ECE 1778 – Creative Applications for Mobile Devices

January 2019

Programming Assignment P2

Advanced UI, Authentication, Database, File Storage

The goal of this assignment is to learn how to build more complex user interfaces and to use Google's Firebase system to authenticate users, and store use data in a backend server without having to manually manage any servers. The first sections of this document give you the specification of the how the software should behave. The later sections guide you through the learning and use of Firebase. **Please note this assignment is a significant amount of work and we strongly suggest that you begin working on it as soon as possible.**

1 Assignment Specification

NOTE: before starting this assignment, please go back and read 'Braiden Brousseau's Guide To Quality Apps' that was part of Assignment P1. Your assignment should obey those guidelines, as part of the grade will be assigned for fulfilling those requirements.

This assignment will build upon Assignment P1 in two major ways: first, it will integrate Google's Firebase Authentication system to make the user login/registration that you built in P1 to actually manage user accounts in a persistent way. Secondly, it will add a new feature that allows users to take, post, and view photos. These photos will be saved to the mobile backend (implemented using Firebase) and will be displayed in an Instagram-like "feed."

1.1 Application Feature Specification

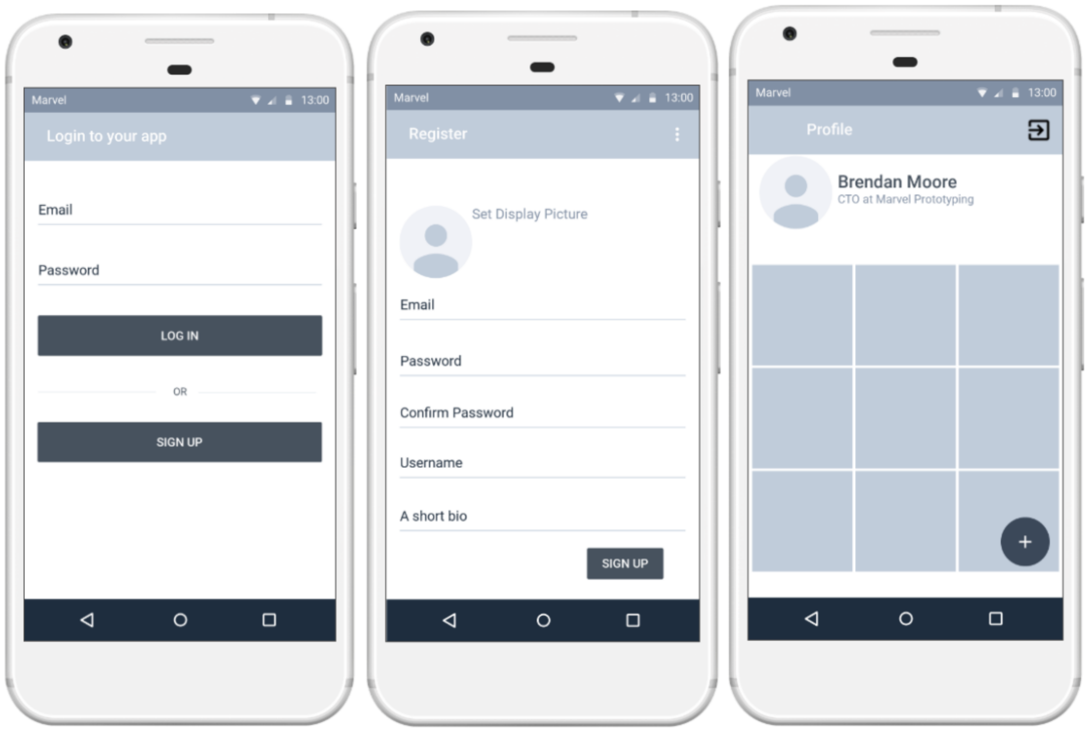
The PhotoSharing app should have the following features, building on those created in assignment P1:

1. Sign-in Page: taken from P1 and **expanded**
    a. The App should launch to this page if a user is not logged in; otherwise it should launch to the Profile page.
    b. The "Log In" button should attempt to authenticate a user's given email and password. If login succeeds, the "Profile Page" should be loaded, but if it fails, it should give an appropriate message on the sign-in page.
    c. The "Sign Up" button should take you to the "Registration Page" as in P1.
2. Registration Page: taken from P1 and **expanded**

    a. Prompts the user for all the same data as in P1 (display picture, email, password, username, bio). A Firebase User should be created for the given email and password (using Firebase Authentication) .
    b. The display picture taken by the user should be saved in Firebase Storage.
    c. The user’s username, bio, and a reference to the location of the display picture in storage (a URL or a path within the storage bucket) should be saved as a document within Firebase Cloud Firestore. All of these documents should be saved within a collection called ‘users’ in the Firestore database.
    d. Once the above steps are completed, the user should be taken to the “Profile Page”

