How to Upload Images to Firebase from an Android App

by Chinedu Izuchukwu 15 Nov 2017

Difficulty: Beginner Length: Medium Languages: English

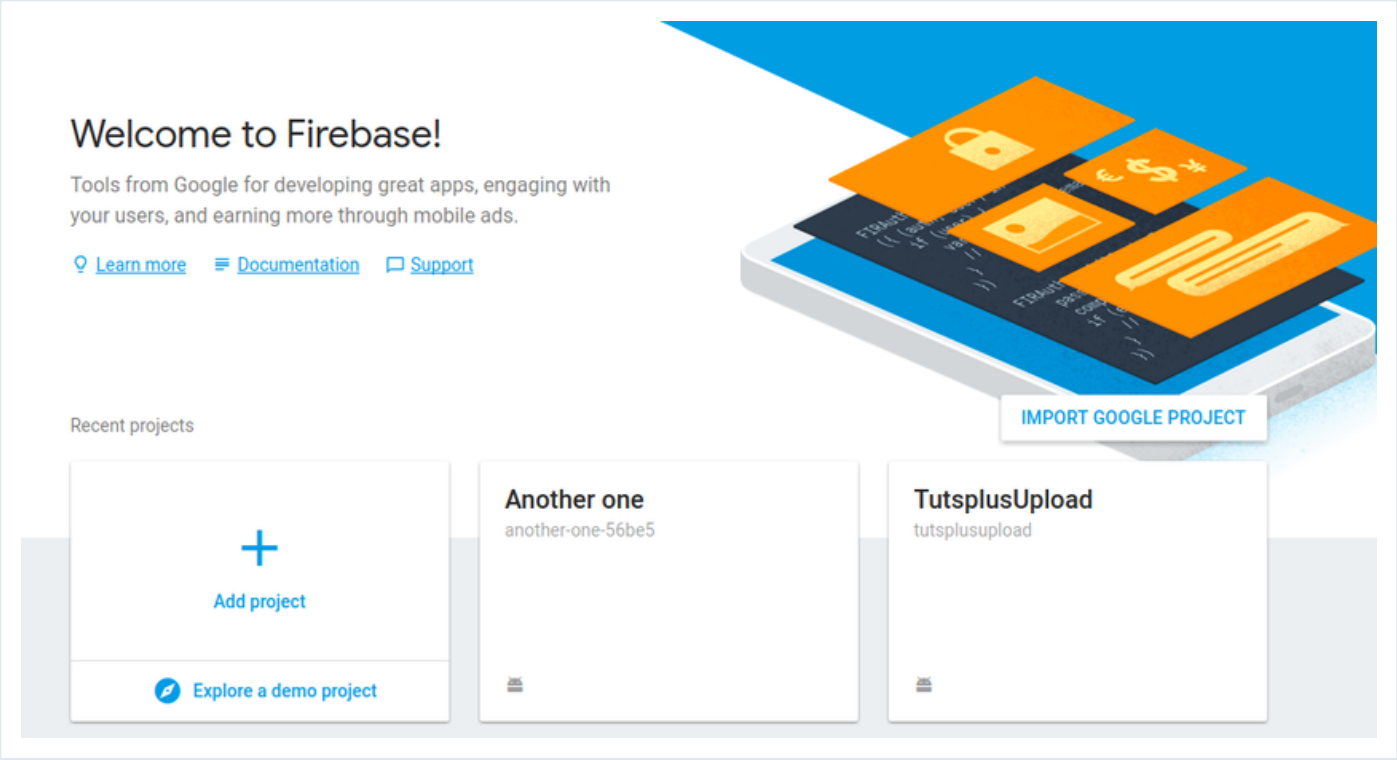
Android SDK

Firebase is a mobile and web application development platform, and Firebase Storage provides secure file uploads and downloads for Firebase apps. In this post, you'll build an Android application with the ability to upload images to Firebase Storage.

