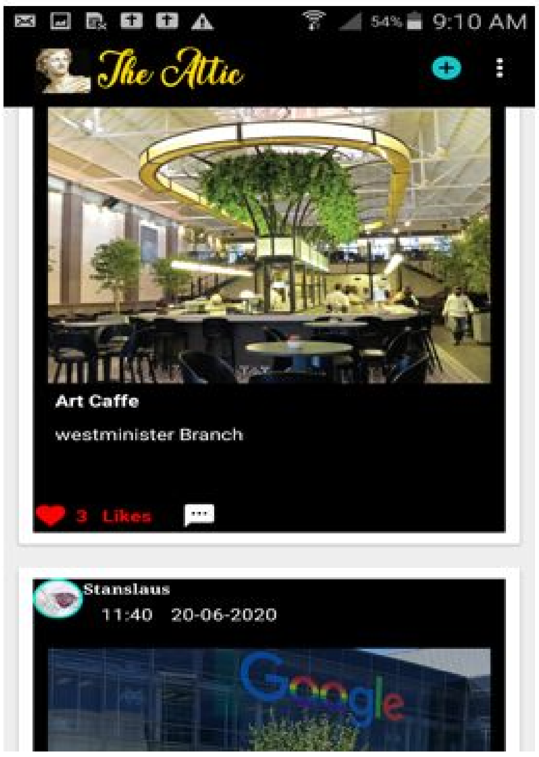
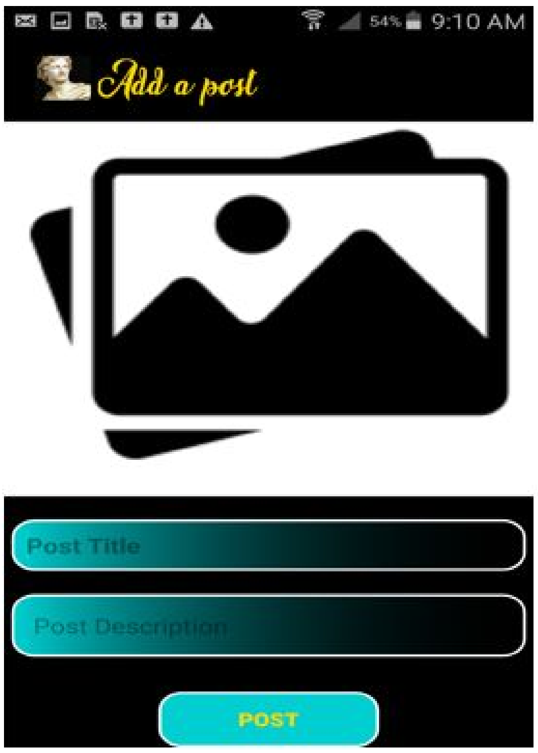
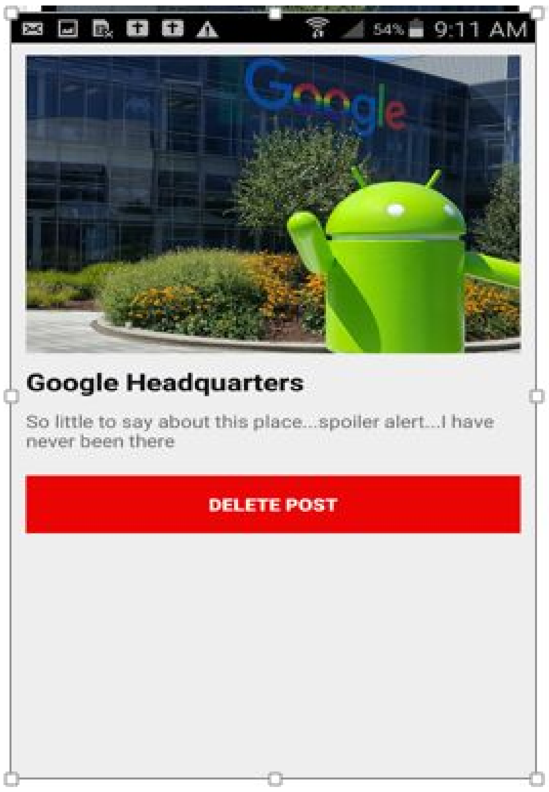
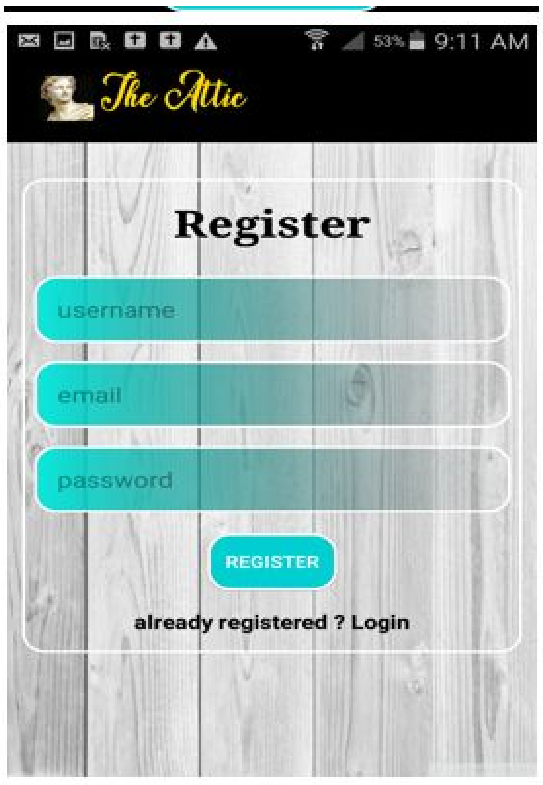
Instructor: Stanslaus Mwongela

Task: Creating a live streaming Blog using Firebase






54% 9:11 AM
Google Headquarters
So little to say about this place...spoiler alert...I have never been there
DELETE POST



Getting Started:

What we will learn:

By the virtue of this Lab, you'll learn a few other things like

1. Creating users with email and password
2. Registering and Signing in Users
3. Logging out signed-in Users
4. Writing to and Reading from database
5. Deleting data from Firebase within the app
6. Creating User Profiles and uploading images
7. Associating Posts with respective owners.
8. Implementing the like functionality

Create a **new project** using the **basic activity template** and give it your preferred name, package name and a minimum SDK of 18 so that it will run in most devices.

In my case I'm assigning the project name as "**The Attic**"….I thought this will make a good name for my blog…why…really?

I adapted this from Sherlock Holmes, a famous saying by Arthur **Conan Doyle**

"I consider that a man's brain originally is like a little empty attic, and you have to stock it with such furniture as you choose. A fool takes in all the lumber of every sort that he comes across, so that the knowledge which might be useful to him gets crowded out, or at best is jumbled up with a lot of other things, so that he has a difficulty in laying his hands upon it. Now the skillful workman is very careful indeed as to what he takes into his brain-attic. He will have nothing but the tools which may help him in doing his work, but of these he has a large assortment, and all in the most perfect order. It is a mistake to think that that little room has elastic walls and can distend to any extent. Depend upon it there comes a time when for every addition of knowledge you forget something that you knew before. It is of the highest importance, therefore, not to have useless facts elbowing out the useful ones."

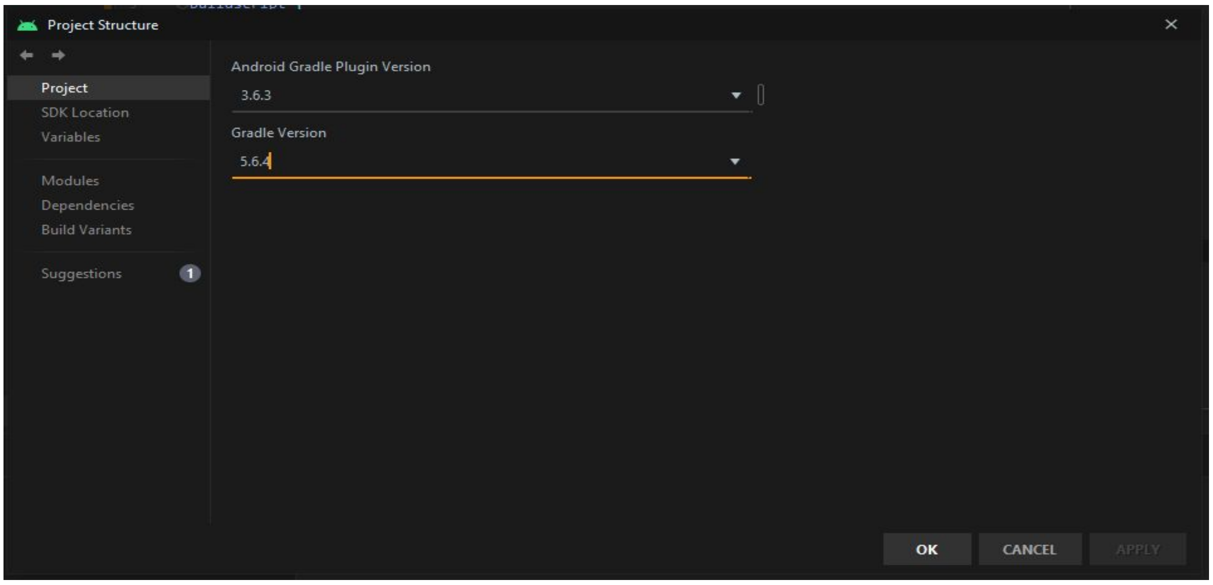
Setting up and adding Firebase to your project:

Add firebase to your project as described in the below link.

Reference: https://firebase.google.com/docs/android/setup

P.s: Ensure your **gradle version is 4.0 or later** and **Android gradle plugin version is 3.2.1 or later**

Please note there is a difference between the gradle version and Android gradle plugin version, e.g. my gradle version is 5.6.4 and my Android gradle plugin is 3.6.3. To check your gradle version click on File- Project Structure and you're a pop-up will appear with your gradle version



Android Gradle Plugin Version
3.6.3
Gradle Version
5.6.4

Initial Set-Up

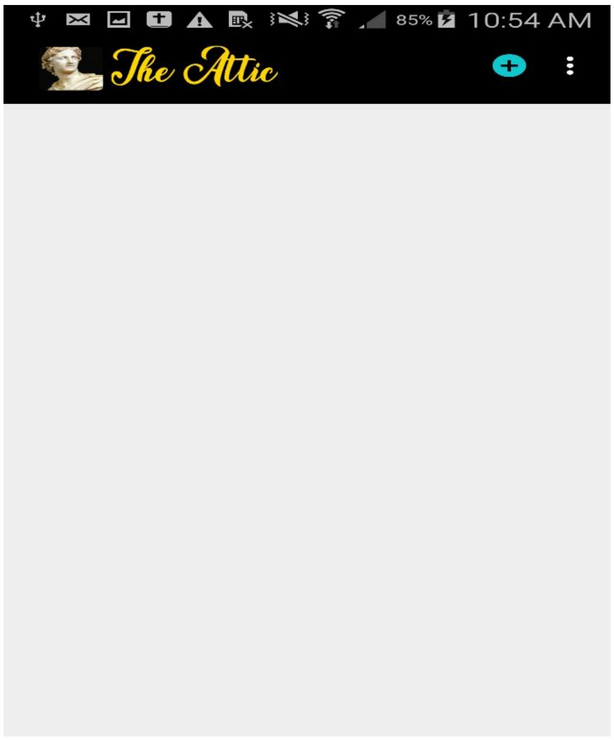
I have uploaded all the resources I used in this project on E-learning. You can edit the resources file to suit your User interface needs

The Main Activity

We will use the main activity to display card views of fetched user posts, interaction with the card view items, add post intent and log out intent.

