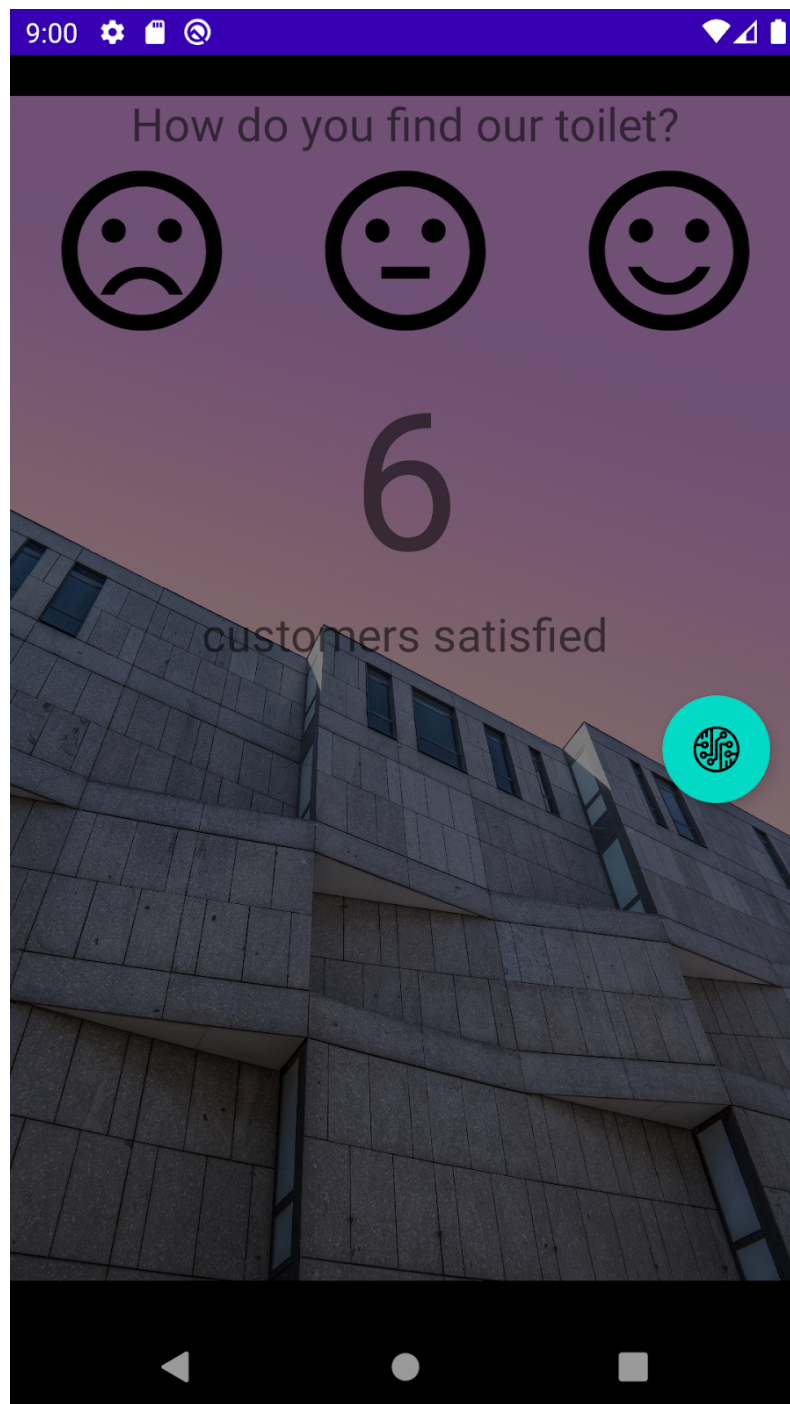


Lesson 5 - Firebase

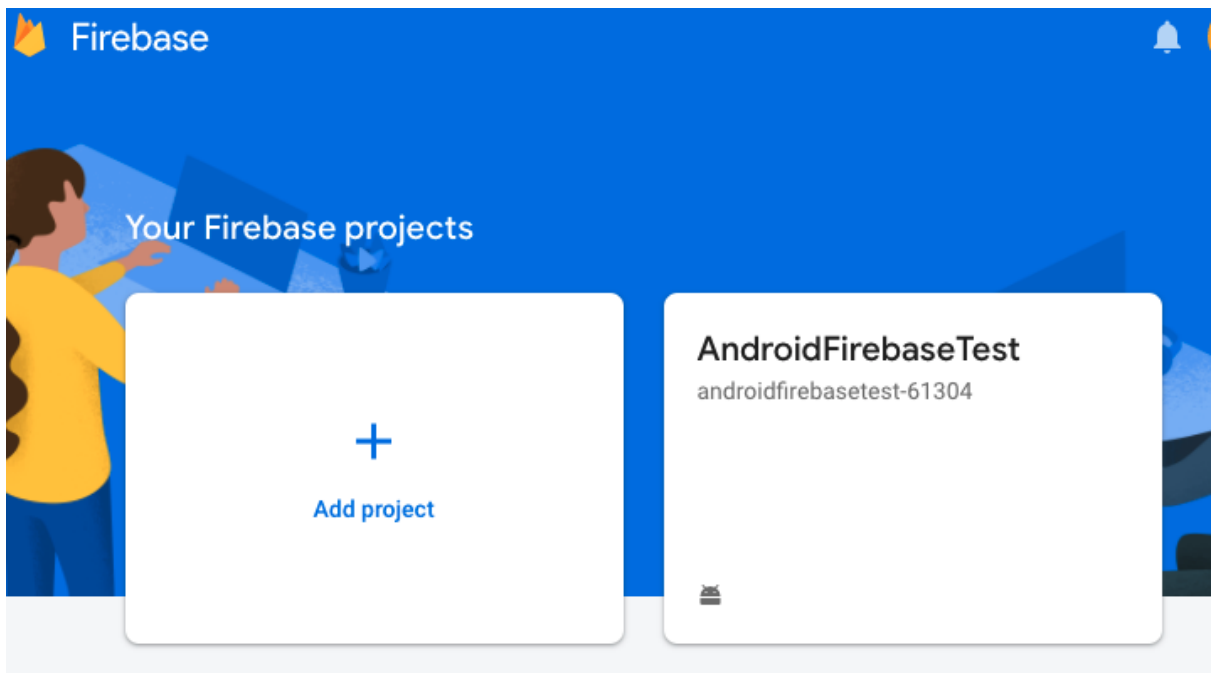
Introduction

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.

