CMU 04-801: Advanced Mobile Application Development
Week 7: Firebase Authentication

Firebase Authentication

https://firebase.google.com/docs/auth
Firebase provides sign-in flows for email/password, email link, phone authentication, Google Sign-In, Facebook Login, Twitter Login, and GitHub Login.
To sign a user in:

1. Get authentication credentials from the user
2. Pass these credentials to Firebase Authentication
3. Firebase will verify the credentials and return a response to the client

FirebaseUI Auth provides a drop-in auth solution that handles the UI flows for signing in users with all the methods.

Firebase Console
In your Firebase console go into Authentication and in the Sign-in tab enable whichever provider you'll be using.
Confirm that in your project's settings you have your app hooked up to your database.
You'll also need to add the SHA1 fingerprint of your app to your Project Settings.
https://developers.google.com/android/guides/client-auth
Open a terminal and run the keytool utility provided with Java to get the SHA-1 fingerprint of the certificate.
Mac:
keytool -list -v -alias androiddebugkey -keystore ~/.android/debug.keystore
Windows:
keytool -list -v -alias androiddebugkey –keystore %USERPROFILE%\.android\debug.keystore

The default password for the debug keystore is android
Then the fingerprint is printed to the terminal. Copy the SHA1 fingerprint.

If you don't have the JDK installed, or it can't be found in your path, you'll get an error as keytool is part of the JDK(can be found in the bin directory). Either install it or add it to your path. The JDK is embedded in Android Studio but it might not be added to your path. Downloading it might be the simplest way to deal with this error.

In Firebase go into your project and click on the gear and go to Project Settings. Scroll down and add a fingerprint. The debug fingerprint is enough to get this working.

Security Rules
https://firebase.google.com/docs/firestore/security/get-started?authuser=0
Firestore lets you define the security rules for the collections and documents in your database.

https://firebase.google.com/docs/firestore/security/rules-structure?authuser=0
Firestore security rules always begin with the following declaration:

```
service cloud.firestore {
  match /databases/{database}/documents {
    // ...
```

```
  }
}
```
Basic rules consist of a `match` statement specifying a document path and an `allow` expression detailing when reading the specified data is allowed:
The `match /databases/{database}/documents` declaration specifies that rules should match any Cloud Firestore database in the project. Currently each project has only a single database named `(default)`.
For all documents in all collections:
```
match /{document=**}
```

All match statements should point to documents, not collections. A match statement can point to a specific document, as in `match /cities/SF` or use the wildcard {} to point to any document in the specified path, as in `match /cities/{city}`.

https://firebase.google.com/docs/firestore/security/rules-conditions?authuser=0
You can set up conditions for your security rules. A condition is a boolean expression that determines whether a particular operation should be allowed or denied. Use security rules for conditions that check user authentication, validate incoming data, or access other parts of your database.
Allow statements let you target your rules for read, write, delete, etc.
This rule allows authenticated users to read and write all documents in the cities collection:
```
service cloud.firestore {
  match /databases/{database}/documents {
    match /cities/{city} {
      allow read, write: if request.auth.uid != null;
    }
  }
}
```

When we set our database up in test mode it opened read and write access open to the public for all documents in our database.
Now that we're going to want to use authentication let's set up our security rules so a user must be authenticated to write to the database. We'll continue to allow public access to read from the database.
```
service cloud.firestore {
  match /databases/{database}/documents {
    match /{document=**} {
      allow read;
      allow write: if request.auth.uid != null;
    }
  }
}
```

You must Publish your rules to save the changes.
You can test your Firestore security rules in the console. In the database rules tab there is a simulator you can use to test different types of actions on different parts of your database with different authentication rules.

Note that once you've changed the rules if you run your app without implementing authentication you won't see any data.

Google Authentication
https://developers.google.com/identity/sign-in/android/sign-in
To implement authentication there's some settings you must change first.

Add the Firebase auth SDKs to your app by adding the following dependencies to your app Gradle file.

```
implementation 'com.google.firebase:firebase-auth:19.2.0'
implementation 'com.google.android.gms:play-services-auth:17.0.0'
```

Recipes app
I'm adding Google authentication to the Recipes app (Recipes auth).
Add the Firebase auth SDKs to your app by adding the following dependencies to your app Gradle file.

```
implementation 'com.google.firebase:firebase-auth:19.2.0'
implementation 'com.google.android.gms:play-services-auth:17.0.0'
```

Now we're ready to implement authentication.

Create a new empty activity called GoogleGoogleSignInActivity
Check Generate Layout File
Check Launcher Activity

Since we just created a launcher activity if you go into the AndroidManifest file you'll see we now have two activities with the category launcher. If you run it this way two launcher icons will get created so for MainActivity you should remove the category and action.

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

I added an image to res/drawable that I'll use on my sign-in page.
In the activity_google_sign_in layout file add an ImageView and set the resource to the image you just added. Add missing constraints.
Add a google sign-in button in the XML (I couldn't figure out how to do this in design mode).
https://developers.google.com/identity/sign-in/android/sign-in#add_the_google_sign-in_button_to_your_app

```
<com.google.android.gms.common.SignInButton
    android:id="@+id/login_with_google"
    android:layout_width="match_parent"
    android:layout_height="wrap_content" />
```

Change the width if you don't want it the width of the screen. Add needed constraints.