1.1 Let's set- up the UI of the Main Activity

Initial Set-Up



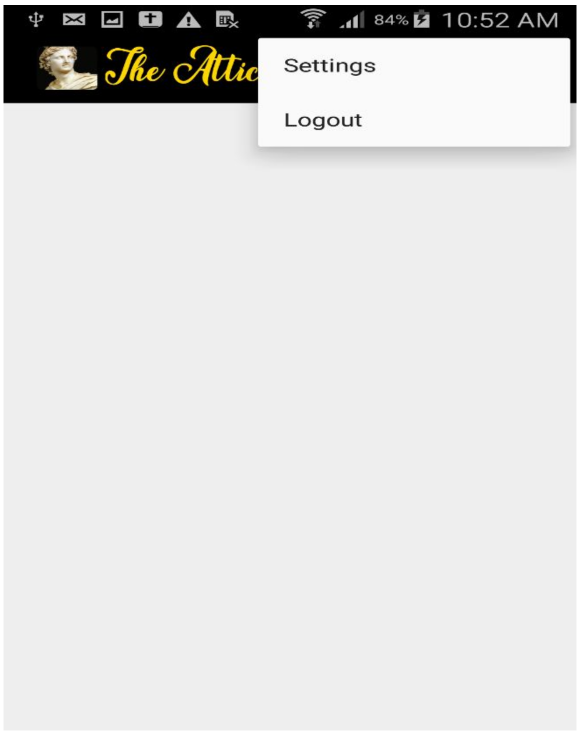


Figure 1.1: Main Activity Layout

Layouts to update to suit as in figure 1.1:

activity_main.xml , content_main.xml

activity_main .xml   uses resources from this files:

styles.xml

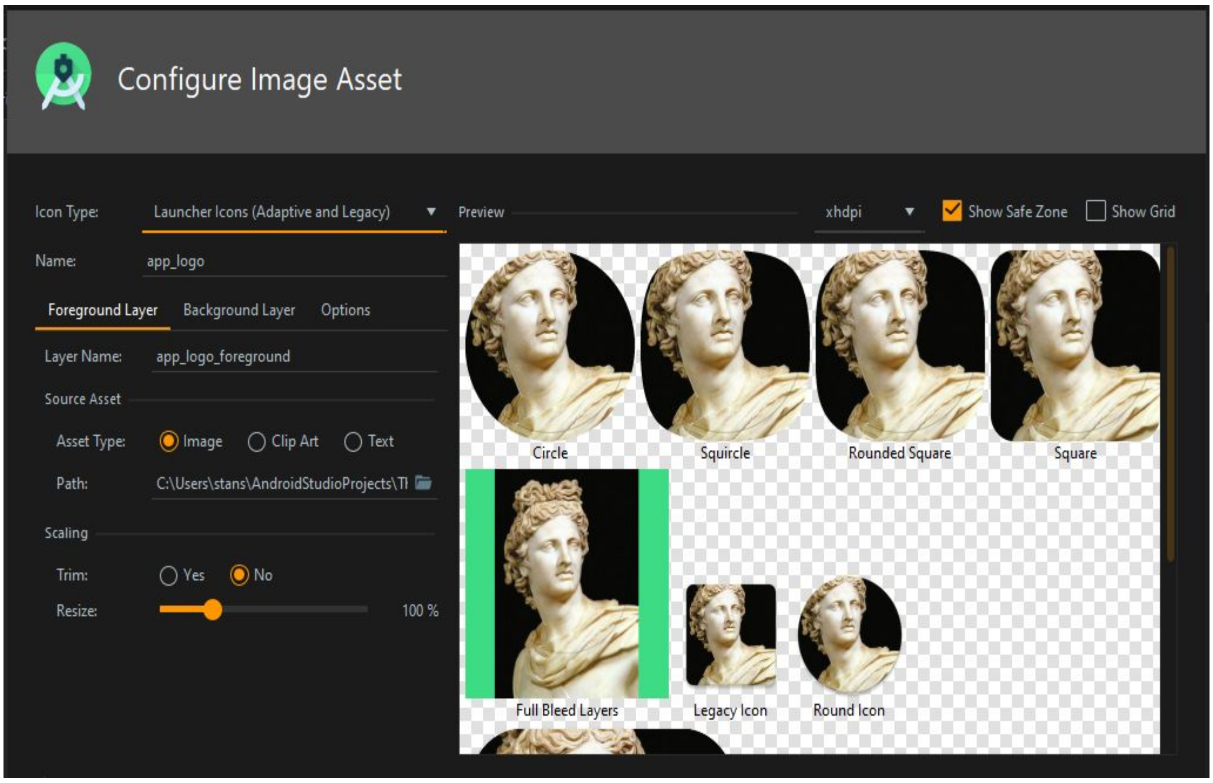
colors.xml

fonts- ensure to include this folder in your project in the res folder

mipmap-for the app logo attribute in activity_main.xml, add an image to your project by importing it as an image asset: Navigate to resource manage, click on the + sign, select image asset, set the icon type as Launcher Icons (Adaptive and legacy), set the name as app_logo, set the asset type as an image, select the path where you have stored the image you want to use as your app logo. Click the next button then finish. See figure 1.2.

Menu_main.xml-menu items

Drawables- "@drawable/add_icon”


Configure Image Asset

Icon Type: Launcher Icons (Adaptive and Legacy)

Name: app_logo

Foreground Layer | Background Layer | Options

Layer Name: app_logo_foreground

Source Asset

Asset Type: [✓ Image] Clip Art Text

Path: C:\Users\stans\AndroidStudioProjects\Tl

Scaling

Trim: Yes [✓ No]

Resize: 100 %

Preview: xhdpi, [✓ Show Safe Zone], Show Grid

Figure 1.2: Adding an image asset

1.2 In your manifest, under the Main activity tag, **delete** the attribute android theme since our styles.xml has changed

```
android:theme="@style/AppTheme.NoActionBar"
```

1.3 Add this permission since our app will use internet to connect to firebase

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

1.4 Delete First Fragment and Second Fragment java files and associated layouts since we will not be using fragments in our current project.

1.5 In your main activity code delete the boiler plate code for setting up the floating action button.

1.6 Instantiate your toolbar correctly by referencing to its ID as set in activity_main.xml.

1.7 Run your app and confirm it corresponds to figure 1

We will update our main activity later to populate blog posts in card views for authenticated users.

2. The Register Activity

Add a new activity using the empty activity template in your package and name it the Register Activity, we will use this activity to register users using their emails and password.

2.1 Define the layout for Register Activity

Having created this activity and named it RegisterActivity, you should now have a layout file (*activity_register.xml*) open this layout file and create the registration fields as per the layout I shared with you. activity_register.xml uses other resources from the drawable folder (@drawable/layoutsyle and @drawable/buttonstyle, @drawable/wood). I have used this to style a layout, edit texts and a button, feel free to edit this layout file and drawables to fit your liking.

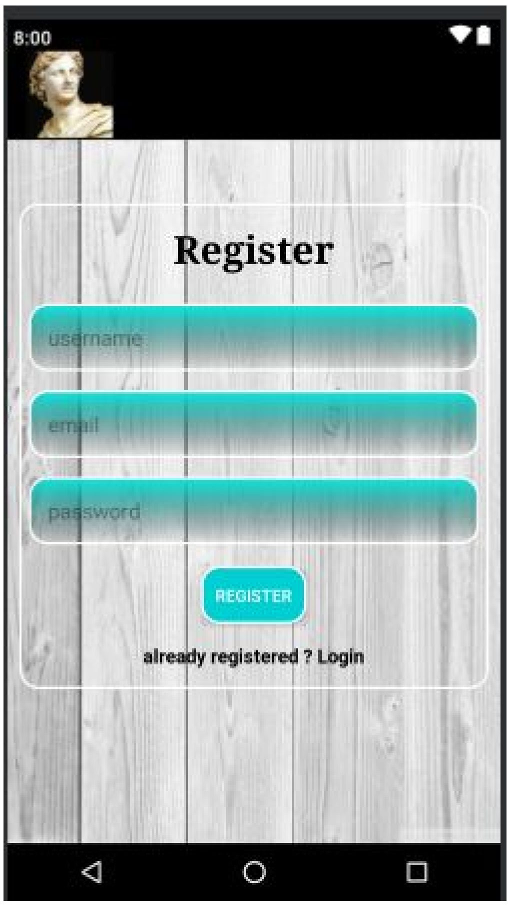


Fig 2.1 activity_register

2.2 Implement Register Functionality

We are going right ahead into the RegisterActivity java class and create all the functionalities we want

So I'm going to summarize what's going on in this class, I'll focus on what happens when the register button is clicked.

Create Users with email and Password

We are going to set an onClickListener on the registerBtn to monitor for click events which will then register a user on our database (when clicked) with the details provided in the EditText fields. First, we define String variables to store the values coming from the EditText fields and then do a check to make certain that none of the fields are empty. We then call the createUserWithEmailAndPassword() method on the FirebaseAuthentication instance.

This method takes in two arguments (*Email and Password*) which are primarily the String variables we used to store the users' email and password fields from the EditText objects.

Then we will attach an *onCompleteListener()* which will then implement the *onComplete()* method where we'll store this registered user on our database with respect to their unique id's. If this task is successful we go ahead and get the registered users id, attach the id to our database reference [userDetailsReference.*child(user_id)*] and then set the Username and Image on the users' unique path (*current_users_db*). Then we make a Toast to show the user that they've been successfully registered and then launch the Profile Activity for them to set their custom display name and profile photo.

Steps

2.2.1 Add the dependencies for the Authentication and RealTime Database Android library to your module (app-level) Gradle file (usually app/build.gradle (Module:App)) and sync the changes.

```
implementation 'com.google.firebase:firebase-auth:19.3.1'
implementation 'com.google.firebase:firebase-database:19.3.0'
```

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.

To use the Real time database (remember this is where we will be saving our users), navigate to the Database section of the Firebase console. You'll be prompted to select an existing Firebase project. Follow the database creation workflow.