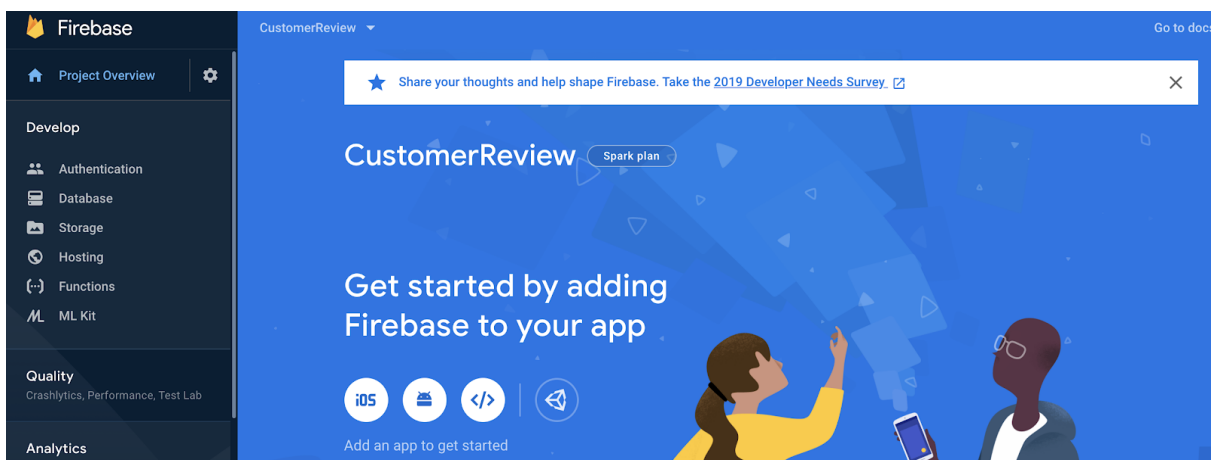


Set up your firebase project

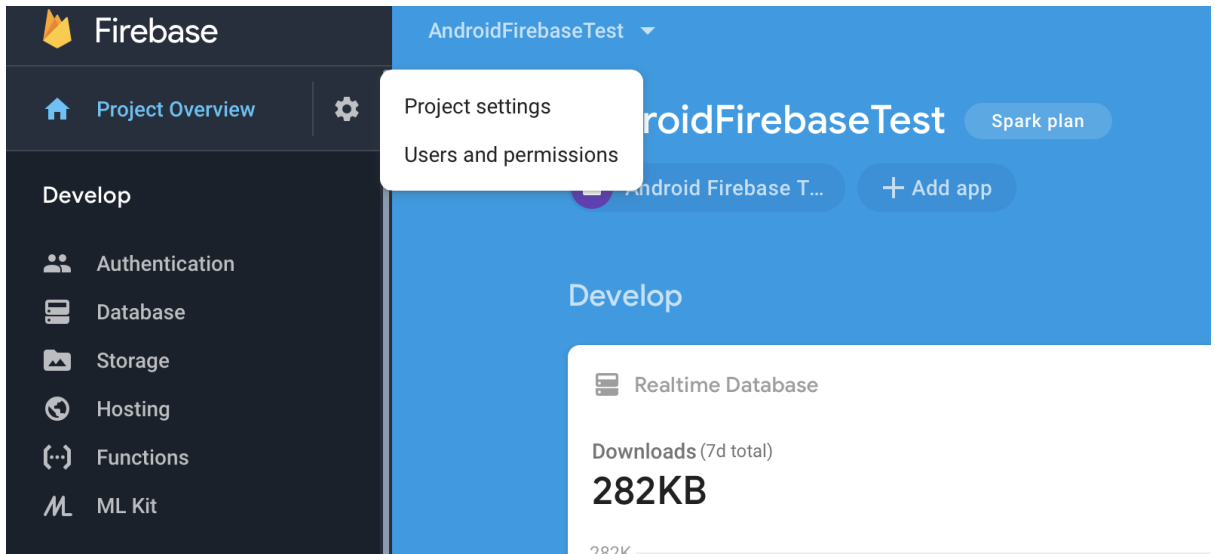
1. Start a new project in Android studio, you will need it in Step 5. OR Alternatively, launch the starter project that you are provided with.
2. Go to console.firebase.google.com and create a new project by selecting Create a Project if you are the first time user of firebase, or add a project if you are an existing user of firebase. You would not need to add Google Analytics to your project at this point.



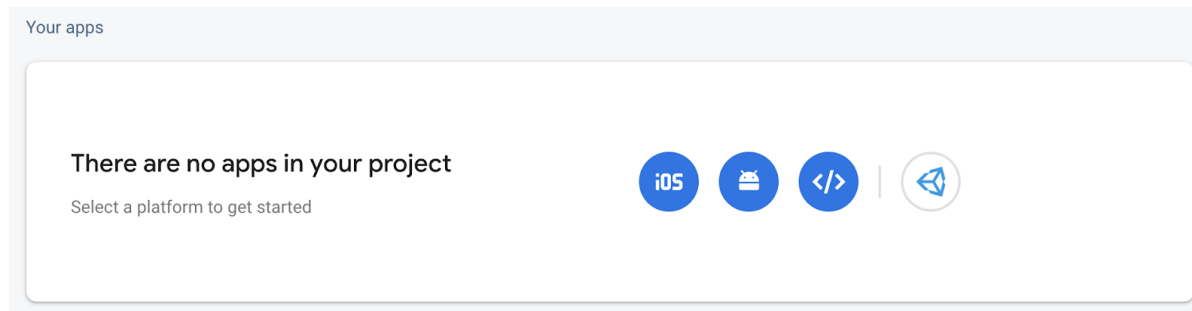
3. After the project set-up is completed, you should see this screen. Click on the android icon.



- 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.



4. You are required to enter some information:

- a. The package name. Go to your android studio project and get the package name.

