Workshop Schedule

Day 1: Android Studio basics, basic app creation, interactivity

Day 2: Debugging, Intents, Layouts, Lists, Introduction to Firebase

Day 3: Firebase Integration, Android App Development Process, alternatives, expanding knowledge

Workshop resources

https://github.com/slimechips/AndroidIAP

All resources for this workshop can be found here

Notes:

The text in orange are meant for advanced users

1. Adding Firebase to Your Project

(Credits to https://firebase.google.com/docs/)

You might have realised that so far, we have created a relatively functional app. However, there are many problems with this app that make it impractical in real life. For example:

1. There is no login system
2. There is no internet database. The votes you have for each item are purely local, and get reset every time you reset your app.

One way to resolve the issue of values getting reset everytime is to use **SharedPreferences**, which save your session. However, this does not solve the issue of the votes being only local.

Firebase is a useful platform that can solve both these issues. With Firebase, setting up a database is easy, and it also simplifies your login and authentication system. It even has a Machine Learning Kit that you can use.

Firebase is generally regarded to be a very simplified solution to things that a mobile app needs. In reality, you app requires a dedicated backend server to handle all internet requests, but for the purpose of many proof-of-concepts and small projects, Firebase is sufficient. Integrating Firebase into your App is also very easy.

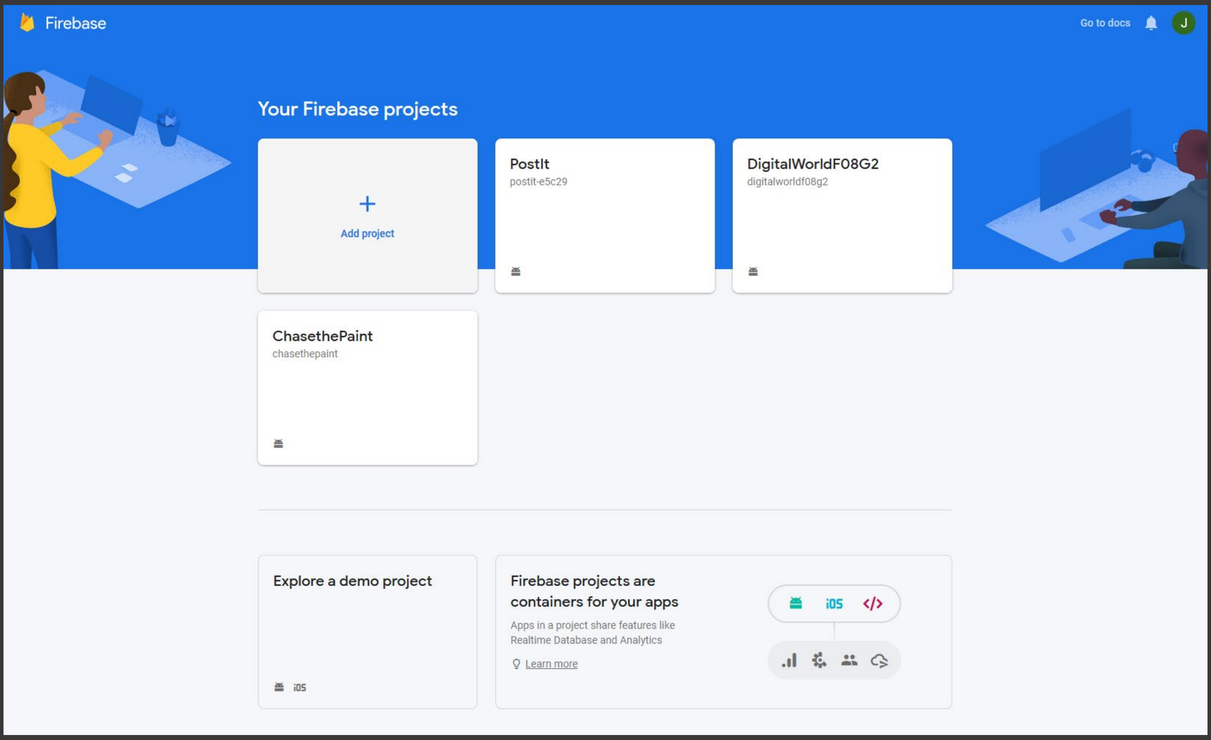
For today, we will just focus on setting up a Firebase account.

Set up your Firebase Account

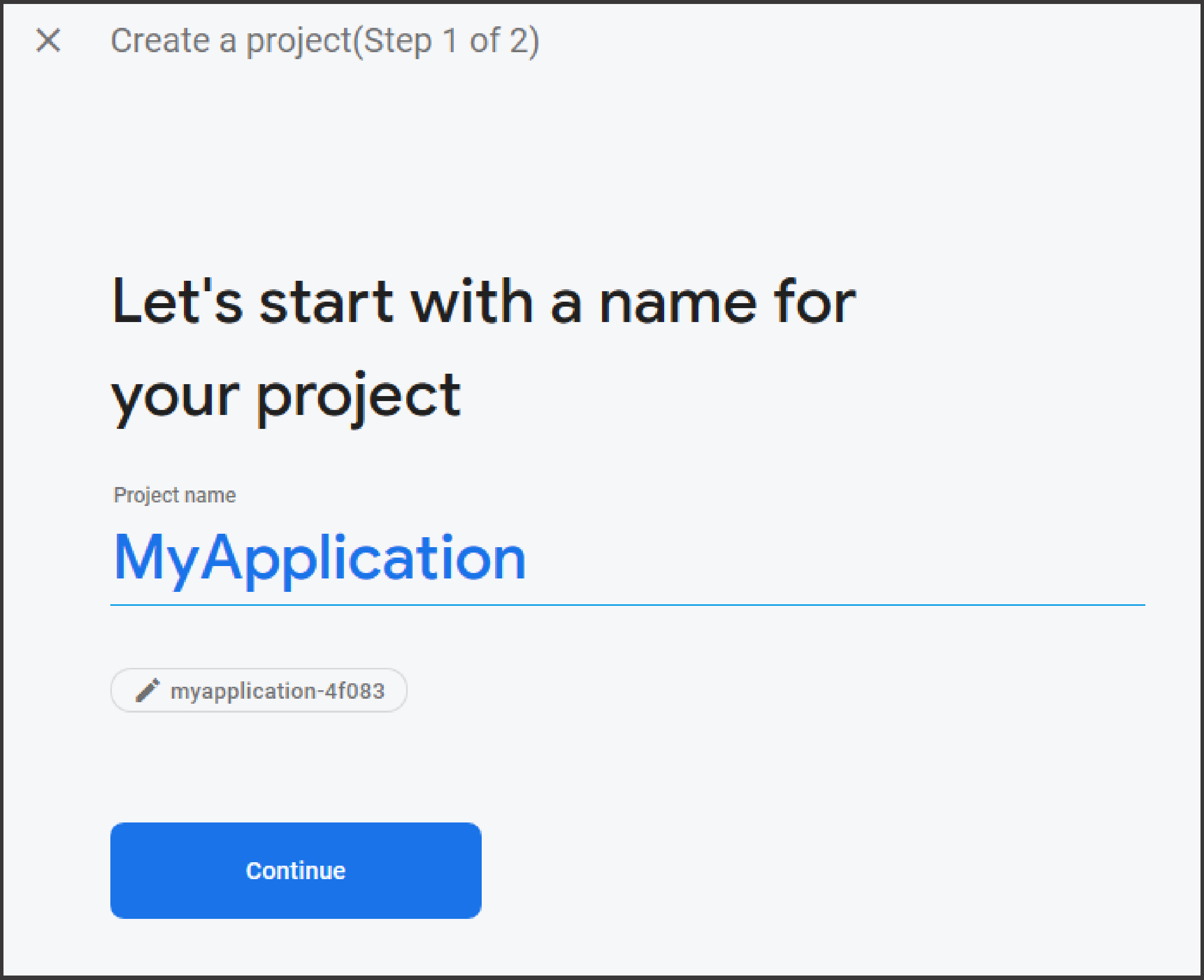
Go to https://firebase.google.com, and sign in with a google account.

Create a Firebase Project

Go to the firebase console at https://console.firebase.google.com/.

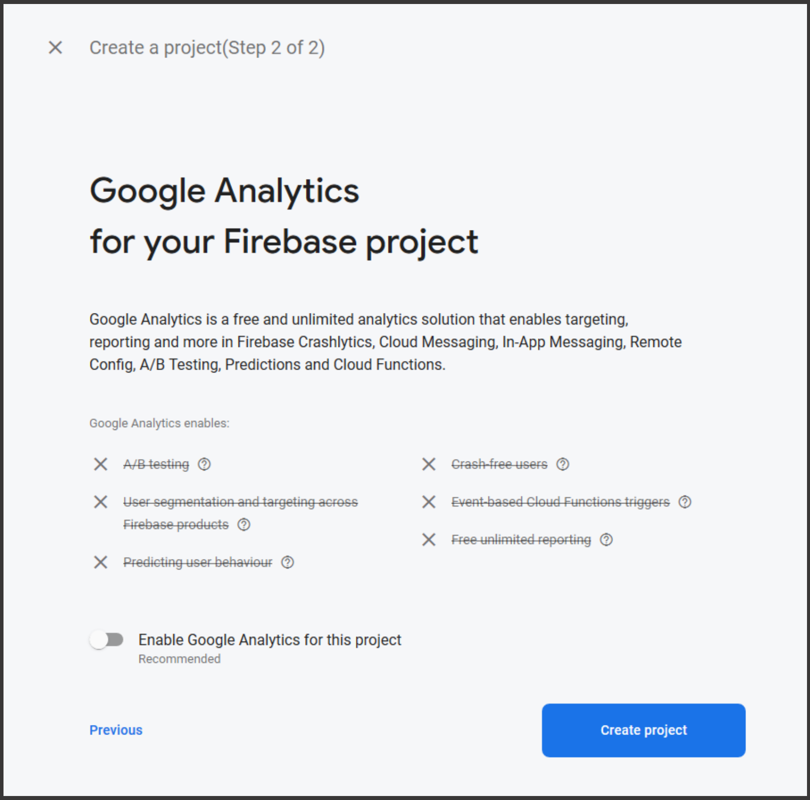


Click on the big **Add Project** button.


Create a project(Step 1 of 2)
Let's start with a name for your project
Project name
MyApplication
myapplication-4f083
Continue

Enter your Application name.

Disable Google Analytics and create your project.



