ATLS 4320/5320: Advanced Mobile Application Development
Week 7: iOS and Firebase

Firebase
https://firebase.google.com/
Google's Firebase is a backend-as-a-service (BaaS) that allows you to get an app with a server-side real-time database up and running very quickly. With SDKs available for the web, Android, iOS, and a REST API it lets you easily sync data across devices/clients on iOS, Android, and the Web.
Along with storage it provides user authentication, static hosting, and the ability to cache offline.
Get started by logging in with your Google login to create a Firebase account
Get Started:
Create a new Firebase project called Recipes.
A project in Firebase stores data that can be accessed across multiple platforms so you can have an iOS, Android, and web app all sharing the same project data. For completely different apps use different projects.
Then you'll be taken to the dashboard where you can manage the Firebase project.

https://firebase.google.com/docs/ios/setup

Before we go further we should create our app in Xcode.
Create a new single view app called recipes.
In Xcode go into the target's general tab and copy the bundle identifier.
Back in Firebase(add another app) click "Add Firebase to iOS App" and paste in the iOS bundle ID.
Download the GoogleService-Info.plist file.
Drag the file into your Xcode project making sure "Copy items" is checked and your target is checked. (or File | Add Files to project)
Close the project in Xcode.

Cocoapods
CocoaPods manages library dependencies for your Xcode projects.
The dependencies for your projects are specified in a single text file called a Podfile. CocoaPods will resolve dependencies between libraries, fetch the resulting source code, then link it together in an Xcode workspace to build your project.

In the terminal type
`sudo gem install cocoapods`
(if you've used cocoapods before just do a `pod update`)

Then navigate to the location of your Xcode project. (System Preferences Keyboard > Shortcuts > Services. Find "New Terminal at Folder" in the settings and click the box. Now, when you're in Finder, just right-click a folder > Services > New Terminal at Folder or cd and drag the folder in and hit enter.)
`pod init`
Open up the Podfile this created and add the pods that you want to install.
```
pod 'Firebase/Core'
pod 'Firebase/Database'
pod 'Firebase/Auth'
pod 'GoogleSignIn'
```

This will add the prerequisite libraries needed to get Firebase up and running in your iOS app, along with Firebase Analytics. Other pods are available for other Firebase functionality.

Save the Podfile
Make sure you've closed the project in Xcode
```
pod install
```
Now when you open the project make sure you open the xcworkspace file (not the xcodeproj file)
Make sure your project builds without errors at this point.

Xcode setup
To connect Firebase when your app starts up, add initialization code to your AppDelegate class.
```
import Firebase

    func application(_ application: UIApplication,
didFinishLaunchingWithOptions launchOptions: [UIApplicationLaunchOptionsKey:
Any]?) -> Bool {
        // Override point for customization after application launch.
        //initializes installed Firebase libraries
        FirebaseApp.configure()
        return true
    }
```

Firebase Database
Before we build the app, go back into the Firebase console and click on Database.
The Firebase Realtime Database is a cloud-hosted database. Data is stored as JSON and synchronized in realtime to every connected client. When you build cross-platform apps with our Android, iOS, and JavaScript SDKs, all of your clients share one Realtime Database instance and automatically receive updates with the newest data.

Note the URL at the top. We'll use this URL to store and sync data in our app.
https://recipes-apierce.firebaseio.com/

To get an idea of how our app will work, from a data perspective, we'll enter some data manually.

Click the + in the recipes-apierce row
Enter a recipe name into the name field (these must be unique)
Click + in this new row
Enter "name" and a name of a recipe in the value
Click + in the recipes row
Enter "url" and the url for that recipe
Click Add

This is an example of what a recipe will look like.
The data is stored as JSON which should look familiar.

Firebase Data Structure
Firebase iOS Structure Data https://firebase.google.com/docs/database/ios/structure-data
Some things to consider when looking at Firebase Realtime Database is the structure of your data:

- Your data should lend itself to being represented in a NoSQL environment.
    - 2+-way relationships can make switching to NoSQL difficult.
- Your application or website needs realtime updates of data.
- The structure of your data changes frequently or is not defined. Data like this is ideal for NoSQL.
- Your application can live without powerful aggregation functions and advanced querying.