Select a starting mode for your Firebase Security Rules:

Test mode: Good for getting started with the mobile and web client libraries, but allows anyone to read and overwrite your data. After testing, make sure to review the Understand Firebase Realtime Database Rules section. To get started for this lab, select test mode.

Locked mode: Denies all reads and writes from mobile and web clients. Your authenticated application servers can still access your database.

Click Done.

2.2.2 Declare and initialize instances of Firebase Authentication, Firebase Database, Firebase Database Reference and the views in our activity_register.xml. See full Register Activity below with comments to better understand.

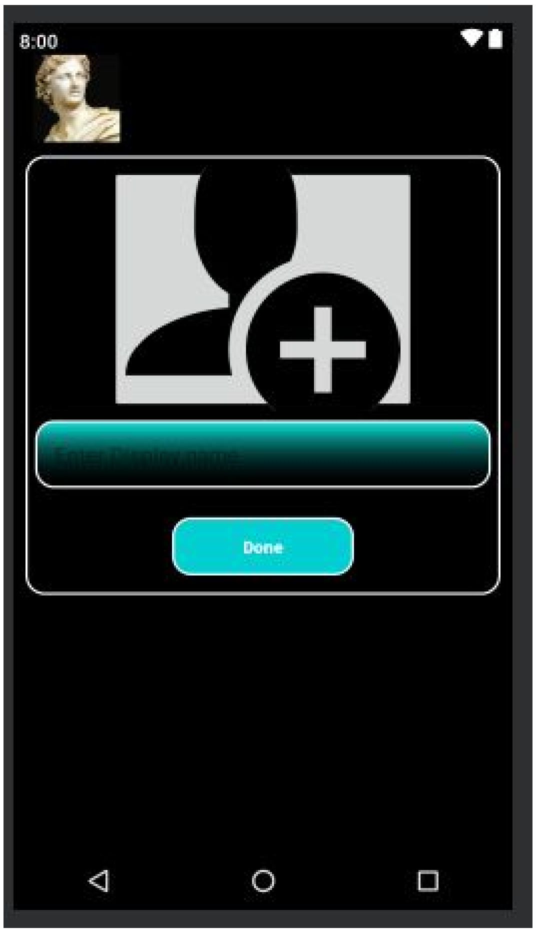
2.2.3 For already registered users we want to redirect them to the login page directly without registering them again, setOnClickListener on the textView object of redirecting users to login Activity. Before completing this step create a **Login Activity** first using the empty activity template and then implement an Intent to launch the login activity See full Register Activity below with comments to better understand.

2.2.4 Set on click listener to the register button so as to implement the method for creating users with email and password and also save user details in the real time database. Further this button will also prompt the opening of the **Profile Activity** where we will set users profile photo and custom display name. Hence, create a **Profile Activity** using the empty activity template. See full Register Activity below with comments to better understand.

```java
package com.mwongela.theattic;

import androidx.annotation.NonNull;
import androidx.appcompat.app.AppCompatActivity;
import androidx.appcompat.widget.Toolbar;

import android.content.Intent;
import android.os.Bundle;
import android.text.TextUtils;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import android.widget.TextView;
import android.widget.Toast;

import com.google.android.gms.tasks.OnCompleteListener;
import com.google.android.gms.tasks.Task;
import com.google.firebase.auth.AuthResult;
import com.google.firebase.auth.FirebaseAuth;
import com.google.firebase.database.DatabaseReference;
import com.google.firebase.database.FirebaseDatabase;

public class RegisterActivity extends AppCompatActivity {

    //Declare instances of the views
    private Button registerBtn;
    private EditText emailField, usernameField, passwordField;
    private TextView loginTxtView;
    //Declare an instance of Firebase Authentication
    private FirebaseAuth mAuth;
    //Declare an instance of FireBase Database
    private FirebaseDatabase database;
    //Declare an instance of FireBase Database Reference;
    // A Database reference is a node in our database, e.g the node users to store user
details
    private DatabaseReference userDetailsReference;


    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_register);
        //inflate the tool bar
        Toolbar toolbar = findViewById(R.id.tool_bar);
        setSupportActionBar(toolbar);

        //Initialize the views
        loginTxtView = findViewById(R.id.loginTxtView);
        registerBtn = findViewById(R.id.registerBtn);
        emailField = findViewById(R.id.emailField);
        usernameField = findViewById(R.id.usernameField);
        passwordField = findViewById(R.id.passwordField);

 // Initialize an Instance of Firebase Authentication  by calling the getInstance() method
        mAuth = FirebaseAuth.getInstance();
  // Initialize an Instance of Firebase Database by calling the getInstance() method
        database = FirebaseDatabase.getInstance();
 //Initialize an Instance of Firebase Database reference by  calling the database instance,
getting a reference  using the get reference() method on the database,and creating a new
child node, in our case "Users" where we will store details of registered users
        userDetailsReference = database.getReference().child("Users");

// For already registered user we want to redirect them to the login page directly without
registering them again
//For this function , setOnClickListener on the textView  object of redirecting user to
login Activity
//Create a Login Activity first using the empty activity template
//Implement an intent to launch this
        loginTxtView.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                Intent loginIntent = new Intent(RegisterActivity.this,
LoginActivity.class);
                startActivity(loginIntent);
            }
        });
 //set an on click listener on your register button, on clicking this button we want to
get: the username, email , password the user enters on edit text fields
 // further we to open a new activity called profile activity where will allow our users to
set a custom display name and their profile image


        registerBtn.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                //create a toast
                Toast.makeText(RegisterActivity.this, "LOADING...",
Toast.LENGTH_LONG).show();
                //get the username entered
                final String username = usernameField.getText().toString().trim();
                //get the email entered
                final String email = emailField.getText().toString().trim();
                //get the password entered
                final String password = passwordField.getText().toString().trim();
                //Validate to ensure that the user has entered email and username
                if (!TextUtils.isEmpty(email) && !TextUtils.isEmpty(username) &&
!TextUtils.isEmpty(password)) {
 //Create a new createAccount method that takes in an email address and password, validates
them, and then creates a new user with the [createUserWithEmailAndPassword] using the
instance of  Firebase Authentication (mAuth) We created and calls addOnCompleteListener

                    mAuth.createUserWithEmailAndPassword(email,
password).addOnCompleteListener(new OnCompleteListener<AuthResult>() {
                        @Override
                        public void onComplete(@NonNull Task<AuthResult> task) {
//override the onComplete method where we'll store this registered user on our database
with respect to their unique id's.

 // Create a string variable to get the  user id of currently  registered user
          String user_id = mAuth.getCurrentUser().getUid();
 //create a child node database reference to attach the user_id to the users node
          DatabaseReference current_user_db = userDetailsReference.child(user_id);
   // set the Username and Image on the users' unique path (current_users_db).
                            current_user_db.child("Username").setValue(username);
                            current_user_db.child("Image").setValue("Default");
   // make a Toast to show the user that they've been successfully registered and then
                            Toast.makeText(RegisterActivity.this, "Registration
Successful", Toast.LENGTH_SHORT).show();
//Create Profile Activity using empty activity template
 //  launch the Profile activity for user to set their preferred profile
            Intent profIntent = new Intent(RegisterActivity.this, ProfileActivity.class);
                   profIntent.addFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP);
                   startActivity(profIntent);

                        }
                    });
                } else {

                    Toast.makeText(RegisterActivity.this, "Complete all fields",
Toast.LENGTH_SHORT).show();
                }
            }
        });
    }

}
```

3. The Profile Activity

Having created this activity in step 2.2.4 and named it Profile Activity, you should now have a layout file (*activity_profile.xml*) open this layout file and create the views as per the layout I shared with you. ***activity_profile.xml*** uses other resources from the drawable folder (mip-map, @drawable/layoutsyle and @drawable/buttonstyle, @drawable/add_profile). I have used this to style a layout, edit Text, button and image button. Feel free to edit this layouts drawables to fit your liking.



Implementing the functionality of Profile Activity

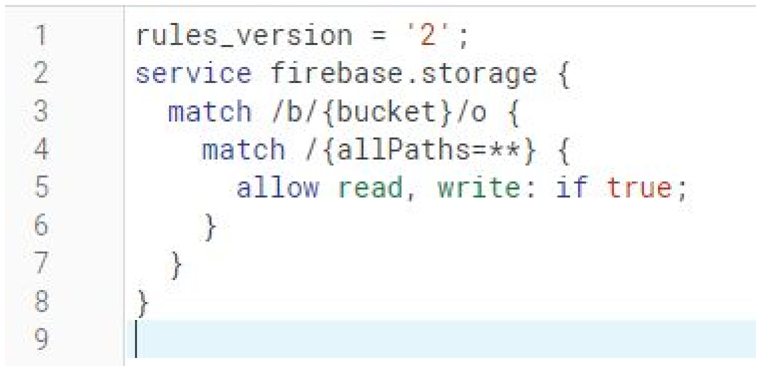
In this activity we will use implicit intent to get and set the preferred user profile image and upload to Firebase Storage under a child node, profile images. After the upload we will then proceed to a download task and get the download url of the image as in Firebase Storage and together with the custom display name, upload this on our real-time database, under the database reference "Users" in the current registered user id node. We will prompt the upload using the done button, hence we will need to set OnClick listener for this button. Further we will use this button to launch the Login Activity.

3.1 Create a default Storage bucket

From the navigation pane of the Firebase console, select **Storage**, then click **Get started**.
Review the messaging about securing your Storage data using security rules. Select a location for your default Storage bucket.
This location setting is your project's *default Google Cloud Platform (GCP) resource location*. Note that this location will be used for GCP services in your project that require a location setting, specifically, your Cloud Firestore database and your App Engine app (which is required if you use Cloud Scheduler).
Warning: After you set your project's default GCP resource location, you cannot change it.

Click **Done**.
During development, consider setting up your rules for public access.