Create a project(Step 2 of 2)

Google Analytics
for your Firebase project

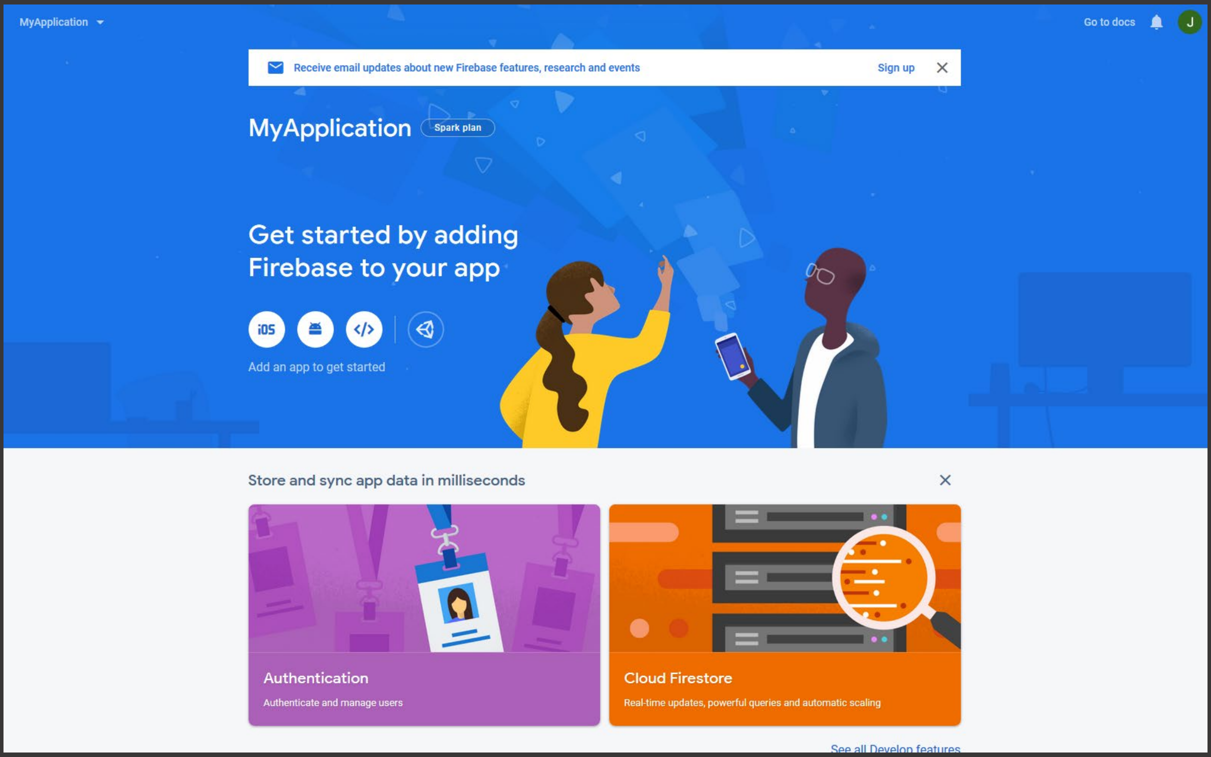
Google Analytics is a free and unlimited analytics solution that enables targeting, reporting and more in Firebase Crashlytics, Cloud Messaging, In-App Messaging, Remote Config, A/B Testing, Predictions and Cloud Functions.

Google Analytics enables:

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

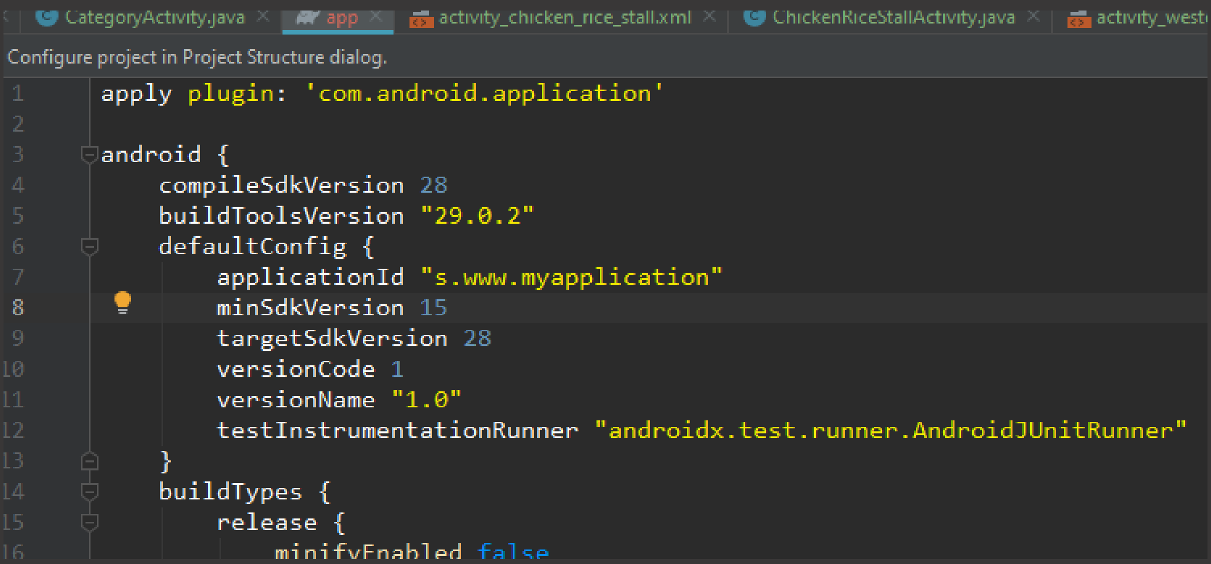
Enable Google Analytics for this project
Recommended

After it is done, click on the Android Button on the main page.



MyApplication
Go to docs
Receive email updates about new Firebase features, research and events
Sign up
MyApplication
Spark plan
Get started by adding Firebase to your app
iOS
Add an app to get started
Store and sync app data in milliseconds
Authentication
Authenticate and manage users
Cloud Firestore
Real-time updates, powerful queries and automatic scaling
See all Develop features

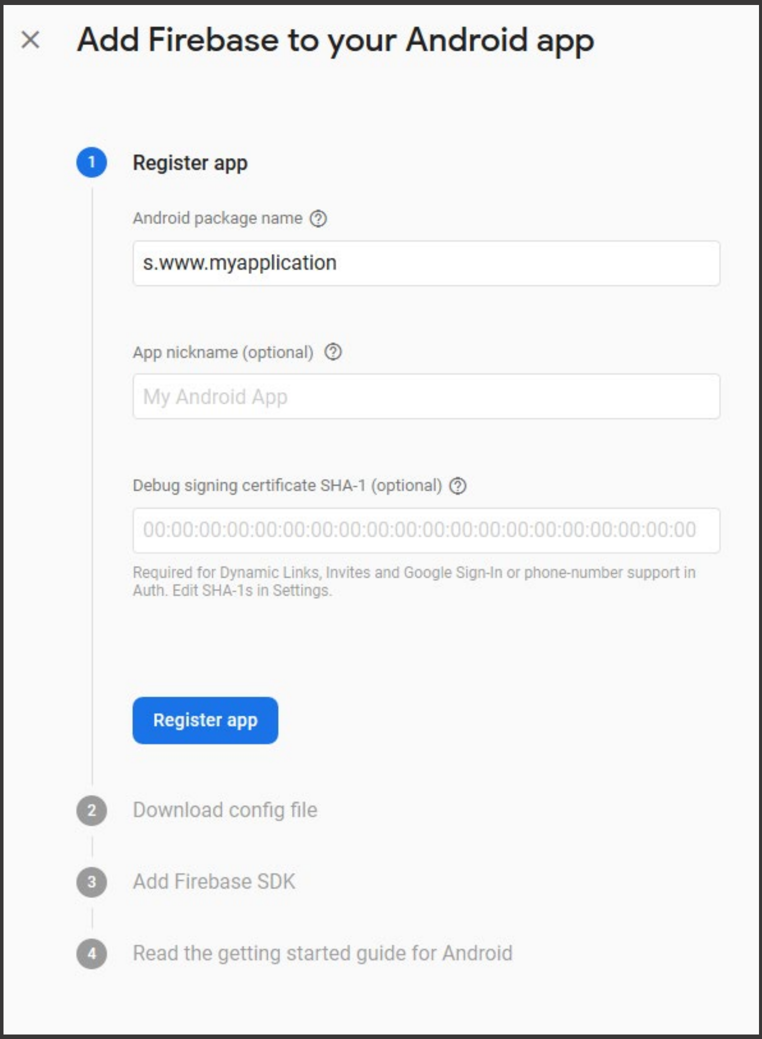
Find your application name in the **build.gradle(Module:app)** file in your project. You can find the build.gradle file under **Gradle Scripts.** Look for the **applicationId.**


CategoryActivity.java
app
activity_chicken_rice_stall.xml
ChickenRiceStallActivity.java
activity_west

Configure project in Project Structure dialog.

```
apply plugin: 'com.android.application'

android {
    compileSdkVersion 28
    buildToolsVersion "29.0.2"
    defaultConfig {
        applicationId "s.www.myapplication"
        minSdkVersion 15
        targetSdkVersion 28
        versionCode 1
        versionName "1.0"
        testInstrumentationRunner "androidx.test.runner.AndroidJUnitRunner"
    }
    buildTypes {
        release {
            minifyEnabled false
```

Copy the applicationId into the Android Package Name on Firebase.



Add Firebase to your Android app

1 Register app

Android package name: s.www.myapplication

App nickname (optional): My Android App

Debug signing certificate SHA-1 (optional): 00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00

Required for Dynamic Links, Invites and Google Sign-In or phone-number support in Auth. Edit SHA-1s in Settings.

2 Download config file

3 Add Firebase SDK

4 Read the getting started guide for Android

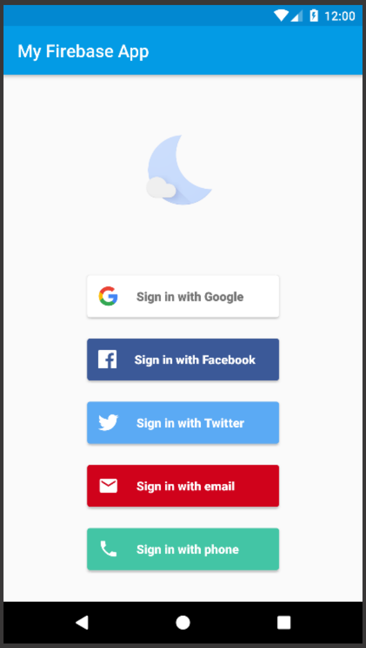
Follow the rest of the instructions on Firebase.

2. Setting up Firebase for Authentication

You may require users to login to an app for various purposes, but login systems are often complicated, requiring various security authorisation protocols. Implementation of such flows often require public-key infrastructure, id tokens etc. Firebase has an authentication system and UI that can implement all of the following for you. You just need to add a button to link to Firebase's authentication system, and then let Firebase do the work. Then you wait for Firebase to return a result to you, like whether or not the authentication or sign up was successful.

Adding FirebaseUI to our Project

FirebaseUI is a prebuilt authentication UI, that provides whatever Authentication providers we want to specify. Their UI looks something like this:



Our implementation for authentication shall be as follows:

1. User clicks on the button in MainActivity to sign-in
2. MainActivity will bring up the above FirebaseUI to handle the sign-in
3. When the sign-in process is completed by FirebaseUI,
    a. **If successful:** Redirect user to the Category Selection page. Also Toast a message to welcome the user.
    b. If **unsuccessful**: **Toast** an error message and stay on MainActivity.

