ATLS 4320: Advanced Mobile Application Development
Week 9: iOS and Firebase Authentication

Many apps want to identify users. Knowing a user's identity lets you authenticate users, save custom data, and provide the same personalized experience across all of the user's devices.

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. We're going to use FirebaseUI to implement authentication using Google.

Google sign-in
https://firebase.google.com/docs/auth/ios/firebaseui
In the Firebase console go into Authentication Sign-in Method and enable Google.

For our Xcode project this requires the following pods in the Podfile (alternatively you can use the FirebaseUI pod and that includes all the components):

```
pod 'FirebaseUI/Auth'
pod 'FirebaseUI/Google'
```

Then you need to do pod install again (needed every time you add new pods).

I ran into pod dependency issues where the FirebaseFirestoreSwift pod was causing the other pods to be updated to 6.34 and then I got the error "no visible @interface for firauth declares the selector 'use emulator with host:port:". The fix was to specify the version of FirebaseFirestoreSwift in the podfile.

```
pod 'FirebaseFirestoreSwift', '~> 7.0-beta'
pod update
```

Check the version of your pods and if they're 6.34 do a pod update to get to 7.7.

```
cat Podfile.lock |grep Firebase/
```

You will get warning re deprecated methods in some of the Firebase Auth frameworks and you can ignore these.
If you get the warning about converting to Swift 5 you can go ahead and do this (takes a while).

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.

Go into Cloud Firestore | Rules to change the rules.

```
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.

Google Authentication
Back in your app go into the GoogleService-Info.plist configuration file and look for the `REVERSED_CLIENT_ID` key. Copy the value of that key.
Go into your target's Info tab and expand the URL Types section and add a new URL scheme and paste the REVERSED_CLIENT_ID key into the URL Schemes field. Leave the other fields blank.

In RecipeTableViewController.swift import the FirebaseUI framework to work with Firebase authentication and Google sign-in.

```
import FirebaseUI
```

We also need our RecipeTableViewController to conform to the Auth delegate.

```
class RecipeTableViewController: UITableViewController, FUIAuthDelegate
```

Declare a variable to hold the Firebase AuthUI object which we'll use to manage Firebase Authentication.

```
var authUI: FUIAuth!
```

Then in viewDidLoad() we'll initialize the Firebase AuthUI object and set its delegate.

```
        authUI = FUIAuth.defaultAuthUI()
        authUI?.delegate = self
```

Logging in and out
We're going to set up our app so all users can see the list of recipes but will need to log in to add or delete receipes.
In the storyboard select the navigation controller and in the attributes inspector check Shows Toolbar. This should make a toolbar visible in the storyboard so you can add two bar button items and a flexible space bar button item between them (this kept crashing Xcode so I had to drag it into the document hierarchy to add it). These should have been added and are visible in the Recipes scene.
Make the button on the right Login and the left button Logout.
Connect these as actions to the RecipeTableViewController class, login and logout, respectively.
We could also connect these as outlets if we want to disable and enable the buttons when appropriate (I didn't do this in my example).

Back in RecipeTableViewController we'll implement these and use alerts so the user knows that they've successfully logged in or logged out.

```
    @IBAction func login(_ sender: Any) {
        //authentication providers
        let providers: [FUIAuthProvider] = [FUIGoogleAuth(authUI: authUI!)]
        authUI?.providers = providers
        if authUI?.auth?.currentUser == nil {
            // get the sign-in method selector
            let authViewController = authUI?.authViewController()
```

```
            // present the auth view controller
            present(authViewController!, animated: true, completion: nil)
        } else {
            //already signed in
            let name = authUI?.auth?.currentUser!.displayName
            let alert=UIAlertController(title: "Firebase", message: "You're
already logged into Firebase  \(name!)", preferredStyle:
UIAlertController.Style.alert)

            //create a UIAlertAction object for the button
            let okAction=UIAlertAction(title: "OK", style:
UIAlertAction.Style.default, handler: nil)
            alert.addAction(okAction)
            self.present(alert, animated: true, completion: nil)
            //print("\(authUI?.auth?.currentUser) is the currently logged
in")
        }
    }


    @IBAction func logout(_ sender: Any) {
        do{
            try authUI?.signOut()
            let alert=UIAlertController(title: "Firebase", message: "You've
been logged out of Firebase", preferredStyle: UIAlertController.Style.alert)

            //create a UIAlertAction object for the button
            let okAction=UIAlertAction(title: "OK", style:
UIAlertAction.Style.default, handler: nil)
            alert.addAction(okAction)
            self.present(alert, animated: true, completion: nil)
        } catch {
            print("You were not logged out")
        }
    }
```

We'll also implement the auth UI delegate method that's called when a user signs in.