After opening your created storage bucket click on the rules tab and to set them for public access edit them as follows:



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

```

3.2 Add the dependency for the Cloud Storage for Firebase Android library to your module (app-level) Gradle file

```
implementation 'com.google.firebase:firebase-storage:19.1.1'
```

3.3 Declare and initialize instances of Firebase Storage, Firebase Storage Reference, Firebase Authentication, Firebase Database Reference and the views in our activity_profile.xml. See full Profile Activity below with comments to better understand.

3.4 Set on click listener on the image button so as to allow users to pick their profile photo from their gallery. For the purpose of accessing the user's external storage, we'll have to ask for the user's permission to access their device storage by adding this line of code into the application's *manifest.xml* file. Open your applications *manifest.xml* file and add the code below to it.

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

3.5 Upload the photo to firebase storage, get the download link and save the download link in the current user node which is a child of the Users Node you created when registering users in your database. See full Profile Activity below with comments to better understand.

3.6 Get the custom display name from the edit text and further save it to the current user node which is a child of the Users Node you created when registering users in your database. See full Profile Activity below with comments to better understand.

3.7 Launch the Login Activity depending on the success of these tasks. See full Profile Activity below with comments to better understand.

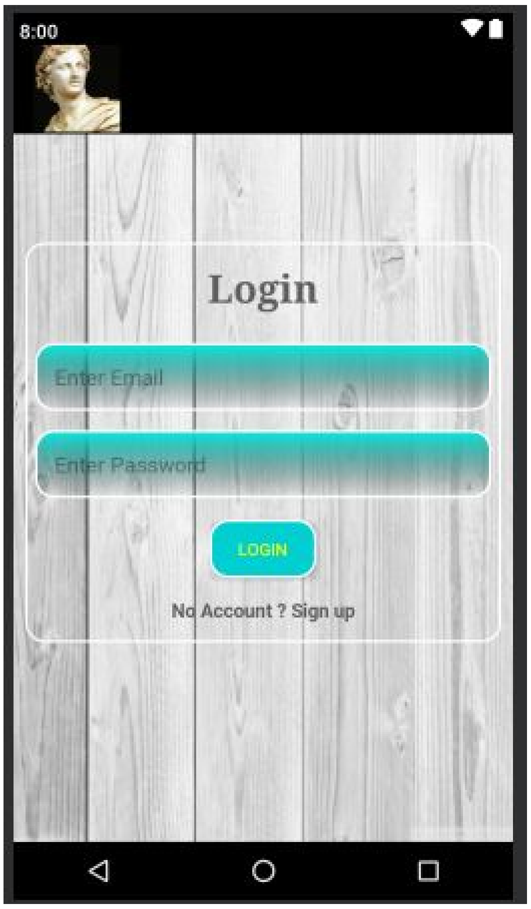
```
package com.mwongela.theattic;

import androidx.annotation.NonNull;
import androidx.appcompat.app.AppCompatActivity;
import androidx.appcompat.widget.Toolbar;

import android.app.ProgressDialog;
import android.content.Intent;
import android.net.Uri;
import android.os.Bundle;
import android.text.TextUtils;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import android.widget.ImageButton;
import android.widget.Toast;

import com.google.android.gms.tasks.OnCompleteListener;
import com.google.android.gms.tasks.OnSuccessListener;
import com.google.android.gms.tasks.Task;
import com.google.firebase.auth.FirebaseAuth;
import com.google.firebase.auth.FirebaseUser;
import com.google.firebase.database.DataSnapshot;
import com.google.firebase.database.DatabaseError;
import com.google.firebase.database.DatabaseReference;
import com.google.firebase.database.FirebaseDatabase;
import com.google.firebase.database.ValueEventListener;
import com.google.firebase.storage.FirebaseStorage;
import com.google.firebase.storage.StorageReference;
import com.google.firebase.storage.UploadTask;

public class ProfileActivity extends AppCompatActivity {

    // Declare instances if the views
    private EditText profUserName;
    private ImageButton imageButton;
    private Button doneBtn;
  // Declare an instance of firebase authentication
    private FirebaseAuth mAuth;
    //Declare an Instance of the database reference  where we will be saving the profile
photo and custom display name
    private DatabaseReference mDatabaseUser;
    //Declare an Instance of the Storage reference where we will upload the photo
    private StorageReference mStorageRef;
    // Declare an Instance of URI for getting the image from our phone, initialize it to
null
    private Uri profileImageUri = null;
//  since we want to get a result (getting and setting image) we will start the implicit
intent using the method startActivityForResult()

//startActivityForResult() method will require two arguments the intent and the request
code

// Declare  and initialize  a private final static int  that will serve as our request code
    private final static int GALLERY_REQ = 1;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_profile);
        //inflate the tool bar
        Toolbar toolbar = findViewById(R.id.tool_bar);
        setSupportActionBar(toolbar);
        //Initialize the  instances of the views
        profUserName = findViewById(R.id.profUserName);
        imageButton = findViewById(R.id.imagebutton);
        doneBtn = findViewById(R.id.doneBtn);
       //Initialize the instance of Firebase authentications
        mAuth = FirebaseAuth.getInstance();
  //We want to set the profile for specific, hence get the user id of the current user and
assign it to a string variable
        final String userID = mAuth.getCurrentUser().getUid();
 //Initialize the database reference where you have your registered users and get the
specific user reference using the user ID
        mDatabaseUser =
FirebaseDatabase.getInstance().getReference().child("Users").child(userID);
```

```
 //Initialize the firebase storage reference where you will store the profile  photo images
        mStorageRef = FirebaseStorage.getInstance().getReference().child("profile_images");
 //set on click listener on the image button so as to allow users to pick their  profile
photo from their gallery
        imageButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                // create an implicit intent for getting the images
                Intent galleryIntent = new Intent(Intent.ACTION_GET_CONTENT);
                //set the type to images only
                galleryIntent.setType("image/*");
//since we need results, use the method  startActivityForResult() and pass the intent and
request code you initialized
                startActivityForResult(galleryIntent, GALLERY_REQ);
            }
        });
// on clicking the images we want to get the name and the profile photo, then later save
this on a database reference for a specfic user
        doneBtn.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                //get the custom display name entered by the user
                final String name = profUserName.getText().toString().trim();
                //validate to ensure that the name and profile image are not null
                if (!TextUtils.isEmpty(name) && profileImageUri != null) {

 //create Storage reference node, inside profile_image storage reference where you will
save the profile image
                    StorageReference profileImagePath =
mStorageRef.child("profile_images").child(profileImageUri .getLastPathSegment());
 //call the putFile() method passing the profile image the user set on the storage
reference where you are uploading the image
 //further call addOnSuccessListener on the reference to listen if the upload task was
successful,and get a snapshot of the task
          profileImagePath .putFile(profileImageUri ).addOnSuccessListener(new
OnSuccessListener<UploadTask.TaskSnapshot>() {
                        @Override
        public void onSuccess(UploadTask.TaskSnapshot taskSnapshot) {
            //if the upload of the profile image was successful get the download url
                      if(taskSnapshot.getMetadata()!=null){
                      if(taskSnapshot.getMetadata().getReference()!=null){
 //get the download url from your storage, use the methods getStorage() and
getDownloadUrl()
                      Task<Uri> result=taskSnapshot.getStorage().getDownloadUrl();
 //call the method addOnSuccessListener to determine if we got the download url
                        result.addOnSuccessListener(new OnSuccessListener<Uri>() {
                          @Override
                          public void onSuccess(Uri uri) {
                        //convert the uri to a string on success
                         final String profileImage =uri.toString();
       // call the method push() to add values on the database reference of  a specific
user
                          mDatabaseUser.push();
   //call the method addValueEventListener to publish the additions in  the database
reference of a specific user
                       mDatabaseUser.addValueEventListener(new ValueEventListener() {
                            @Override
                     public void onDataChange(@NonNull DataSnapshot dataSnapshot) {
           //add the profilePhoto and displayName for the current user
                               mDatabaseUser.child(" displayName").setValue(name);

mDatabaseUser.child("profilePhoto").setValue(profileImage).addOnCompleteListener(new
OnCompleteListener<Void>() {
                                  @Override
                               public void onComplete(@NonNull Task<Void> task) {
                                   if(task.isSuccessful()){
```

```
                              //show a toast to indicate the profile was updated

Toast.makeText(ProfileActivity.this, "Profile Updated", Toast.LENGTH_SHORT).show();
                                       //launch the login activity
               Intent login= new Intent(ProfileActivity.this, LoginActivity.class);

                                   startActivity(login);
                                                            }
                                                        }
                                                    });
                                                }

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

                                                }
                                            });

                                        }
                                    });
                                }
                            }

                        }
                    });

                }
            }
        });

    }
    @Override
    //override this method to get the  profile image set it in the image button view
    protected void onActivityResult(int requestCode, int resultCode, Intent data) {
        super.onActivityResult(requestCode, resultCode, data);

           if (requestCode == GALLERY_REQ && resultCode == RESULT_OK) {
               //get the image selected by the user
            profileImageUri  = data.getData();
            //set in the image button view
            imageButton.setImageURI(profileImageUri );
        }
    }
}
```

4. The Login Activity

Having created this activity in step 2.2.3 and named it Login Activity, you should now have a layout file (*activity_login.xml*) open this layout file and create the views as per the layout I shared with you. ***activity_login.xml*** uses other resources from the drawable folder (mip-map, @drawable/layoutsyle, @drawable/wood and @drawable/buttonstyle). I have used this to style a layout, edit Texts and a button. Feel free to edit these drawables to fit your liking.



Implementing the functionality of Login Activity

In this activity we will be login user using the emails and password they registered with and on success launch the Main activity

4.1 Declare and initialize instances, Firebase Authentication, Firebase Database Reference and the views in our activity_login.xml. See full Login Activity with comments to better understand.

4.2 Set on click listener on the login button so as to allow users login using their email and password. Use the method signInWithEmailAndPassword() on the authentication instance and create a method that checks the user existence in our database reference "Users" using their User ID. See full Login Activity with comments to better understand.

4.3 If Login is successful, launch the main activity. See full Login Activity below with comments to better understand. See full Login Activity with comments to better understand.

4.4 At this point you can test the functionality of Registration, Profile and Login. Edit the manifest and set the Intent launcher filters to Register activity to launch the Register activity before any other activity. Observe your console back end specifically, Authentication, storage and real time database

```
package com.mwongela.theattic;
```

```java
import androidx.annotation.NonNull;
import androidx.appcompat.app.AppCompatActivity;
import androidx.appcompat.widget.Toolbar;

import android.content.Intent;
import android.os.Bundle;
import android.text.TextUtils;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import android.widget.Toast;

import com.google.android.gms.tasks.OnCompleteListener;
import com.google.android.gms.tasks.Task;
import com.google.firebase.auth.AuthResult;
import com.google.firebase.auth.FirebaseAuth;
import com.google.firebase.database.DataSnapshot;
import com.google.firebase.database.DatabaseError;
import com.google.firebase.database.DatabaseReference;
import com.google.firebase.database.FirebaseDatabase;
import com.google.firebase.database.ValueEventListener;

public class LoginActivity extends AppCompatActivity {

    private EditText loginEmail, loginPass;
    private FirebaseAuth mAuth;
    private DatabaseReference mDatabaseUsers;
    private Button loginBtn;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_login);
        //inflate the tool bar
        Toolbar toolbar = findViewById(R.id.tool_bar);
        setSupportActionBar(toolbar);
        //Initialize the views
        loginBtn = findViewById(R.id.loginBtn);
        loginEmail =findViewById(R.id.login_email);
        loginPass = findViewById(R.id.login_password);
        //Initialize the Firebase Authentication instance
        mAuth = FirebaseAuth.getInstance();
        //Initialize the database reference where you have the child node Users
        mDatabaseUsers = FirebaseDatabase.getInstance().getReference().child("Users");
       //Set on click listener on the login button
        loginBtn.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                Toast.makeText(LoginActivity.this, "PROCESSING....",
Toast.LENGTH_LONG).show();
                // get the email and password entered by the user
                String email = loginEmail.getText().toString().trim();
                String password = loginPass.getText().toString().trim();