To get started, add the following line to your **Module: app Gradle File,** under **dependencies**:

```
implementation 'com.google.firebase:firebase-auth:19.2.0'
implementation 'com.firebaseui:firebase-ui-auth:4.3.1'
```

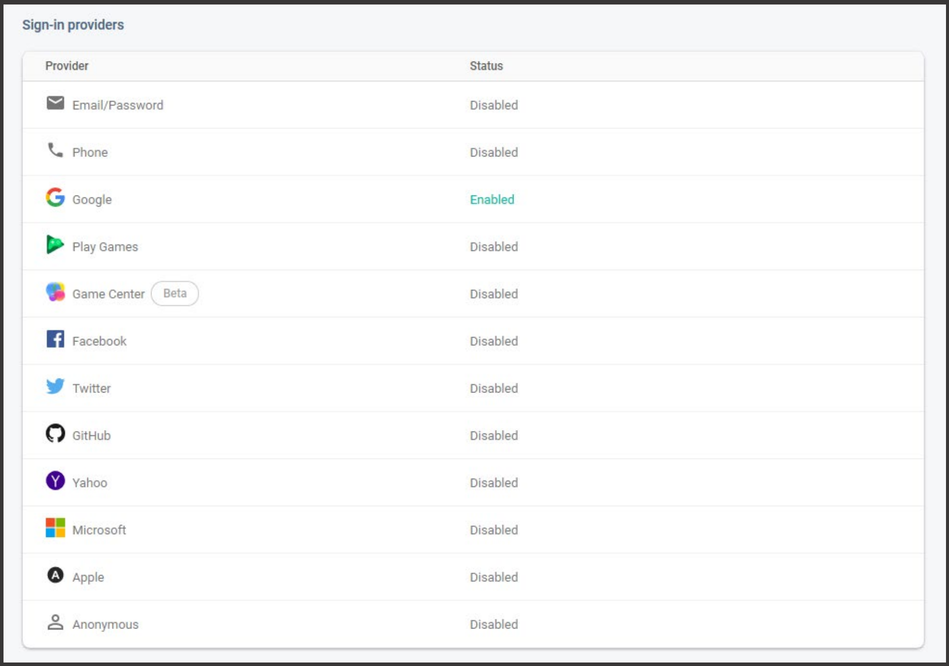
Also, you will need to upgrade your **minSdkVersion** to 16 like so

```
android {
    compileSdkVersion 28
    buildToolsVersion "29.0.2"
    defaultConfig {
        applicationId "s.www.myapplication"
        minSdkVersion 16
        targetSdkVersion 28
        versionCode 1
        versionName "1.0"
        testInstrumentationRunner
"androidx.test.runner.AndroidJUnitRunner"
    }
```

Remember to press **Sync Now** on the notification bar that appears near the top.

Enabling Authentication Providers in Firebase Console

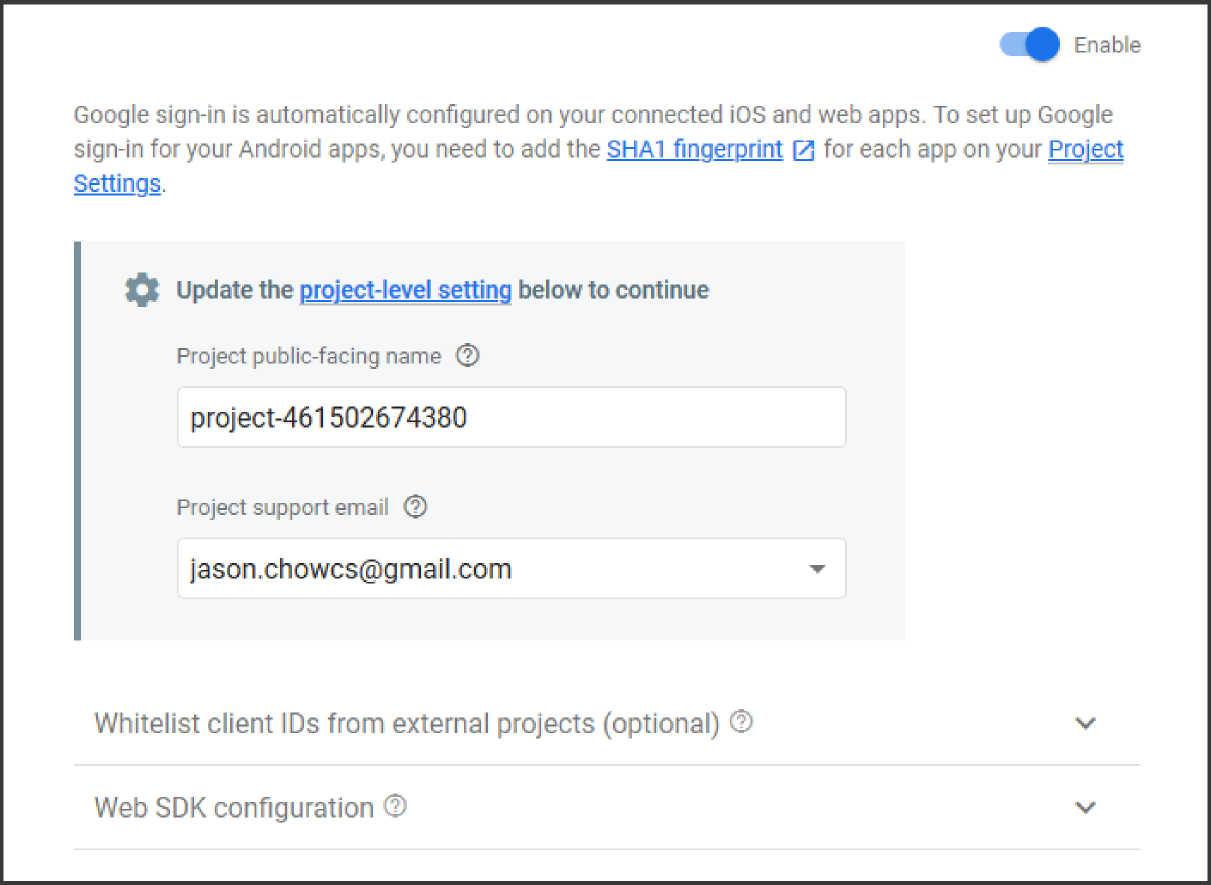
To use an authentication provider, you need to enable it in the Firebase console. Go to the Sign-in Method page in the Firebase Authentication section to enable Email/Password sign-in and any other identity providers you want for your app. For now, let us just try using **Google Sign-in**.



Sign-in providers

Provider	Status
Email/Password	Disabled
Phone	Disabled
Google	Enabled
Play Games	Disabled
Game Center Beta	Disabled
Facebook	Disabled
Twitter	Disabled
GitHub	Disabled
Yahoo	Disabled
Microsoft	Disabled
Apple	Disabled
Anonymous	Disabled

In Google’s case, they will require you to provide a **SHA1 fingerprint** of our app’s signing certificate for our app to enable their sign-in.



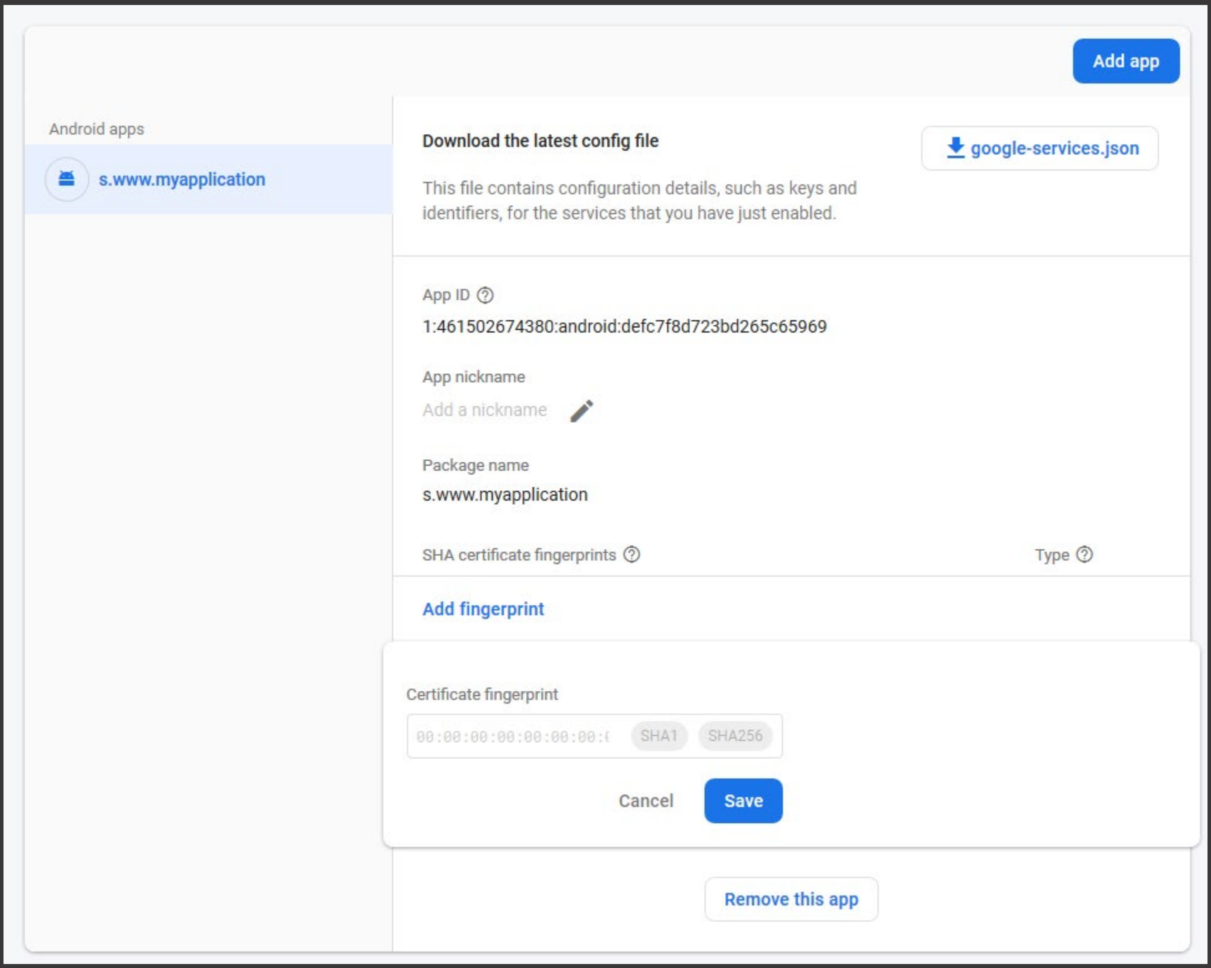
Google sign-in is automatically configured on your connected iOS and web apps. To set up Google sign-in for your Android apps, you need to add the SHA1 fingerprint for each app on your Project Settings.

Update the project-level setting below to continue

Project public-facing name: project-461502674380

Project support email: jason.chowcs@gmail.com

Click on the project settings. It will bring you to the project settings page, where there is a field for you to add your SHA1 fingerprint.



Download the latest config file

This file contains configuration details, such as keys and identifiers, for the services that you have just enabled.

