**CSEE 5590**

**Assignment: Lab 4**

**Team Details:**

**Name: Duy Hoang Ho Class ID: 7**

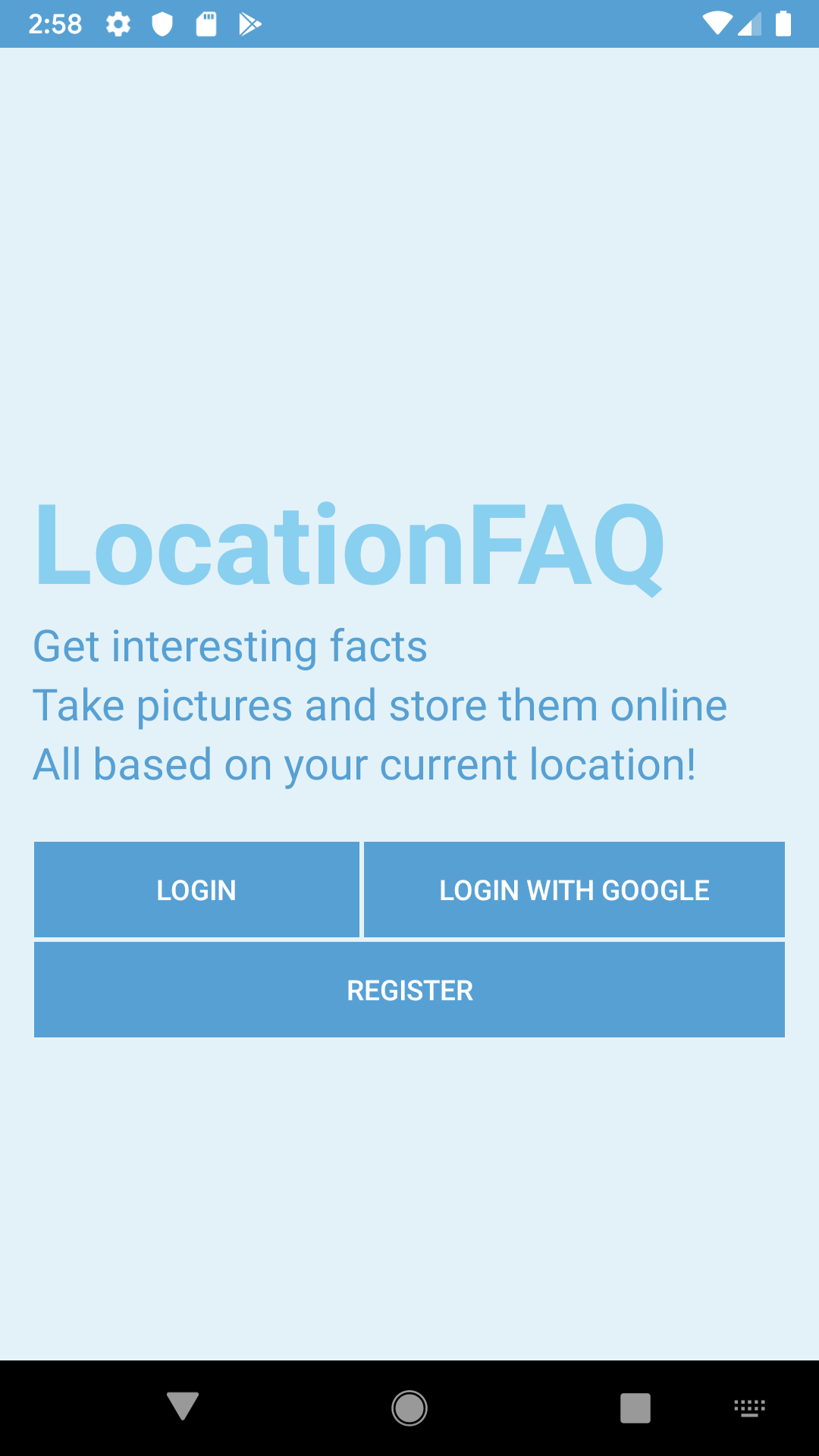
**Name: Tonderai Kambarami Class ID: 8**

**Name: Cameron Knight Class ID: 12**

**Name: Evan Wike Class ID: 32**

**Projects:**

[**Location Expert (Mobile)**](https://github.com/benamreview/CSEE5590-Group1-Labs/wiki/Lab-4-Wiki#location-expert-mobile)



**Demo Video:**

[**https://youtu.be/4COTRTepHkA**](https://youtu.be/4COTRTepHkA)

**Location Expert (Mobile)**

**I. Introduction**

This project is an application that retrieves current logged-in user’s location (in terms of latitude and longitude) and produces relevant information and facts about the current user’s country, such as currency, bordering neighbors, languages, and capital city. Users will be able to check in different countries at different times and take photos of their current location. All of the provided information will be properly stored in the database for more convenient retrieval afterwards.

**II. Objective**

The purpose of this application is to build a basic user authentication system on Android, along with the ability to incorporate several different hardware extensions such as the GPS, the camera, and REST APIs requests. All forms of authentication should be accurate and straightforward and there should be a mechanism for checking whether the credentials such as the username/email address/Google account are valid or not (validation process). If conditions for a proper account registration are not satisfied, the validation process should affirmatively halt the signup process and thereby prevent the user from continuing with the inappropriate credentials. Furthermore, user’s entered information and past checked-in locations are expected to be stored properly in a database in order to be retrieved and displayed whenever the user logs in at another point in time. Along the same line, all the edits of the information should be updated accordingly and accurately reflected once the user is redirected to the main page. As of the location record, they are indexed and ordered by the visited countries. If the user visits any country multiple times, some parameters such as last checked-in time and photos will be modified accordingly (merged or overwritten). This application will contain 4 main features:

* Sign Up/Authenticate a new account via an email address.
* Sign Up/Authenticate an existing Google username.
* Check-in with current location and retrieve facts about it
* Take pictures of current experience with camera.

**III. Approach**

Both authentication mechanisms are implemented with Google Firebase authentication feature (Auth object) but with different options. The first feature needs activating the email/password feature of Firebase authentication while the latter needs the Google Social Login feature instead.

The email/password feature contains the signup and signin methods that will help authenticate and register users conveniently. The Google Social Login will help expedite the sign-up process because users are allowed to proceed with their existing/current Google account without having to go through the registration process again. All the user’s entered information (such as name, age, university, major, and phone number) is stored in the Firestore database, the latest version of Google Firebase’s real-time databases. Before Firestore, our other applications tended towards the initial version of the Firebase database, which was called Realtime database. However, Firestore is more intuitive, object-oriented, and performance-oriented. Getting familiar with Firestore will help in the long run. Records are stored in a document under the user account’s ID, which is uniquely created for every single user, regardless of which login mechanism (regular/social) is activated.

For example, when user [bobsmith@yahoo.com](mailto:bobsmith@yahoo.com) signs up successfully, his account will be given an ID “*f12saZsow0*”, which will then be used as an entry to the database document. Each document will contain JSON-formatted list of objects such as name, age, university, major, and phone number. This mechanism of storing information under user’s ID instead of user’s username will make sure records are indeed unique and generally avoid any faulty behaviors from the false-positive authentication situations.

Location records are stored with a separate collection WITHIN the user record (please refer to the screenshots section below to help clarify the database structure). The collection will contain a history of all the countries visited by user, along with their facts, user’s checked-in location and time, and photographs of the locations (if any).

The user can also upload images to a photo gallery. The images are loaded to firebase storage, with a reference to the image url saved to the database. Images are downloaded an displayed using a GridLayout.

**VI. Screenshots**

Separate Activity for Google Login (a separate pop-up screen with the default google login screen will ask user to sign in with the current credentials)



Main function that’s in charge of signing the user in with Google login. If the authentication is successful, a callback function is then executed -> updateDB(user) where the user is the returned user object.