                if (!TextUtils.isEmpty(email)&& !TextUtils.isEmpty(password)){
                    // use firebase authentication instance you create and call the method
signInWithEmailAndPassword method passing the email and password you got from the views
                    //Further call the addOnCompleteListener() method to handle the
Authentication result

mAuth.signInWithEmailAndPassword(email,password).addOnCompleteListener(new
OnCompleteListener<AuthResult>() {
                        @Override
                        public void onComplete(@NonNull Task<AuthResult> task) {
                            if (task.isSuccessful()){
                                //create a method that will check if the user exists in our
database reference
                                checkUserExistence();
                            }else {
                      //if the user does not exist in the database reference throw a toast
                               Toast.makeText(LoginActivity.this, "Couldn't login, User not
found", Toast.LENGTH_SHORT).show();
                            }
                        }
                    });
                }else {
                   //if the fields for email and password were not completed show a toast
                    Toast.makeText(LoginActivity.this, "Complete all fields",
Toast.LENGTH_SHORT).show();
                }
            }
        });
    }
     //check if the user exists
    public void checkUserExistence(){
       //check the user existence of the user using the user id in users database reference
        final String user_id = mAuth.getCurrentUser().getUid();
//call the method addValueEventListener on the database reference of the user to determine
if the current userID supplied exists in our database reference
        mDatabaseUsers.addValueEventListener(new ValueEventListener() {
            @Override
            public void onDataChange(DataSnapshot dataSnapshot) {
  //get a dataSnapshot of the users database reference to determine if current user exists

                if (dataSnapshot.hasChild(user_id)){
                    //if the users exists direct the user to the Main Activity
                    Intent mainPage = new Intent(LoginActivity.this, MainActivity.class);
                    startActivity(mainPage);
                }else {
                    //if the user id does not exist show a toast
                    Toast.makeText(LoginActivity.this, "User not registered!",
Toast.LENGTH_SHORT).show();
                }
            }

            @Override
            public void onCancelled(DatabaseError databaseError) {

            }
        });
    }
}
```

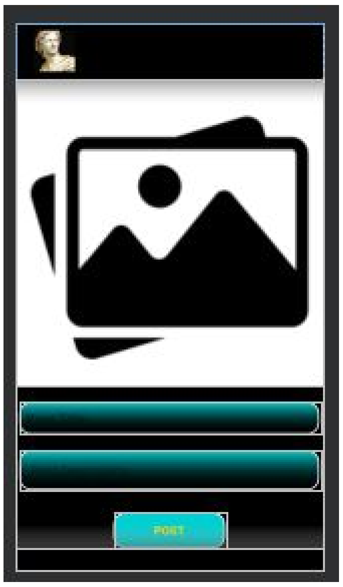
5. The Post Activity

Add a new activity using the empty activity template in your package and name it the Post Activity, we will use this activity to create posts that will be uploaded to our real time database and populated in the Main activity

5.1 Define the layout for Post Activity

Having created this activity and named it Post Activity, you should now have a layout file (*activity_post.xml*) open this layout file and create the views as per the layout I shared with you. activity_post.xml uses other resources from the drawable folder

(@drawable/buttonstyle, @drawable/add_photo). I have used this to style a layout, edit texts , image button and a button, feel free to edit this drawables to fit your liking



Implementing the functionality of the Post Activity

5.2 Declare and get instances of the Firebase objects and view objects we created in the *activity_post.xml* layout file, read values from them and with the help of the Post Button, send all the data to our Firebase database. See full Post Activity with comments to better understand.

5.3 Inside the *onCreate()* method, we initialized all the view and Firebase Objects that will be relevant for handling the required task in this Class. See full Post Activity with comments to better understand.

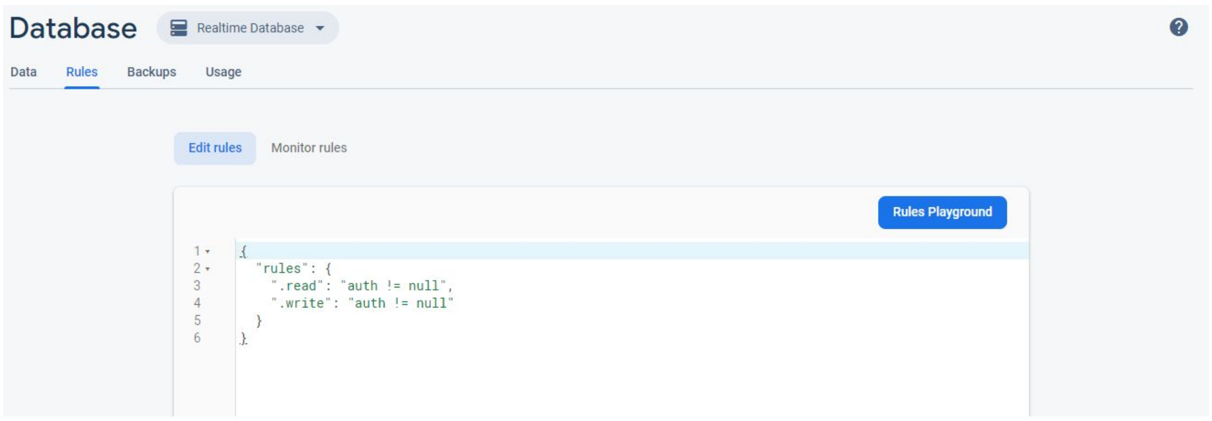
5.4 Set up the ImageButton to access the user's device gallery and pick the desired image for the post using an intent. See full Post Activity with comments to better understand.

5.5 proceed to set up the post button such that upon clicking it, it'll store the page contents to our Firebase database. To achieve this, first we "got" the values coming from the *EditText* fields and "stored" them into String variables, then we also got the date and time of the post ,then did a check to make certain that the fields are not empty, after which we then called an instance of the Firebase *StorageReference* where we specified the path to store the post images.

Then to store the post *Title*, *Description, Post Image, Date-Time, display name of the person posting and profile image* into our database, we called an instance of Firebase *DatabaseReference* and added a child to it (new post) where we'll then store all the values. See full Post Activity with comments to better understand.

5.6 On successful posting launch the Main Activity

5.7 Since we authenticated users, you'll need to log into your firebase console and open the *RULES* tab under Database and Storage and set the Read and Write rules as follows so that only authenticated users can post data to it.



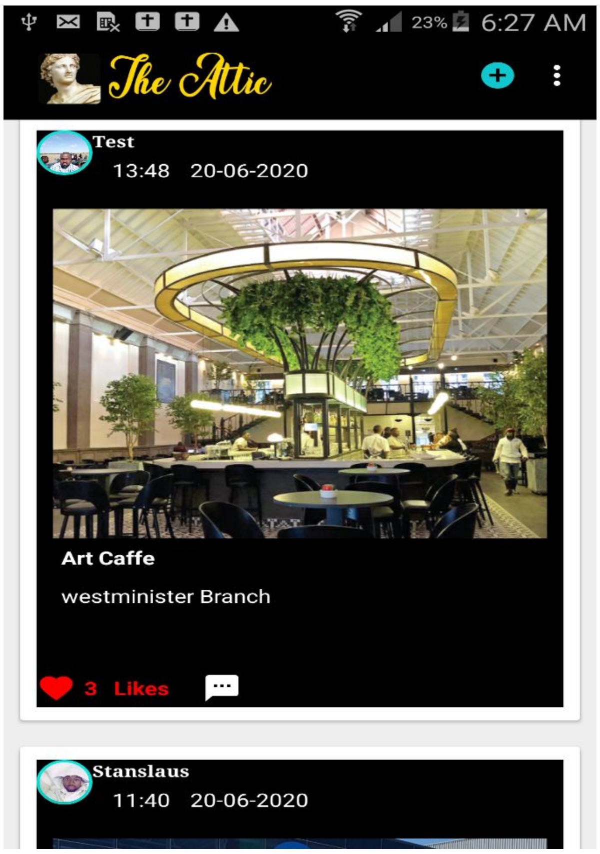
```
{
  "rules": {
    ".read": "auth != null",
    ".write": "auth != null"
  }
}
```

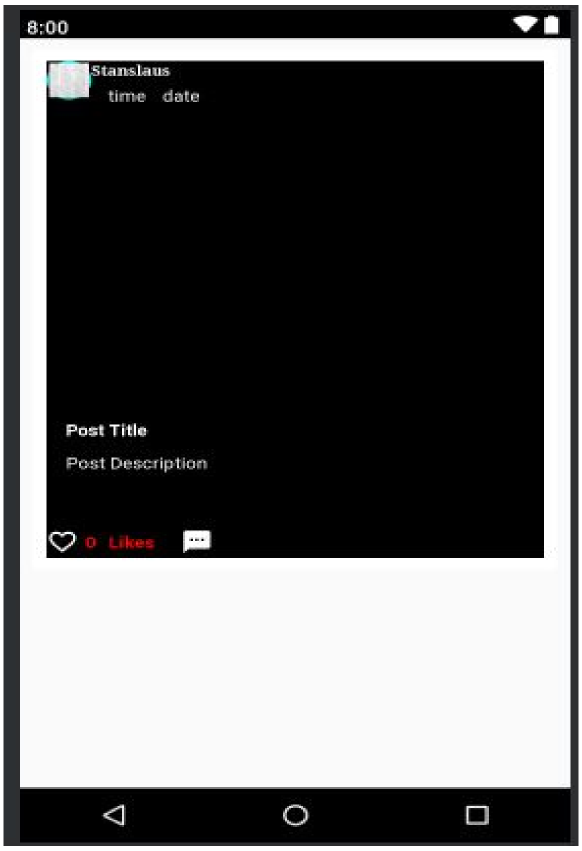
```java
package com.mwongela.theattic;

import androidx.annotation.NonNull;
import androidx.appcompat.app.AppCompatActivity;
import androidx.appcompat.widget.Toolbar;

import android.content.Intent;
import android.net.Uri;
import android.os.Bundle;
import android.text.TextUtils;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import android.widget.ImageButton;
import android.widget.Toast;

import com.google.android.gms.tasks.OnCompleteListener;
import com.google.android.gms.tasks.OnSuccessListener;
import com.google.android.gms.tasks.Task;
import com.google.firebase.auth.FirebaseAuth;
import com.google.firebase.auth.FirebaseUser;
import com.google.firebase.database.DataSnapshot;
import com.google.firebase.database.DatabaseError;
import com.google.firebase.database.DatabaseReference;
import com.google.firebase.database.FirebaseDatabase;
import com.google.firebase.database.ValueEventListener;
import com.google.firebase.storage.FirebaseStorage;
import com.google.firebase.storage.StorageReference;
import com.google.firebase.storage.UploadTask;

import java.text.SimpleDateFormat;
import java.util.Calendar;

public class PostActivity extends AppCompatActivity {

    // Declare the view objects
    private ImageButton imageBtn;
    private EditText textTitle;
    private EditText textDesc;
    private Button postBtn;