App ID

1:461502674380:android:defc7f8d723bd265c65969

App nickname

Add a nickname

Package name

s.www.myapplication

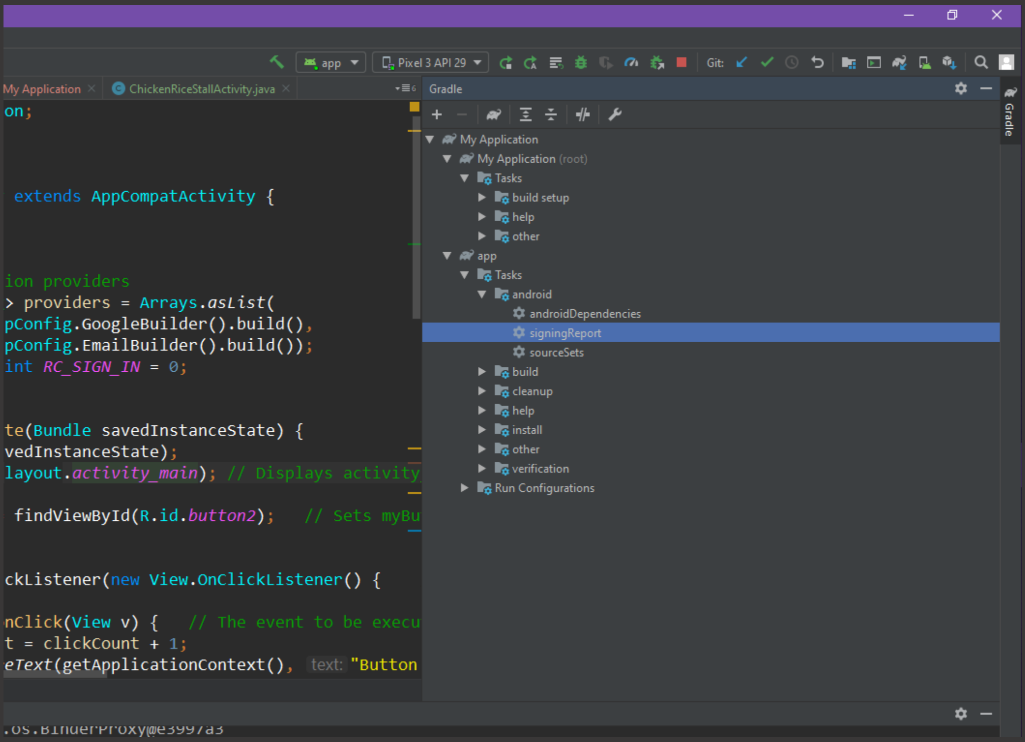
SHA certificate fingerprints

Type

Certificate fingerprint

00:00:00:00:00:00:00:0

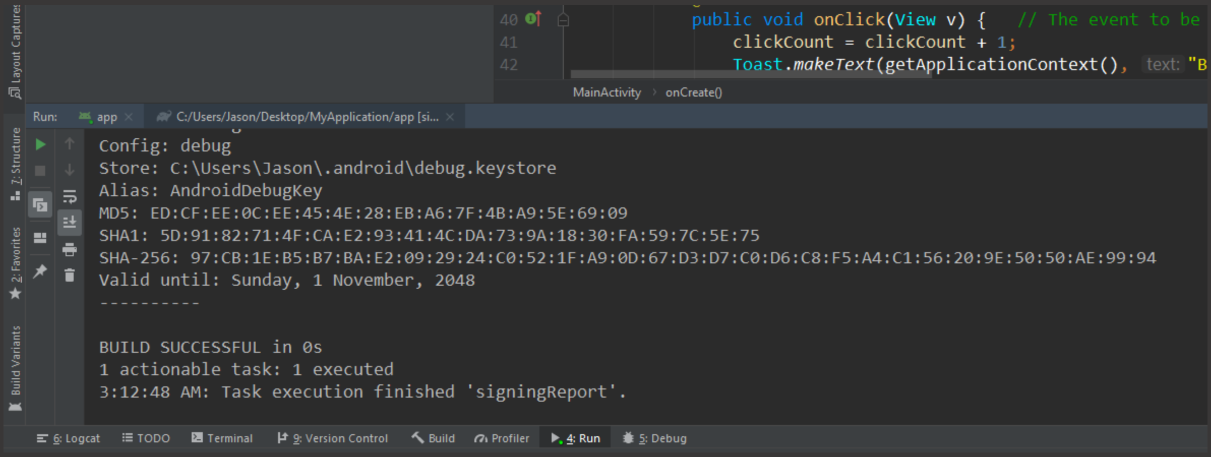
We will need to generate this SHA1 fingerprint. To do so, open the gradle tab on the right of **Android Studio.** Open **app -> Tasks -> android -> signingReport**.



```
on;


 extends AppCompatActivity {


ion providers
> providers = Arrays.asList(
pConfig.GoogleBuilder().build(),
pConfig.EmailBuilder().build());
int RC_SIGN_IN = 0;


te(Bundle savedInstanceState) {
vedInstanceState);
layout.activity_main); // Displays activity

 findViewById(R.id.button2);   // Sets myBu


ckListener(new View.OnClickListener() {

nClick(View v) {   // The event to be execu
t = clickCount + 1;
eText(getApplicationContext(),  text: "Button
```

```
.os.BinderProxy@e3997a3
```

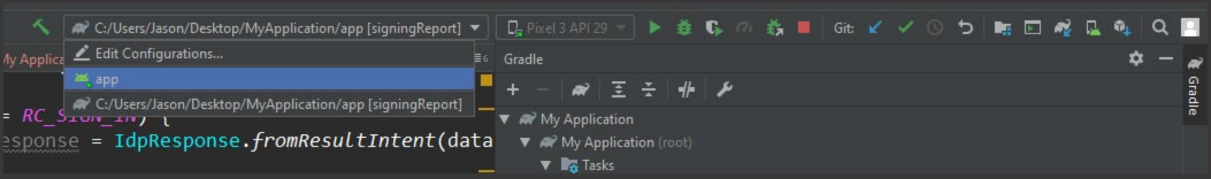
The console should pop out and you will see your generated SHA1 fingerprint here:



```
Config: debug
Store: C:\Users\Jason\.android\debug.keystore
Alias: AndroidDebugKey
MD5: ED:CF:EE:0C:EE:45:4E:28:EB:A6:7F:4B:A9:5E:69:09
SHA1: 5D:91:82:71:4F:CA:E2:93:41:4C:DA:73:9A:18:30:FA:59:7C:5E:75
SHA-256: 97:CB:1E:B5:B7:BA:E2:09:29:24:C0:52:1F:A9:0D:67:D3:D7:C0:D6:C8:F5:A4:C1:56:20:9E:50:50:AE:99:94
Valid until: Sunday, 1 November, 2048
----------

BUILD SUCCESSFUL in 0s
1 actionable task: 1 executed
3:12:48 AM: Task execution finished 'signingReport'.
```

Copy the SHA1 into the **Add fingerprint** field of the Firebase console. Done!

Remember to set back the build configuration in your Android studio to **app**.


C:/Users/Jason/Desktop/MyApplication/app [signingReport]
Edit Configurations...
app
C:/Users/Jason/Desktop/MyApplication/app [signingReport]
Pixel 3 API 29
Gradle
My Application
My Application (root)
Tasks

Bringing up FirebaseUI with a Button Click

Before we start changing our Java Code, let's change the button display text! Go to **activity_main.xml** and change the button text to **Sign In**.

Go to **MainActivity.java** and we shall add the function to bring up FirebaseUI with the button click. First, we need to create a sign in intent with our preferred sign-in methods. In this case, it would just be **Google**.

Add the following code the before the **onCreate** function.

```
List<AuthUI.IdpConfig> providers = Arrays.asList(
        new AuthUI.IdpConfig.GoogleBuilder().build());
private static final int RC_SIGN_IN = 0;
```

We are stating the providers that we are choosing to use here. Note that you will probably need to import some things. **RC_SIGN_IN** is a sort of tag that we will be assigning to our sign-in intent.

Now, add the following code **in the onClick function** of our button onClickListener.

```
// Create and launch sign-in intent
startActivityForResult(
        AuthUI.getInstance()
```

```
                    .createSignInIntentBuilder()
                    .setAvailableProviders(providers)
                    .build(),
          RC_SIGN_IN);
```

Also, remove the two lines of code that create the intent for the new Category Activity and start it. We do not want to move on to the next Activity until the sign-in process is successful.

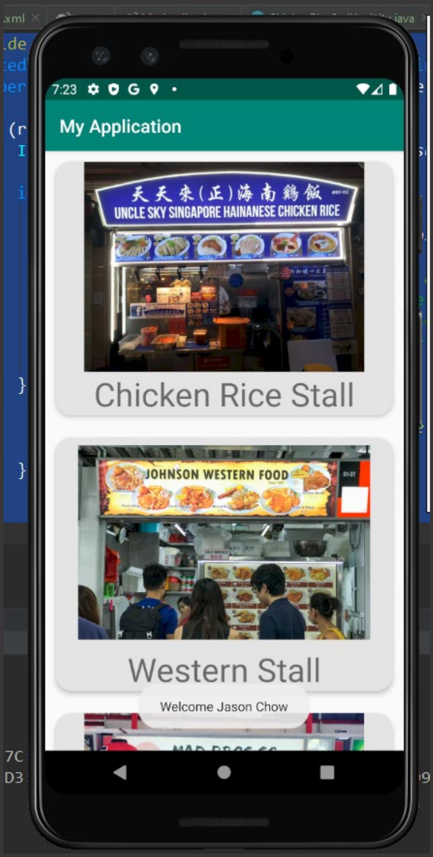
Add the following line of code **after the entire block of the onCreate function.** This function is executed when FirebaseUI is done with the authentication process.

We check whether the result does indeed have the sign-in tag, then we check whether it was successful or not.

```
@Override
protected void onActivityResult(int requestCode, int resultCode, Intent data) {
    super.onActivityResult(requestCode, resultCode, data);

    if (requestCode == RC_SIGN_IN) {
        IdpResponse response = IdpResponse.fromResultIntent(data);

        if (resultCode == RESULT_OK) {
            // Successfully signed in
            FirebaseUser user = FirebaseAuth.getInstance().getCurrentUser();

            // TODO: Add code here to Welcome the user with a Toast and go to 
CategoryActivity
            // Remember Toast.makeText and remember to show()!
            // You can get the user's name with user.getDisplayName()
            // You can get the text "Welcome <user>" with the following code:
            // "Welcome " + user.getDisplayName()
        } else {
            // Sign in failed
            // TODO: Add code here to Toast to the user that sign-in failed!

        }
    }
}
```

Fill in the TODOs! Some hints have been provided to you already!

After you are done, your sign-in flow should be complete!



Note that at any point in the app, you can check the current user with the following code

```
FirebaseUser user =
FirebaseAuth.getInstance().getCurrentUser();
```

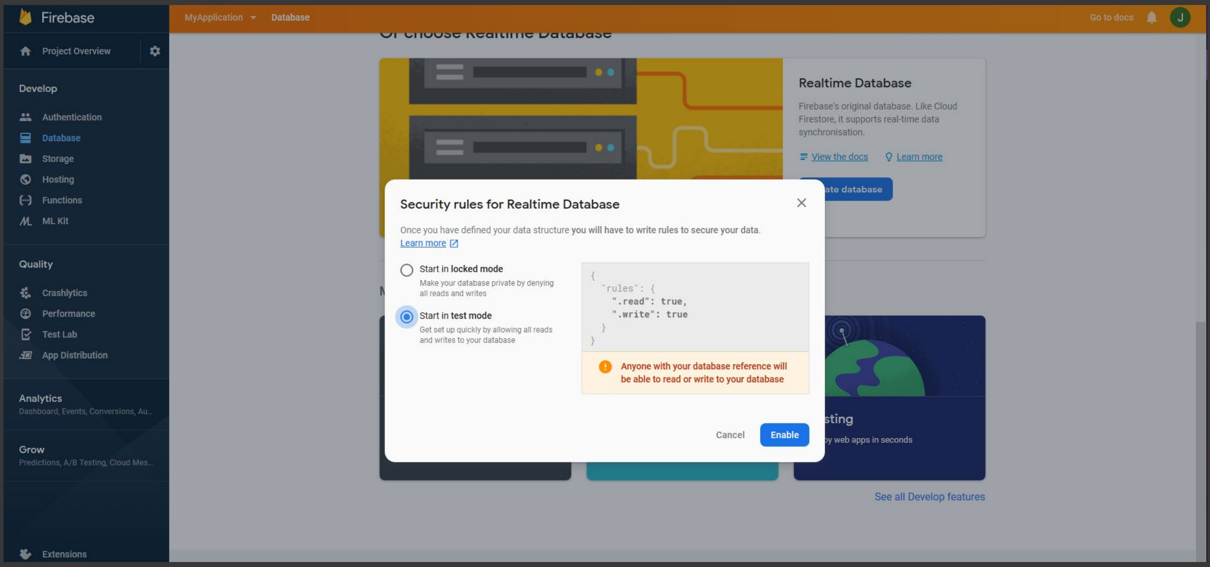
3. Setting up Firebase Database

Recall the voting page for our items that we did last session? We can now connect that to the database. We will

1. Store the number of votes for each item on the database
2. Every time we click on the vote button, we will
    a. Increment the number of votes by 1
    b. **Store** the new number of votes on the database
3. Every time the number of votes on the database is updated, we will update the displayed vote count on our page
4.