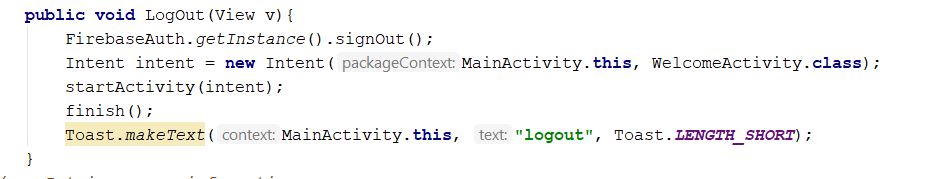
updateDB function that either updates current user information to the document in Firestore, or create a new document in the Firestore if the user just signs in for the first time



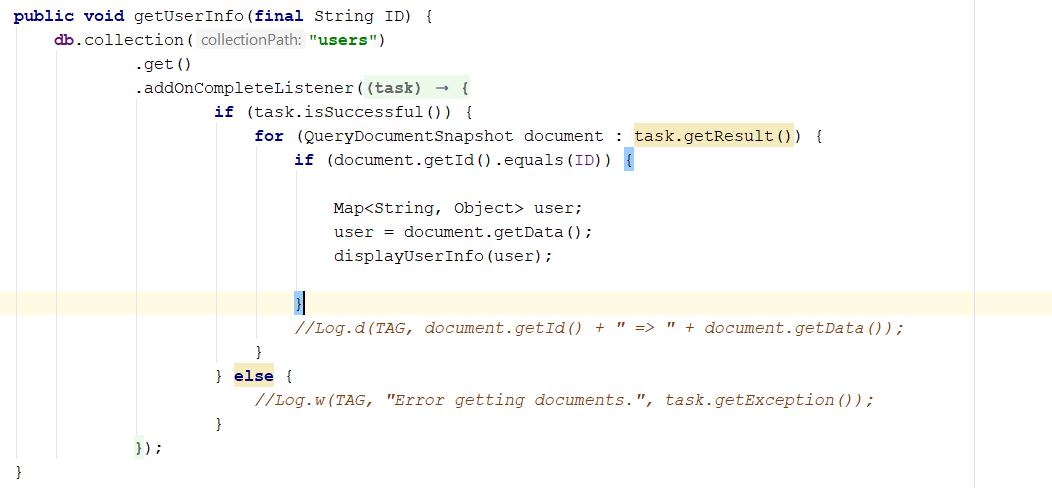
Profile image and User profile display



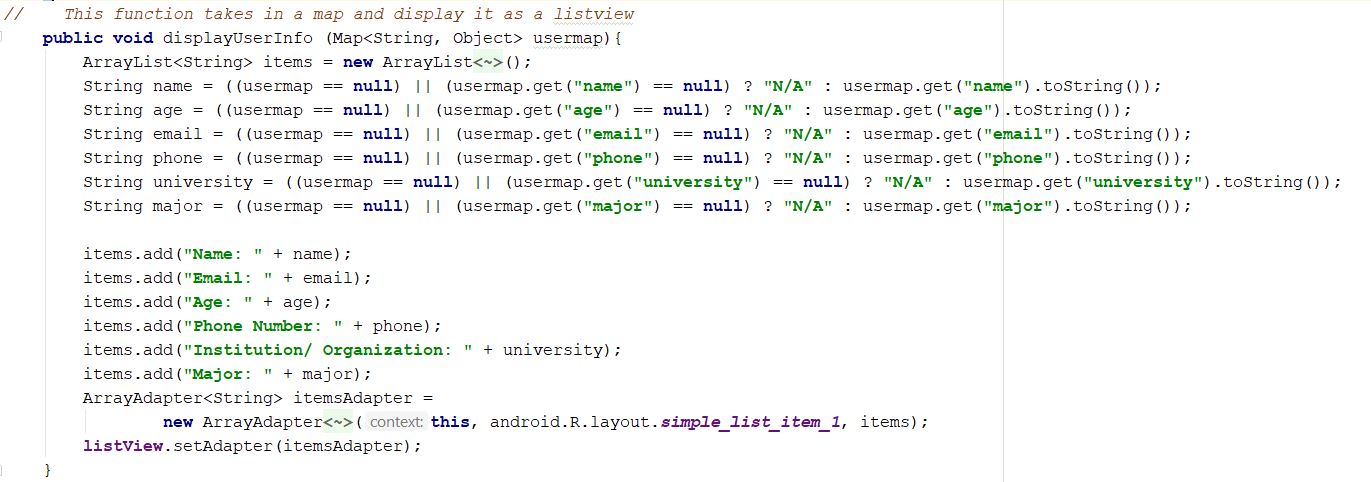
LogOut() that is used in both authentication cases



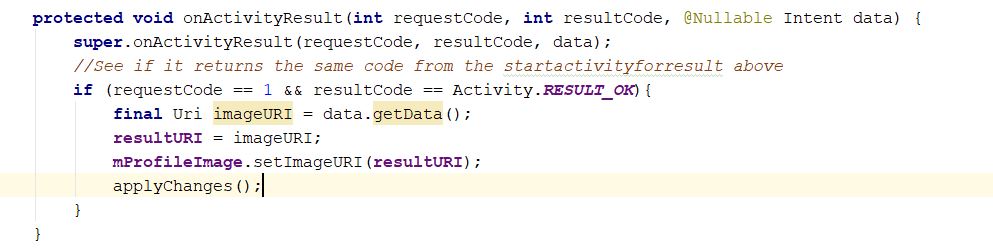
getUserInfo() function will take care of retrieving user object from the Firestore database. If successfully, it will pass the returned object to another function for display and presentation



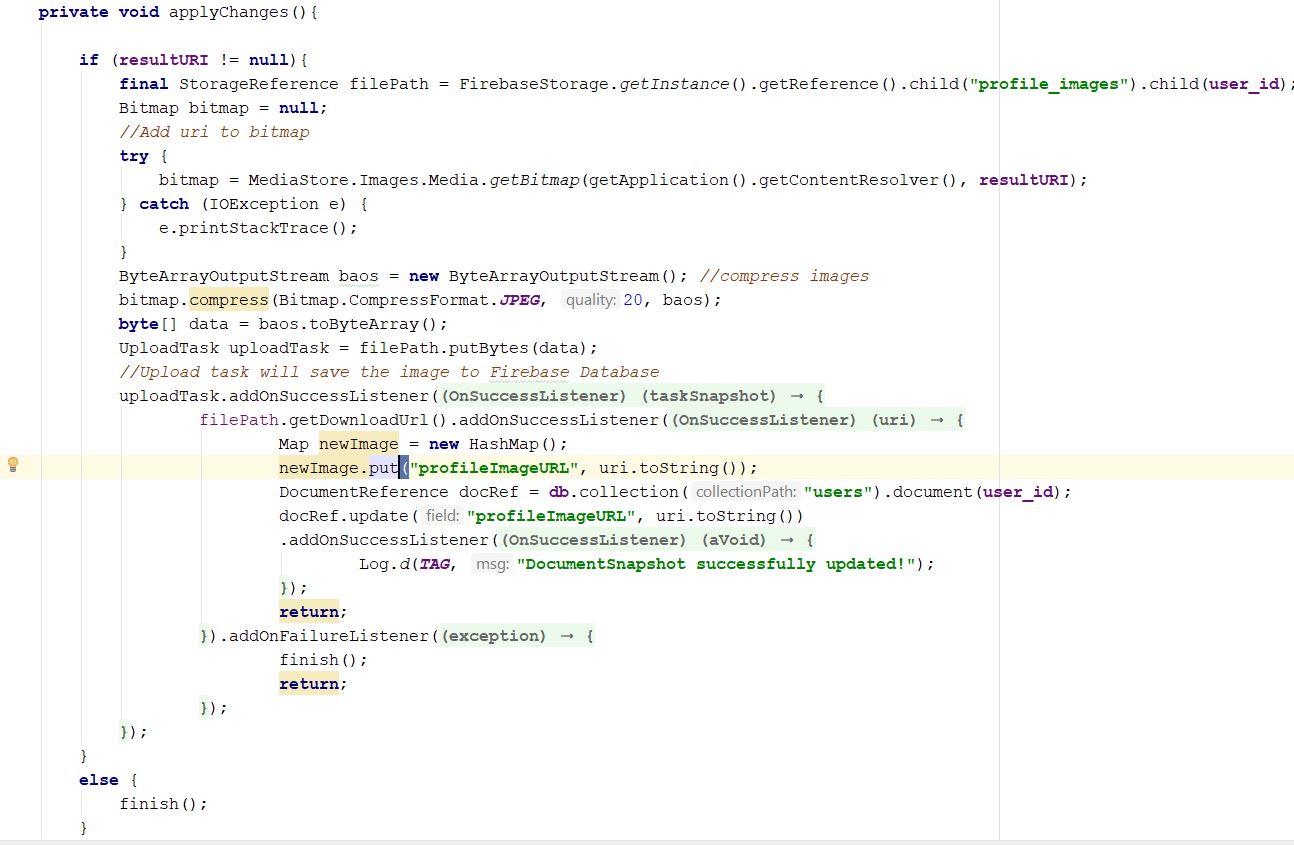
Main function that displays the forwarded user object onto a list view on the front end.