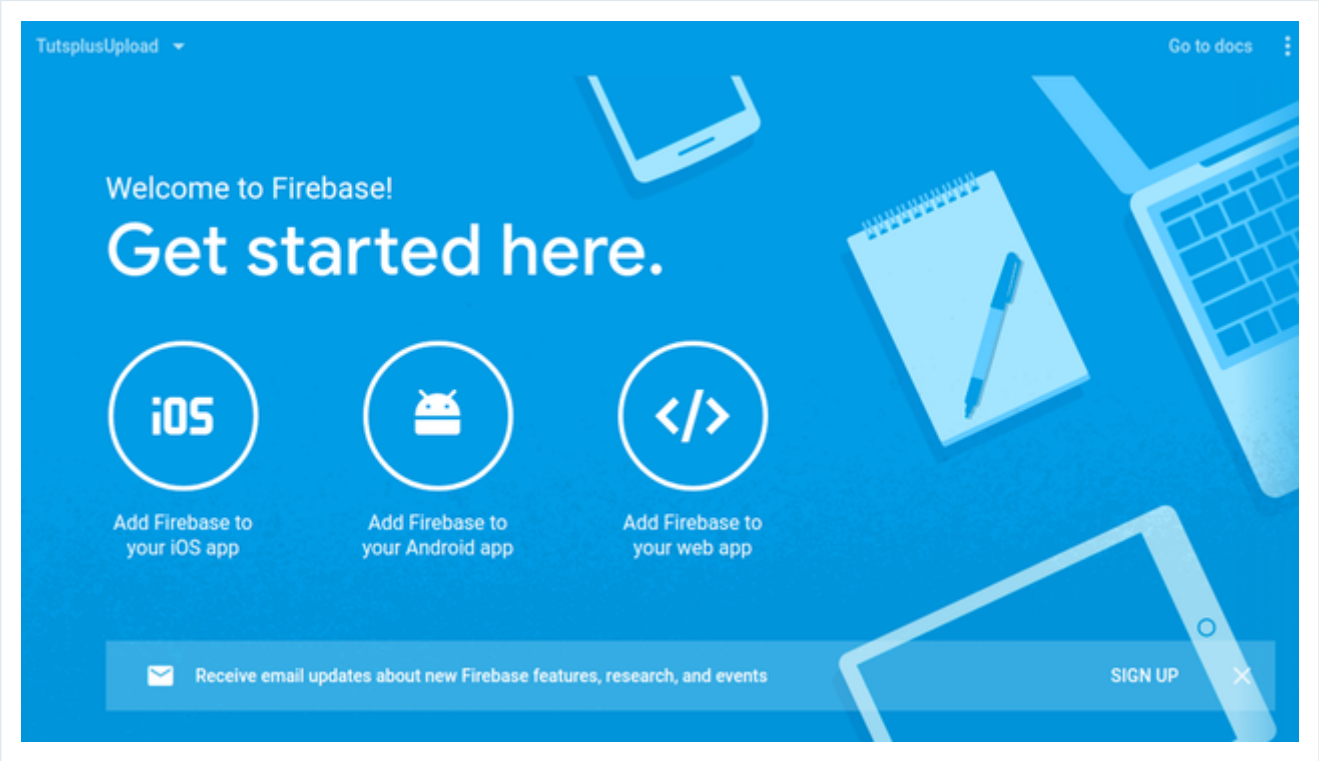
Firebase Setup

If you don't have a Firebase account yet, you can create one at the Firebase home page.