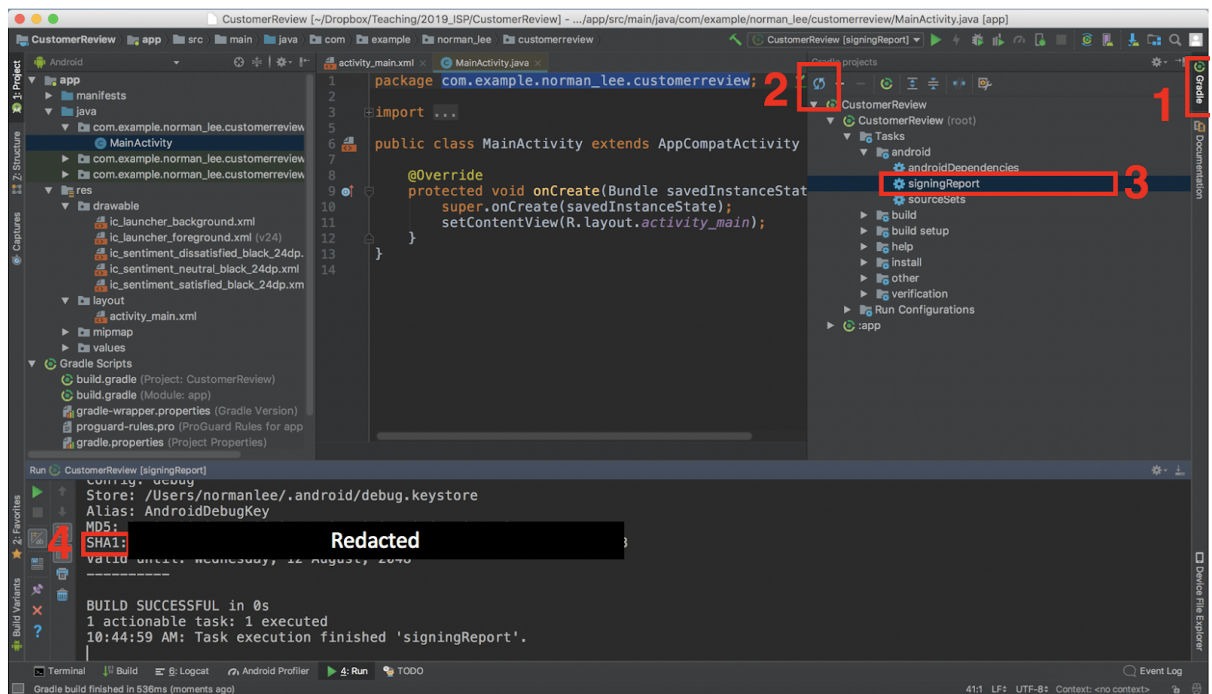
As an example, my package name is

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:



5. The website is user friendly and will tell you what to do next. In general,
 - a. You are given a **google-services.json** file to be put in your project. Be sure to put it in the correct location. Click Next.
 - b. You are also told how to modify the gradle files. For this step, change back to the Android view. At the time of writing (Oct 2021), I see this:

3
Add Firebase SDK
Instructions for Gradle | [Unity](#) [C++](#)

The Google services plugin for [Gradle](#) loads the `google-services.json` file that you just downloaded. Modify your build.gradle files to use the plugin.

Project-level build.gradle (<project>/build.gradle):

```
buildscript {
    repositories {
        // Check that you have the following line (if not, add it):
        google() // Google's Maven repository
    }
    dependencies {
        ...
        // Add this line
        classpath 'com.google.gms:google-services:4.3.4'
    }
}

allprojects {
    ...
    repositories {
        // Check that you have the following line (if not, add it):
        google() // Google's Maven repository
        ...
    }
}
```

App-level build.gradle (<project>/<app-module>/build.gradle):

```
apply plugin: 'com.android.application'
// Add this line
apply plugin: 'com.google.gms:google-services'

dependencies {
    // Import the Firebase BoM
    implementation platform('com.google.firebase:firebase-bom:25.12.0')

    // Add the dependencies for the desired Firebase products
    // https://firebase.google.com/docs/android/setup#available-libraries
}
```

- (1) Go to the **Project-level gradle file** and add in the line that is given to you.
- (2) Go to the **App-level gradle file** and add in the line that is given to you.
- (3) Sync your gradle files. (look for the elephant icon button to the right of the Run and Debug buttons in Android Studio)

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

Firestore vs Realtime Database

At the time when this note was written Oct 2020, 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.