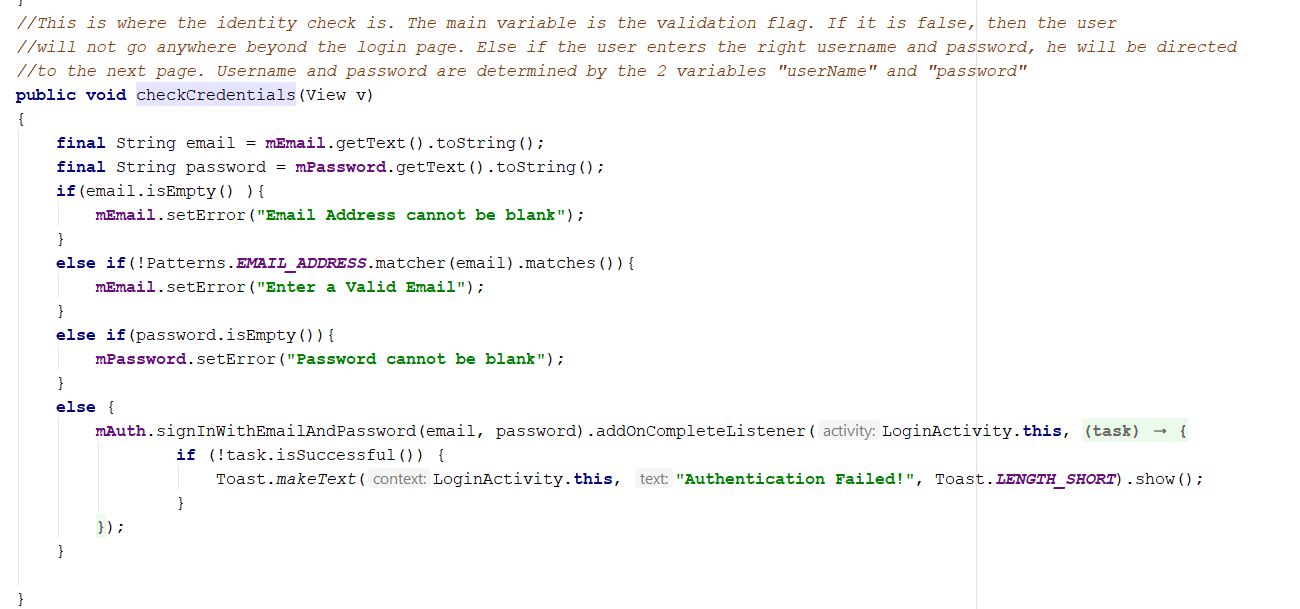
Profile image. If an image is selected from the phone, this will save the URI and upload it to Firebase Storage.



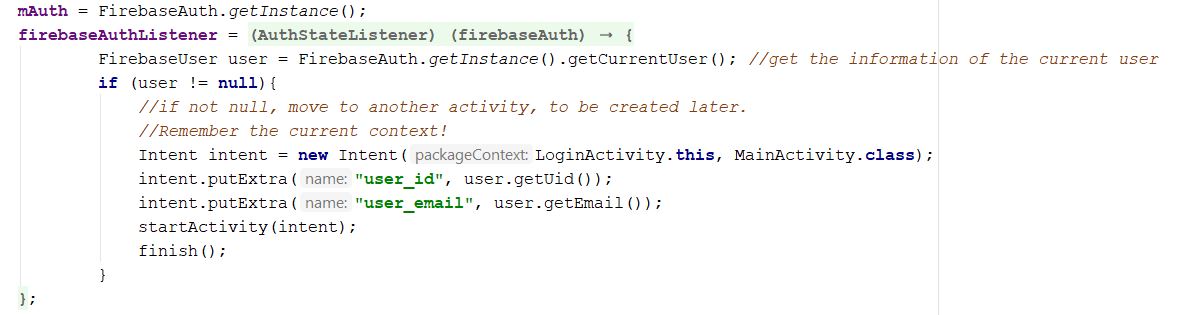
This will process the image, upload it to Firebase storage, and display it in the user’s main page.



This function verifies user’s credentials



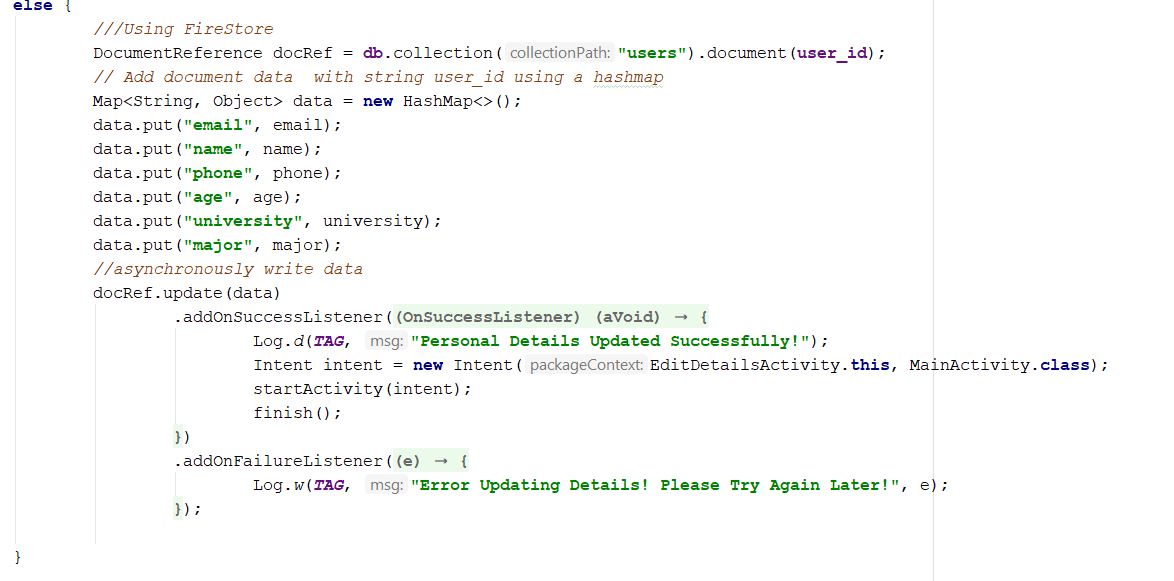
This function listens to any state change (not signed in vs signed in) and redirects accordingly.