Firebase Realtime Database sacrifices functionality for speed.

Firebase Authentication

Before we implement authentication we need to make it public so our app can read and write.
Click on Authentication and in sign-in method disable all the sign-in providers.
Also back in Database go to the Rules tab and make sure they're public with read and write both set to true so our app can access the database.

```
"rules": {
    ".read": true,
    ".write": true
}
```

Xcode

Delete the ViewController.swift file.
Go into the Storyboard and delete the view controller.
Add a table view controller and embed it in a navigation controller.
Make the navigation controller the Initial View Controller.
Give the table view controller the title "Recipes".
Add a bar button on the right and change System Item to Add.
For the table view cell make the style Basic and give it a reuse identifier "recipecell".
Add a View Controller to the right of the Table View Controller and embed it in a navigation controller.
Give it the title "Add Recipe".
Add a bar button item on the right side of the navigation bar and change it to Save.
Add a bar button item on the left side of the navigation bar and change it to Cancel.
Add a label and textfield for the user to add a recipe name and connect it as recipename.
Add another textfield for the url and connect it as recipeurl. (I added default text of http://)
Use auto layout to add needed constraints for this view.

Add two Cocoa touch classes to control these.
Call the first RecipeTableViewController and subclass UITableViewController.
Call the second AddViewController and subclass ViewController.

Back in the storyboard change the two views to use these classes.

Connect the textfields in Add Recipe as recipename and recipeurl respectively.

In order to navigate back create an unwind method in RecipeTableViewController.swift

```
@IBAction func unwindSegue(segue:UIStoryboardSegue){
}
```

Create a show segue from the Add bar button (use the document hierarchy) to the Navigation Controller for the Add View Controller and give it the identifier "addrecipe".

In the storyboard connect the Cancel and Save button to the Exit icon and chose the unwindSegue method. Name the segues "cancelsegue" and "savesegue".

You should be able to run it and navigate back and forth.

Data Model

Create a struct called Recipe for your data model.

```
class Recipe {
    var name: String
    var url: String
}
```

In RecipeTableViewController create an array of recipes to hold our recipe data.

```
var recipes = [Recipe]()
```

Now you need to define a Firebase database reference. This is called a reference because it refers to a location in Firebase where data is stored.
Firebase iOS Getting Started https://firebase.google.com/docs/database/ios/start

```
import Firebase
var ref: DatabaseReference!
```

Then in viewDidLoad you assign this reference. It knows which database to reference from the data in your GoogleService-Info.plist file.

```
ref = Database.database().reference()
```

Reading Data
Firebase iOS Read and Write Data https://firebase.google.com/docs/database/ios/read-and-write
Now we need to set up an event listener that fires when attached and then every time data in your Firebase app changes. Any time you read Firebase data, you receive the data as a DataSnapshot.
Add this in viewDidLoad()

```
//set up a listener for Firebase data change events
//this event will fire with the initial data and then all data changes
        ref.observe(DataEventType.value, with: {snapshot in
            self.recipes=[]
            //DataSnapshot represents the Firebase data at a given time
            //loop through all the child data nodes
            for recipe in snapshot.children.allObjects as! [DataSnapshot]{
                if let recipeValue = recipe.value as? [String: String],
                //get value as a Dictionary
                    let json = try? JSONEncoder().encode(recipeValue),
                //encode as JSON
                    let newRecipe = try? JSONDecoder().decode(Recipe.self,
from: json) //decode JSON to Recipe
                {
                    //add recipe to recipes array
                    self.recipes.append(newRecipe)
                }
            }
            self.tableView.reloadData()
        })
```

There is also a library called CodableFirebase that helps use the Codable protocol with custom types and Firebase.