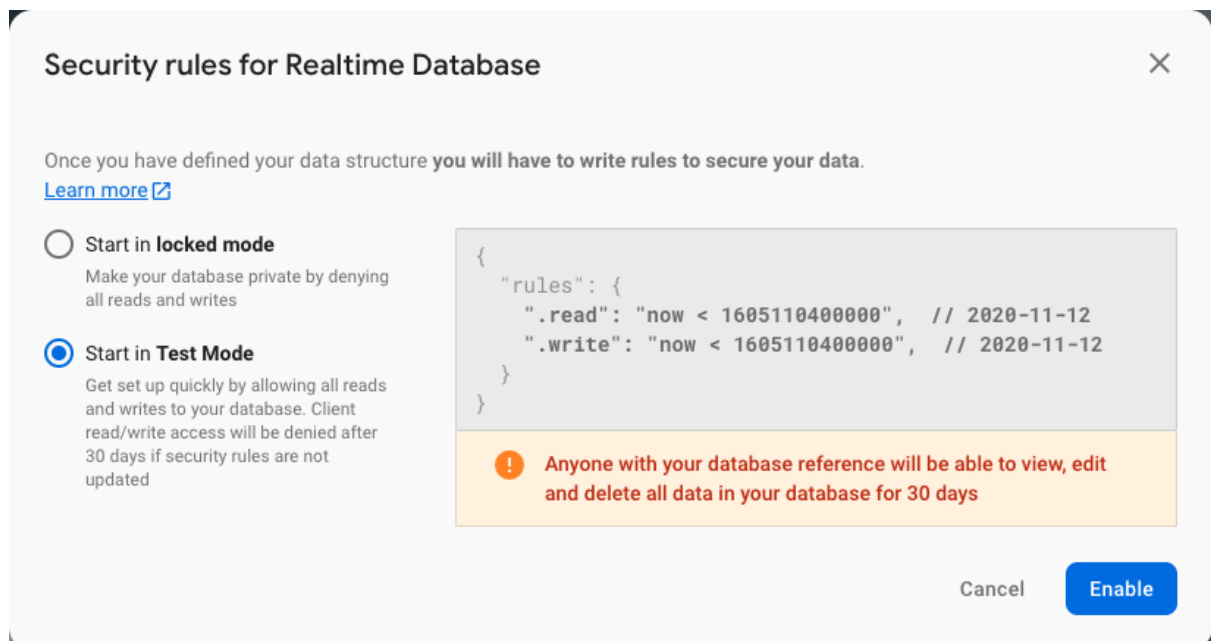
Initialize your databases

Set up the Firebase Realtime Database.

This is the **Database** item on the left-hand menu in your firebase console.

https://console.firebase.google.com/project/<your_project>/overview

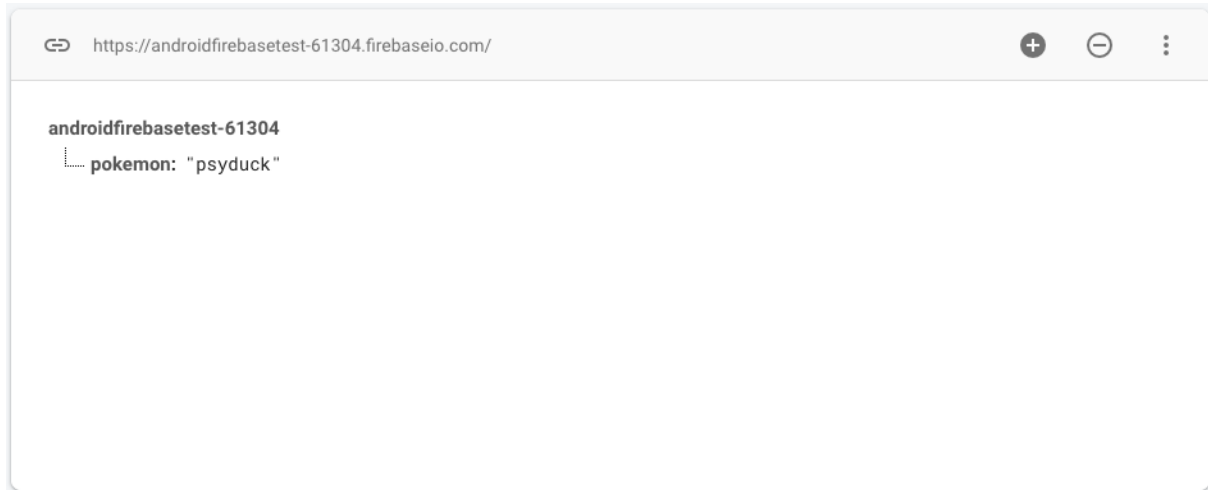
It will prompt you to set a security rule. Let's go with the **Test Mode** to eliminate the need of setting up the authentication for the app. We can update the rules later in the **rule** tab.



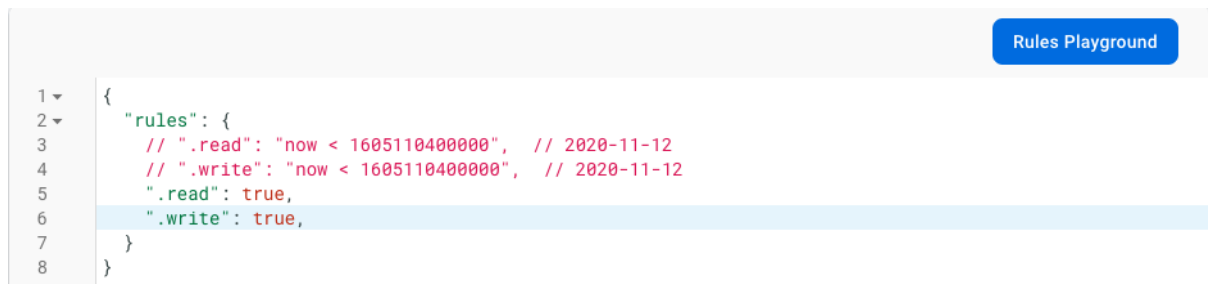
Initially your database will be empty.