    //Declare an Instance of the Storage reference where we will upload the post photo
    private StorageReference mStorageRef;
 //Declare an Instance of the database reference  where we will be saving the post details
```

```
   private DatabaseReference databaseRef;
   //Declare an Instance of firebase authentication
    private FirebaseAuth mAuth;
    //Declare an Instance of the database reference  where we have user details
    private DatabaseReference mDatabaseUsers;
    //Declare a Instance of currently logged in user
    private FirebaseUser mCurrentUser;
    // Declare  and initialize  a private final static int  that will serve as our request
code


    private static final int GALLERY_REQUEST_CODE = 2;
  // Declare an Instance of URI for getting the image from our phone, initialize it to null
    private Uri uri = null;



    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_post);
        //inflate the tool bar
        Toolbar toolbar = findViewById(R.id.tool_bar);
        setSupportActionBar(toolbar);
        // initializing  view objects
        postBtn = findViewById(R.id.postBtn);
        textDesc = findViewById(R.id.textDesc);
        textTitle = findViewById(R.id.textTitle);
        //Initialize the storage reference
        mStorageRef = FirebaseStorage.getInstance().getReference();
        //Initialize the database reference/node where you will be storing posts
        databaseRef = FirebaseDatabase.getInstance().getReference().child("Posts");
        //Initialize an instance of  Firebase Authentication
        mAuth = FirebaseAuth.getInstance();
        //Initialize the instance of the firebase user
        mCurrentUser = mAuth.getCurrentUser();
        //Get currently logged in user
        mDatabaseUsers =
FirebaseDatabase.getInstance().getReference().child("Users").child(mCurrentUser.getUid());
        // initialize the image button
        imageBtn = findViewById(R.id.imgBtn);
        //picking image from gallery
        imageBtn.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                Intent galleryIntent = new Intent(Intent.ACTION_GET_CONTENT);
                galleryIntent.setType("image/*");
                startActivityForResult(galleryIntent, GALLERY_REQUEST_CODE);
            }
        });
        // posting to Firebase
        postBtn.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                Toast.makeText(PostActivity.this, "POSTING...", Toast.LENGTH_LONG).show();
                //get title and desc from the edit texts
                final String PostTitle = textTitle.getText().toString().trim();
                final String PostDesc = textDesc.getText().toString().trim();
                //get the date and time of the post

                java.util.Calendar calendar = Calendar.getInstance();
                SimpleDateFormat currentDate=  new SimpleDateFormat("dd-MM-yyyy");
                final String saveCurrentDate=currentDate.format(calendar.getTime());

                java.util.Calendar calendar1 = Calendar.getInstance();
                SimpleDateFormat currentTime=  new SimpleDateFormat("HH:mm");
        final String  saveCurrentTime=currentTime.format(calendar1.getTime());
        // do a check for empty fields
        if (!TextUtils.isEmpty(PostDesc) && !TextUtils.isEmpty(PostTitle)) {

  //create Storage reference node, inside post_image storage reference where you will
save the post image
          StorageReference filepath =
mStorageRef.child("post_images").child(uri.getLastPathSegment());
 //call the putFile() method passing the post image the user set on the storage reference
where you are uploading the image
 //further call addOnSuccessListener on the reference to listen if the upload task was
successful,and get a snapshot of the uploaded image
          filepath.putFile(uri).addOnSuccessListener(new
OnSuccessListener<UploadTask.TaskSnapshot>() {
            @Override
            public void onSuccess(UploadTask.TaskSnapshot taskSnapshot) {
          //if the upload of the post image was successful get the download url
              if (taskSnapshot.getMetadata() != null) {
                if (taskSnapshot.getMetadata().getReference() != null) {
//get the download url from your storage use the methods getStorage() and getDownloadUrl()
           Task<Uri> result = taskSnapshot.getStorage().getDownloadUrl();
      //call the method addOnSuccessListener to determine if we got the download url
               result.addOnSuccessListener(new OnSuccessListener<Uri>() {
                    @Override
                    public void onSuccess(Uri uri) {
                      //convert the uri to a string on success
                      final String imageUrl = uri.toString();

                  Toast.makeText(getApplicationContext(), "Successfully
Uploaded", Toast.LENGTH_SHORT).show();
        // call the method push() to publish the values on the database reference
                  final DatabaseReference newPost = databaseRef.push();
                     //adding post contents to database
reference,call addValueEventListener so as to set the values
           mDatabaseUsers.addValueEventListener(new ValueEventListener() {
                   @Override
                 public void onDataChange(DataSnapshot dataSnapshot) {

newPost.child("title").setValue(PostTitle);

newPost.child("desc").setValue(PostDesc);

newPost.child("postImage").setValue(imageUrl);

newPost.child("uid").setValue(mCurrentUser.getUid());

newPost.child("time").setValue(saveCurrentTime);

newPost.child("date").setValue(saveCurrentDate);
 //get the profile photo and display name of the person posting

newPost.child("profilePhoto").setValue(dataSnapshot.child("profilePhoto").getValue());

newPost.child("displayName").setValue(dataSnapshot.child("displayName").getValue()).
                              addOnCompleteListener(new
OnCompleteListener<Void>() {

              @Override
             public void onComplete(@NonNull Task<Void> task) {
                                if (task.isSuccessful()){
                        //launch the main activity after posting
           Intent intent = new Intent(PostActivity.this, MainActivity.class);

                       startActivity(intent);
                                  }
                                }
```

```
                                                            });
                                                }

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

                                                }
                                            });
                                        }
                                    });
                                }
                            }
                        }
                    });
                }
            }
        });
    }


    @Override
    // image from gallery result
    protected void onActivityResult(int requestCode, int resultCode, Intent data) {
        super.onActivityResult(requestCode, resultCode, data);

        if (requestCode == GALLERY_REQUEST_CODE && resultCode == RESULT_OK){
            //get the image selected by the user
            uri = data.getData();
            //set the image
            imageBtn.setImageURI(uri);
        }
    }
}
```

Back to Main Activity
Rendering
At this point we can post to Firebase, the next task is to have this post appear on our homepage(The main activity) from where we can view them. First off we need to create a layout file that will hold the data coming from Firebase, we'll do this with *CardViews* as it'll be the best to handle the task given the nature of the App. So head on over to layout in resources and create a new layout resource file, I named mine as card_items. Define the layout of the card views using my layout as a guide.





Having created the layout to hold the individual posts coming from Firebase, it's time to create our *Recyclerview* Layout in the content_main.xml. This view will hold the Cards we just created in the layout above. So open your content_main.xml file (came with the basic activity template) and create your Recyclerview. Content_main.xml is the Layout that renders on the activity_main.xml.

Recap: *What we have just done is create a Cardview layout that will contain the contents of the blog post made by the user for the purpose of showing it to other users. To achieve this, we also created a Recyclerview layout where we'll be posting the individual Cards (populating the recyclerview layout with the cardview layout).*

Now let's proceed to the *MainActivity.java* file and write the code that will achieve this desired functionality (to render the post from our database to the defined view objects).

Before that, we need to create a model class that will act like an *Adapter* to help us bind the data from the server down to the respective view objects, hence, create a new java class called "Attic". This is how your model should look like

```
package com.mwongela.theattic;

public class Attic {
   //declare the variable
   private String title, desc, postImage, displayName, profilePhoto, time, date;
   //create a constructor
```

```
        public Attic(String title, String desc, String postImage, String displayName,
String profilePhoto, String time, String date) {
            this.title = title;
            this.desc = desc;
            this.postImage=postImage;
            this.displayName = displayName;
            this.profilePhoto=profilePhoto;
            this.time=time;
            this.date=date;
        }
         //requires an empty constructor
        public Attic() {
        }
       // setters
        public void setPostImage(String postImage){
            this.postImage=postImage;

        }
        public void setProfilePhoto(String profilePhoto) {
            this.profilePhoto = profilePhoto;
        }

        public void setDisplayName(String displayName) {
            this.displayName = displayName;
        }
        public void setTitle(String title) {
            this.title = title;
        }

        public void setDesc(String desc) {
            this.desc = desc;
        }

        public void setTime(String time){
            this.time=time;
        }
        public void setDate(String date){
            this.date=date;
        }
      //getters
       public String getDisplayName() {
        return displayName;
    }

        public String getPostImage() {
            return postImage;
        }

        public String getTitle() {
            return title;
        }

        public String getDesc() {
            return desc;
        }
        public String getProfilePhoto()
        {
            return profilePhoto;
        }
        public String getTime(){
            return time;
        }
        public String getDate(){
            return date;
        }
```

```
}
```

Before we proceed let’s add the implementations we will need for our Firebase Recycler adapter and loading images

```
implementation 'com.firebaseui:firebase-ui-database:3.1.3'

implementation 'com.squareup.picasso:picasso:2.5.2'
implementation 'com.github.bumptech.glide:glide:4.11.0'
```

We will then initialize the necessary Firebase Objects and the Recyclerview widget. Then handle the user login which we’ll soon get into but in the meantime, observe the code and understand how simple it is to login users with Firebase.

Next, we create a static inner class that extends *Recyclerview.ViewHolder*. This is primarily where we’ll “set” the data coming from the server to its respect view object using the Attic model class we had earlier created.

Then we override the onStart() method to check if the user is logged in , after which we then fire up the FirebaseRecyclerAdapter that will take in two parameters (the Attic model class and the Viewholder static inner class that we just created). There after we tell our adapter to start listening for events

A new instance of the *FirebaseRecyclerAdapter* will then take in all the three parameters which are basically the components we’ll need to successfully bind the incoming data to the viewholders (i.e the Blogzone class, the *ViewHolder* class, and the database instance).

The FirebaseRecyclerAdapter will implement a protected method called *onBindViewHolder()* which takes in three parameters (*model class, the ViewHolder class and an integer variable called position*). With these, we can go ahead and populate our viewHolder class.

In the called *onBindViewHolder()* method, we then call the *set()* method on the *viewHolder* instance to populate it with values from the Attic model class. We also use a final String variable to store the *post_key* that will allow us to identify a particular post and associate it with its corresponding position on the recyclerview.

This will come in handy while clicking on a particular card to like it or open it’s contents on a different Activity (**SinglePostActivity**) as we just did with the *onClickListener()* set on the viewHolder .

Then finally to define an Adapter for our recyclerview, we do

```
recyclerView.setAdapter(adapter);
adapter.notifyDataSetChanged();
```

Next we implement the functions of **Logging out users and launching the post activity from the main activity**

Finally you need to override the onStop() method to tell the adapter to stop listening when the user stops using the app

P.s If you had initially set the launcher to register activity when testing , revert it to Main activity since in our main activity we can now check if a user is registered or not, and if not we direct him to the registration page

Next we will create Persistence for your database by extending your application and calling the method setPersistenceEnabled()

To do this, go to your package where you have your java files, create new Java class, give it the name FirebaseHandler, set the superclass as **android.app.Application,** click ok and then inside the class, override the create method and set Persistence. Finally you will add this sub-application to the manifest by adding this line on your application tag