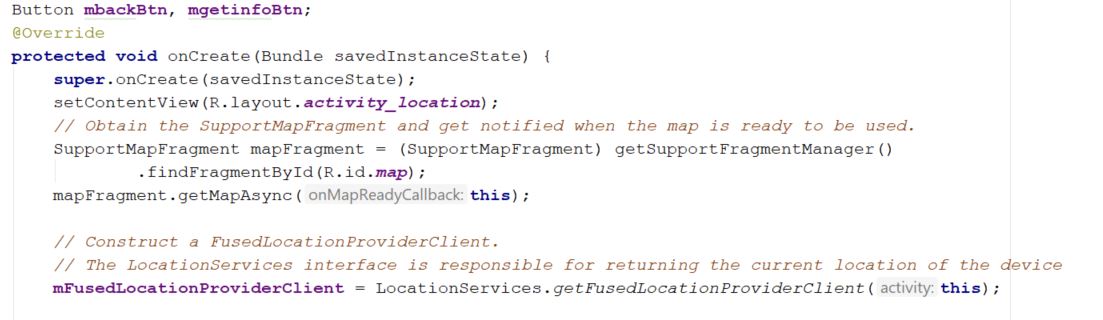
This function (although having the same name as the applychanges for profile image function) is responsible for credential validation process such as username is not in form of an email address, or password is not long enough, there is a blank/unfilled textbox.



This function ensures that the latest user-entered information is updated and reflected properly in the application



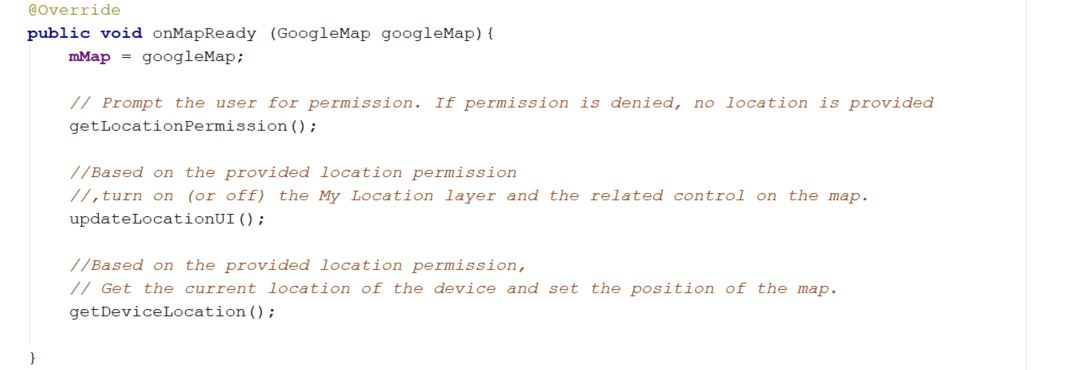
Location fragment required for the Location Activity



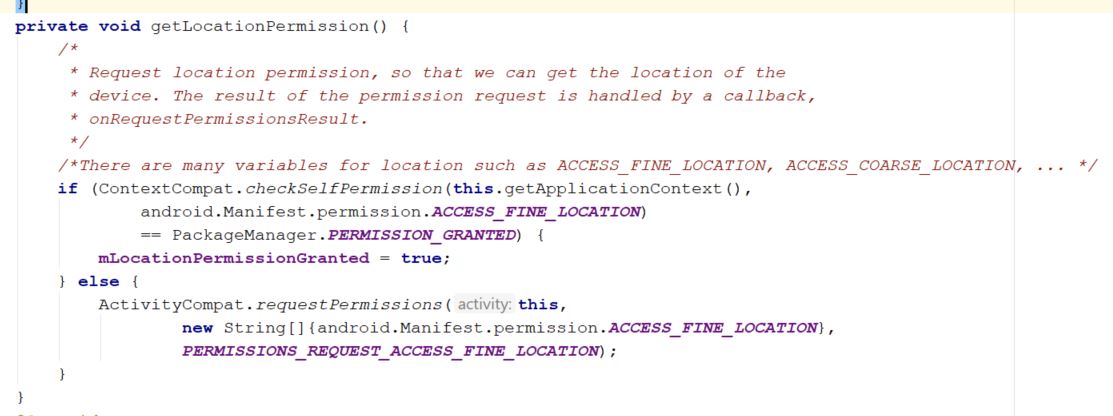
When this getinfo button is clicked, it redirects user to the next activity that displays facts about the current location of the user



High-level functions of the map when it is ready to load. These functions are portrayed in better detail below



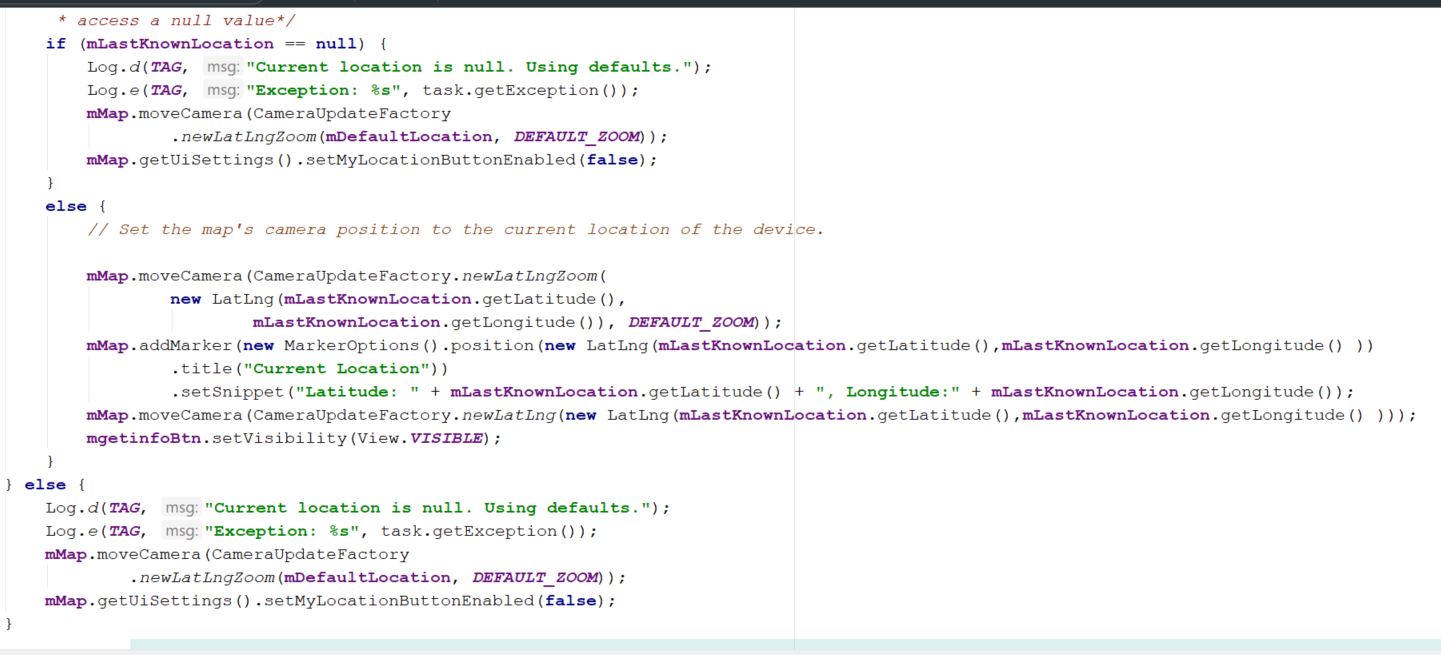
Permission for the map access is important, which determines whether this feature of the application will essentially be able to perform or not at all.



Update the UI location based on the granted (or ungranted) permission. If not granted, map will be set to empty, displaying nothing.



This portion of the getDeviceLocation() is responsible for displaying the recently retrieved latitude and longitude on the map along with a marker.



This code resides on the GetInfoActivity which is responsible for initializing the Firebase authentication object, the Firebase database, retrieving intent-stored variables, along with calling the Rest API request (the API in this case is the Country API), which in turn returns a JSON-formatted object that will then be stored in a JSONObject (of Android) in order to be accessed later by the subsequent functions.