Update the table view delegate methods as usual.

```
override func numberOfSections(in tableView: UITableView) -> Int {
      return 1
    }

    override func tableView(_ tableView: UITableView, numberOfRowsInSection
section: Int) -> Int {
      return recipes.count
    }

    override func tableView(_ tableView: UITableView, cellForRowAt
indexPath: IndexPath) -> UITableViewCell {
        let cell = tableView.dequeueReusableCell(withIdentifier:
"recipecell", for: indexPath)
        let recipe = recipes[indexPath.row]
        cell.textLabel!.text = recipe.name
        return cell
    }
```

You should now be able to run the app and see the data you entered directly into Firebase. If you add or delete data through the Firebase console you will see your app automatically updated.

Writing Data

Now let's save new recipes and write them to Firebase.

In AddViewController add variables for the recipe name and url and implement prepareForSegue.

```
    var addedrecipe = String()
    var addedurl = String()

    override func prepare(for segue: UIStoryboardSegue, sender: Any?) {
        if segue.identifier == "savesegue"{
            if recipename.text?.isEmpty == false {
                addedrecipe = recipename.text!
                addedurl = recipeurl.text!
            }
        }
    }
```

Back in RecipeTableViewController let's update unwindsegue() to save our data.

```
    @IBAction func unwindSegue(segue: UIStoryboardSegue){
        if segue.identifier == "savesegue" {
            let source = segue.source as! AddViewController
            if source.addedrecipe.isEmpty == false {
                //create new recipe object
                let newRecipe = Recipe(name: source.addedrecipe, url:
source.addedurl)
                //add recipe to recipes array
                recipes.append(newRecipe)
                //create Dictionary
                let newRecipeDict = ["name": source.addedrecipe, "url":
source.addedurl]
                //create a child reference in Firebase where the key value
```

```
is the recipe name
                let reciperef = ref.child(source.addedrecipe)
                //write data to Firebase
                reciperef.setValue(newRecipeDict)
            }
        }
    }
```

Note: I'm not checking that a url was entered or that it's a valid url, this should be done at some point. You also might want the key value something more useful like a user id or something else useful as the name ends up being duplicated.
To make typing in the simulator easier you might want to set it to use an external keyboard.
When you run this, check in Firebase to make sure the data was added.

Deleting items

To delete items in RecipeTableViewController uncomment

```
    override func tableView(_ tableView: UITableView, canEditRowAt
indexPath: IndexPath) -> Bool {
    return true
    }
```

Delete the recipe from Firebase by uncommenting and implementing

```
    override func tableView(_ tableView: UITableView, commit editingStyle:
UITableViewCellEditingStyle, forRowAt indexPath: IndexPath) {
        if editingStyle == .delete {
            let recipe = recipes[indexPath.row]
            //create a child reference in Firebase where the key value is
the recipe name
            let reciperef = ref.child(recipe.name)
            // Delete the row from Firebase
            reciperef.ref.removeValue()
        }
}
```

Offline

You can enable local persistence for the case where you don't have Internet access so offline updates will apply to your Firebase database once a connection has been made. This does not save data between app restarts.

Add the following to the end of `application(_:didFinishLaunchingWithOptions:)`, before `return true`

```
        Database.database().isPersistenceEnabled = true
```

Detail View

In the main storyboard add a new view controller and add a web view that fills up the whole view. Add an activity indicator on top of the web view. (it must be below the web view in the document hierarchy). In the attributes inspector check Hides When Stopped but make sure Hidden is unchecked (down below)
Create a new class called WebViewController that subclasses UIViewController to control this view.
Back in the storyboard change the view to use this new class.
Connect the web view and activity indicator as webView and webSpinner.

Setup needed autoresizing/constraints.
Create a show segue from the RecipeTableViewController cell to the new view and give it an identifier "showdetail".
Before leaving the storyboard go to the Master view and change the accessory on the cell to a disclosure indicator to give the user the visual cue that selecting the row will lead to more information.