```
android:name=".FirebaseHandler"
```

Below is how your Firebase Persistence handler should look like

```
package com.mwongela.theattic;

import android.app.Application;

import com.google.firebase.database.FirebaseDatabase;

public class FirebaseHandler extends Application {
    @Override
    public void onCreate() {
        super.onCreate();
        FirebaseDatabase.getInstance().setPersistenceEnabled(true);
    }
}
```

My final manifest including all the labels

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.mwongela.myblog">

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

    <application
        android:name=".FirebaseHandler"
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <activity android:name=".ProfileActivity"
            android:label="@string/set_profile"
            >

        </activity>
        <activity android:name=".SinglePostActivity" />
        <activity android:name=".LoginActivity"
```

```
                 android:label="@string/app_name"/>
        <activity android:name=".RegisterActivity"
                 android:label="@string/app_name"/>

        <activity android:name=".PostActivity"
            android:label="@string/addpost"/>


        <activity
            android:name=".MainActivity"
            android:label="@string/app_name"
            >
        <intent-filter>
            <action android:name="android.intent.action.MAIN" />

            <category android:name="android.intent.category.LAUNCHER" />
        </intent-filter>

    </activity>
    </application>

</manifest>
```
Final Main Activity with all the comments

```
package com.mwongela.theattic;

import android.content.Context;
import android.content.Intent;
import android.os.Bundle;

import com.firebase.ui.database.FirebaseRecyclerAdapter;
import com.firebase.ui.database.FirebaseRecyclerOptions;
import com.firebase.ui.database.SnapshotParser;
import com.google.android.material.floatingactionbutton.FloatingActionButton;
import com.google.android.material.snackbar.Snackbar;
import com.google.firebase.auth.FirebaseAuth;
import com.google.firebase.auth.FirebaseUser;
import com.google.firebase.database.DataSnapshot;
import com.google.firebase.database.DatabaseError;
import com.google.firebase.database.DatabaseReference;
import com.google.firebase.database.FirebaseDatabase;
import com.google.firebase.database.Query;
import com.google.firebase.database.ValueEventListener;
import com.squareup.picasso.Picasso;

import androidx.annotation.NonNull;
import androidx.appcompat.app.AppCompatActivity;
import androidx.appcompat.widget.Toolbar;
import androidx.recyclerview.widget.LinearLayoutManager;
import androidx.recyclerview.widget.RecyclerView;

import android.view.LayoutInflater;
import android.view.View;
import android.view.Menu;
import android.view.MenuItem;
import android.view.ViewGroup;
import android.widget.ImageButton;
import android.widget.ImageView;
import android.widget.LinearLayout;
import android.widget.TextView;
import android.widget.Toast;

public class MainActivity extends AppCompatActivity {

    private RecyclerView recyclerView;
    private DatabaseReference likesRef;
```

```java
    private FirebaseAuth mAuth;
    // private FirebaseUser mCurrentUser;
    private FirebaseAuth.AuthStateListener mAuthListener;
    Boolean likeChecker =false;
    private FirebaseRecyclerAdapter adapter;
    String currentUserID =null;
    //boolean likeChecker;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        //inflate the tool bar
        Toolbar toolbar = findViewById(R.id.tool_bar);
        setSupportActionBar(toolbar);

        //initialize recyclerview
        recyclerView = findViewById(R.id.recyclerView);
        LinearLayoutManager linearLayoutManager = new LinearLayoutManager(this);
        //Reverse  the layout so as to display the most recent post at the top
        linearLayoutManager.setReverseLayout(true);
        recyclerView.setLayoutManager(linearLayoutManager);
        recyclerView.setHasFixedSize(true);

        //get the database reference where you will fetch posts
       // mDatabase = FirebaseDatabase.getInstance().getReference().child("Posts");
        //Initialize the database reference where you will store likes
       likesRef = FirebaseDatabase.getInstance().getReference().child("Likes");
        //get an instance of firebase authentication
        mAuth = FirebaseAuth.getInstance();
             //get currently logged in user
            FirebaseUser currentUser = mAuth.getCurrentUser();
            if (currentUser == null) {
                // if user is not logged in refer him/he ro the register activity
                Intent loginIntent = new Intent(MainActivity.this, RegisterActivity.class);
                loginIntent.addFlags(Intent.FLAG_ACTIVITY_CLEAR_TASK);
                startActivity(loginIntent);
            }

        }

        @Override
        protected void onStart() {
            //

            super.onStart();
            FirebaseUser currentUser = mAuth.getCurrentUser();
            //check to see if the user is logged in
            if (currentUser != null) {
                //if user is logged in populate the Ui With card views
                updateUI(currentUser);
               // Listen to the events on the adapter
            adapter.startListening();

            }

         }

        private void updateUI(final FirebaseUser currentUser) {
          //create and initialize an instance of Query that retrieves all posts uploaded
            Query query = FirebaseDatabase.getInstance().getReference().child("Posts");
   // Create and initialize an instance of Recycler Options passing in your model clas
FirebaseRecyclerOptions<Attic> options = new FirebaseRecyclerOptions.Builder<Attic>().
                    setQuery(query, new SnapshotParser<Attic>() {
                        @NonNull
                        @Override
                        //Create a snapshot of your model
                        public Attic parseSnapshot(@NonNull DataSnapshot snapshot) {
                            return new Attic(snapshot.child("title").getValue().toString(),
                                    snapshot.child("desc").getValue().toString(),
                                    snapshot.child("postImage").getValue().toString(),
                                    snapshot.child("displayName").getValue().toString(),
                                    snapshot.child("profilePhoto").getValue().toString(),
                                    snapshot.child("time").getValue().toString(),
                                    snapshot.child("date").getValue().toString());

                        }
                    })
                    .build();
        // crate a fire base adapter passing in the model, and a View holder
     // Create a  new ViewHolder as a public inner class that extends RecyclerView.Holder,
outside the create , start and update the Ui methods.
       /Then implement the methods onCreateViewHolder and onBindViewHolder
      //Complete all the steps in the AtticViewHolder before proceeding to  the methods
onCreateViewHolder, and onBindViewHolder
            adapter = new FirebaseRecyclerAdapter<Attic, AtticViewHolder>(options) {


                @NonNull
                @Override
                public AtticViewHolder onCreateViewHolder(@NonNull ViewGroup parent, int
viewType) {
                    //inflate the layout where you have the card view items
 View view = LayoutInflater.from(parent.getContext()).inflate(R.layout.card_items, parent,
false);
                    return new AtticViewHolder(view);
                }

  @Override
 protected void onBindViewHolder(@NonNull AtticViewHolder holder, int position, @NonNull
Attic model) {
   // very important for you to get the post key since we will use this to set likes and
delete a particular post
                    final String post_key = getRef(position).getKey();
                    //populate the card views with data
                    holder.setTitle(model.getTitle());
                    holder.setDesc(model.getDesc());
                    holder.setPostImage(getApplicationContext(), model.getPostImage());
                    holder.setUserName(model.getDisplayName());
                    holder.setProfilePhoto(getApplicationContext(),
model.getProfilePhoto());
                    holder.setTime(model.getTime());
                    holder.setDate(model.getDate());
                    //set a like on a particular post
                    holder.setLikeButtonStatus(post_key);
 //add  on click listener on the a particular post to  allow opening this post on a
different screen
                    holder.post_layout.setOnClickListener(new View.OnClickListener() {
                        @Override
                        public void onClick(View view) {
         //launch the screen single post activity on clicking a particular cardview item
           //create this activity using the empty activity template
           Intent singleActivity = new Intent(MainActivity.this, SinglePostActivity.class);
                            singleActivity.putExtra("PostID", post_key);
                            startActivity(singleActivity);
                        }
                    });
                    // set the onclick listener on the button for liking a post
                    holder.likePostButton.setOnClickListener(new View.OnClickListener() {

                        @Override
                        public void onClick(View v) {
```

```
  // initialize the like checker to true, we are using this boolean variable to determine
if a post has been liked or dislike
                            // we declared this variable on to of our activity class
                            likeChecker = true;
                            //check the currently logged in user using his/her ID
                            FirebaseUser user =
FirebaseAuth.getInstance().getCurrentUser();
                            if (user != null) {
                                currentUserID=user.getUid();
                            } else {
                                Toast.makeText(MainActivity.this,"please
login",Toast.LENGTH_SHORT).show();

                            }
                              //Listen to changes in the likes database reference
                            likesRef.addValueEventListener(new ValueEventListener() {
                                @Override
                                public void onDataChange(@NonNull DataSnapshot
dataSnapshot) {
                                    if (likeChecker.equals(true)) {
      // if the current post has a like, associated to the current logged and the user
clicks on it again, remove the like, basically this means the user is disliking the post
if (dataSnapshot.child(post_key).hasChild(currentUserID)) {

likesRef.child(post_key).child(currentUserID).removeValue();
                                            likeChecker = false;
                                        } else {
 //here the user is liking, set value on the like

likesRef.child(post_key).child(currentUserID).setValue(true);
                                            likeChecker = false;
                                        }
                                    }
                                }

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

                                }
                            });
                        }
                    });
                }
            };
            recyclerView.setAdapter(adapter);
            adapter.notifyDataSetChanged();
        }

        @Override
        protected void onStop() {
        super.onStop();
        FirebaseUser currentUser = mAuth.getCurrentUser();
         if (currentUser != null) {
               adapter.stopListening();
                   }

                }


    public class AtticViewHolder extends RecyclerView.ViewHolder{
        //Declare the view objects in the card view
        public TextView post_title;
        public TextView post_desc;
        public ImageView post_image;
        public TextView postUserName;
        public ImageView user_image;
        public TextView postTime;
        public TextView postDate;
        public LinearLayout post_layout;
        public ImageButton likePostButton, commentPostButton;
        public TextView displayLikes;
        //Declare an int variable to hold the count  of likes
        int countLikes;
        //Declare a string variable to hold  the user ID of currently logged in user
        String currentUserID;
        //Declare an instance of firebase authentication
        FirebaseAuth mAuth;
        //Declare a database reference where you are saving  the likes
        DatabaseReference likesRef;
        //create constructor matching super
        public AtticViewHolder(@NonNull View itemView) {
            super(itemView);
            //Initialize the card view item objects
            post_title = itemView.findViewById(R.id.post_title_txtview);
            post_desc = itemView.findViewById(R.id.post_desc_txtview);
            post_image = itemView.findViewById(R.id.post_image);
            postUserName = itemView.findViewById(R.id.post_user);
            user_image = itemView.findViewById(R.id.userImage);
            postTime = itemView.findViewById(R.id.time);
            postDate = itemView.findViewById(R.id.date);
            post_layout = itemView.findViewById(R.id.linear_layout_post);
            likePostButton = itemView.findViewById(R.id.like_button);
            commentPostButton = itemView.findViewById(R.id.comment);
            displayLikes = itemView.findViewById(R.id.likes_display);

            //Initialize a database reference where you will store  the likes
            likesRef = FirebaseDatabase.getInstance().getReference().child("Likes");
        }
        // create yos setters, you will use this setter in you onBindViewHolder method
        public void setTitle(String title){

            post_title.setText(title);
        }
        public void setDesc(String desc){

            post_desc.setText(desc);
        }
        public void setPostImage(Context ctx, String postImage){

            Picasso.with(ctx).load(postImage).into(post_image);
        }
        public void setUserName(String userName){

            postUserName.setText(userName);
        }
        public void  setProfilePhoto(Context context,String profilePhoto){
            Picasso.with(context).load(profilePhoto).into(user_image);

        }
        public void setTime(String time) {
            postTime.setText(time);
        }
        public void setDate(String date) {
            postDate.setText(date);
        }
        public void setLikeButtonStatus(final String post_key){
           //we want to know who has like a particular post, so let's get the user using
their user_ID
            FirebaseUser user = FirebaseAuth.getInstance().getCurrentUser();
            if (user != null) {
                currentUserID = user.getUid();
            } else {
                Toast.makeText(MainActivity.this,"please login",Toast.LENGTH_SHORT).show();
            }

      // Listen to changes in the database reference of Likes
            likesRef.addValueEventListener(new ValueEventListener() {
                @Override
                public void onDataChange(@NonNull DataSnapshot dataSnapshot) {
                    //define post_key in the in the onBindViewHolder method
                    //check if a particular post has been liked
                    if(dataSnapshot.child(post_key).hasChild(currentUserID)){
                        //if liked get the number of likes
                        countLikes=(int) dataSnapshot.child(post_key).getChildrenCount();
                        //check the image from initial dislike to like
                        likePostButton.setImageResource(R.drawable.like);
                       // count the like and display them in the textView for likes
                        displayLikes.setText(Integer.toString(countLikes));
                    }else {
                          //If disliked, get the current number of likes
                        countLikes=(int) dataSnapshot.child(post_key).getChildrenCount();
                        // set the image resource as disliked
                        likePostButton.setImageResource(R.drawable.dislike);
                        //display the current number of likes
                        displayLikes.setText(Integer.toString(countLikes));
                    }

                }

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

                }
            });
        }
    }
    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        // Inflate the menu; this adds items to the action bar if it is present.
        getMenuInflater().inflate(R.menu.menu_main, menu);
        return true;
    }
    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        int id = item.getItemId();
        if (id == R.id.action_settings) {
            return true;
        }
        //implement the functionality of the add icon, so that the user on clicking it
launches the post activity
        else if (id == R.id.action_add) {
            Intent postIntent=new Intent(this,PostActivity.class);
            startActivity(postIntent);
           // on clicking logout, log the user out
        } else if (id == R.id.logout){
            mAuth.signOut();
            Intent logouIntent = new Intent(MainActivity.this, RegisterActivity.class);
            logouIntent.addFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP);
            startActivity(logouIntent);
        }
        return super.onOptionsItemSelected(item);
    }
}
```