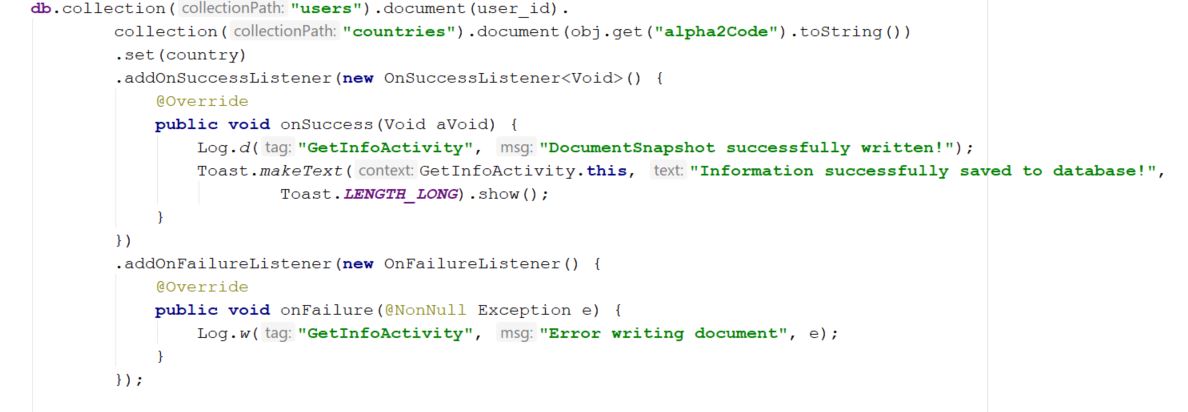
Initializing the array adapter and item list to populate the list view.



This function takes in the JSONobject returned by the API call above and extract all the necessary information to store in the Firebase database.



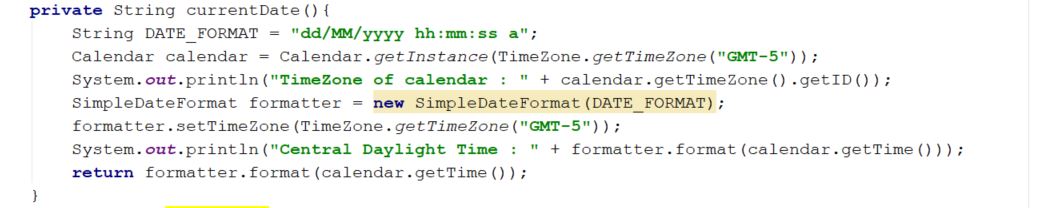
This is where storing and saving the Firebase database instance happens. All of the necessary information have been clumped into the country object will be pushed to the Firebase database.



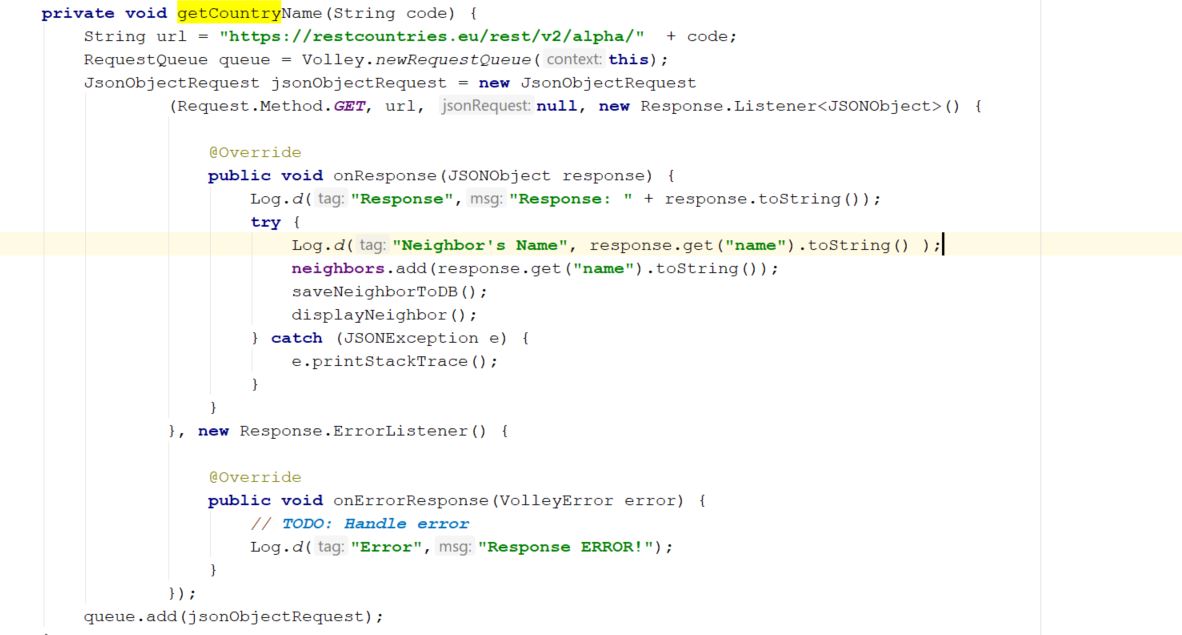
displayInfo is in charge of using the JSONObject from the API and populating the listview of the current activity’s layout



This function is a helper function for styling the current date format as the check-in time of the user. The Time Zone is adjusted to Missouri timezone.



This function converts all the bordering neighbors of the current country to its full name using the API for each of the neighbor’s code. For example, neighbors of the US will be MEX and CAN, which will then be translated to Mexico and Canada via this function and the API calls.



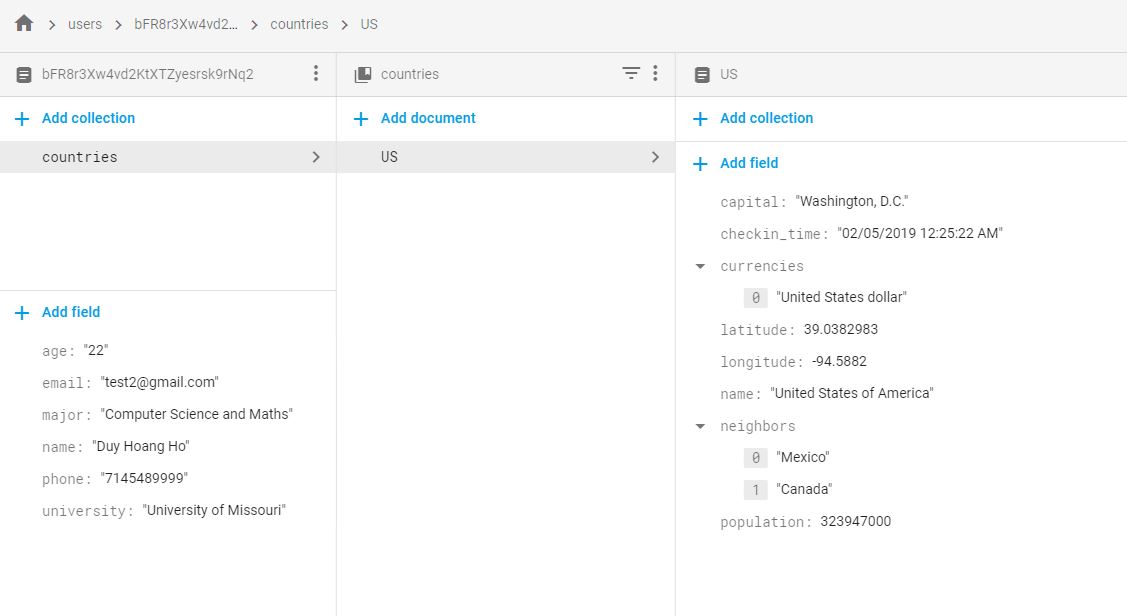
Because the function above is an asynchronous function, it may return its results at a different point in time, having this function ready to be called whenever the asynchronous ones finish would come in handy, especially writing to the database.



