Advanced Mobile Application Development

Recipe App: Firebase Authentication

For the final demo of the semester, we'll be adding user authentication to our recipe app via FirebaseUI and Google Sign In. The process is relatively simple and only requires a few changes to the app which is nice.

Enable Authentication

We'll start by enabling Google sign in via the Firebase console.

Go to Firebase Console and **open the project**. **Go to** the *Authentication* page in the *Develop* section of the sidebar. In the *Sign-In Method* tab, **select** *Google* from the list, **provide** a support email, and **toggle** the *Enable* switch in the top right corner of the card.

And that's all we need to do to enable the authentication!

Add Gradle Dependency

Open the app-level *build.gradle* file and **add this line to the dependencies**:

```
implementation 'com.firebaseui:firebase-ui-auth:6.2.0'
```

Sync Gradle files using the pop-up at the top of the window.

Changes to UI

Before we implement the sign in, we'll make some changes to our UI. We need some sort of "Sign In" and there are a variety of different ways we could do this. We could add a new activity or fragment, set that as the entry point to the app, and then navigate to MainActivity after successful authentication, but I came across an interesting component called a ViewSwitcher that would simplify the process and allow us to use MainActivity as a Sign In Screen AND the host activity for the actual app content.

ViewSwitcher works by having exactly two child views (we'll use ConstraintLayouts) and gives the ability to switch between the two at runtime. This means that we can just make some modifications to *activity_main.xml* and we don't need to create an entirely new Activity or Fragment with its own layout. ViewSwitcher also handles animation.

To start, **add this code** to the top of *activity_main.xml*

```
<ViewSwitcher xmlns:android="http://schemas.android.com/apk/res/android"
  android:id="@+id/signInSwitcher"
  android:layout_width="fill_parent"
```

```
  android:layout_height="fill_parent">

<!--   initial view, no sign in -->
  <androidx.constraintlayout.widget.ConstraintLayout
     xmlns:android="http://schemas.android.com/apk/res/android"
     xmlns:app="http://schemas.android.com/apk/res-auto"
     android:layout_width="match_parent"
     android:layout_height="match_parent">

     <Button
        android:id="@+id/signInButton"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Sign In"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />
  </androidx.constraintlayout.widget.ConstraintLayout>
```

Be sure to close the ViewSwitcher tag after the other constraint layout.

We will now only see our “Sign In” screen in the design view of the xml, but rest assured that the other constraint layout still exists for us to toggle between.

Next, go to *MainActivity.kt* and **create class properties for the ViewSwitcher and new Sign In Button**:

```
private lateinit var viewSwitcher: ViewSwitcher
private lateinit var signInButton: Button
```

Then instantiate them in onCreate() after we setContentView():

```
viewSwitcher = findViewById(R.id.signInSwitcher)
signInButton = findViewById(R.id.signInButton)
```

Implement FirebaseUI

FirebaseUI works by launching an Activity for a result and passing a list of the authentication providers that you are using in your app. Then, we’ll override the *onActivityResult()* method to handle successful/unsuccessful operation after the FirebaseUI activity finishes.

Start by creating a list property of providers in *MainActivity*:

```
private val providers = arrayListOf(AuthUI.IdpConfig.GoogleBuilder().build())
```

Next, **add a click listener to the sign-in button** in the *onCreate()* method of *MainActivity*:

```
signInButton.setOnClickListener {
  // Create and launch sign-in intent
  startActivityForResult(
```

```
      AuthUI.getInstance()
          .createSignInIntentBuilder()
          .setAvailableProviders(providers)
          .build(),
      1
    )
}
```

Then, **add a method** to override *onActivityResult*:

```
override fun onActivityResult(requestCode: Int, resultCode: Int, data: Intent?) {
  super.onActivityResult(requestCode, resultCode, data)

  if (requestCode == 1) {
      val response = IdpResponse.fromResultIntent(data)

      if (resultCode == Activity.RESULT_OK) {
          val user = FirebaseAuth.getInstance().currentUser
          Toast.makeText(this,"Welcome, ${user?.displayName}!", Toast.LENGTH_LONG).show()
          viewSwitcher.showNext()
      } else {
          if(response != null) {
              Log.e(LOG_TAG, response.error?.localizedMessage!!)
              Toast.makeText(this,"Could not complete sign in, try again!", Toast.LENGTH_LONG).show()
          }
      }
  }
}
```

At this point we can **run our app** to see the sign-in screen at work. You'll notice that we still haven't done anything to the database to associate the data with the authenticated user so we'll need to do that next.

Database Modifications

Instead of having a top level "Favorites" collection in our database, we'll use top level collections associated with the authenticated user's id. Everything else should be the same.

Make the following modifications to the FavoritesRepository.kt

Add a class property to get and store the currently authenticated user:

```
private val firebaseUser = FirebaseAuth.getInstance().currentUser
```

Next, in the *init()* method, we need to **verify that the user is authenticated (they should be) and then attach the listener to the collection based on their id**:

```
init {
  if(firebaseUser != null) {
      db.collection(firebaseUser.uid)
```

```
        .addSnapshotListener { snapshot, e ->
            if (e != null) {
                Log.e(LOG_TAG, "Listen failed.", e)
                return@addSnapshotListener
            }
            if (snapshot != null) {
                parseAllData(snapshot)
            } else {
                Log.d(LOG_TAG, "Current data: null")
            }

        }
    }
}
```

Next, **we need to verify authentication and add the appropriate collection based on ID** in the *addFavorite* method:

```
fun addFavorite(recipe: RecipeDetails) {
    if(firebaseUser != null) {
        val recipeMap = recipeDetailsToHashMap(recipe)

        db.collection(firebaseUser.uid).document(recipe.id.toString())
            .set(recipeMap)
            .addOnSuccessListener {
                Log.i(LOG_TAG, "Added favorite success!")
            }
            .addOnFailureListener { exception ->
                Log.w(LOG_TAG, "Error adding document.", exception)
            }
    }
}
```

And we'll have to do the same when we remove:

```
fun removeRecipeFromFavorites(id: Int) {
    if(firebaseUser != null) {
        db.collection(firebaseUser.uid).document(id.toString()).delete()
    }
}
```

And that's it! **Run the app** to verify that the user's data is tied to the account and **check the Firebase console** to make sure that everything looks right there.

There's a lot more that you can do with authentication like profile management and such, but that's somewhat outside the scope of this class and it's well documented in the FirebaseUI Docs so feel free to check that out if you want to know more there.