Wow….. that was a lot of code yeah? I agree, however, if you did everything as you should, then on successfully posting to Firebase, your post should immediately render on your homepage (*activity_main.xml*) as it appeared in the app image I posted earlier.

Finally!!!

6 Single Post Activity

One more functionality we'd like to add is to have click events on the cards we used to populate the Recyclerview so that users can click on a particular post and have it open up in another activity where they can view all the post contents and even delete the post if they are the ones that posted it. So we'll go ahead and create another Activity called SinglePostActivity.

Define the layout for Single Post Activity

Having created this activity and named it Post Activity, you should now have a layout file (*activity_single_post.xml*) open this layout file and create the views as per the layout I shared with you. activity_single_post.xml uses other resources from the drawable folder.

Implement the functionality for single Post Activity

This layout simply has an ImageView to hold the post image, two TextViews to hold the post Title and Description and a delete Button to delete the post.

The first thing we'll do will be to go back to our MainActivity class and extract the position of the clicked item. To achieve this we created a variable that stored the post_key of every unique

FYI. We already did this

Then we also set an onClickListener on the viewHolder instance inside the onBindViewHolder() method and pass in the post_key with an intent so as to open it up in the SinglePostActivity with its corresponding details.

So then let's set up the SinglePostActivity class to receive all the data coming in from the "clicked" Post on the recyclerview.

So here what we did was first get the Intent we passed from the MainActivity class and retrieve the information into a String variable (post_key). At this point, the post_key variable represents every post on our database.

So we get references to the view objects we created in the layout (activity_single_post.xml) and also get references to our Firebase objects. On the FirebaseDatabase reference, we pass the post_key as a child and attach a addValueEventListener() method which will implement the onDataChanged() method where we'll then store the individual values from our data snapshots into String variables. We'll then set these values on the view objects of the [activity_single_post.xml] file and boom our post will successfully appear in the Single Post tActivity upon click from the recyclerview holder.

Delete Post

Next, we hook up the delete button to delete the post from the database within the app. Since the post is associated with it's post_key and user_id, it becomes very simple to delete the post but only when the user_id of the currentUser matches the user_id of the post such that another user cannot delete someone else's post.

So first things first, to delete the post we set an onClickListener on the delete Button and inside the onClick() method, we call the removeValue() method on the FirebaseAuth instance. then use an intent to move the user back to the MainActivity.

Next we'd like to make sure that only the users who made that post will be able to see this Delete button so inside the SinglePostActivity, you may have noticed that we initially made the button invisible

deleteBtn.setVisibility(View.INVISIBLE);

To make the button visible to the post owner we then do a check to see if the current user's unique id matches the post uid which we passed with the intent. Then, if the current users id matches with the post uid, we make the deleteBtn visible.

If (mAuth.getCurrentUser().getUid().equals(post_uid)){

deleteBtn.setVisibility(View.VISIBLE);

} }

So we are done with the delete button. At this point we have a fully functional App. Users can register, login, make a post, view the post and even delete the post.

Below is the full Single Post Activity, Add comments to it:

```java
package com.mwongela.theattic;

import androidx.appcompat.app.AppCompatActivity;

import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.ImageView;
import android.widget.TextView;

import com.google.firebase.auth.FirebaseAuth;
import com.google.firebase.database.DataSnapshot;
import com.google.firebase.database.DatabaseError;
import com.google.firebase.database.DatabaseReference;
import com.google.firebase.database.FirebaseDatabase;
import com.google.firebase.database.ValueEventListener;
import com.squareup.picasso.Picasso;

public class SinglePostActivity extends AppCompatActivity {
    private ImageView singelImage;
    private TextView singleTitle, singleDesc;
    String post_key = null;
    private DatabaseReference mDatabase;
    private Button deleteBtn;
    private FirebaseAuth mAuth;

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

        singelImage = findViewById(R.id.singleImageview);
        singleTitle = findViewById(R.id.singleTitle);
        singleDesc = findViewById(R.id.singleDesc);

        mDatabase = FirebaseDatabase.getInstance().getReference().child("Posts");
        post_key = getIntent().getExtras().getString("PostID");
        deleteBtn = findViewById(R.id.deleteBtn);
        mAuth = FirebaseAuth.getInstance();
        deleteBtn.setVisibility(View.INVISIBLE);
        deleteBtn.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {

                mDatabase.child(post_key).removeValue();

                Intent mainintent = new Intent(SinglePostActivity.this,
MainActivity.class);
                startActivity(mainintent);
            }
        });


        mDatabase.child(post_key).addValueEventListener(new ValueEventListener() {
            @Override
            public void onDataChange(DataSnapshot dataSnapshot) {
                String post_title = (String) dataSnapshot.child("title").getValue();
                String post_desc = (String) dataSnapshot.child("desc").getValue();
                String post_image = (String) dataSnapshot.child("postImage").getValue();
                String post_uid = (String) dataSnapshot.child("uid").getValue();

                singleTitle.setText(post_title);
                singleDesc.setText(post_desc);
                Picasso.with(SinglePostActivity.this).load(post_image).into(singelImage);
                if (mAuth.getCurrentUser().getUid().equals(post_uid)){

                    deleteBtn.setVisibility(View.VISIBLE);
                }
            }

            @Override
            public void onCancelled(DatabaseError databaseError) {

            }
        });
    }
}
```

Run your app and confirm it work appropriately

To Do

Implement the comment button, and also add a share button for the cardview.

Complete the single post activity to get the items for the post I have not included

My final gradle files

Project level

```
/ Top-level build file where you can add configuration options common to all
sub-projects/modules.

buildscript {

    repositories {
        google()
        jcenter()

    }
    dependencies {
        classpath 'com.android.tools.build:gradle:3.6.3'
        classpath 'com.google.gms:google-services:4.3.3'

        // NOTE: Do not place your application dependencies here; they belong
        // in the individual module build.gradle files
    }
}

allprojects {
    repositories {
        google()
        jcenter()

    }
}

task clean(type: Delete) {
    delete rootProject.buildDir
}
```

Module Level

```
apply plugin: 'com.android.application'
apply plugin: 'com.google.gms.google-services'
android {
    compileSdkVersion 29
    buildToolsVersion "29.0.3"

    defaultConfig {
        applicationId "com.mwongela.theattic"
        minSdkVersion 18
        targetSdkVersion 29
        versionCode 1
        versionName "1.0"

        testInstrumentationRunner "androidx.test.runner.AndroidJUnitRunner"
    }

    buildTypes {
        release {
            minifyEnabled false
            proguardFiles getDefaultProguardFile('proguard-android-optimize.txt'),
'proguard-rules.pro'
        }
    }

}

dependencies {
    implementation fileTree(dir: 'libs', include: ['*.jar'])

    implementation 'androidx.appcompat:appcompat:1.1.0'
    implementation 'com.google.android.material:material:1.1.0'
    implementation 'androidx.constraintlayout:constraintlayout:1.1.3'
    implementation 'androidx.navigation:navigation-fragment:2.2.2'
    implementation 'androidx.navigation:navigation-ui:2.2.2'
    testImplementation 'junit:junit:4.12'
    androidTestImplementation 'androidx.test.ext:junit:1.1.1'
    androidTestImplementation 'androidx.test.espresso:espresso-core:3.2.0'

    implementation 'com.google.firebase:firebase-analytics:17.4.3'
    implementation 'com.google.firebase:firebase-auth:19.3.1'
    implementation 'com.google.firebase:firebase-database:19.3.0'
    implementation 'com.google.firebase:firebase-storage:19.1.1'
    implementation 'com.firebaseui:firebase-ui-database:3.1.3'

    implementation 'com.squareup.picasso:picasso:2.5.2'
    implementation 'com.github.bumptech.glide:glide:4.11.0'
}
```