Initialising the Database

Let's get started. Head back to the Firebase Console, select **Database** from the left navigation, and Create a new **Realtime Database** (not Cloud Firestore!). Select **test mode**.



Firebase's original database. Like Cloud Firestore, it supports real-time data synchronisation.

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

```
{
  "rules": {
    ".read": true,
    ".write": true
  }
}
```

Anyone with your database reference will be able to read or write to your database

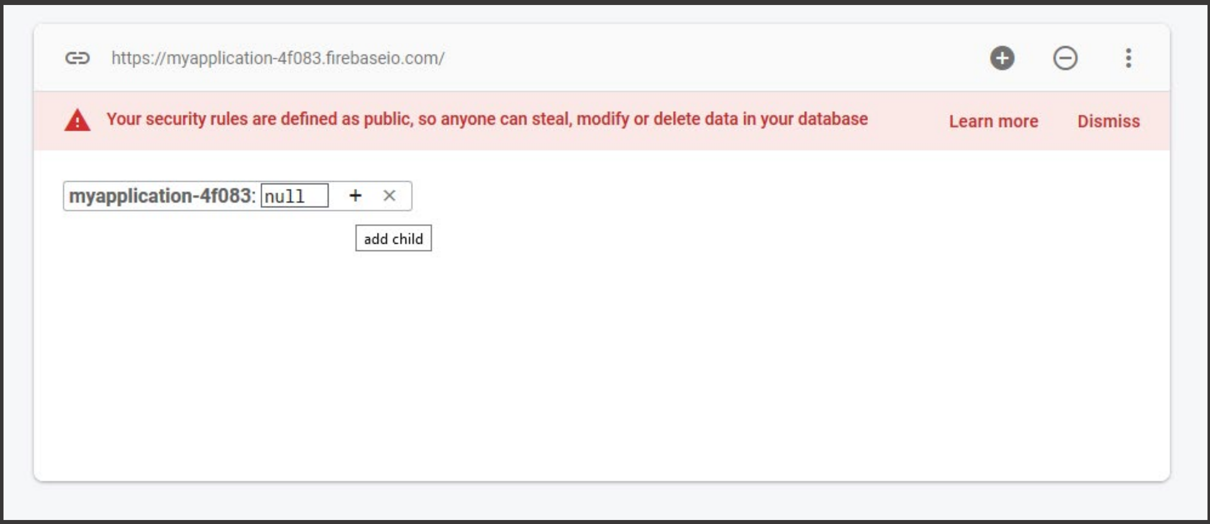
Cancel | Enable

Adding some data to our database

What's a database without some data? Let us add some data! The way Firebase database works is a **NoSQL** format. It is stored as a JSON format file, where data takes on a nested structure. Let us just create some data with the following format

- Myapplication-root
  - <category_name>
    - <item_name>
    - <item_name>

To do so, mouse over the initial item, and click on the + sign. This will create a new child from our root.

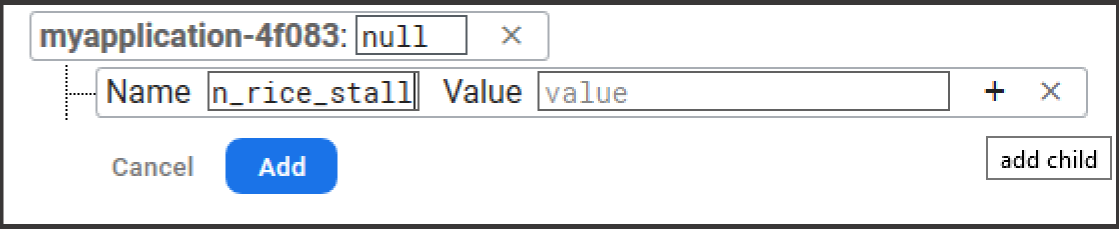

https://myapplication-4f083.firebaseio.com/

Your security rules are defined as public, so anyone can steal, modify or delete data in your database

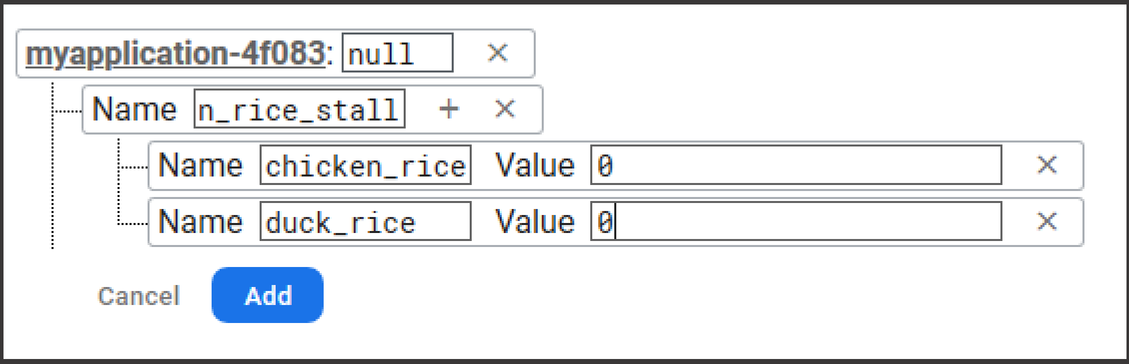
Learn more Dismiss

myapplication-4f083: null

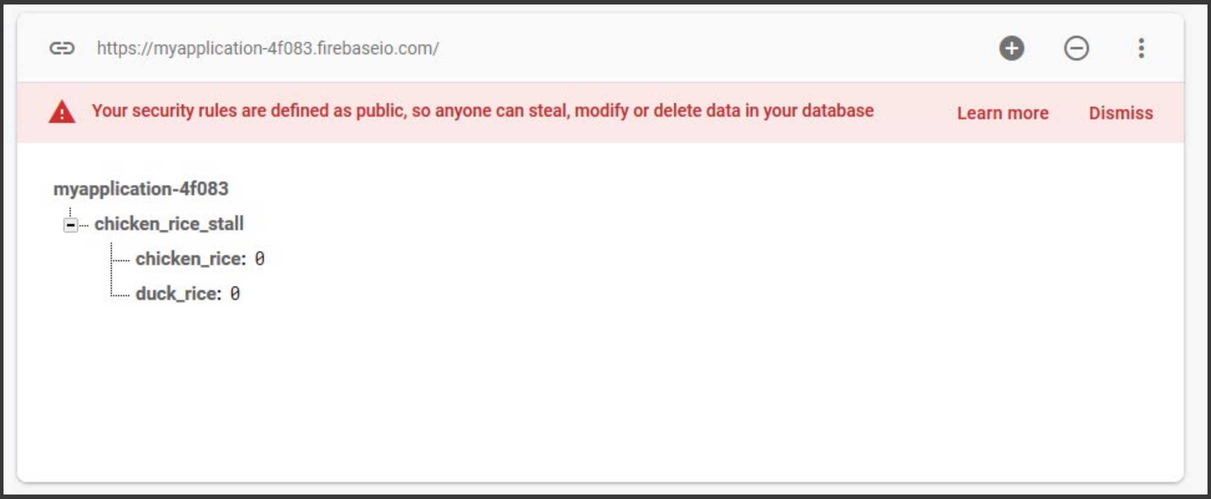
I am going to give this new child a name of chicken_rice_stall, while leaving the value empty, because I want chicken_rice_stall to have children. Then I will mouse over it and click on the + button again, twice in fact to create two new children.


myapplication-4f083: null
Name n_rice_stall Value value

I will give my new items the name of chicken_rice and duck_rice, while initialising their values to 0, since they start with 0 votes.


myapplication-4f083: null
Name n_rice_stall
Name chicken_rice Value 0
Name duck_rice Value 0

Now that I am done, I will click the **Add** button, and the database values have now been updated! It looks like that now:


https://myapplication-4f083.firebaseio.com/

Your security rules are defined as public, so anyone can steal, modify or delete data in your database

myapplication-4f083
chicken_rice_stall
chicken_rice: 0
duck_rice: 0

Getting a Reference to the Database from Android App

First, I will need to add the dependency for Firebase Database in our Android App. Once again, go to the **Module:app build.gradle** file, and add the following under **dependencies:**

```
implementation 'com.google.firebase:firebase-
database:19.2.0'
```

Remember to press **Sync Now** on the notification bar that appears near the top.

NOTE: AT THIS POINT, I RECOMMEND RESTARTING THE ANDROID PHONE (NOT THE EMULATOR APPLICATION) IF YOU ARE USING THE EMULATOR, BECAUSE YOU MAY FACE INTERNET CONNECTIVITY GLITCHES OTHERWISE.

I will head back to the **ChickenRiceStallActivity.java** which contained my item page.

First, I will initialise my votes as 0. I will also create a variable to store my reference to the Firebase Database as such:

```
int chickenRiceVoteGood = 0;
int duckRiceVoteGood = 0;
DatabaseReference mDatabase;
```

Now in the **onCreate** function, I need to get the actual reference to the database. Use the following code, after **setContentView**:

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

Setting up References to Item Data in Database

Now that I have a reference, I just need to get the values from the database, by telling the database where the data that I need is located at.

So the first step is to specify where the data is located. In other words I will need to set up a reference point. I will declare my references first before onCreate:

```
DatabaseReference chickenRiceRef;
DatabaseReference duckRiceRef;
```

Then after my definition for mDatabase, I will define the references for the items:

```
chickenRiceRef = mDatabase.child("chicken_rice_stall").child("chicken_rice");
duckRiceRef = mDatabase.child("chicken_rice_stall").child("duck_rice");
```

These references refer to the locations of the values in the Firebase database. Note how I defined my references, in relation to the hierarchy of my data in the database.

Retrieving Values from the Database

Now that the references have been set, what is left is to actually retrieve the data. We want to

1. Retrieve the data when the page is first loaded

2. We also want to retrieve the data again whenever the data in the database has been updated (could be by another phone).