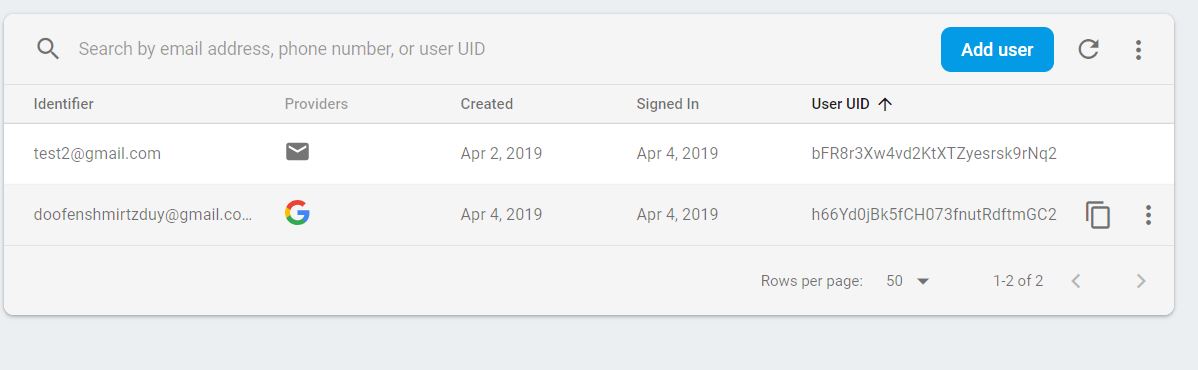
Slightly similar to the above function’s role, this function is called whenever the async function finishes its jobs and passes its result to this function for styling and presentation of the activity’s layout.



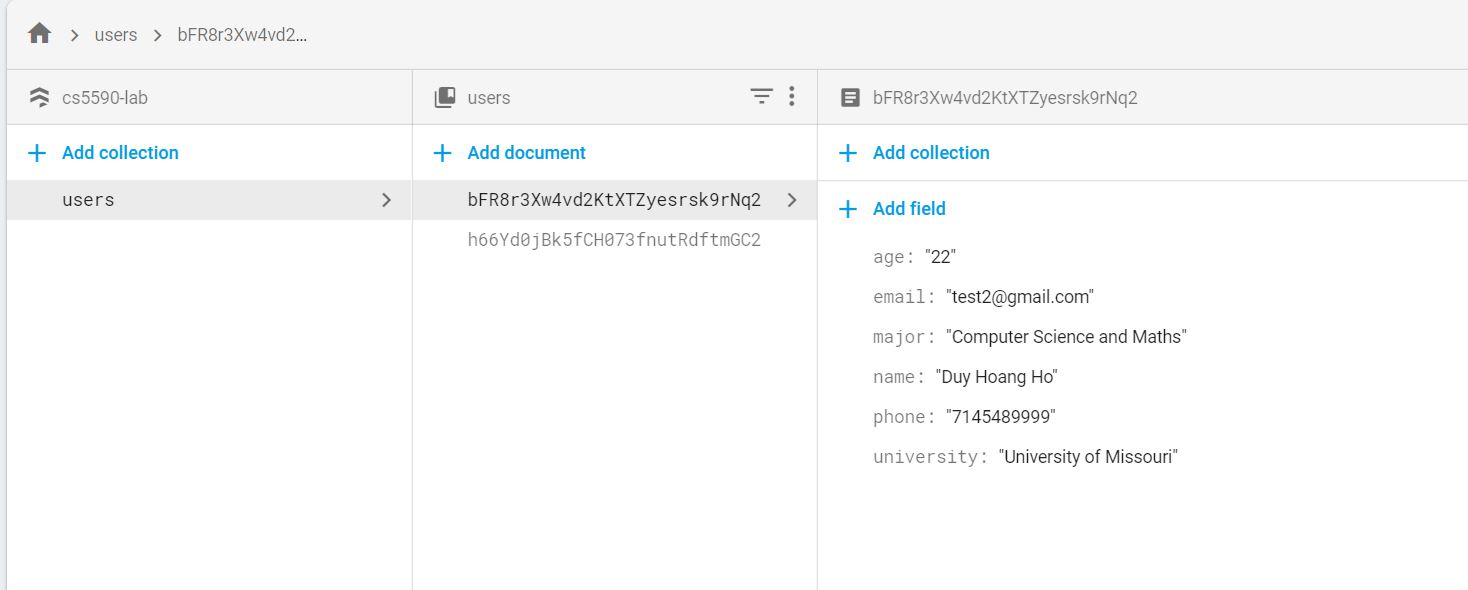
Firebase’s main view of the location collection and documents. Check-in history is stored by names of visited countries.