3. Profile Page: take from P1 and **expanded**
    a. The top of the page still displays the user’s display picture, user name, and bio, as in P1
    b. **New**: Beneath the display pic, name, and bio, the profile page will display a scrolling list, 3 columns wide, of thumbnails of all of photos that have been ‘posted’ to the service by the currently logged in user. Clicking on a thumbnail displays photo in full screen. Photos should be listed in chronological order, from newest to oldest.
    c. **New**: Somewhere on the Profile Page there should be a button to launch the camera so that a photo can be taken. Once taken, the picture should be shown to the user and confirmation requested. Once confirmed, the photo should be “posted”. Posting a photo saves the photo in Firebase Storage. A reference to the photo’s location in Firebase Storage should be saved in the Firestore database.
    d. **New**: Somewhere on the Profile Page there should be a button to log out the user and return them to the Sign-in Page.

An example of the Sign-In, Registration, and Profile Pages are shown below. Note that these are only suggestions for how you may choose to lay out and style the app features/elements specified above.



1.2 Additional Application Behavior

In addition to all the behavior specified above, the application should *also* behave in the following ways:

1. Uninstalling and reinstalling the app (or clearing the app's data) should not delete any data from the cloud storage "backend". If a user were to delete and reinstall the app and then log back in to their account, the app should display all their original data on the Profile page (i.e., display picture, username, bio, and photos). This will be tested.
2. The app should support multiple users. For example, a user should be able to register an account and upload photos to that account ("account 1"). Afterwards, if they were to logout and register a new account ("account 2"), account 1's data should still persist and should be accessible if the user logs out of account 2 and logs back in to account 1. This will be tested.
3. The profile page for a particular user should only show photos belonging to that user, and not all the photos belonging to all users in storage/database. Therefore, all photo references in the database should also store the associated user's unique identifier (UID, see `FirebaseUser.getUid()` in the Firebase Authentication Reference). In addition to the UID you should also store the timestamp of the photo so that images can be shown in the correct chronological order.

2 References and Documentation

To begin integrating Firebase into your app and learning how to use its features, please read through the links below.

2.1 Setting up Firebase dependencies

Begin by adding the basic Firebase setup to your app, according to the following guides:

Android: https://firebase.google.com/docs/android/setup
iOS: https://firebase.google.com/docs/ios/setup

Next, add Firebase **Authentication** can be added according to the following guides:

Android: https://firebase.google.com/docs/auth/android/start
iOS: https://firebase.google.com/docs/auth/ios/start

Next, add Firebase **Firestore** can be added by according to the following guide:

Android & iOS: https://firebase.google.com/docs/firestore/quickstart

Finally, add Firebase **Storage** can be added according to the following guides:

Android: https://firebase.google.com/docs/storage/android/start
iOS: https://firebase.google.com/docs/storage/ios/start

2.2 Firebase Sample Apps

Google provides very helpful sample apps complete with source code and written guides which demonstrate how to use all the necessary features of Authentication, Firestore, and Storage which you will need to complete the labs. We provide links to these sample apps below.

Authentication – user ID and password authentication
Android: https://github.com/firebase/quickstart-android/tree/master/auth
iOS: https://github.com/firebase/quickstart-ios/tree/master/authentication

Firestore – storage of information in Google's cloud servers
Android: https://github.com/firebase/quickstart-android/tree/master/firestore
iOS: https://github.com/firebase/quickstart-ios/tree/master/firestore

Storage – larger file storage in the cloud.
Android: https://github.com/firebase/quickstart-android/tree/master/storage
iOS: https://github.com/firebase/quickstart-ios/tree/master/storage

3 Important Notes

This section provides some clarification on the specification and also gives some important information which will save you from common pitfalls. Also, you can see a video of one possible implementation here.

3.1 Note on Storing Images

A production-quality app would store images in multiple resolutions and cache local copies of images as they were downloaded. For example, a 100 x 100 pixel image would be used for thumbnails and the original resolution when greater detail is requested. Doing this is an optimization *outside the scope* of these assignments. We recommend not storing the highest resolution images your camera provides but instead to downscale them to 1024 x 1024 pixel resolution. Make sure to also encode the images into a compressed format, such as **jpg** or **png**.

