ATLS 4320: Advanced Mobile Application Development
Week 8: 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. Providing this as a service means you don't have to set up a database, write all the code to synchronize the data, and figure out security and authentication. Firebase stores data as JSON in the cloud in a NoSQL database.
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.

Features: https://firebase.google.com/products/

- Database
- Cloud Storage
- Authentication
- Hosting
- Cross platform support – web, iOS, Android

Firebase was founded by Andrew Lee and James Tamplin in 2011, officially launched in April 2012, and purchased by Google two years later.

Get Started
https://firebase.google.com/ Get Started
Get started by logging in with your Google login to create a Firebase account
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.
Firebase has two databases, the Realtime Database (Firebase) and Cloud Firestore. Both are noSQL databases but Firestore is their newer version so we'll use that. (Be careful of the difference between Firebase and Firestore online as they're different and have different structures, methods, etc)
Then you'll be taken to the dashboard where you can manage the Firebase project.

Cloud Firestore
https://firebase.google.com/docs/firestore
Cloud Firestore is a flexible, scalable NoSQL database for mobile, web, and server development. It keeps your data in sync across client apps through real-time listeners and offers offline support for mobile and web.

Get Started
https://firebase.google.com/docs/firestore/quickstart

1. From the console's navigation pane, select **Database**, then click **Create database** for Cloud Firestore.
   - Start in test mode.
     - Good for getting started with the mobile and web client libraries
     - Allows anyone to read and overwrite your data.
   - Select the multi-regional location nam5(us-central)

Data Model

https://firebase.google.com/docs/firestore/data-model
Cloud Firestore is a NoSQL, document-oriented database. Unlike a SQL database, there are no tables or rows. Instead, you store data in documents, which are organized into collections.
Documents

- Document is the unit of storage
- A document is a lightweight record that contains fields, which map to values.
- Each document is identified by a unique id or name
- Each document contains a set of key-value pairs.
- All documents must be stored in collections.
- Documents within the same collection can all contain different fields or store different types of data in those fields.
- However, it's a good idea to use the same fields and data types across multiple documents, so that you can query the documents more easily.
- The names of documents within a collection are unique. You can provide your own keys, such as user IDs, or you can let Cloud Firestore create random IDs for you automatically.

Collection

- A collection is a container for documents
- A collection contains documents and nothing else.
- A collection can't directly contain raw fields with values, and it can't contain other collections.
- You can use multiple collections for different related data (orders vs users)

You do not need to "create" or "delete" collections. After you create the first document in a collection, the collection exists.

Every document in Cloud Firestore is uniquely identified by its location within the database. To refer to a location in your code, you can create a *reference* to it.
A reference is a lightweight object that just points to a location in your database. You can create a reference to a collection or a document.

```
let collection = Firestore.firestore().collection("restaurants")

let document =
Firestore.firestore().collection("restaurants").document(id)
```

Let’s create a collection and the first document using the console so we know our collection exists.

In the console go into Cloud Firestore | Data
Create a collection called recipes.
Now create a document using auto id with the fields name, and url.

Note that because we’re in test mode our security rules are public and you’ll see the following in the rules tab:

```
rules_version = '2';
service cloud.firestore {
  match /databases/{database}/documents {
    match /{document=**} {
      allow read, write;
```

```
    }
  }
}
```

You'll want to change this when not in test mode or when using authentication.

Firebase and iOS
https://firebase.google.com/docs/ios/setup
Before we go further let's create an app in Xcode and get Firebase integrated.
Create a new app in Xcode called Recipes.
In Xcode go into the target's general tab and copy the bundle identifier.
Back in Firebase in Project settings | General click the "Add App" button, select iOS and follow the steps.

2. Paste in the iOS bundle ID
3. Download the GoogleService-Info.plist file
    a. 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`)

4. 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 specify the iOS version
`platform :ios, '10.0'`

Add the pods that you want to install.
```
pod 'Firebase/Firestore'
pod 'FirebaseFirestoreSwift'
```

This will add the prerequisite libraries needed to get Firestore up and running in your iOS app. FirebaseFirestore gives you access to the Firestore API and FirebaseFirestoreSwift helps integrate Firestore with your Swift models.

Other pods are available for other Firebase functionality.
Save the Podfile
Make sure you've closed the project in Xcode

In the terminal install the pods you just added to the podfile.

```
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 (this will take awhile).
For the warnings around updating the target to 12, you can either do that or uncheck them.
If you get a warning to convert to Swift 5, you can do that.
If you get the warnings "The iOS Simulator deployment target 'IPHONEOS_DEPLOYMENT_TARGET' is set to 8.0, but the range of supported deployment target versions is 9.0 to 14.0.99." this is a cocoapod issue.
You can ignore it or add to the end of your podfile (after 'end'):

```
post_install do |installer|
  installer.pods_project.targets.each do |target|
    target.build_configurations.each do |config|
      config.build_settings.delete 'IPHONEOS_DEPLOYMENT_TARGET'
    end
  end
end
```

And then do a pod install and that should remove the warnings.

Xcode setup

5. 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
    }
```

Read Data

There are two ways to retrieve data stored in Cloud Firestore. Either of these methods can be used with documents, collections of documents, or the results of queries:

- Call a method to get the data.
- Set a listener to receive data-change events.