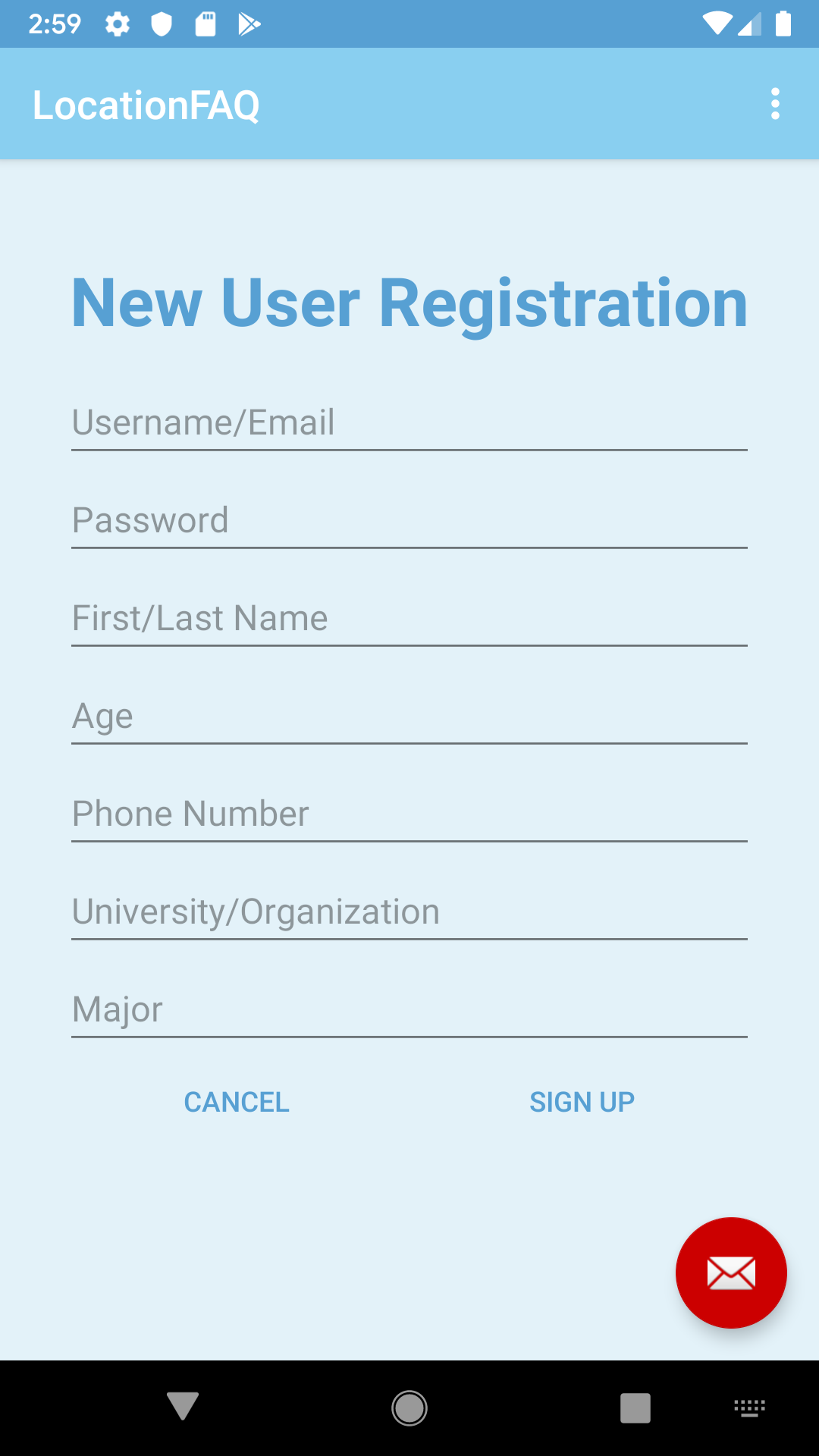
Firebase Authentication Dashboard



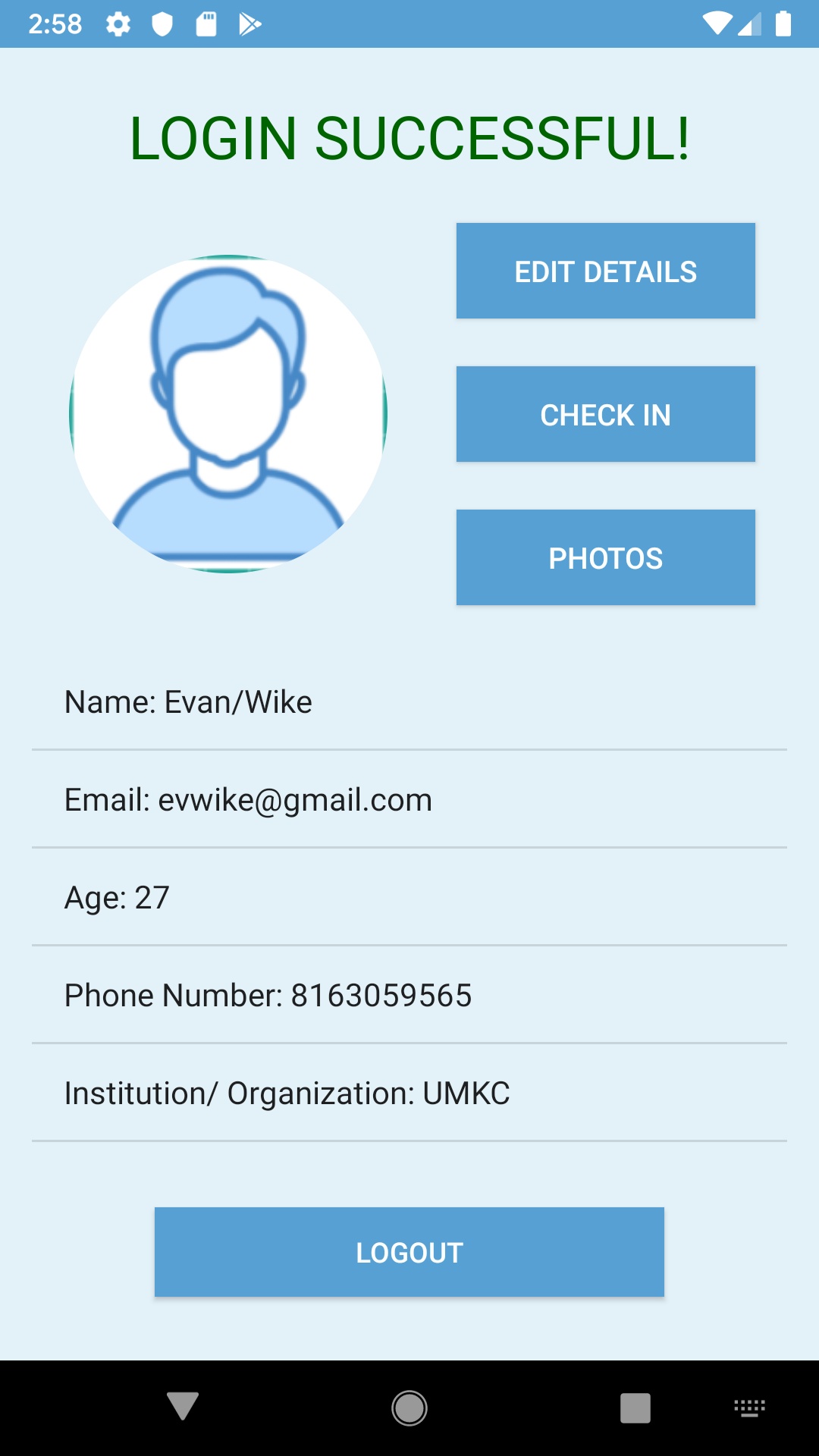
Firestore’s document database



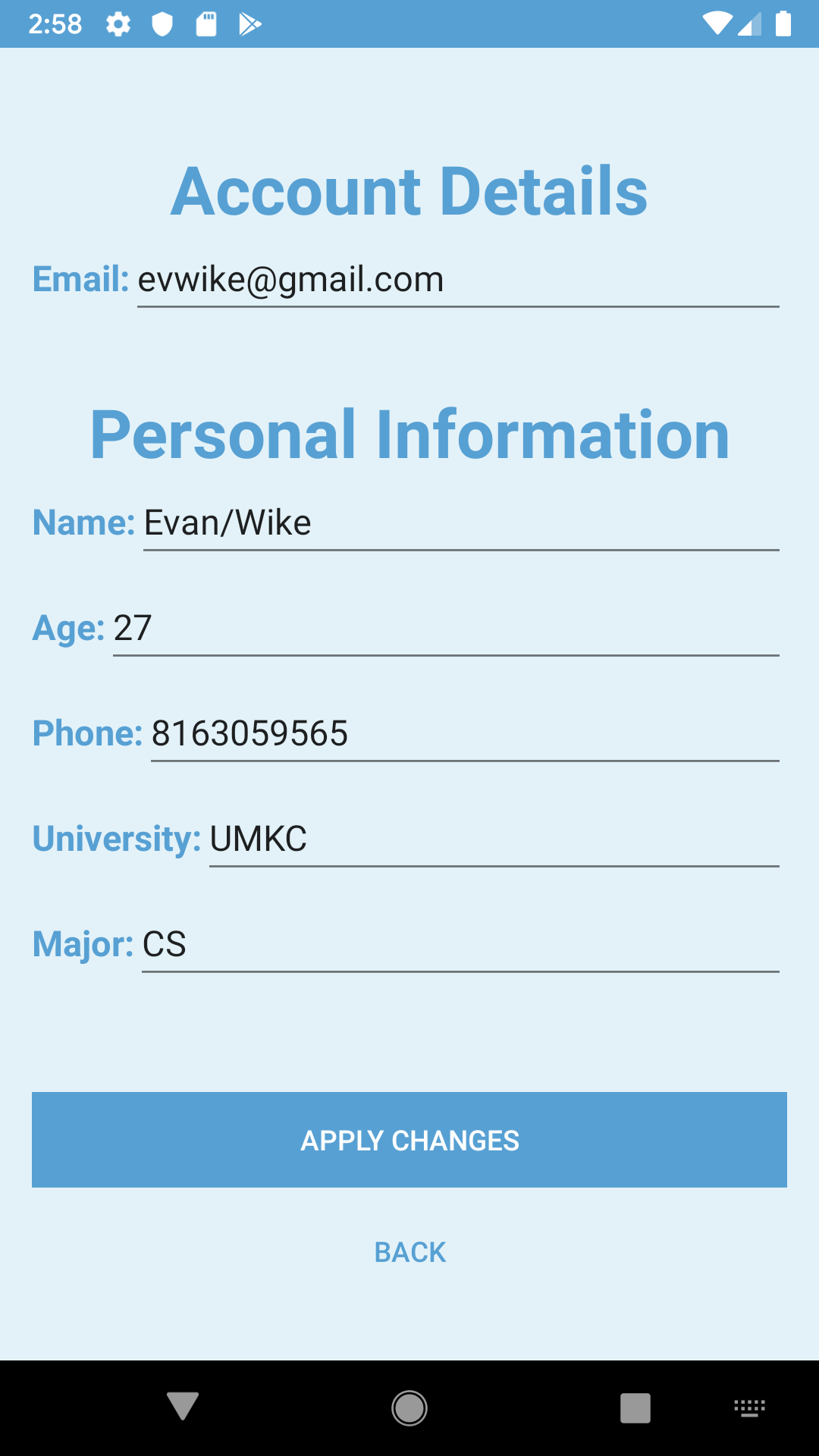
Register



Logged in



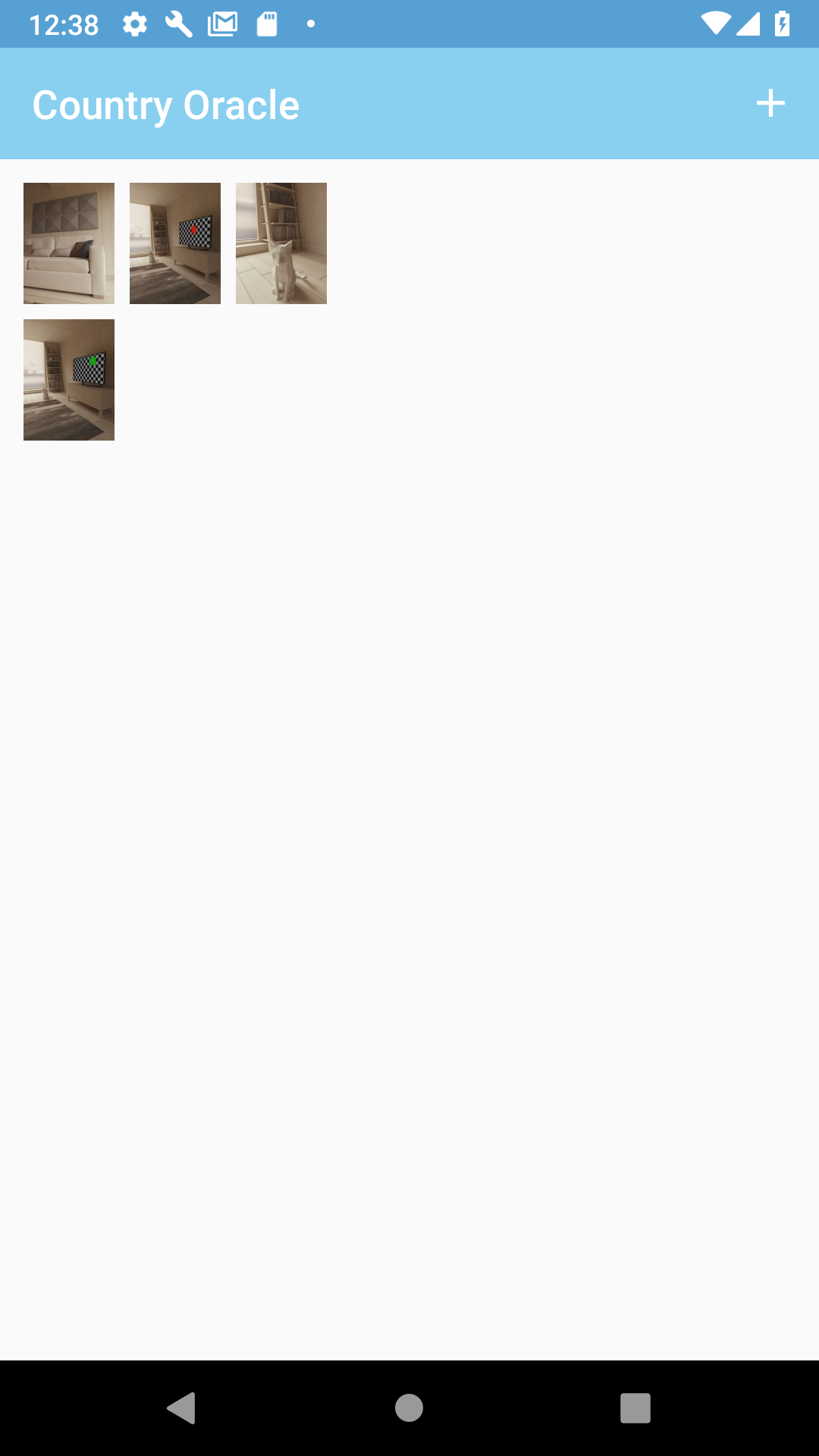
Account Details



Location



Photo gallery



Camera



Images are loaded from the images collection:

mFirebaseFirestore.collection("images").get().addOnCompleteListener(new OnCompleteListener<QuerySnapshot>() {

@Override

public void onComplete(@NonNull Task<QuerySnapshot> task) {

if (task.isSuccessful()) {

for (QueryDocumentSnapshot document : task.getResult()) {

Log.d("PHOTO", document.getString("url"));

downloadImage(document.getString("url"));

}

}

}

});

Helper method to load images:

private void downloadImage(final String url) {

StorageReference reference = mFirebaseStorage.getReferenceFromUrl(url);

final long ONE\_MEGABYTE = 1024 \* 1024;

reference.getBytes(ONE\_MEGABYTE).addOnSuccessListener(new OnSuccessListener<byte[]>() {

@Override

public void onSuccess(byte[] data) {

ImageView imageView = new ImageView(PhotosActivity.this);

Bitmap bitmap = BitmapFactory.decodeByteArray(data, 0, data.length);

imageView.setImageBitmap(bitmap);

mGridLayout.addView(imageView);

// Data for "images/island.jpg" is returns, use this as needed

}

});

}

Upload the image, save the url, and update the gallery:

@Override

public void onActivityResult(int requestCode, int resultCode, Intent data) {

if (requestCode == REQUEST\_IMAGE\_CAPTURE && resultCode == RESULT\_OK) {

final ProgressDialog progressDialog = new ProgressDialog(this);

progressDialog.setTitle("Uploading...");

progressDialog.show();

Bitmap bitmap = (Bitmap) data.getExtras().get("data");

ByteArrayOutputStream baos = new ByteArrayOutputStream();

bitmap.compress(Bitmap.CompressFormat.JPEG, 100, baos);

byte[] bitmapData = baos.toByteArray();

