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 a variety of platforms.

Firebase offers two cloud-based, client-accessible database solutions that support realtime data syncing:

- **Cloud Firestore** is Firebase's newest database for mobile app development. It builds on the successes of the Realtime Database with a new, more intuitive data model. Cloud Firestore also features richer, faster queries and scales further than the Realtime Database.
    - Stores data in a standard NoSQL structure as a collection of documents
    - Similar to MongoDB
    - Easy to scale and store complex data structures
    - Easy to query
- **Realtime Database** is Firebase's original database. It's an efficient, low-latency solution for mobile apps that require synced states across clients in realtime.
    - Stores data in a single JSON tree (can get very large)
    - Easy to store simple data, but difficult to store complex hierarchical data
    - More difficult to query and filter

Along with storage it provides user authentication, static hosting, and the ability to cache offline.

Create Project

Get started by **logging in** with your Google login to the Firebase console to create a Firebase account
Create a new Firebase project called *Miles* or something.
We'll be using the 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

Create App

Before we go further we should create our app in Xcode.
Create a new single view app called *miles*.
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/Firestore'
pod 'Firebase/Auth'
pod 'GoogleSignIn'
pod 'FirebaseFirestoreSwift'
```

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 and **run**

```
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 UIKit
import FirebaseCore
import FirebaseFirestore

@UIApplicationMain
class AppDelegate: UIResponder, UIApplicationDelegate {
```

```
    func application(_ application: UIApplication, didFinishLaunchingWithOptions
launchOptions: [UIApplication.LaunchOptionsKey: Any]?) -> Bool {
        // Override point for customization after application launch.
        FirebaseApp.configure()

        let db = Firestore.firestore()
        print(db) // silence warning
        return true
    }
```

Create Firebase Database

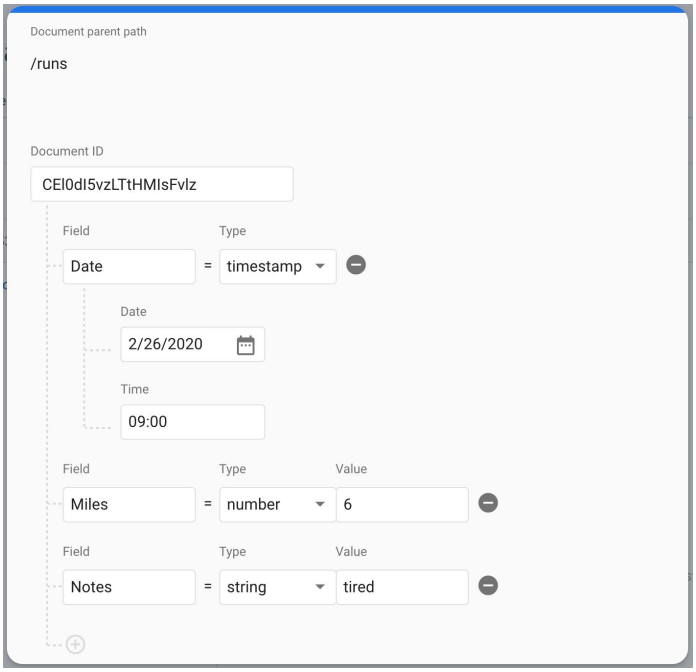
Before we build the app, go back into the Firebase console and click on Database.

Click “Create Database” under the Firestore header
Select test mode in the startup options
Choose us-central for location

Create Initial Collection and Document

Click “Start collection”
Give it collection id: *runs*

For our first document, we’ll define the structure for our runs and the associated data
Click **Auto-id** and add the following fields/values



Miles App

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.
Set the title of the navigation bar in the table view controller to “Logged Runs”.
Add a bar button on the right and **change** System Item to Add.
For the table view cell **make** the style *Right Detail* and **give** it a reuse identifier “*RunCell*”.
Set the cell’s accessory to *Disclosure Indicator*

Add a regular View Controller
Add a Navigation Bar and pin it to top, left, and right
Give it the title “New Run”.
Add a bar button item on the right side of the navigation bar and **change** the system item to *Save*.
Set the style to *Done*
Add a bar button item on the left side of the navigation bar and **change** it to Cancel.

Create *Present Modally* segue from *Add* button in the Table View to the new View Controller
Give the segue an “*AddRun*” identifier in the Attributes inspector

Add a *three* labels, *a* date picker, a text field, and text view
Change labels to reflect content
On mile textfield **set** a min width constraint
Stackview the labels with their input views
Pin the stackviews to one another

Change the keyboard type to *Decimal Pad* for the miles Text Field
Change the return key to *Done* for the Text View
Connect both delegates to the view controller

Add two Cocoa touch classes to control these.
Call the first *RunTableViewController* and **subclass** UITableViewController.
Call the second *AddRunController* and **subclass** ViewController.

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

Open assistant editor for view controller
Create outlet connections for the text edit (*notesTextEdit*), the textfield (*milesTextField*), and the date picker (*datePicker*)

Conform to delegates for text field and text view

