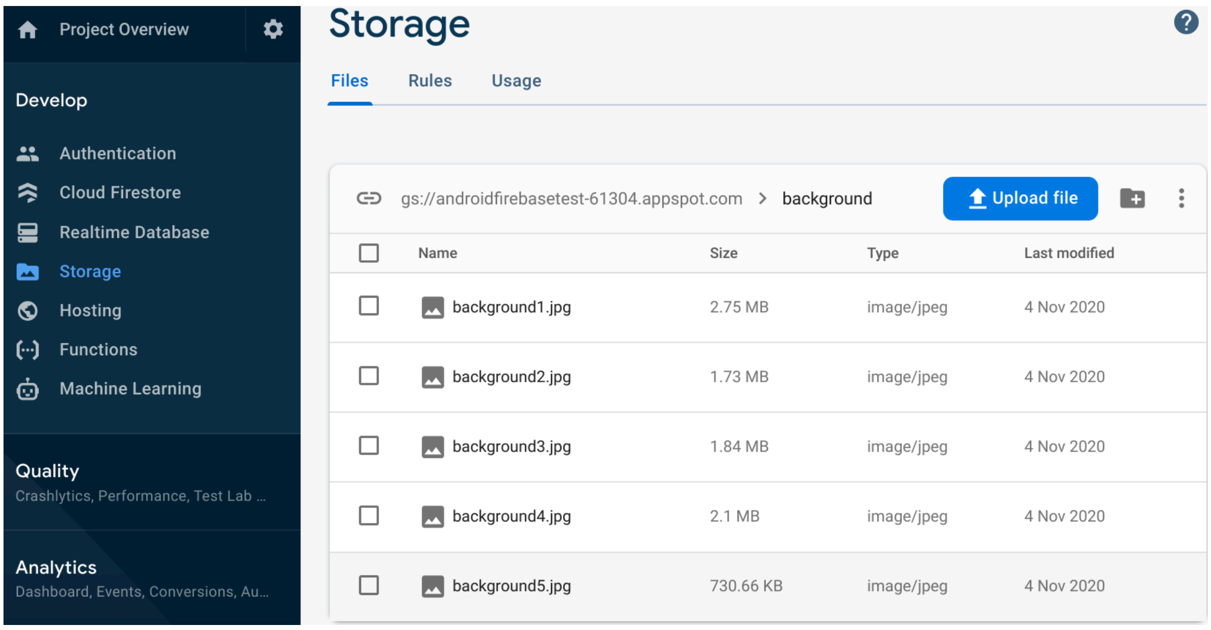
Applying the Single Responsibility Principle

Code to write to / read from Firebase Storage is encapsulated in a **FirebaseStorageOperations** class, whose methods are then executed by **MainActivity**. There are also static utility methods in **FireBaseUtils** to upload and download images.

Preparation

Upload the images in res/drawable/ to the Firebase Storage associated with this app.

Put the images in a folder called "backgrounds"



Name	Size	Type	Last modified
background1.jpg	2.75 MB	image/jpeg	4 Nov 2020
background2.jpg	1.73 MB	image/jpeg	4 Nov 2020
background3.jpg	1.84 MB	image/jpeg	4 Nov 2020
background4.jpg	2.1 MB	image/jpeg	4 Nov 2020
background5.jpg	730.66 KB	image/jpeg	4 Nov 2020

TODO 14.1 Get a reference to the root note of the firebase storage

- **TODO 14.1a [MainActivity]** We instantiate a **`FirebaseStorageOperations`** object
- **TODO 14.1b [FirebaseStorageOperations]**

  We write a class **`FirebaseStorageOperations`** and put some of these statements in the constructor.

```
final StorageReference storageRef =
FirebaseStorage.getInstance().getReference();
```

TODO 14.2 Add a onClickListener to imageViewBackground, so that when the background image is clicked, download the image "background/background2.jpg" and set it as the source of imageViewBackground

This todo illustrates the code necessary to download an image in a simple context.