final String uuid = UUID.randomUUID().toString();

final StorageReference ref = mStorageReference.child("images/" + uuid);

ref.putBytes(bitmapData).addOnSuccessListener(new OnSuccessListener<UploadTask.TaskSnapshot>() {

@Override

public void onSuccess(UploadTask.TaskSnapshot taskSnapshot) {

progressDialog.dismiss();

Toast.makeText(PhotosActivity.this, "Uploaded", Toast.LENGTH\_SHORT).show();

}

}).addOnFailureListener(new OnFailureListener() {

@Override

public void onFailure(@NonNull Exception e) {

progressDialog.dismiss();

Toast.makeText(PhotosActivity.this, "Failed " + e.getMessage(), Toast.LENGTH\_SHORT).show();

}

}).addOnProgressListener(new OnProgressListener<UploadTask.TaskSnapshot>() {

@Override

public void onProgress(UploadTask.TaskSnapshot taskSnapshot) {

double progress = (100.0 \* taskSnapshot.getBytesTransferred() / taskSnapshot

.getTotalByteCount());

progressDialog.setMessage("Uploaded " + (int) progress + "%");

}

}).continueWithTask(new Continuation<UploadTask.TaskSnapshot, Task<Uri>>() {

@Override

public Task<Uri> then(@NonNull Task<UploadTask.TaskSnapshot> task) throws Exception {

if (!task.isSuccessful()) {

throw task.getException();

}

// Continue with the task to get the download URL

return ref.getDownloadUrl();

}

}).addOnCompleteListener(new OnCompleteListener<Uri>() {

@Override

public void onComplete(@NonNull Task<Uri> task) {

if (task.isSuccessful()) {

Uri downloadUri = task.getResult();

Map<String, String> doc = new HashMap<>();

String url = downloadUri.toString();

doc.put("url", url);

mFirebaseFirestore.collection("images").add(doc);

downloadImage(url);

} else {

// Handle failures

// ...

}

}

});

}

}

**V. References:**

[Firestore Tutorials](https://firebase.google.com/docs/firestore/quickstart)

[Glide Library for Profile Image](https://github.com/bumptech/glide)

[Rest Country API](https://www.programmableweb.com/api/rest-countries)