This is necessary because if you store everything as super-high resolution images and then load them 10 or more at a time on the profile screen you may run into a problem: during a long day of development you might get close to the daily bandwidth limits for the Firebase storage *free* tier. Downscaling the images before uploading will help keep your usage within the free tier limits.

3.2 Note on Firebase Firestore and Storage

While reading the app specification you may have been wondering why it is necessary to both save images in Storage **and** save a reference to the image's location within Storage into the Firestore database. This is because the Storage API doesn't allow you to simply list all the files that exist there – in order to get a reference to a file in Storage you need to know where it is beforehand. The workaround here is to maintain a piece of information (the file path) in the database for each image you upload to storage. Then, when you want to download or delete images from storage, you simply query the database for all image paths and then use those to get StorageReferences. Once you have the StorageReference for an image file, you can download or delete that image from Storage.

3.3 Suggested Organization for Data in Firestore Database

First, familiarize yourself with Firestore's data model, here:

https://firebase.google.com/docs/firestore/data-model

There are many ways to organize your data, and you're free to choose your own organization. One way that you can organize your data in Firestore is to maintain two collections: a collection of "user" documents called "users", and a collection of photo references called "photos".

The “users” collection would be a flat collection of “user” documents, where each “user” document uses the associated user’s UID as its key, and saves the associated username, bio, and reference to display picture as data:

```
users
    <UID of user 1>
    username: "AdaLovelace"
    bio: "First Programmer Ever"
    displayPicPath: <UID of user
    1>/displayPic.jpg
    <UID of user 2>
    name: "AlanTuring"
    bio: "Cryptography Legend"
    displayPicPath: <UID of user 2>/displayPic.jpg
```

The “photos” collection is similar to “users”. Each “photo” document has the UID of the owner’s Firebase User account, a reference to the location of the actual picture in Firebase storage, and the timestamp for the photo. Note that the key for the “photo” documents can be auto generated by Firestore in this case, by calling `add()` instead of `set()` when adding these photo documents to the photos collection. A visualization of this “photos” collection is provided:

```
photos
    <Key of photo 1>
    uid : <UID of owner>
    storageRef : <UID of owner>/1546281942.jpg
    timestamp : 1546281942
    <Key of photo 2>
    uid: <UID of owner>
    storageRef : <UId of owner>/1546282040.jpg
    timestamp : 1546282040
```

Finally, if you were to use this layout for the Firestore database, you would need to organize your Storage bucket into folders for each user. (The name of the folder would be the user’s UID). Inside each folder, there would be one file called “displayPic.jpg”, which would be that user’s Display Picture, and all other photos just need a unique name. In this example given, we use the timestamp since it is highly unlikely that a particular user will take two photos at the exact same time. Note that this makes the “timestamp” field within the “photo” document redundant, since you can just parse the timestamp out of the storageRef field.

3.4 Tip for building the Profile Page UI

We strongly suggest that you create the grid of image thumbnails within the Profile page using a RecyclerView with a GridLayoutManager (the equivalent in iOS is a UICollectionView). In a later assignment you will be asked to create another photo feed with a slightly different layout, and doing so will be much easier if you can reuse your RecyclerView + GridLayoutManager with slightly different parameters. For android users, avoid putting your RecyclerView inside a NestedScollView, doing so will cause every element of the list to be rendered even when not being displayed on the screen. This can cause dramatic performance issues as the size of the list grows, especially when loading images.

Note that it is perfectly OK for the view element at the top of the profile page (which contains the user's display picture, username, and bio) to stay fixed at the top of the screen and not move as you scroll through the grid of photos. This behavior is not consistent with Instagram but is fine for the purpose of these assignments.

4 Grading and Submitting

Important Note on Marking:

8/10 marks are assigned to meeting the above specifications.
2/10 will be based on the quality of the User Interface and User Experience.

Please refer to Braiden Brousseau's guide to quality apps to understand the guidelines for good quality user interface and user experience.

Due: Tuesday February 5th, at 6pm, marked out of 10, 0.5 marks off every hour late.

Submit your zip file through the **Quercus Assignment P2** page in this course**.**

1. Android developers: a zip file containing your final Android application file (.apk); use your student number as the filename. Also submit the complete Android Studio project directory in a separate zip file.

2. iOS developers: you must submit the complete project directory, including source, in a zip file. Use your student number as the filename. Please do your development on the 10.1 version of the SDK, and make sure that you haven't included any files by reference.

Please test your submitted zip file before submission.