In WebViewController import WebKit and adopt the WKNavigationDelegate protocol

```
import WebKit
class WebViewController: UIViewController, WKNavigationDelegate
```

Define a variable to hold the web address.

```
var webpage : String?
```

Set up the web view's delegate in viewDidLoad()

```
webView.navigationDelegate = self
```

Write a method to load a web page.

```
    func loadWebPage(_ urlString: String){
        //the urlString should be a propery formed url
        guard let weburl = webpage
            else {
                print("no web page found")
                return
        }
        //create a URL object
        let url = URL(string: weburl)
        //create a URLRequest object
        let request = URLRequest(url: url!)
        //load the URLRequest object in our web view
        webView.load(request)
    }
```

Call this method from viewDidLoad()

```
loadWebPage()
```

Implement the two delegate methods that are called when the web page starts and stops loading.

```
    //WKNavigationDelegate method that is called when a web page begins to
load
    func webView(_ webView: WKWebView, didStartProvisionalNavigation
navigation: WKNavigation!) {
        webSpinner.startAnimating()
    }

    //WKNavigationDelegate method that is called when a web page loads
successfully
    func webView(_ webView: WKWebView, didFinish navigation: WKNavigation!)
{
        webSpinner.stopAnimating()
    }
```

In RecipeTableViewController implement prepare(for:) to send the detail view the data it needs.

```
    override func prepare(for segue: UIStoryboardSegue, sender: Any?) {
        if segue.identifier == "showdetail" {
            let detailVC = segue.destination as! WebViewController
            let indexPath = tableView.indexPath(for: sender as!
UITableViewCell)!
            let recipe = recipes[indexPath.row]
            //sets the data for the destination controller
            detailVC.title = recipe.name
            detailVC.webpage = recipe.url
        }
    }
```

We really should check that the url is valid so the view doesn't hang or crash.

With Apple's updated app transport security to load web pages not available through https you need to add the following to your Info.plist

```
<key>NSAppTransportSecurity</key>
<dict>
       <key>NSAllowsArbitraryLoadsInWebContent</key>
       <true/>
</dict>
```

Key	Type	Value
Information Property List	Dictionary	(14 items)
App Transport Security Settings	Dictionary	(1 item)
Allow Arbitrary Loads in Web Content	Boolean	YES
Localization native development region	String	en
Executable file	String	$(EXECUTABLE_NAME)
Bundle identifier	String	$(PRODUCT_BUNDLE_IDENTIFIER)
InfoDictionary version	String	6.0
Bundle name	String	$(PRODUCT_NAME)
Bundle OS Type code	String	APPL
Bundle versions string, short	String	1.0
Bundle version	String	1
Application requires iPhone environment	Boolean	YES
Launch screen interface file base name	String	LaunchScreen
Main storyboard file base name	String	Main
Required device capabilities	Array	(1 item)
Supported interface orientations	Array	(3 items)

Authentication

https://firebase.google.com/docs/auth/ios/google-signin

Firebase supports authentication using email/password, google, facebook, twitter, or github. They even offer a pre-built UI you can use if you're supporting all of these.

We're going to implement authentication using Google. This requires the following pods in the Podfile:

pod 'Firebase/Auth'

pod 'GoogleSignIn'

(if you needed to add these you need to do pod install again).

In the Firebase console go into Authentication Sign-in Method and enable Google.

Then go into Database | Rules that we had changed to be wide open, and change it back so users must be authenticated.

```
  "rules": {
    ".read": "auth != null",
    ".write": "auth != null"
  }
```

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 box on the configuration page. Leave the other fields blank.
Repeat this with your Bundle Identifier that you can get in the General tab. Add another URL type with that value as the URL scheme.

In the app delegate import the needed frameworks

```
import Firebase
import GoogleSignIn
```

Configure the the FirebaseApp object and initialize sign-in in application(_:didFinishLaunchingWithOptions:)

```
// Initialize sign-in
GIDSignIn.sharedInstance().clientID = FirebaseApp.app()?.options.clientID
```

Implement the application:openURL:options: method of your app delegate. The method should call the handleURL method of the GIDSignIn instance, which will properly handle the URL that your application receives at the end of the authentication process. (a different method is needed for iOS 8 and earlier)

```
    @available(iOS 9.0, *)
    func application(_ application: UIApplication, open url: URL, options:
[UIApplicationOpenURLOptionsKey : Any])
        -> Bool {
            return GIDSignIn.sharedInstance().handle(url,

sourceApplication:options[UIApplicationOpenURLOptionsKey.sourceApplication]
as? String,
                                                     annotation: [:])
    }
```

Create a new Cocoa Touch class called LoginViewController and subclass UIViewController.
In your storyboard drag out a new View Controller and make its class LoginViewController.
Make it the Initial View Controller.
We could change the view to be the GIDSignInButton class but we can't control its layout using autoresizing or autolayout in interface builder, so we'll create the button programmatically.