- **TODO 14.1a [MainActivity]** Execute the **`displayBackground2()`** method in **`FirebaseStorageOperations`** with the necessary inputs
- **TODO 14.1b [FirebaseStorageOperations]** In **`displayBackground2()`** within **`FirebaseStorageOperations`**
  - Obtain a reference to background2.jpg
  - Execute **FirebaseUtils.downloadToImageView()** to download the image

```
StorageReference bgImgRef =
storageRef.child("background/background2.jpg");
FireBaseUtils.downloadToImageView(MainActivity.this, bgImgRef,
imageViewBackground);
```

The static method `downloadToImageview` takes a context, a Firebase storage reference and an imageview object as inputs, then download the image stored in Firebase storage and set it as the source of the imageview
You may refer to `FireBaseUtils.downloadToImageView` to check the implementation details.

TODO 14.3 Modify the onClick method definition in 14.2 such that it will first list all the images stored under the "background" folder in the Firebase Storage. Given the list result, it then picks one image randomly from the list and download it to `imageViewBackground`

We now adapt TODO 14.2 such that a *random image* is displayed, selected from Storage.

- **TODO 14.3a [MainActivity]** Execute the **`getRandomImageFromBackground()`** method in **`FirebaseStorageOperations`** with the necessary inputs
- **TODO 14.3b [FirebaseStorageOperations]** In **`getRandomImageFromBackground()`** within **`FirebaseStorageOperations`**
    - We call **`listAll()`** on **`backgroundStorageReference`** to get a **`Task<ListResult>`** object. This encapsulates the background process required to download all the child nodes i.e. the image files in the /background folder.
    - Next, we call **`addOnSuccessListener`** on **`listResultTask`**, and pass an anonymous **`OnSuccessListener<ListResult>`** task to it. This encapsulates the instructions to be carried out when the download is completed.

- **For TODO 14.3b**, you can also consider using the **`continueWithTask()`** callback as follows.

```
Task<ListResult> taskListResult =
storageRef.child("background").listAll();
taskListResult.continueWithTask(new Continuation<ListResult,
Task<byte[]>>() {
   @Override
   public Task<byte[]> then(@NonNull Task<ListResult> task) throws
Exception {
         ListResult listResult = task.getResult();
         ArrayList<StorageReference> refs = new
ArrayList<>(listResult.getItems());
         Random r = new Random();
         int p = r.nextInt(refs.size()-1);
         StorageReference ref = refs.get(p);
         return FireBaseUtils.downloadToImageView(MainActivity.this, ref,
imageViewBackground);
   }
});
```

TODO 14.4 Create a button which, when clicked, brings the user to the image gallery where an image is selected, and then uploaded to Storage

- **TODO 14.4a [MainActivity]** For this you have to recall Implicit intents in Lesson 4.
  - Create the launcher for the implicit intent to the image gallery.
  - Within the onActivityResult() callback of the launcher:
    - Get the URI of the image selected
    - Execute the `uploadUriToStorage()` method in `FirebaseStorageOperations` with the necessary inputs

- **TODO 14.4b [FirebaseStorageOperations]** In `uploadUriToStorage()` within `FirebaseStorageOperations`
  - For the image to be uploaded, we first extract the filename from the URI using `FireBaseUtils.getFileName`
  - We then create a reference to this filename on Storage, so that any file uploaded will be associated with this reference
  - With the URI, we extract the bitmap of the image
  - Finally, upload the bitmap to Storage using `FireBaseUtils.uploadImageToStorage`
- **TODO 14.4c [MainActivity]** Now we are ready to allow the user to activate the previous steps by a button click by creating an intent and passing it to the launcher.

```
uploadButton.setOnClickListener(new View.OnClickListener(){
    @Override
    public void onClick(View view) {
        Intent intent = new Intent(Intent.ACTION_PICK,
MediaStore.Images.Media.EXTERNAL_CONTENT_URI);
        launcherGallery.launch(intent);
    }
});
```