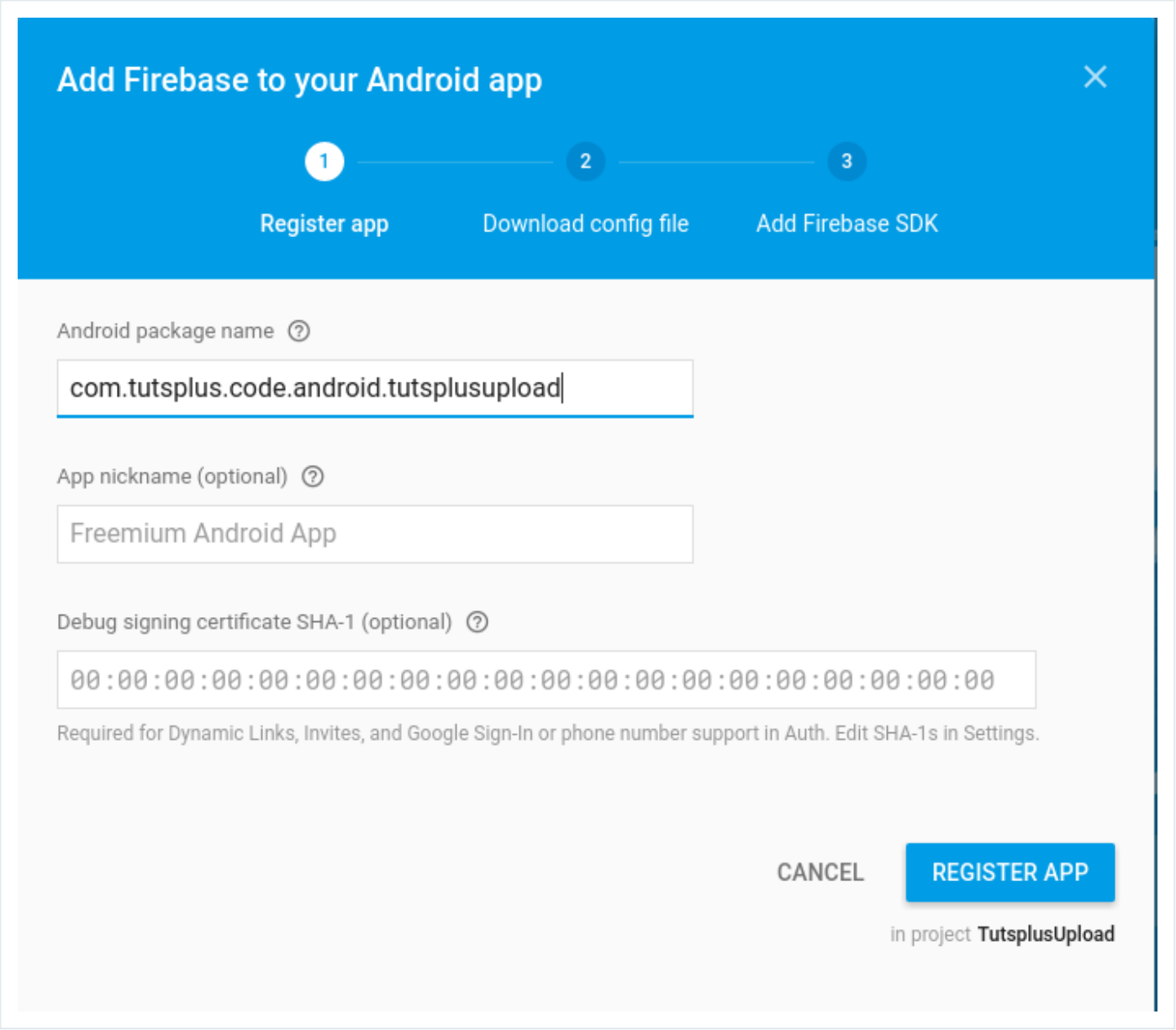
Once your account is set up, go to your Firebase console, and click the **Add Project** button to add a new project.



Enter your project details and click the **Create Project** button when done. On the next page, click on the link to **Add Firebase to your Android app**.