In the LoginViewController import the needed frameworks

```
import Firebase
import GoogleSignIn
```

Adopt the needed the `GIDSignInDelegate` and GIDSignInUIDelegate protocol.

```
class LoginViewController: UIViewController, GIDSignInDelegate,
GIDSignInUIDelegate
```

The GIDSignInDelegate and GIDSignInUIDelegate protocols handle Google sign in and out
https://developers.google.com/identity/sign-in/ios/api/protocol_g_i_d_sign_in_delegate-p
https://developers.google.com/identity/sign-in/ios/api/protocol_g_i_d_sign_in_u_i_delegate-p

In viewDidLoad() assign the GIDSignIn and ui delegates and then define and add the Google sign in button.

```
        GIDSignIn.sharedInstance().uiDelegate = self
        GIDSignIn.sharedInstance().delegate = self

        //define Google sign in button
        let googleSignInButton = GIDSignInButton()
        googleSignInButton.style = .wide
        googleSignInButton.center = view.center
        view.addSubview(googleSignInButton)
```

The GIDSignInButton class lets you choose a style and colorScheme for the button.
https://developers.google.com/identity/sign-in/ios/api/interface_g_i_d_sign_in_button

Now we have to implement the GIDSignInDelegate methods. Users might log in using different methods but they pass the authentication to Firebase so all users are logged in with Firebase.

```
    func sign(_ signIn: GIDSignIn!, didSignInFor user: GIDGoogleUser!,
withError error: Error?) {
        if let error = error {
            print(error.localizedDescription)
            return
        }
        guard let authentication = user.authentication
            else {
                return
           }
        let credential = GoogleAuthProvider.credential(withIDToken:
authentication.idToken,
                                                       accessToken:
authentication.accessToken)

        Auth.auth()?.signIn(with: credential) { (user, error) in
           if let error = error {
                print(error.localizedDescription)
                return
            }
            print("User logged in with Google")
}

    func sign(_ signIn: GIDSignIn!, didDisconnectWith user: GIDGoogleUser!,
withError error: Error!) {
        let firebaseAuth = Auth.auth()
        do {
            try firebaseAuth.signOut()
```

```
        } catch {
            print(error.localizedDescription)
        }
    }
```

The first time you log in it will take you to Google to sign in. Right now we just go back to the login screen. In your Firebase console go to Authentication and look at Users and you should see yourself. The app will remember you have logged in and you won't be prompted again.

Now once you log in let's have our app go to our RecipeTableViewController.
Back in the storyboard create a segue from the login view controller(yellow circle icon) to the recipe table view's navigation controller and chose Manual Segue Present Modally. Give it the identifier "showRecipes".

Back in LoginViewController let's use an alert to tell the user they're logged in and then in the completion block segue to our RecipeTableViewController.

Update sign(_:didSignInFor: withError:)

```
            //create a UIAlertController object
            let alert=UIAlertController(title: "Firebase", message: "Welcome
to Firebase " + (user?.displayName)!, preferredStyle:
UIAlertControllerStyle.alert)
            //create a UIAlertAction object for the button
            let okAction=UIAlertAction(title: "OK", style:
UIAlertActionStyle.default, handler: {action in
                //perform segue
                self.performSegue(withIdentifier: "showRecipes", sender:
nil)
            })
            alert.addAction(okAction)
            self.present(alert, animated: true, completion: nil)
```

Now when you run the app and click the button to sign in you should get an alert that tells you you're logged in and when you click OK it takes you to the RecipeTableViewController.