```
    func authUI(_ authUI: FUIAuth, didSignInWith user: User?, error: Error?)
{
        // handle user and error as necessary
        guard let authUser = user else { return }
        //create a UIAlertController object
            let alert=UIAlertController(title: "Firebase", message: "Welcome
to Firebase \(authUser.displayName!)", preferredStyle:
UIAlertController.Style.alert)

            //create a UIAlertAction object for the button
            let okAction=UIAlertAction(title: "OK", style:
UIAlertAction.Style.default, handler: nil)
            alert.addAction(okAction)
            self.present(alert, animated: true, completion: nil)
```

```
        guard let authError = error else { return }

        let errorCode = UInt((authError as NSError).code)

        switch errorCode {
        case FUIAuthErrorCode.userCancelledSignIn.rawValue:
            print("User cancelled sign-in");
            break

        default:
            let detailedError = (authError as
NSError).userInfo[NSUnderlyingErrorKey] ?? authError
            print("Login error: \((detailedError as!
NSError).localizedDescription)");
        }
    }
```

Note that we can access the auth object's currentUser and any of their properties. currentUser is nil if the user is not logged in.
If you run this and log in you can then go to your Firebase console and in Authentication in the Users tab you will see your login identifier.

You can also customize the pre-built FirebaseUI authentication view controller by implementing `authPickerViewController(forAuthUI authUI: FUIAuth) -> FUIAuthPickerViewController`
In this method you would create and customize your own `FUIAuthPickerViewController` object.

Now let's update adding and deleting recipes so these tasks can only be done if the user is logged in.

I created a method that checks to see if you're logged in and returns a Boolean. I want to make sure the user knows why they can't add or delete a recipe so I add an alert if they're not logged in.

```
    func isUserSignedIn() -> Bool {
        if authUI?.auth?.currentUser == nil {
            //create a UIAlertController object
            let alert=UIAlertController(title: "Firebase", message: "Please
login to save your recipes", preferredStyle: UIAlertController.Style.alert)
            //create a UIAlertAction object for the button
            let okAction=UIAlertAction(title: "OK", style:
UIAlertAction.Style.default, handler: nil)
            alert.addAction(okAction)
            self.present(alert, animated: true, completion: nil)
            return false
        } else {
            return true
        }
    }
```

We'll call this method before deleting a recipe.

```
    override func tableView(_ tableView: UITableView, commit editingStyle:
UITableViewCell.EditingStyle, forRowAt indexPath: IndexPath) {
        if editingStyle == .delete {
            if isUserSignedIn(){
                // Delete the row from the data source
                if let recipeID = recipes[indexPath.row].id {
                    recipeDataHandler.deleteRecipe(recipeID: recipeID)
                }
            }
        }
}
```

Run the app, logout (it will remember you if you've already logged in), and try to delete a recipe.

Adding a recipe is done when the segue from the AddRecipeViewController unwinds so let's try to add this in the unwindSegue(segue:) method.

```
    @IBAction func unwindSegue(segue:UIStoryboardSegue){
        if segue.identifier == "savesegue" {
            if isUserSignedIn(){
                let source = segue.source as! AddRecipeViewController
                if source.addedrecipe.isEmpty == false {
                    recipeDataHandler.addRecipe(name: source.addedrecipe,
url: source.addedurl)
                }
            }
        }
    }
```

Run the app, logout, and try to add a recipe. It doesn't add the recipe but you never see the alert. In the debug area you'll see the error:

```
Attempt to present <UIAlertController: 0x7f9b5188e000> on
<UINavigationController: 0x7f9b5200da00> (from
<RecipesAuth.RecipeTableViewController: 0x7f9b51507790>) which is
already presenting <RecipesAuth.AddRecipeViewController:
0x7f9b51406a90>.
```

The error explains what's happening pretty well. We can't present an alert while it's in the middle of unwinding and presenting the RecipeTableViewController.
What would be even better is to not even let the user go to the AddRecipeViewController if they're not logged in.
Undo the changes you made to the unwindSegue(segue:) method.
Instead we'll use a method in the UIViewController class that determines whether a segue should be performed and returns a boolean. The default is true so you only need this if that's not always the case. And we have multiple segues in this class so we want to make sure we only return false for the addrecipe segue if the user is not signed in.

```
    override func shouldPerformSegue(withIdentifier identifier: String,
sender: Any?) -> Bool {
        if identifier == "addrecipe" {
            if isUserSignedIn(){
                return true
```

```
            } else {
                return false
            }
        }
        else {
            return true
        }
    }
```

Now run the app and if you're not signed in when you tap the + you'll get the alert that you're not logged in and the segue won't fire, and you'll remain in the RecipeTableViewController. If you are logged in the app will transition to AddRecipeViewController and you'll be able to add a new recipe.

You can imagine in a fully functional app perhaps we'd be showing recipes of the week, or most highly rated recipes, and then if the user logged in they could also save some recipes to their own account. It's nice to have some functionality even without making the user log in as many will be turned off by that requirement and not use the app. This way we're showing the value of the app and making logging in optional for additional functionality.