GoogleSignInActivity.java
All authentication logic will be in GoogleSignInActivity.java
Create the class data members we'll need. It doesn't matter what value you give RC_SIGN_IN, we're just using a constant so it's easy to check.

```
private GoogleSignInClient googleSignInClient;
private static final int RC_SIGN_IN = 9001;
```

```
private FirebaseAuth firebaseAuth;
private SignInButton signInButton;
```

Update onCreate() to configure Google sign-in.

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

    //get the shared instance of the FirebaseAuth object
    firebaseAuth = FirebaseAuth.getInstance();

    //configure Google sign-in
    GoogleSignInOptions options = new
GoogleSignInOptions.Builder(GoogleSignInOptions.DEFAULT_SIGN_IN)
            .requestIdToken(getString(R.string.default_web_client_id))
            .build();

    //creates a new instance of GoogleSignInClient
    googleSignInClient = GoogleSignIn.getClient(this, options);

    signInButton = findViewById(R.id.login_with_google);
    signInButton.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            signIn();
        }
    });
}
```

In the GoogleSignInOptions object you can add requestEmail() to request the user's email as well. signIn() will give you an error because we need to create that method.

```
private void signIn() {
    //gets an Intent to start the Google sign-in flow
    Intent signInIntent = googleSignInClient.getSignInIntent();
    //start Google sign-in intent. onActivityResult() will be called with
the requestCode when it's finished
    startActivityForResult(signInIntent, RC_SIGN_IN);
}
```

Starting the intent prompts the user to select a Google account to sign in with. After the user signs in, you can get a GoogleSignInAccount object for the user in the activity's onActivityResult method.

```
//called when Google sign-in finishes
@Override
protected void onActivityResult(int requestCode, int resultCode, Intent
data) {
    super.onActivityResult(requestCode, resultCode, data);

    //result returned from launching the Intent from
GoogleSignInApi.getSignInIntent()
```

```
    if (requestCode == RC_SIGN_IN) {
        //returns a completed Task containing an error or
GoogleSignInAccount object
        Task<GoogleSignInAccount> task =
GoogleSignIn.getSignedInAccountFromIntent(data);
        try {
            //Google Sign In was successful
            //returns Google account
            GoogleSignInAccount account =
task.getResult(ApiException.class);
            firebaseAuthWithGoogle(account);
        } catch (ApiException e) {
            // Google Sign In failed
            Log.e("LOGIN", "Google sign in failed", e);
            Toast.makeText(this, "Login Unsuccessful", Toast.LENGTH_SHORT)
                 .show();
        }
    }
}
```

The GoogleSignInAccount object contains information about the signed-in user, such as the user's name. firebaseAuthWithGoogle() will give you an error because we need to create that method.

```
private void firebaseAuthWithGoogle(GoogleSignInAccount account){
    String idToken = account.getIdToken();
    final String name = account.getDisplayName();

    //get Firebase credential
    AuthCredential credential = GoogleAuthProvider.getCredential(idToken,
null);

    //Firebase authentication
   firebaseAuth.signInWithCredential(credential).addOnCompleteListener(this,
new OnCompleteListener<AuthResult>() {
        @Override
        public void onComplete(@NonNull Task<AuthResult> task) {
            Log.d("SIGNIN", "signInWithCredential:onComplete:" +
task.isSuccessful());
            if (task.isSuccessful()) {
                Toast.makeText(GoogleSignInActivity.this, "Login successful
" + name, Toast.LENGTH_SHORT).show();
                Intent intent = new Intent(GoogleSignInActivity.this,
MainActivity.class);
                startActivity(intent);
            }else {
                Log.e("SIGNIN", "signInWithCredential" +
task.getException());
                Toast.makeText(GoogleSignInActivity.this, "Authentication
failed.", Toast.LENGTH_SHORT).show();
            }
        }
    });
}
```

At this point you should be able to run it, log in, and then go to the main recipes page.

Note that I'm using the newer GoogleSignInClient as outlined in the Android Firebase Google authentication documentation. The older GoogleApiClient is used in many tutorials and will still work. You can read about the differences and why GoogleSignInClient is suggested in this Android Developers blog post https://android-developers.googleblog.com/2017/11/moving-past-googleapiclient_21.html.

It would be nice to check to see if the user is currently signed in when initializing your Activity which you can do in onStart(). I have this commented out in my example, and I suggest you do the same while testing, or once you're logged in you won't see the login activity and won't be able to log in.

```
@Override
protected void onStart() {
    super.onStart();

    //check to see if the user is already logged in
    if(firebaseAuth.getCurrentUser()!=null){
        startActivity(new
Intent(getApplicationContext(),MainActivity.class));
    }
}
```

Logout

There's already an options menu set up in MainActivity so I changed the menu item to be Logout by changing the value in the strings.xml file. I also changed the menu_main.xml file to have the id action_logout and then implemented the onOptionsItemSelected() method to sign out the user currently signed into Firebase. I also call finish() on the activity which destroys it and goes to the previous activity.

```
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    // Handle action bar item clicks here. The action bar will
    // automatically handle clicks on the Home/Up button, so long
    // as you specify a parent activity in AndroidManifest.xml.
    int id = item.getItemId();

    //noinspection SimplifiableIfStatement
    if (id == R.id.action_logout) {
        Toast.makeText(MainActivity.this, "Logged out",
Toast.LENGTH_SHORT).show();
        FirebaseAuth.getInstance().signOut();
        finish();
        return true;
    }
    return super.onOptionsItemSelected(item);
}
```

This is a minimal example to introduce you to Firebase authentication on Android.
Firebase also has an AuthStateListener that's called when there is a change in the authentication state. If different accounts have different data in Firebase you'd need to organize your database differently so recipes are specific to a user.