ATLS 4320: Advanced Mobile Application Development
Week 15: 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. You can use the UI provided as is or customize it.
https://firebase.google.com/docs/auth/android/firebaseui

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
To implement authentication you must add the FirebaseUI auth library to your app by adding the following dependencies to your app Gradle file.

```
implementation 'com.firebaseui:firebase-ui-auth:6.4.0'
```

Recipes app
I'm adding Google authentication to the Recipes app (Recipes auth).
Add the FirebaseUI auth library to your app by adding the dependencies to your app Gradle file.
Your app Gradle file should have these dependencies:

```
implementation platform('com.google.firebase:firebase-bom:27.0.0')
implementation 'com.google.firebase:firebase-firestore-ktx'
implementation 'com.firebaseui:firebase-ui-firestore:7.1.1'
implementation 'com.firebaseui:firebase-ui-auth:6.4.0'
implementation "androidx.lifecycle:lifecycle-viewmodel-ktx:2.3.1"
```

Now we're ready to implement authentication.

Create a new empty activity called GoogleSignInActivity
Uncheck Generate Layout File as FirebaseUI will be handling the UI for authentication
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 intent-filter with the category launcher and action main.

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

We're also going to need some additional strings in strings.xml.

```
<string name="signin_success">Successfully signed in user</string>
<string name="signin_fail">Sign in unsuccessful</string>
<string name="action_logged_out">Logged Out</string>
<string name="action_logout">Logout</string>
<string name="action_login">Login</string>
```

GoogleSignInActivity.kt
All authentication logic will be in GoogleSignInActivity.kt
Create a class level constant that will be used during authentication. It doesn't matter what value you give SIGN_IN_REQUEST_CODE, we're just using a constant so it's easy to check.

```
private val SIGN_IN_REQUEST_CODE = 1969
```

We'll create a method to launch the FirebaseUI sign-in flow.
I disabled SmartLock as it was not enabled on my emulator and gave me errors.

```
fun googleLogIn(){
    // choose authentication providers
    val providers = arrayListOf(
        AuthUI.IdpConfig.GoogleBuilder().build()
        //others would go here
    )

    // Create and launch sign-in intent
    startActivityForResult(
        AuthUI.getInstance()
            .createSignInIntentBuilder()
            .setAvailableProviders(providers)
            .setIsSmartLockEnabled(false)
            .build(),
        SIGN_IN_REQUEST_CODE)
}
```

When the sign-in flow is complete, you will receive the result in onActivityResult().

```
//called when Google sign-in finishes
override fun onActivityResult(requestCode: Int, resultCode: Int, data:
Intent?) {
    super.onActivityResult(requestCode, resultCode, data)

    if (requestCode == SIGN_IN_REQUEST_CODE) {
        val response = IdpResponse.fromResultIntent(data)

        if (resultCode == Activity.RESULT_OK) {
            // Successfully signed in
            val user = FirebaseAuth.getInstance().currentUser
            Toast.makeText(this, "${getString(R.string.signin_success)}
${user?.displayName}", Toast.LENGTH_LONG).show()
            val intent = Intent(this, MainActivity::class.java)
            startActivity(intent)
        } else {
            // Sign in failed. If response is null the user canceled the
            // sign-in flow using the back button. Otherwise check
            // response.getError().getErrorCode() and handle the error.
            Toast.makeText(this, "${getString(R.string.signin_fail)}",
Toast.LENGTH_LONG).show()
        }
    }
}
```

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

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.

```
<string name="action_logout">Logout</string>
```

In menu_main.xml I updated the id for the menu item and the title to use the updated string resource

```
android:id="@+id/action_logout"
android:title="@string/action_logout"
```

In MainActivity update 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 fun onOptionsItemSelected(item: MenuItem): Boolean {
    // 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.
    return when (item.itemId) {
        R.id.action_logout -> {
            AuthUI.getInstance()
                .signOut(this)
                .addOnCompleteListener {
                    Toast.makeText(this,
"${getString(R.string.action_logged_out)}", Toast.LENGTH_LONG).show()
                    finish()
                }
            true
        }
        else -> super.onOptionsItemSelected(item)
    }
}
```

You should be able to now log out and go back to the previous activity which is empty.
It would be nice if the user had the option to log back in.
We can add an options menu to GoogleSignInActivity.

In the menu directory create a new menu resource file.
File name: menu_signin
Root element: menu
Source set: main
Directory name: menu

Update menu_signin

```
<menu xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    tools:context="com.example.recipes.GoogleSignInActivity">
    <item
        android:id="@+id/action_login"
        android:orderInCategory="100"
        android:title="@string/action_login"
        app:showAsAction="never" />
</menu>
```

Then in GoogleSignInActivity we'll implement the methods to add the menu and then handle the menu selection to log in.

```
override fun onCreateOptionsMenu(menu: Menu): Boolean {
    // Inflate the menu; this adds items to the action bar if it is present.
    menuInflater.inflate(R.menu.menu_signin, menu)
    return true
}

override fun onOptionsItemSelected(item: MenuItem): Boolean {
    // 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.
    return when (item.itemId) {
        R.id.action_login -> {
            googleLogIn()
            true
        }
        else -> super.onOptionsItemSelected(item)
    }
}
```

This is a basic 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.