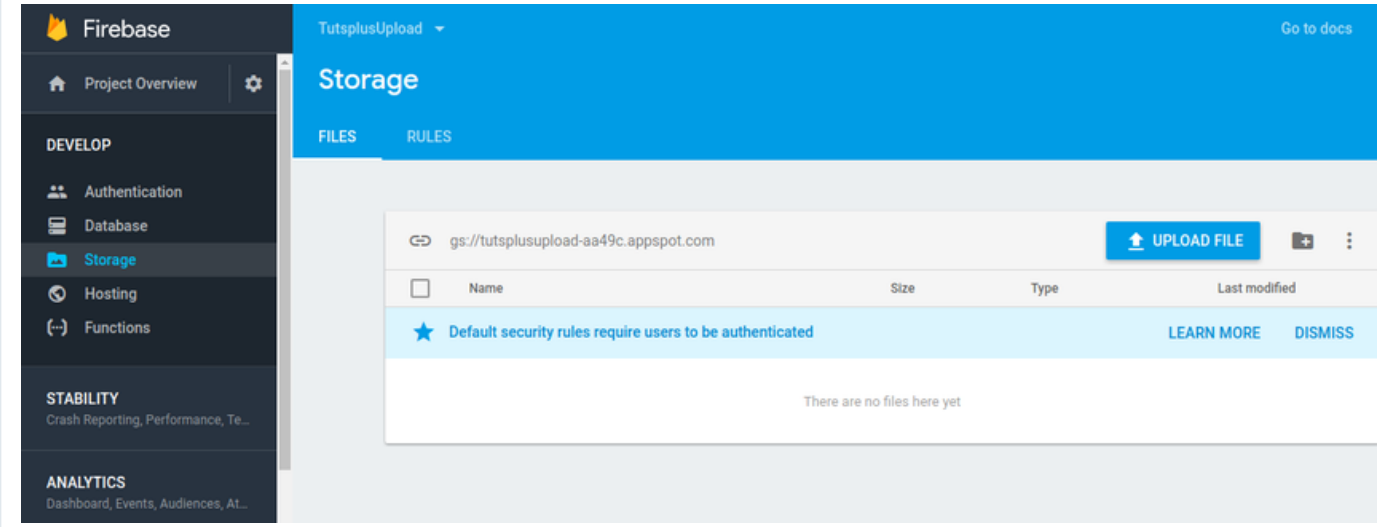
Enter your application package name. My application package is **com.tutsplus.code.android.tutsplusupload**. Note that the package name is namespaced with a unique string that identifies you or your company. An easy way to find this is to open your `MainActivity` file and copy the package name from the top.



When done, click on **Register App**. On the next page, you will be given a **google-services.json** to download to your computer. Copy and paste that file into the app folder of your application. (The path should be something like **TutsplusUpload/app**.)

Set Firebase Permissions

To allow your app access to Firebase Storage, you need to set up permissions in the Firebase console. From your console, click on **Storage**, and then click on **Rules**.



Paste the rule below and publish.

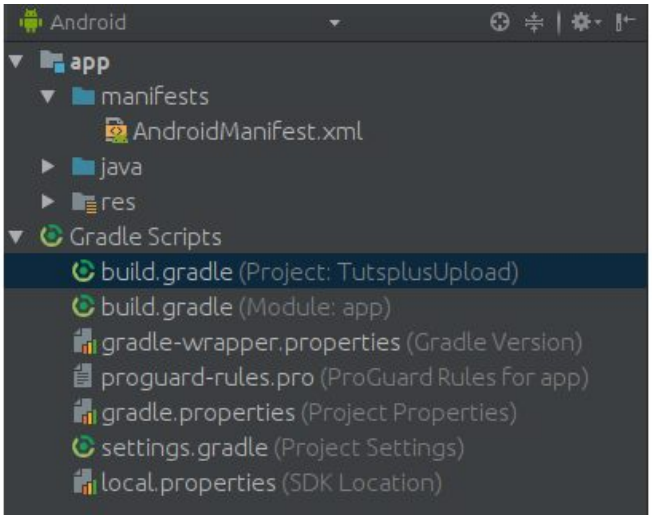
```
service firebase.storage {
  match /b/{bucket}/o {
    match /{allPaths=**} {
      allow read, write: if true;
    }
  }
}
```

This will allow read and write access to your Firebase storage.

Create the Application

Open up Android Studio, and create a new project. You can call your project anything you want. I called mine **TutsplusUpload**.

Before you proceed, you'll need to add a couple of dependencies. On the left panel of your Android Studio, click on **Gradle Scripts**.



Open **build.gradle (Project: TutsplusUpload)**, and add this line of code in the dependencies block.

```
classpath 'com.google.gms:google-services:3.0.0'
```

Next, open **build.gradle (Module: app)** to add the dependencies for Firebase. These go in the dependencies block also.

```
compile 'com.google.firebase:firebase-storage:9.2.1'
compile 'com.google.firebase:firebase-auth:9.2.1'
```

Finally, outside the dependencies block, add the plugin for **Google Services**.

```
apply plugin: 'com.google.gms.google-services'
```

Save when done, and it should sync.