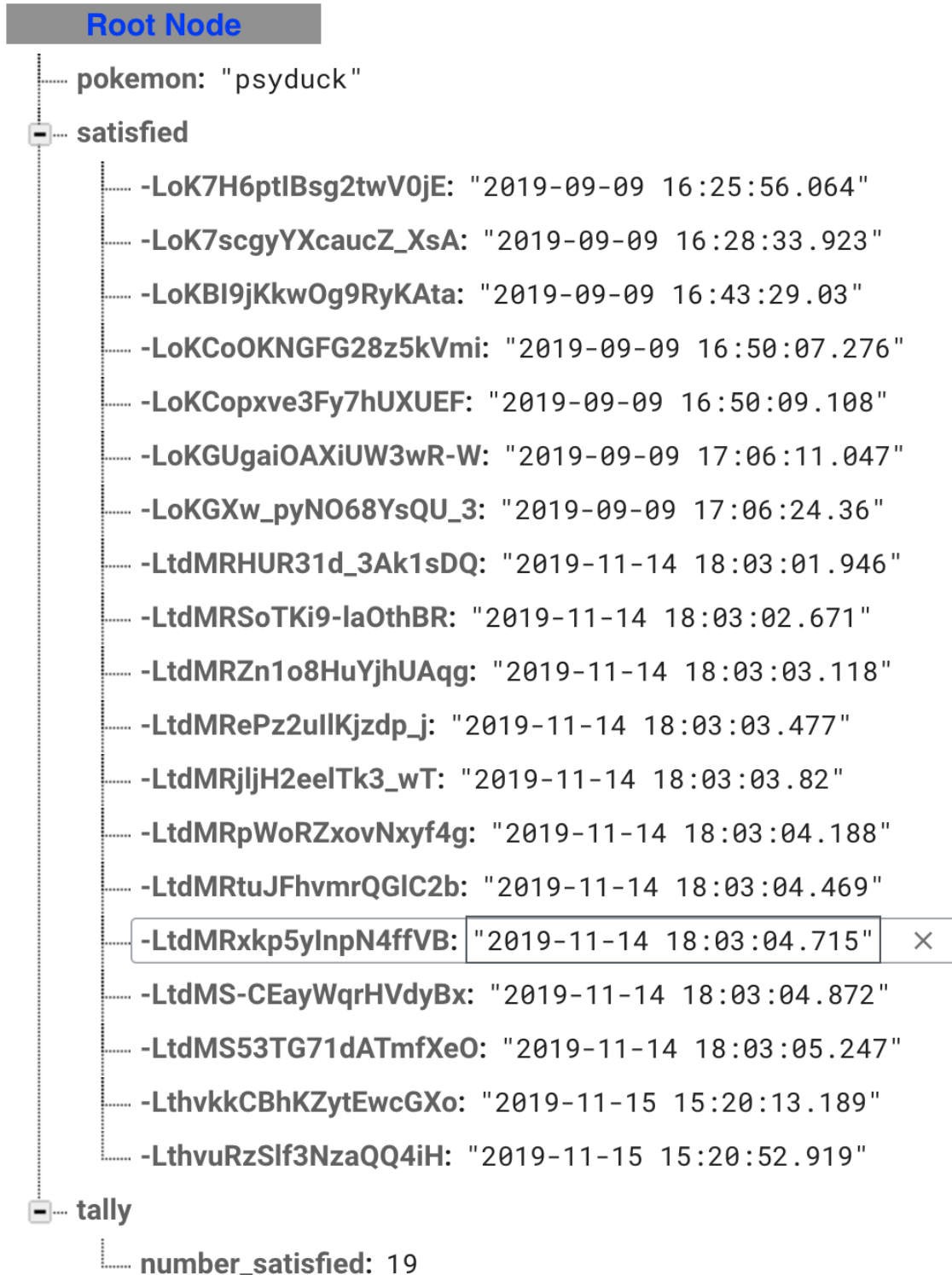
Set it up so that it will contain one key-value pair.



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



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



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.