Methods:
https://firebase.google.com/docs/firestore/query-data/get-data
Get a single document:

```
let docRef = db.collection("cities").document("SF")

docRef.getDocument { (document, error) in
    if let document = document, document.exists {
        let dataDescription =
document.data().map(String.init(describing:)) ?? "nil"
        print("Document data: \(dataDescription)")
    } else {
        print("Document does not exist")
```

```
    }
}
```

Get a single document and convert it to an object of your model class that adopts the Codable protocol (field names must match).

```
let docRef = db.collection("cities").document("BJ")

docRef.getDocument { (document, error) in
    let result = Result {
      try document?.data(as: City.self)
    }
    // parse document
}
```

Get multiple documents from a collection. You then need to loop through the result or access the one you want. You can use the various where() methods to define your query.

```
db.collection("cities").getDocuments() { (querySnapshot, err) in
    if let err = err {
        print("Error getting documents: \(err)")
    } else {
        for document in querySnapshot!.documents {
            // parse document
        }
    }
}
```

Listeners:
https://firebase.google.com/docs/firestore/query-data/listen
When you set a listener, Cloud Firestore sends your listener an initial snapshot of the data, and then another snapshot each time the document changes.

You can listen to a document with the onSnapshot() method. When listening for changes to a document, collection, or query, you can pass options to control the granularity of events that your listener will receive.

```
db.collection("cities").document("SF")
    .addSnapshotListener { documentSnapshot, error in
      guard let document = documentSnapshot else {
        print("Error fetching document: \(error!)")
        return
      }
      guard let data = document.data() else {
        print("Document data was empty.")
        return
      }
      // parse document
    }
```

An initial call using the callback you provide creates a document snapshot immediately with the current contents of the single document. Then, each time the contents change, another call updates the document snapshot.
Local writes in your app will invoke snapshot listeners immediately. This is because of a feature called "latency compensation." When you perform a write, your listeners will be notified with the new data before the data is sent to the backend.

Queries:
https://firebase.google.com/docs/firestore/query-data/queries
Cloud Firestore provides powerful query functionality for specifying which documents you want to retrieve from a collection or collection group. These queries can also be used with both the methods and listeners above.
By default, a query retrieves all documents that satisfy the query in ascending order by document ID. You can specify the sort order for your data using orderBy(), and you can limit the number of documents retrieved using limit().

Write Data
https://firebase.google.com/docs/firestore/manage-data/add-data
When you write a document to a collection in Cloud Firestore each document needs an identifier. You can explicitly specify a document identifier or have Cloud Firestore automatically generate it. You can also create an empty document with an automatically generated identifier, and assign data to it later.

.setData(data) creates or overwrites a single document and requires a document identifier.
.addDocument(data) adds a document and automatically generates a document identifier for you.
.updateData() lets you update some fields of a document without overwriting the entire document.

Delete Data
.delete() deletes a specific document
Note that deleting a document does NOT delete any subcollections it might have.

To delete an entire collection or subcollection in Cloud Firestore, retrieve all the documents within the collection or subcollection and delete them. This is not recommended from a mobile client.

Access Data Offline
https://firebase.google.com/docs/firestore/manage-data/enable-offline
Cloud Firestore supports offline data persistence. A copy of the Cloud Firestore data that your app is actively using will be cached so your app can access the data when the device is offline. You can write, read, listen to, and query the cached data. When the device comes back online, Cloud Firestore synchronizes any local changes made by your app to the Cloud Firestore backend.

- For Android and iOS, offline persistence is enabled by default. To disable persistence, set the PersistenceEnabled option to false
- For the web, offline persistence is disabled by default

Xcode
Go into the Storyboard and delete the view controller.
Add a table view controller and in the attributes inspector check Is Initial View Controller.
Select the table view cell and make the style Basic and give it a reuse identifier “recipecell”.
Embed the table view controller in a navigation controller.
Select the navigation item in the table view controller and make the title “Recipes”.

Add a bar button on the right and change System Item to Add.
Add a View Controller to the right of the Table View Controller.
Add a navigation bar to the top and set its title to "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.
Add another textfield for the url (I added default text of http://).
Use auto layout to add needed constraints for this view.

Delete the ViewController.swift file (move to trash).
Add two Cocoa touch classes to control these.
Call the first RecipeTableViewController and subclass UITableViewController.
Call the second AddRecipeViewController and subclass ViewController.

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

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

Back in the storyboard change the two controllers to use these classes.
In the AddRecipeViewController connect the textfields in Add Recipe as recipeTextField and urlTextField respectively.