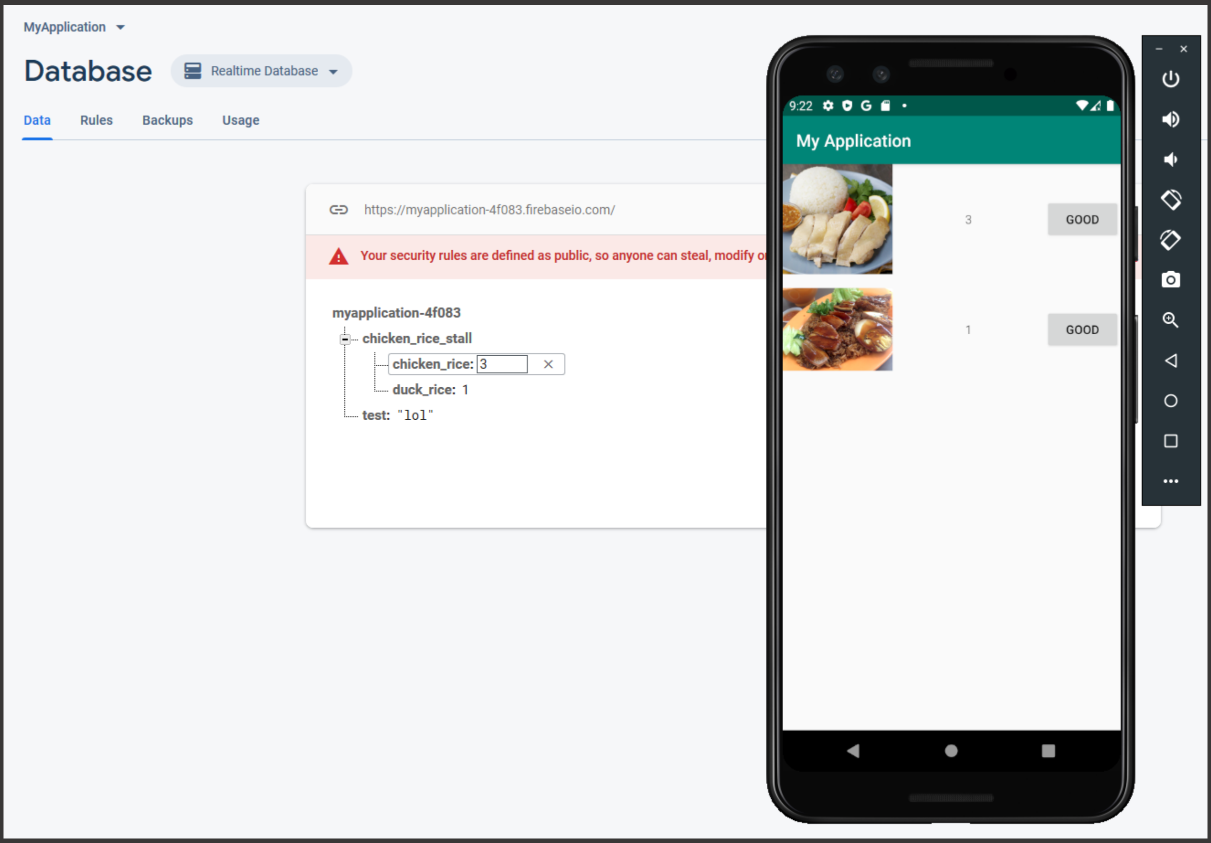
As it turns out, there is a way to implement both operations stated above in a single step. We can add a **ValueEventListener** that by itself periodically checks for updates from the database. The Listener can be attached to a Database Reference (e.g. the reference for each of our items), and it has a function that will be called whenever the database value is updated and also at the start. It will look something like this:

```java
chickenRiceRef.addValueEventListener(new ValueEventListener() {
    @Override
    public void onDataChange(@NonNull DataSnapshot dataSnapshot) {
        // This code in this function is executed when the db value is updated
        // The following line of code retrieves the updated db value as integer
        // And it also sets the chickenRiceVoteGood variable to the updated value
        // TODO: Update the text on the page with this updated value
        chickenRiceVoteGood = dataSnapshot.getValue(int.class);
    }

    @Override
    public void onCancelled(@NonNull DatabaseError databaseError) {

    }
});
```

Note the similarity of this code with adding **onClickListeners**. Whenever the database value is updated, we want to update the text on the page as well, so you just need to implement that part yourself.

Remember to implement the ValueEventListener for the other items as well.

You can test your implementation, by changing the values manually on the Firebase database. The text on your application page should change accordingly.


MyApplication

Database

Realtime Database

Data Rules Backups Usage

https://myapplication-4f083.firebaseio.com/

Your security rules are defined as public, so anyone can steal, modify o

```
myapplication-4f083
  chicken_rice_stall
    chicken_rice: 3
    duck_rice: 1
  test: "lol"
```

Writing/Updating values to the Database

Previously, we manually incremented the number of votes within the local app context whenever the button was clicked. We now want to increment the number of votes on the online database whenever the button is clicked. We have already implemented updating the page text whenever there is an update on the online database, so **we do not need to care about updating our page text anymore**. We just need to care about writing new values to the database on button clicks.

Firstly, we need to remove the **setText** codes that we had previously implemented inside the **onClickListeners**. We can keep the code that we used to increment the variables. Now, we just need to implement the database update inside our onClickListeners.

The proper way to update values on Firebase is to use the **setValue** method on our item database references. Like so:

```
chickenRiceRef.setValue(chickenRiceVoteGood);
```

All we did here is put the new value that we want to update into the database, inside the **setValue** method.

Remember to implement the setValue for other items as well!

After that is done, the end result should have been achieved. Clicking the button now updates the values on the application page, as well as on the Firebase database. If you manually change the values on the Firebase Database, or use another phone to increment the votes, the values should update on your phone as well!

Bonus: If you have previously implemented the percentage rating system where you make use of both good and bad votes, you can/should integrate the bad votes and store them into Firebase as well

4. Distributing your Android Application

It is great that we have a functioning app right now, but how do we distribute the application for use?

One way is to publish your app on the Google Play Store. Publishing your app on Google Play Store obviously means you can target a very broad audience, but the main drawback is that it is not free. You will need to sign up for a Google Developer Account, and that account will cost you 25USD, which may not be worth it, especially if you are planning to make your app free.

An alternative is publish your App on Amazon App Store. You do not have to pay for publishing on Amazon App Store, but be prepared for a smaller user base.

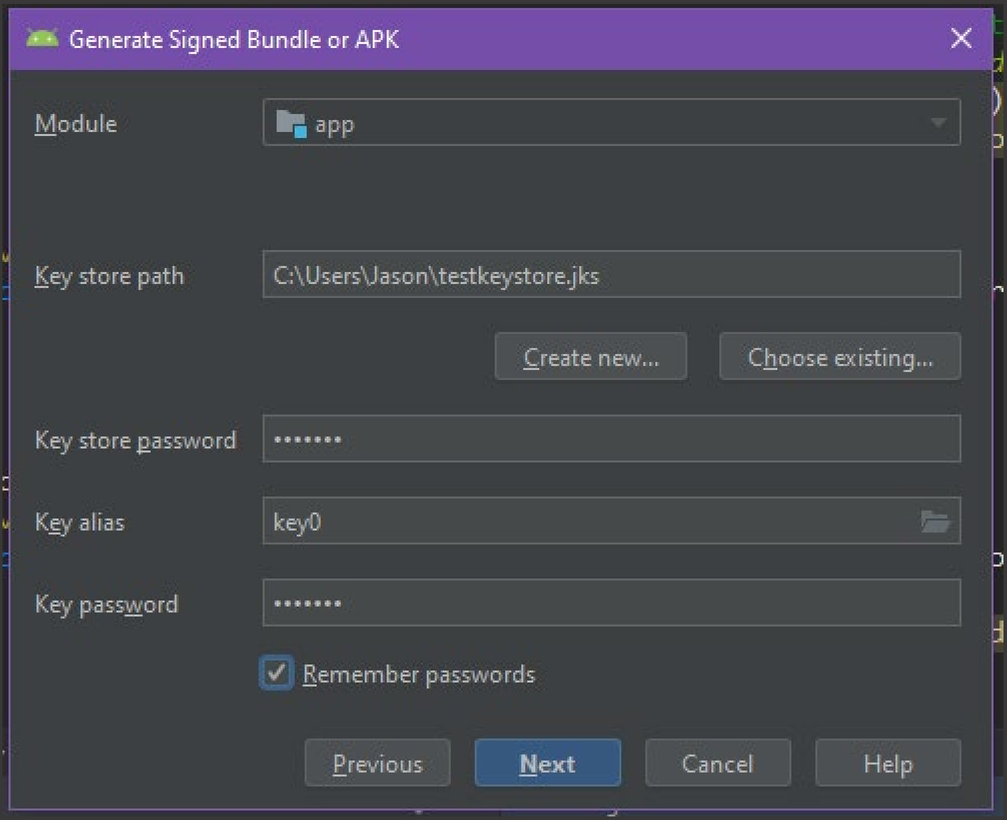
If you have a very specific audience in mind for your app, such as SUTD, you can instead upload your file APK on a file-hosting site or your own website and share the download link instead through publicity channels. However, do note that users will need to allow on their phone installation from 3rd party sources.

Generating an unsigned APK

You can simply generate an unsigned apk distribution by going to **Build -> Build APK/Bundles -> Build APK.** You can get your APK from app/build/outputs/apk/debug. Try sending the APK to yourself through e.g. Telegram, and run it!

Generating a signed APK

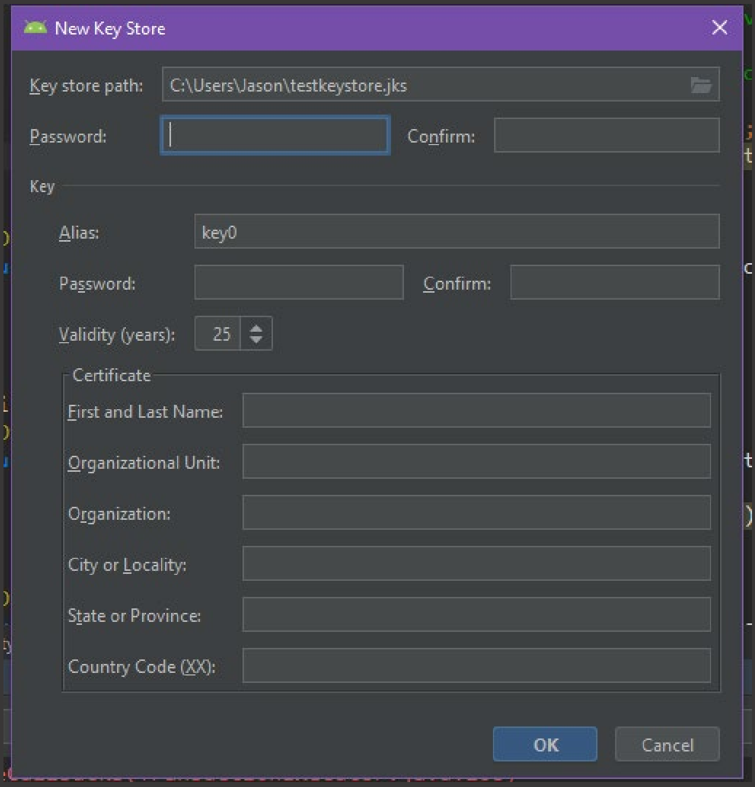
Normally, you will need to sign your APK before releasing it for distribution. To do so, go to **Build -> Generate Signed Bundle / APK,** then select APK. You will see something like this.