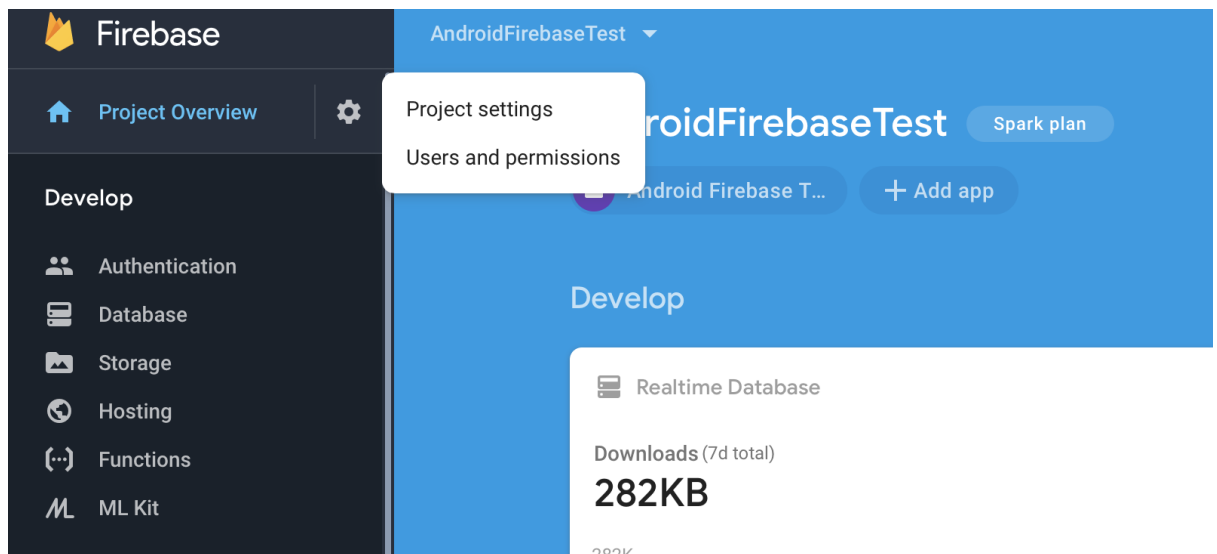
```

1  rules_version = '2';
2  service firebase.storage {
3    match /b/{bucket}/o {
4      match /{allPaths=**} {
5        // allow read, write: if request.auth != null;
6        allow read, write;
7      }
8    }
9  }
10

```

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

Initializing the connection instance

Before performing any operation to the realtime database, we need to establish the connection.

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

Recall that Firebase realtime database is a NoSQL database whose data are stored in a JSON-like structure. The last statement above gives us the reference to the root node of the database.

Database design

Though database design is not the focus in this module, we still have to think about how to organize the data. 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

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" }
```

To retrieve the value back, we need to use the event listener.

Event Listeners

We can add event listeners to the data nodes to perform actions w.r.t the data change, i.e. display the update value on screen.

```
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());
        }
    });
```

What if I want to make the “// action” part customizable?

Pass in an anonymous class object! Recall delegation from Lesson 1.

Working with List data

To append an entry to a list of data under a node,

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

Note that push() will create an unique key for the value being added.

To handle change of list events, we normally can use child event listeners

```
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());
}
});

```

There are situations in which we need to get a copy of the stored list, we need a different event handler.

```

mNodeRefSatisfied
    .addValueEventListener(new ValueEventListener() {
        @Override
        public void onDataChange(DataSnapshot dataSnapshot) {
            for (DataSnapshot postSnapshot:
                  dataSnapshot.getChildren()) {
                // TODO: handle the post
            }
        }

        @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. 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.

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)`;

```
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.

```
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

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:25.12.0')
    implementation 'com.google.firebase:firebase-database'
```

...

TODO 13.1 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