Create an action segue Present Modally from the Add bar button (use the document hierarchy) to the AddRecipeViewController 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. Note the properties have the same names as the fields in Firestore documents.

```
struct Recipe: Codable, Identifiable {
    @DocumentID var id: String?
    var name: String
    var url: String
}
```

The Recipe struct must adopt the Codable protocol and the data member names must match the keys in the documents in Firestore (or use coding keys) so Swift can convert a Firestore document to an instance of our custom class.
The Identifiable protocol enables you to define a property, in our case id, that is unique.
@DocumentID marks a property so Firestore will automatically map the document ID to the id attribute. We need this unique id so later we know which document the user is trying to access, such as in a delete or update, in Firestore.

Accessing Firestore Data
Let's create a new data handler class RecipeDataHandler to handle the data in Firestore.

We need to import Firebase in order to use it.

```
import FirebaseFirestore
import FirebaseFirestoreSwift
```

In RecipeDataHandler 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. It knows which database to reference from the data in your GoogleService-Info.plist file.

Firebase iOS Getting Started https://firebase.google.com/docs/firestore/quickstart#ios

```
class RecipeDataHandler {
    let db = Firestore.firestore()
}
```

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)?
```

Firebase iOS Read and Write Data https://firebase.google.com/docs/firestore/query-data/listen#swift
Now we need to set up an event listener that fires when attached and then every time data in your Firestore app changes. Any time you read Firestore data, you receive the data as a document snapshot. We'll create a method where we set up this event listener.

```
    func dbSetup(){
        db.collection("recipes")
            .addSnapshotListener {querySnapshot, error in
                guard let documents = querySnapshot?.documents else {
                    print("Error fetching snapshot results: \(error!)")
                    return
                }
                self.recipeData = documents.compactMap {document->Recipe? in
                    return try? document.data(as: Recipe.self)
                }
                //passing the results to the onDataUpdate closure
                self.onDataUpdate?(self.recipeData)
            }
    }
```

Here I call addSnapshotListener() on my entire collection which adds a listener to receive changes in the collection. I could also define a more specifi query and add a listener to that query.
The listener takes a closure where the first parameter is the querySnapshot, the data from Firestore, and the second is the error. We can get access to all the documents through the documents property. Then we need to parse these documents into our data model.

compactMap(_:) iterates over all the documents and returns an array. It takes a closure that accepts an element(document) of this sequence(documents) as its argument and returns an instance of type Recipe in each iteration, resulting in an array.
The data(as:) method, which is provided by the FirebaseFirestoreSwift module, takes in a decodable struct and will map every document to that type and return an instance.
Note that the Recipe struct must adopt the Codable protocol and the data member names must match the keys in the documents in Firebase (or use coding keys).

Because Firestore itself doesn't require each document in a collection to look the same, it is possible to have documents that can't be converted to the struct you have specified.

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 RecipeDataHandler class.

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

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

```
    func render(){
        recipes=recipeDataHandler.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
        recipeDataHandler.onDataUpdate = {[weak self] (data:[Recipe]) in
self?.render()}
        recipeDataHandler.dbSetup()
}
```

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 Firestore. 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 Firestore.

In RecipeDataHandler add a method to save a new recipe.

```
    func addRecipe(name:String, url:String){
        let recipeCollection = db.collection("recipes")

        //create Dictionary
        let newRecipeDict = ["name": name, "url": url]

        // Add a new document with a generated id
        var ref: DocumentReference? = nil
        ref = recipeCollection.addDocument(data: newRecipeDict)
        {err in
            if let err = err {
                print("Error adding document: \(err)")
            } else {
                print("Document added with ID: \(ref!.documentID)")
            }
        }
    }
```

In AddRecipeViewController 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 recipeTextField.text?.isEmpty == false {
                addedrecipe = recipeTextField.text!
                addedurl = urlTextField.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! AddRecipeViewController
            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 Firestore to make sure the data was added.

Deleting items

To delete items from Firestore let's add a method to RecipeDataHandler

```
    func deleteRecipe(recipeID: String){
        // Delete the object from Firestore
        db.collection("recipes").document(recipeID).delete() { err in
            if let err = err {
                print("Error removing document: \(err)")
            } else {
                print("Document successfully removed!")
            }
        }
    }
```

If you want the Edit button visible in RecipeTableViewController in viewDidLoad uncomment the following line and update it so the edit button will be on the left since the add is on the right. This is optional as we can enable swipe to delete without it.

```
        self.navigationItem.leftBarButtonItem = self.editButtonItem
```

In RecipeTableViewController uncomment and implement these delegate methods

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

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

Detail View

In the main storyboard add a new view controller and add a WebKit 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 (below under Drawing)

Create a new Cocoa Touch class called WebViewController that subclasses UIViewController to control this view.
Back in the storyboard change the scene 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 recipes scene 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){
        //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() and test if no web page was entered. We really should also check that the url is valid so the view doesn't hang or crash.

```
        if webpage == "" {
            loadWebPage("https://www.myrecipes.com/")
        } else {
            loadWebPage(webpage!)
        }
```

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
        }
    }
```

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>
```