Set Up the `MainActivity` Layout

The application will need one activity layout. Two buttons will be needed—one to select an image from your device, and the other to upload the selected image. After selecting the image you want to upload, the image will be displayed in the layout. In other words, the image will not be set from the layout but from the activity.

In your `MainActivity` layout, you will use two layouts—nesting the linear layout inside the relative layout. Start by adding the code for your relative layout.

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:padding="16dp"
    tools:context="com.tutsplus.code.android.tutsplusupload.MainActivity">

</RelativeLayout>
```

The `RelativeLayout` takes up the whole space provided by the device. The `LinearLayout` will live inside the `RelativeLayout`, and will have the two buttons. The buttons should be placed side by side, thus the orientation to be used for the `LinearLayout` will be horizontal.

Here is the code for the linear layout.

```
<LinearLayout
    android:id="@+id/layout_button"
    android:orientation="horizontal"
    android:layout_alignParentTop="true"
    android:weightSum="2"
    android:layout_width="match_parent"
    android:layout_height="wrap_content">

    <Button
        android:id="@+id/btnChoose"
        android:text="Choose"
        android:layout_weight="1"
        android:layout_width="0dp"
        android:layout_height="wrap_content" />

    <Button
        android:id="@+id/btnUpload"
        android:text="Upload"
        android:layout_weight="1"
        android:layout_width="0dp"
        android:layout_height="wrap_content" />
</LinearLayout>
```

From the above code, you can see that both buttons have ids assigned. The ids will be used to target the button from the main activity such that when the button gets clicked, an interaction is initiated. You will see that soon.

Below the `LinearLayout`, add the code for the `ImageView`.

```
<ImageView
```

```
        android:id="@+id/imgView"
        android:layout_width="match_parent"
        android:layout_height="match_parent" />
```

You can also see that the `ImageView` has an `id`; you will use this to populate the layout of the selected image. This will be done in the main activity.

Get `MainActivity` Up

Navigate to your `MainActivity`, and start by declaring fields. These fields will be used to initialize your views (the buttons and `ImageView`), as well as the URI indicating where the image will be picked from. Add this to your main activity, above the `onCreate` method.

```
private Button btnChoose, btnUpload;
private ImageView imageView;

private Uri filePath;

private final int PICK_IMAGE_REQUEST = 71;
```

`PICK_IMAGE_REQUEST` is the request code defined as an instance variable.

Now you can initialize your views like so:

```
//Initialize Views
btnChoose = (Button) findViewById(R.id.btnChoose);
btnUpload = (Button) findViewById(R.id.btnUpload);
imageView = (ImageView) findViewById(R.id.imgView);
```

In the above code, you are creating new instances of `Button` and `ImageView`. The instances point to the buttons you created in your layout.

You have to set a listener that listens for interactions on the buttons. When an interaction happens, you want to call a method that triggers either the selection of an image from the gallery or the uploading of the selected image to Firebase.

Underneath the initialized views, set the listener for both buttons. The listener looks like this.

```
btnChoose.setOnClickListener(new View.OnClickListener() {
```

```
    @Override
    public void onClick(View v) {
      chooseImage();
    }
  });

  btnUpload.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
      uploadImage();
    }
  });
```

This should be in the `onCreate()` method. As I mentioned above, both buttons call a different method. The **Choose** button calls the `chooseImage()` method, while the **Upload** button calls the `uploadImage()` method. Let's add those methods. Both methods should be implemented outside the `onCreate()` method.

Let's start with the method to choose an image. Here is how it should look:

```
private void chooseImage() {
  Intent intent = new Intent();
  intent.setType("image/*");
  intent.setAction(Intent.ACTION_GET_CONTENT);
  startActivityForResult(Intent.createChooser(intent, "Select Picture"), PICK_
}
```

When this method is called, a new `Intent` instance is created. The intent type is set to image, and its action is set to get some content. The intent creates an image chooser dialog that allows the user to browse through the device gallery to select the image. `startActivityForResult` is used to receive the result, which is the selected image. To display this image, you'll make use of a method called `onActivityResult`.

`onActivityResult` receives a request code, result code, and the data. In this method, you will check to see if the request code equals `PICK_IMAGE_REQUEST`, with the result code equal to `RESULT_OK` and the data available. If all this is true, you want to display the selected image in the `ImageView`.

Below the `chooseImage()` method, add the following code.