```
mRootDatabaseRef = FirebaseDatabase.getInstance().getReference();
mNodeRefPokemon = mRootDatabaseRef.child(SAMPLE_NODE);
mNodeRefSatisfied = mRootDatabaseRef.child(SATISFIED);
mNodeRefTally = mRootDatabaseRef.child(TALLY);
```

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

The steps in details

1. Create OnClickListener and set it to imageViewSatisfied
2. In the onClick() method in the listener, create a System current timestamp
3. Push the timestamp as one of the entry to the list items under mNodeRefSatisfied
4. 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

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

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

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

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

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

Part 2 - Pushing and Pulling Images

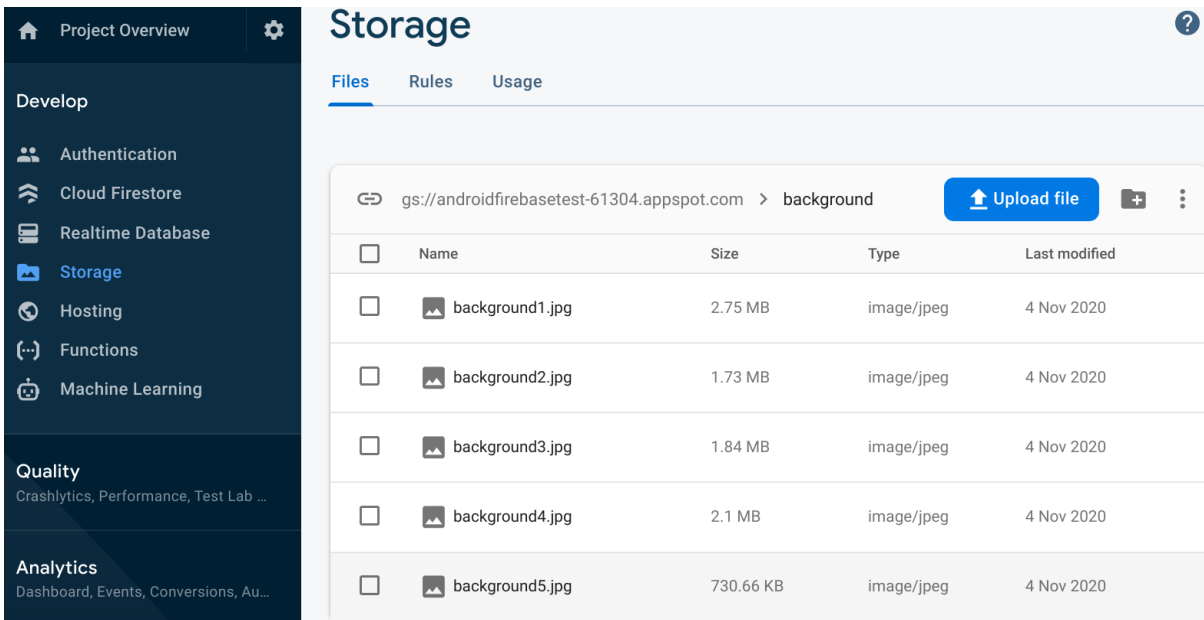
Expected result

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.

Preparation

Upload the images in res/drawable/ to the Firebase Storage associated with this app. Put the images in a folder called “backgrounds”



The screenshot shows the Firebase Storage console interface. On the left is a sidebar with navigation options: Project Overview, Develop (Authentication, Cloud Firestore, Realtime Database, Storage, Hosting, Functions, Machine Learning), Quality (Crashlytics, Performance, Test Lab ...), and Analytics (Dashboard, Events, Conversions, Au...). The main area is titled 'Storage' and shows the 'Files' tab. The breadcrumb path is 'gs://androidfirebasetest-61304.appspot.com > background'. There is an 'Upload file' button and a menu icon. Below is a table listing the files in the 'background' folder:

<input type="checkbox"/>	Name	Size	Type	Last modified
<input type="checkbox"/>	 background1.jpg	2.75 MB	image/jpeg	4 Nov 2020
<input type="checkbox"/>	 background2.jpg	1.73 MB	image/jpeg	4 Nov 2020
<input type="checkbox"/>	 background3.jpg	1.84 MB	image/jpeg	4 Nov 2020
<input type="checkbox"/>	 background4.jpg	2.1 MB	image/jpeg	4 Nov 2020
<input type="checkbox"/>	 background5.jpg	730.66 KB	image/jpeg	4 Nov 2020

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

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

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

```
imageViewBackground.setOnClickListener(new View.OnClickListener(){
    @Override
    public void onClick(View view) {
        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`

```
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 Add an `onClick` listener to the `uploadButton` in which the `onClick` method creates an implicit intent of image gallery (refer to lesson 4: Pokedex app)

```
uploadButton.setOnClickListener(new View.OnClickListener(){
    @Override
    public void onClick(View view) {
        Intent intent = new Intent(Intent.ACTION_GET_CONTENT);
        ...
    }
});
```

TODO 14.5 Define an `onActivityResult` method to retrieve the image selected and returned by the implicit intent. (refer to lesson 4: Pokedex app) Given the image, convert it to `Bitmap`. Call `FirebaseUtils.uploadImageToStorage` to upload it to the firebase storage.

```
protected void onActivityResult(int requestCode, int resultCode, Intent data) {
    if (requestCode == REQUEST_IMAGE_GET && resultCode == RESULT_OK) {
        Uri fullPhotoUri = data.getData();
        try {
            String filename =
                FirebaseUtils.getFileName(MainActivity.this, fullPhotoUri);
            StorageReference storageRef =
                FirebaseStorage.getInstance().getReference();
            StorageReference imageRef =
                storageRef.child("/background/"+filename);
            Bitmap bitmap =
                MediaStore.Images.Media.getBitmap(this.getContentResolver(),
                fullPhotoUri);
            FirebaseUtils.uploadImageToStorage(MainActivity.this,
                imageRef, bitmap);
        } catch (IOException e) {
            Toast.makeText(MainActivity.this, R.string.io_error,
                Toast.LENGTH_LONG);
        }
    } else {
        Toast.makeText(MainActivity.this, R.string.file_not_found,
            Toast.LENGTH_LONG);
    }
}
```