ATLS 4320: 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.
We'll be using the realtime database, not cloud firestore.
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/

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.

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
In name add "recipes" then + and enter any id for the name
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 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.

```
struct Recipe {
    var id: String
    var name: String
    var url: String

    init(id: String, name: String, url: String){
        self.id = id
        self.name = name
        self.url = url
    }
}
```

Let's create a new data model controller class
File | New | File | Swift File
RecipeDataController

We need to import Firebase in order to use it.

```
import Firebase
```

In RecipeDataController 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

```
class RecipeDataController {
    var ref: DatabaseReference!
}
```

We also need an array of recipes to hold our recipe data.

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

We also need a property callback with a closure that we can use to update our data and table in our RecipeTableViewController class.

```
    //property with a closure as its value
    //closure takes an array of Recipe as its parameter and Void as its
return type
    var onDataUpdate: ((_ data: [Recipe]) -> Void)?
```

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.
We'll create a method where we set up this event listener.
We also assign the reference to our database reference. It knows which database to reference from the data in your GoogleService-Info.plist file.

```
    func setupFirebaseListener(){
        ref = Database.database().reference().child("recipes")
        //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.recipeData.removeAll()
            //DataSnapshot represents the Firebase data at a given time
            //loop through all the child data nodes
            for snap in snapshot.children.allObjects as! [DataSnapshot]{
                //print(snap)
                let recipeID = snap.key
                if let recipeDict = snap.value as? [String: String], //get
value as a Dictionary
                    let recipeName = recipeDict["name"],
                    let recipeURL = recipeDict["url"]
                {
                    let newRecipe = Recipe(id: recipeID, name: recipeName,
url: recipeURL)
                    //add recipe to recipes array
                    self.recipeData.append(newRecipe)
                    //print(recipeValue)
                }
            }
            //passing the results to the onDataUpdate closure
            self.onDataUpdate?(self.recipeData)
        })
    }
```

We also need a method to return the array of recipes.

```
    func getRecipes()->[Recipe]{
        return recipeData
    }
```

In RecipeTableViewController we need an array of recipes to hold our recipe data and an instance of our RecipeDataController class.

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

Let's add a method that will be called by the callback every time the data changes.

```
    func render(){
        recipes=recipeData.getRecipes()
        //reload the table data
        tableView.reloadData()
    }
```

Then in viewDidLoad() we'll assign this method to the callback and call the method to set up the event listener.

```
    override func viewDidLoad() {
        super.viewDidLoad()
        //assign the closure with the method we want called to the
onDataUpdate closure
        recipeData.onDataUpdate = {[weak self] (data:[Recipe]) in
self?.render()}
        recipeData.setupFirebaseListener()
```

```
}
```

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 RecipeDataController add a method to save a new recipe.

```
    func addRecipe(name:String, url:String){
        //create Dictionary
        let newRecipeDict = ["name": name, "url": url]

        //create a new ID
        let reciperef = ref.childByAutoId()

        //write data to the new ID in Firebase
        reciperef.setValue(newRecipeDict)
    }
```

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 {
                recipeData.addRecipe(name: source.addedrecipe, url:
source.addedurl)
            }
        }
    }
```

I'm not bothering to add the new recipe to my array or reload the table because the listener we set up will be fired since there was a change to the database and my app will automatically get updated.
Note: I'm not checking that a url was entered or that it's a valid url, this should be done at some point.
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 from Firebase let's add a method to RecipeDataController

```
    func deleteRecipe(recipeID: String){
        // Delete the object from Firebase
        ref.child(recipeID).removeValue()
    }
```

In RecipeTableViewController uncomment

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

Uncommenting and implement

```
    override func tableView(_ tableView: UITableView, commit editingStyle:
UITableViewCell.EditingStyle, forRowAt indexPath: IndexPath) {
        if editingStyle == .delete {
            let recipeID = recipes[indexPath.row].id
            recipeData.deleteRecipe(recipeID: recipeID)
        }
}
```

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()
    }
```

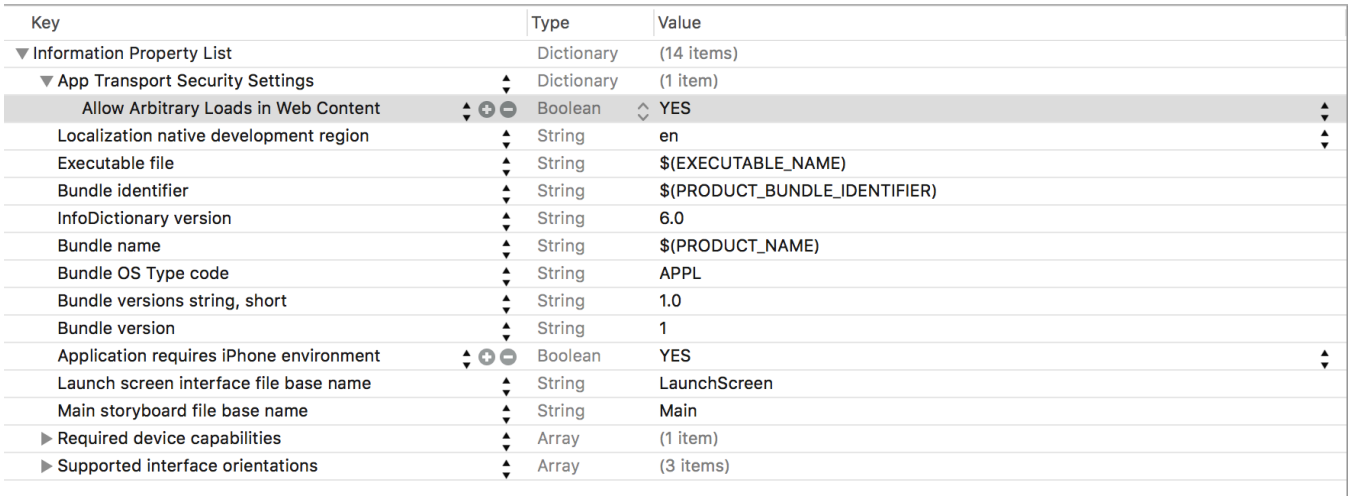
In RecipeTableViewController implement prepare(for:) to send the detail view the data it needs.

```
    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)