```
@Override
protected void onActivityResult(int requestCode, int resultCode, Intent data)
  super.onActivityResult(requestCode, resultCode, data);
  if(requestCode == PICK_IMAGE_REQUEST && resultCode == RESULT_OK
          && data != null && data.getData() != null )
  {
```

```
    filePath = data.getData();
    try {
      Bitmap bitmap = MediaStore.Images.Media.getBitmap(getContentResolver(),
      imageView.setImageBitmap(bitmap);
    }
    catch (IOException e)
    {
      e.printStackTrace();
    }
  }
}
```

Uploading the File to Firebase

Now we can implement the method for uploading the image to Firebase. First, declare the fields needed for Firebase. Do this below the other fields you declared for your class.

```
//Firebase
FirebaseStorage storage;
StorageReference storageReference;
```

`storage` will be used to create a `FirebaseStorage` instance, while `storageReference` will point to the uploaded file. Inside your `onCreate()` method, add the code to do that—create a `FirebaseStorage` instance and get the storage reference. References can be seen as pointers to a file in the cloud.

```
storage = FirebaseStorage.getInstance();
storageReference = storage.getReference();
```

Here is what the `uploadImage()` method should look like.

```
private void uploadImage() {

  if(filePath != null)
  {
    final ProgressDialog progressDialog = new ProgressDialog(this);
    progressDialog.setTitle("Uploading...");
    progressDialog.show();

    StorageReference ref = storageReference.child("images/"+ UUID.randomUUID(
    ref.putFile(filePath)
            .addOnSuccessListener(new OnSuccessListener<UploadTask.TaskSnapsh
              @Override
              public void onSuccess(UploadTask.TaskSnapshot taskSnapshot) {
                progressDialog.dismiss();
                Toast.makeText(MainActivity.this, "Uploaded", Toast.LENGTH_SH
              }
```