Generate Signed Bundle or APK

Module	app
Key store path	C:\Users\Jason\testkeystore.jks
Key store password	•••••••
Key alias	key0
Key password	•••••••

If you don’t have a key store yet, click on **Create new…**, specify a path for your key store and fill in the relevant details.



New Key Store

Key store path: C:\Users\Jason\testkeystore.jks

Password: | Confirm:

Key

Alias: key0

Password: | Confirm:

Validity (years): 25

Certificate

First and Last Name:

Organizational Unit:

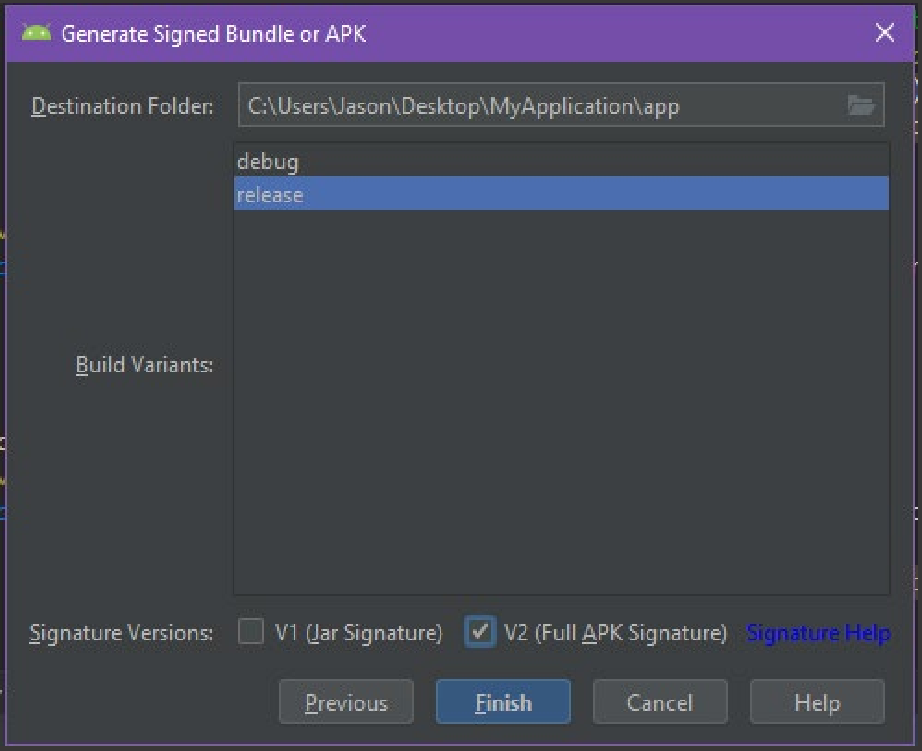
Organization:

City or Locality:

State or Province:

Country Code (XX):

When you are done, click **OK**, then **Next,** then select your build to be for **release** and select **V2** for signature version. Then click **Finish**.



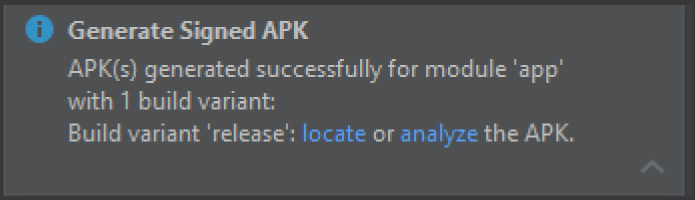
Generate Signed Bundle or APK

Destination Folder: C:\Users\Jason\Desktop\MyApplication\app

Build Variants: debug, release (selected)

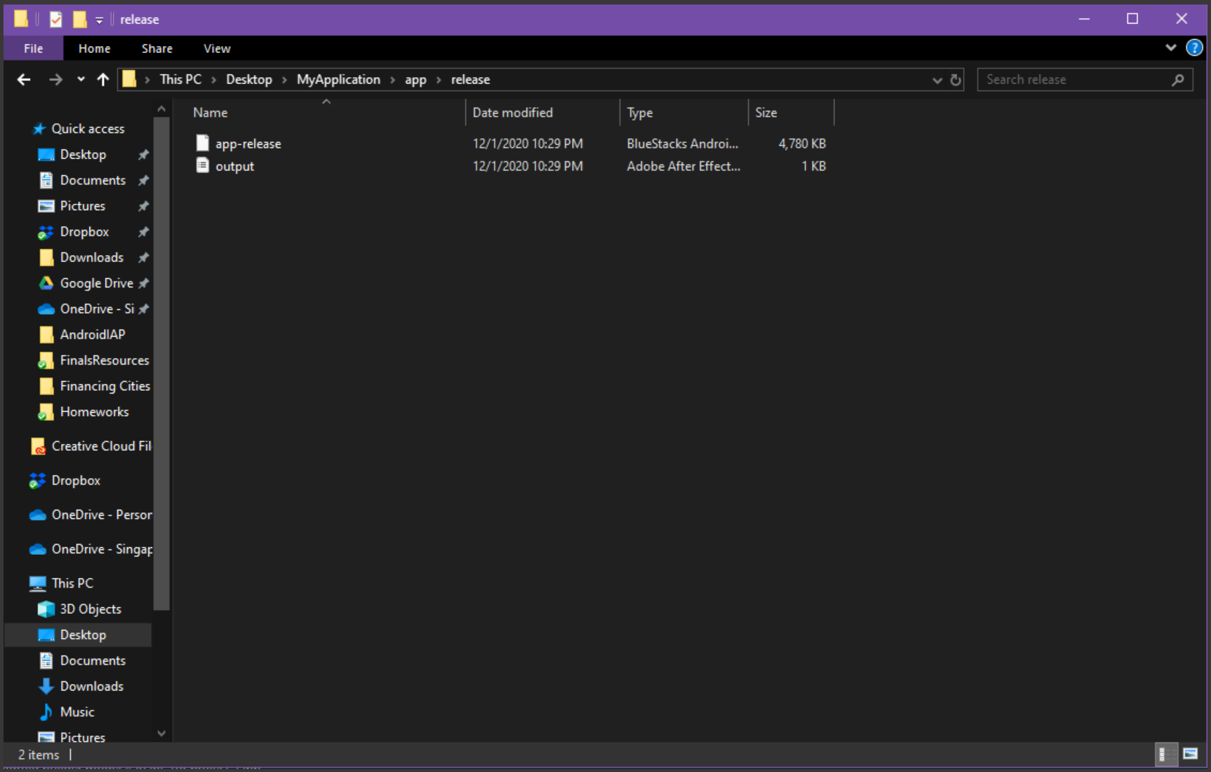
Signature Versions: V1 (Jar Signature) [✓ V2 (Full APK Signature)] Signature Help

Your APK will finish building after some time, and it will show a message. You APK has been created, either click on **Locate**, or go to the **release** folder in your app folder. You will find app-release.apk.


Generate Signed APK

APK(s) generated successfully for module 'app' with 1 build variant:

Build variant 'release': locate or analyze the APK.


This PC > Desktop > MyApplication > app > release

Name	Date modified	Type	Size
app-release	12/1/2020 10:29 PM	BlueStacks Androi...	4,780 KB
output	12/1/2020 10:29 PM	Adobe After Effect...	1 KB

2 items

You may proceed to upload this APK to a file-sharing site, App Store, or send it.

Try sending the APK to yourself through e.g. Telegram, and run it! Note that Firebase will not work due to the SHA-1 being different.

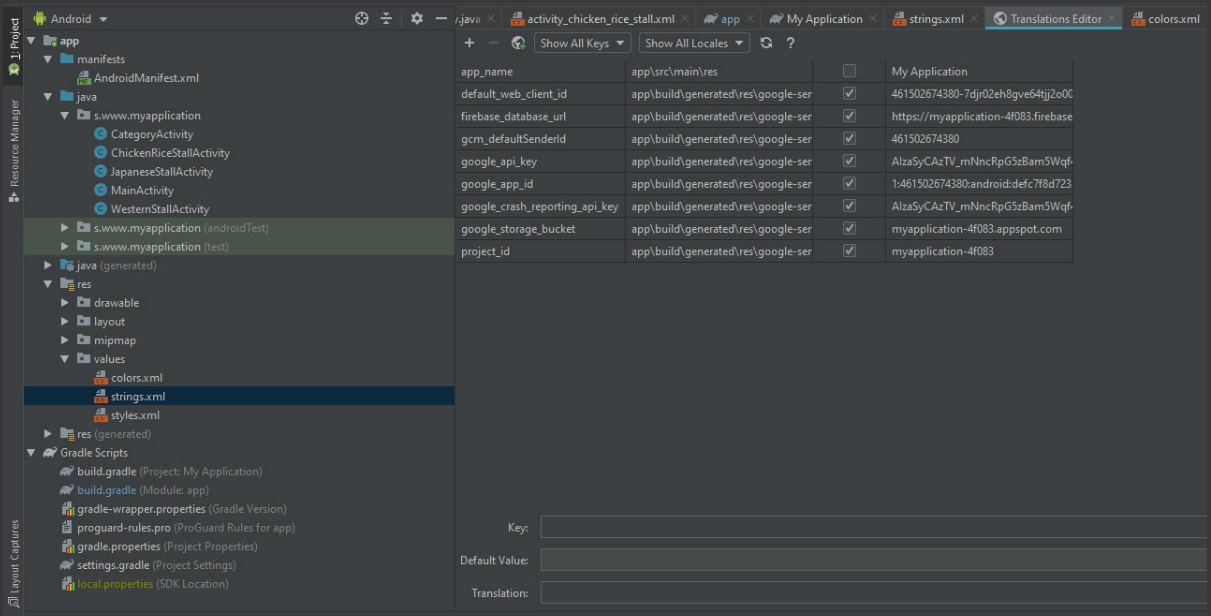
You will realise that building the APK to distribute every time you add a new feature may be troublesome. There are build automation tools out there such as **Visual Studio App Center** that can help you automate this process (though this requires knowledge of Git, but hey, so do many things out there).

5. Best Practices

Using the values Folder

You might have realised that the Android app contains a **values** folder that we have not really touched. In practice, whenever we use any values (e.g. Text, dimensions, colours) in our application page, we should not be writing the values directly into our activity XML file, but take these values from a common XML file shared across multiple activities. This helps to ensure uniformity, and if we need to make a change, the change will be reflected everywhere.

They values are split into different XML files, such as strings.xml for text and colors.xml for colours. Here is an example of how the values are stored, and can be edited.



```
▼ app
  ▼ manifests
      AndroidManifest.xml
  ▼ java
    ▼ s.www.myapplication
        CategoryActivity
        ChickenRiceStallActivity
        JapaneseStallActivity
        MainActivity
        WesternStallActivity
    ▶ s.www.myapplication (androidTest)
    ▶ s.www.myapplication (test)
  ▶ java (generated)
  ▼ res
    ▶ drawable
    ▶ layout
    ▶ mipmap
    ▼ values
        colors.xml
        strings.xml
        styles.xml
  ▶ res (generated)
▼ Gradle Scripts
    build.gradle (Project: My Application)
    build.gradle (Module: app)
    gradle-wrapper.properties (Gradle Version)
    proguard-rules.pro (ProGuard Rules for app)
    gradle.properties (Project Properties)
    settings.gradle (Project Settings)
    local.properties (SDK Location)
```

app_name	app\src\main\res	☐	My Application
default_web_client_id	app\build\generated\res\google-ser	✓	461502674380-7djr02eh8gve64tjj2o00
firebase_database_url	app\build\generated\res\google-ser	✓	https://myapplication-4f083.firebase
gcm_defaultSenderId	app\build\generated\res\google-ser	✓	461502674380
google_api_key	app\build\generated\res\google-ser	✓	AIzaSyCAzTV_mNncRpG5zBam5Wqf
google_app_id	app\build\generated\res\google-ser	✓	1:461502674380:android:defc7f8d723
google_crash_reporting_api_key	app\build\generated\res\google-ser	✓	AIzaSyCAzTV_mNncRpG5zBam5Wqf
google_storage_bucket	app\build\generated\res\google-ser	✓	myapplication-4f083.appspot.com
project_id	app\build\generated\res\google-ser	✓	myapplication-4f083

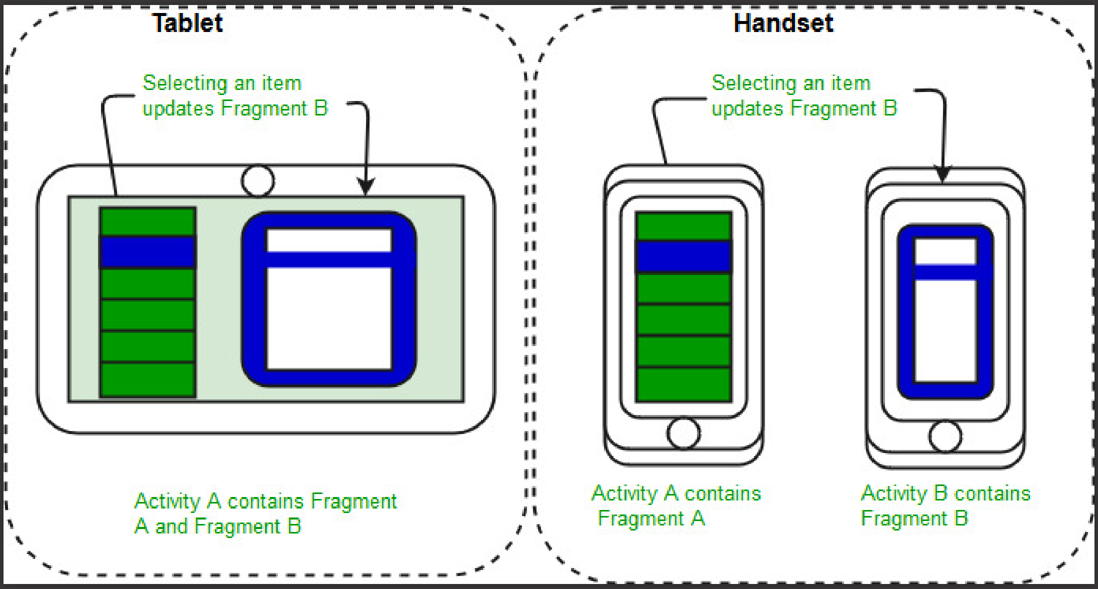
Bonus: Change your implementation of your UI texts and colours by using values from the values folder

Using RecyclerViews/Adapters

You might have found it quite tedious when you had to copy and paste code and elements for each and every Category and item in our sample app. What if these categories and elements were dynamic? You would need to change your code every single time! Instead, what could be done is you could use RecyclerViews/Adapters that simply take in some parameters and automatically churn out the desired list of items for you. Each item is represented by a standard **ViewHolder** format. These parameters could be easily retrieved through something like a database. RecyclerViews and Adapters require a good understanding of Java to implement however, and may be too complicated for beginners. But nevertheless you should keep the existence of these tools in mind, should you decide to go deeper.

Using Fragments

While we have been using Activities to create our new pages, the ideal way is actually to use **Fragments** instead for our different pages. **Activities** are more for splitting up our app into its various logical components, such as Logging in and the main browsing activity.


Tablet
Selecting an item updates Fragment B
Activity A contains Fragment A and Fragment B
Handset
Selecting an item updates Fragment B
Activity A contains Fragment A
Activity B contains Fragment B

Fragments are similar to Activities, in that they can have their own Java and XML file. However, fragment can support dynamic displays and transitions much better. It is usually common to see a single Activity be made up of multiple fragments.

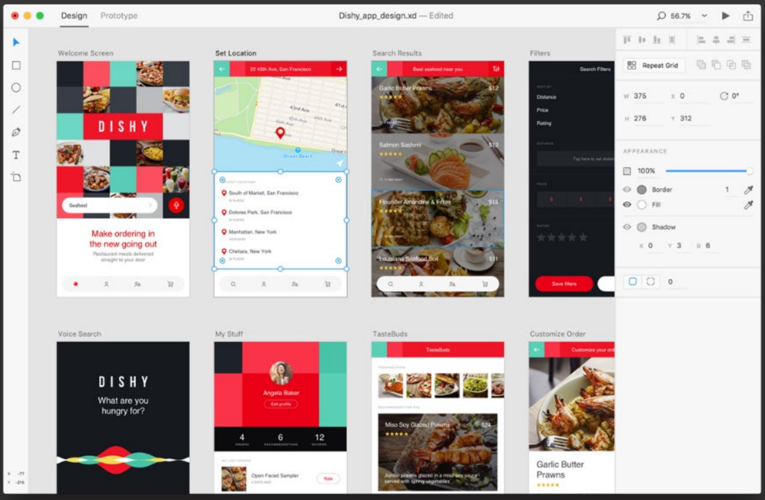
While we are talking about transitions, do note that it is also possible to enable smooth left-drag/right-drag etc transitions using fragments, you just need to specify or create the animation that you want to use, and let something called the **FragmentManager** handle the transitions, going back etc for you.

6. Extra: Do I really need an Android App?

(Note: All the notes past this point are for your own reading)

Depending on your situation, you may find that creating an Android App actually takes up a lot of time, especially if you are trying to make it look nice. In times like this, perhaps making an actual Android App is not the way to go, but a mock-up would be good enough.

In times like this, you can instead use Adobe XD. Adobe XD helps you create an interactive mock-up of your App, while making it very easy for you to build beautiful designs, and there are ready templates that you can download and work from.



When should you use Adobe XD:

1. Short hackathons where time is of the essence, and you just need to present the flow of your app
2. When the Android App is not the main focus of your project
3. You can use XD to create a wireframe or mockup first, then when your project gets approved, you can continue with Android Studio, while referring to your Adobe XD files to build your design

When you should not use Adobe XD:

1. As a standalone submission for a major project/challenge
2. When the user experience is extremely important and needs to be showcased with a proper phone and app
3. When you REALLY need a functioning Android App for people to use

6. Extra: Other forms of Android Development, Cross-Platform Mobile Development

As it turns out, there are many different ways to do Android development. Using Android Studio is considered the “Native” way of doing Android development, the way Google designed it to be, but there are other methods.

For starters, you may choose to use **Kotlin** instead of Java as a programming language. Kotlin is still a native method to do android development, and is in fact recommended by Google over Java. This is due to features like type-safety that come with the Kotlin programming language.

You also do not have to use Android Studio to develop your application, but Android Studio’s IDE really helps you a lot with Android development. There is IntelliJ IDEA, which is practically the same as Android Studio. You can use your favourite text editor of your choice (e.g. Visual Studio Code, Sublime Text, Vim, Emacs, Atom) and manually use Gradle yourself to build your app. But you will start to miss the features that Android Studio provides you with.

For freshmores, as you will soon learn in Digital World, it is possible to use Python, together with a tool called Kivy to build and load applications in Android. However, do note that Kivy is rather limited in its capabilities especially for developing Android applications, and is often more tedious.

Cross-Platform Development

What if your project manager or boss decided that he wanted to develop the app for iOS as well? After all, iOS has a huge market as well, especially in Singapore.

While iOS apps are generally natively developed in XCode and **Swift** programming language, there are a few ways to develop Android and iOS apps simultaneously.

React Native

React Native is a **Javascript** framework that allows you to just that. React Native is developed and maintained by Facebook, and derives some similar styles from **React**, which is another framework used for Web Frontend programming. People who are more used to Javascript and React may find React Native a much more comfortable experience. However, do note that some slight customizations still need to be made to make the app cross-compatible with both iOS and Android. Also, you will still need a Mac to build the iOS app.

Xamarin

Xamarin is yet another framework that accomplishes what **React Native** does, but using **.NET** and **C#**. it is developed by Microsoft. You will have to use Visual Studio to develop your Xamarin applications. Again, if you are familiar with **.NET** and **C#,** Xamarin may be the way to go for you.

Flutter

Flutter by Google is another cross-platform app-development technology. Flutter is more focused towards building a great UI and has good DevOps documentation as well. Flutter uses Dart, which is a programming language similar to Java and Javscript.

Flutter is relatively new and has less community support, but does offer better relative native performance and has great development toolkits and frameworks.

Take note that while React Native, Xamarin and Flutter all claim near-native performance, native Android/iOS development in Java, Kotlin, Objective-C or Swift will always still be better for their respective platforms. If a company has the resources and time to invest in more engineers and resources, developing the Android and iOS apps natively should be the choice. If the company does not want to spend that much resources however, cross-platform development tools are an acceptable alternative.

7. Extra: Backend

While some simple applications don’t require any server support apart from maybe a database, often times an application that is slightly more complex will require some sort of integration with a backend server. A backend server can streamline your operations and ensure no conflicts occur between devices. Couple that with a backend server usually being much more powerful than your mobile device, so it is much more reliable at performing CPU intensive operations, and a backend server also provides a much needed layer of security. For example, you could let the backend server interface with your database instead of directly through the mobile app, which could otherwise cause synchronisation issues.

I am not going to be too specific here since backend is an entirely different topic, but you could start looking at **NodeJs/Express** if you are interested in creating a backend server. **Heroku** is a relatively beginner-friendly platform that also handles your hosting needs for you. If you are a more advanced user, cloud providers like **Amazon Web Services (AWS)**, **Microsoft Azure** or **Google Cloud Platform (GCP)** can provide many advanced features and options for you.

8. Extra: Working in a Team, Development Process

You may realise that developing an Android application solo may be an unsustainable task. This is more so especially when you are developing an app for a company, where expected standards will be higher.

When working in a team, it is almost a given to use Git to share your code with your team members, and to version your application.

Often in companies, there will be DevOps engineers who outline how the development process should be like, as well as monitor the CI/CD (Continuous Integration, Continuous Development) cycle of the application. This is to facilitate and ensure that new updates for the application are rolled out smoothly, with minimal disruptions and bugs.

Usually, before a new feature gets added to the application, something like the following process happens

1. Developer develops new feature and tests on his **local workstation**
2. Developer tries out the new feature on the **development environment** (sandbox). Unit testing happens at this stage
3. **Integration** testing happens with other services, check for side effects
4. Feature is tested rigorously on **QC environment**, QC team will test out new feature and ensure no bugs are introduced
5. Feature is deployed on the **staging environment**, which is a mirror of the actual production environment. This is the final stage the ensure the feature will also work on the production environment.
6. Feature is pushed to the **production** environment, which is the actual environment consumers use.

The above process helps to minimise any risk of downtime or errors that may occur between feature updates, and ensures features are well-tested.