```
            })
            .addOnFailureListener(new OnFailureListener() {
              @Override
              public void onFailure(@NonNull Exception e) {
                progressDialog.dismiss();
                Toast.makeText(MainActivity.this, "Failed "+e.getMessage(), T
              }
            })
            .addOnProgressListener(new OnProgressListener<UploadTask.TaskSnap
              @Override
              public void onProgress(UploadTask.TaskSnapshot taskSnapshot) {
                double progress = (100.0*taskSnapshot.getBytesTransferred()/t
                        .getTotalByteCount());
                progressDialog.setMessage("Uploaded "+(int)progress+"%");
              }
            });
    }
}
```

When the `uploadImage()` method is called, a new instance of `ProgressDialog` is initialized. A text notice showing the user that the image is being uploaded gets displayed. Then a reference to the uploaded image, `storageReference.child()`, is used to access the uploaded file in the **images** folder. This folder gets created automatically when the image is uploaded. Listeners are also added, with toast messages. These messages get displayed depending on the state of the upload.

Set Permission in the App

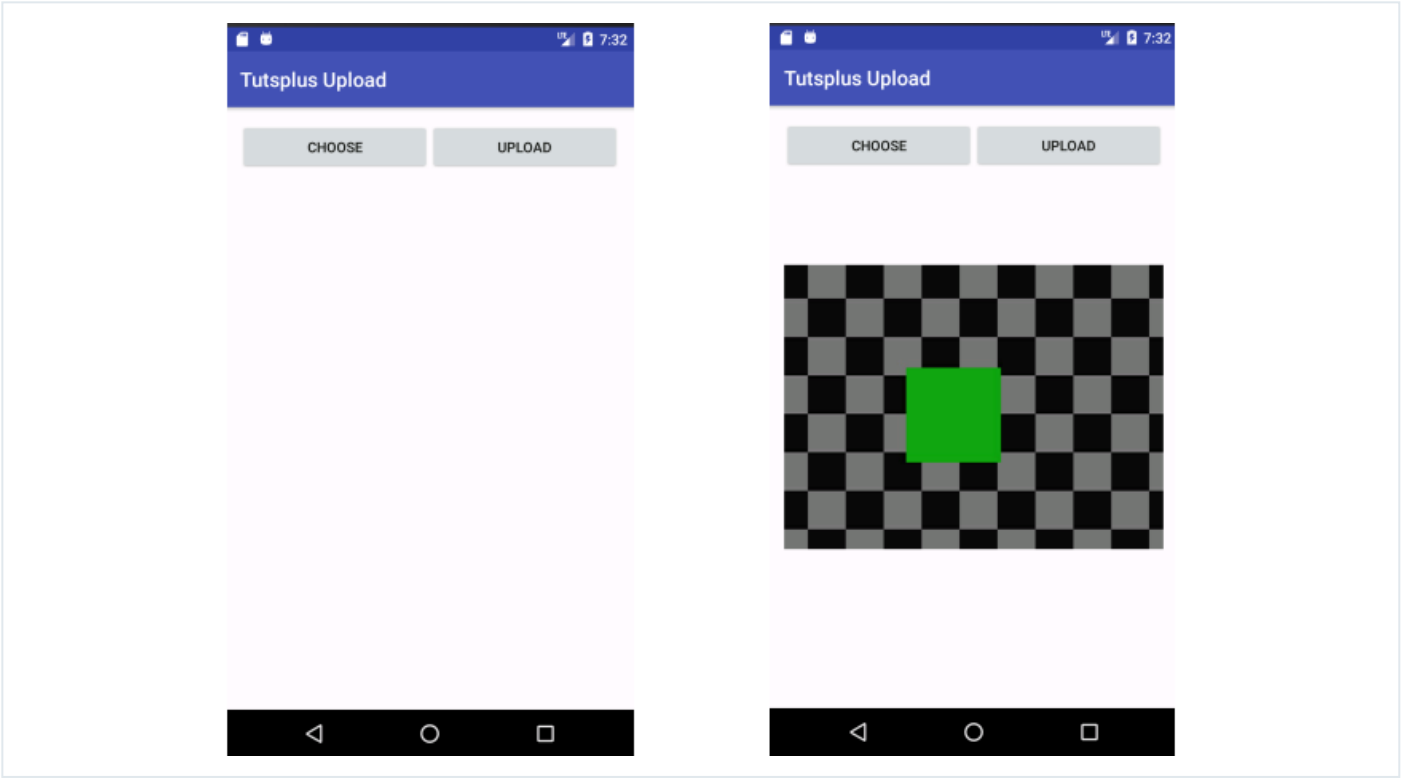
Finally, you need to request permission that your application will make use of. Without this, users of your application will not be able to browse their device gallery and connect to the internet with your application. Doing this is easy—simply paste the following in your **AndroidManifest** file. Paste it just above the `application` element tag.

```
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE" />
```

This requests for permission to use the internet and read external storage.

Testing the App

Now go ahead and run your application! You should be able to select an image and successfully upload it to Firebase. To confirm the image uploaded, go back to your console and check in the **Files** part of your storage.



Conclusion

Firebase provides developers with lots of benefits, and file upload with storage is one of them. Uploading images from your Android application requires you to work with Activities and Intents. By following along with this tutorial, your understanding of Activities and Intents has deepened. I hope you enjoyed it!

Check out some of our other posts for more background about Activities and Intents, or take a look at some of our other tutorials on using Firebase with Android!

ANDROID SDK

What Are Android Intents?

Chinedu Izuchukwu

ANDROID SDK

What Is the Android Activity Lifecycle?

Chinedu Izuchukwu

ANDROID SDK

Firebase for Android: File Storage

Paul Trebilcox-Ruiz

ANDROID SDK

How to Create an Android Chat App Using Firebase

Ashraff Hathibelagal

Chinedu Izuchukwu

Lagos, Nigeria

I enjoy writing code and passing knowledge. When not doing either of them, I watch movies.