```
class AddRunController: UIViewController, UITextFieldDelegate, UITextViewDelegate
```

Add a tap gesture recognizer in `viewDidLoad()`

```
let tapRecognizer = UITapGestureRecognizer()
tapRecognizer.addTarget(self, action: #selector(AddRunController.didTapView))
self.view.addGestureRecognizer(tapRecognizer)
```

And **add** the following methods to dismiss the keyboard

```
func textView(_ textView: UITextView, shouldChangeTextIn range: NSRange,
replacementText text: String) -> Bool {
    if(text == "\n") {
        textView.resignFirstResponder()
        return false
    }
    return true
}

@objc func didTapView(){
  self.view.endEditing(true)
}
```

In order to navigate back **create** an unwind method in `RunTableViewController.swift`

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

In the storyboard **connect** the Cancel and Save button to the Exit icon and **choose** the unwindSegue method
Name the segues “CancelSegue” and “SaveSegue”.

You should be able to **run** it and navigate back and forth. Initial build will take a while because of the dependencies

Data Model

Let’s **create** a new data model controller class
File -> New -> File -> Swift File
RunDataController

We need to **import** Firebase in order to use it.

```
import Firebase
```

Create a struct called *Run* for your data model.

```
struct Run {
    var date: Date
    var miles: Double
    var notes: String
```

```
    var id: String

    func getDate() -> String {
        let formatter = DateFormatter()
        formatter.dateStyle = .short
        return formatter.string(from: date)
    }
}
```

Create data controller class
Add required properties, init method, and method to load/listen for data

```
class RunDataController {
    //store reference to data base
    var db: Firestore!

    //store our local data
    var runData = [Run]()

    //closure to notify view controller of data changes
    var onDataUpdate: (([Run]) -> Void)!


    init() {
        //use the default settings
        let settings = FirestoreSettings()
        Firestore.firestore().settings = settings

        //get reference to our database
        db = Firestore.firestore()
    }

    //fetch data initially and add a listener for any new data
    func loadData() {
        db.collection("runs").addSnapshotListener { querySnapshot, error in
            //make sure we got the collection
            guard let collection = querySnapshot else {
                print("Error fetching collection: \(error!)")
                return
            }

            //get the docs
            let docs = collection.documents
```

```
            //empty our data out
            self.runData.removeAll()

            //append to our list
            for doc in docs {
                //get the data dictionary from the document
                let data = doc.data()
                //get the data fields and downcast to appropriate types
                let date = (data["Date"] as! Timestamp).dateValue()
                let miles = data["Miles"] as! Double
                let notes = data["Notes"] as! String

                //get the id
                let id = doc.documentID
                //construct object
                let run = Run(date: date, miles: miles, notes: notes, id: id)


                self.runData.append(run)
            }

            self.onDataUpdate(self.runData)
        }
    }
}
```

Implement Data Controller in View Controller

class RunTableViewController: UITableViewController {

```
    //instantiate data controller
    var runDC = RunDataController()

    //local data
    var runData = [Run]()

    override func viewDidLoad() {
        super.viewDidLoad()
        //set data update listener
        runDC.onDataUpdate = {[weak self] (data: [Run]) -> Void in
self?.newData(data: data)}

        //load the data
        runDC.loadData()
    }
```

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

    func newData(data: [Run]) {
        //set the data
        runData = data
        //reload the tableview
        tableView.reloadData()
    }


    // MARK: - Table view data source
    override func numberOfSections(in tableView: UITableView) -> Int {
        return 1
    }

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

    override func tableView(_ tableView: UITableView, cellForRowAt indexPath:
IndexPath) -> UITableViewCell {
        let cell = tableView.dequeueReusableCell(withIdentifier: "RunCell", for:
indexPath)

        //get the run object
        let run = runData[indexPath.row]

        //set the labels
        cell.textLabel?.text = run.getDate()
        cell.detailTextLabel?.text = "\(String(run.miles)) mi"

        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 runs and write them to Firebase.
In `RunDataController.swift` **add** a method to save a new run.

```
func writeData(date: Date, miles: Double, notes: String) {
```

```
    // Add a second document with a generated ID.
    db.collection("runs").addDocument(data: [
        "Date": Timestamp(date: date),
        "Miles": miles,
        "Notes": notes,
    ], completion: { err in
        if let err = err {
            print("Error adding document: \(err)")
        } else {
            print("new document added successfully!")
        }
    })
}
```

In `AddRunController.swift` add variables for the run details and implement prepareForSegue.

```
//user input variables
var notes: String?
var date: Date?
var miles: Double?


override func prepare(for segue: UIStoryboardSegue, sender: Any?) {
    //check to make sure we're only saving when the user presses save button
    if segue.identifier == "SaveSegue" {
        //check to make sure they at least entered mileage
        if let userMiles = Double(milesTextField.text!) {
            miles = userMiles
            date = datePicker.date
            if notesTextView.text.isEmpty == false {
                notes = notesTextView.text
            }
        } else {
            print("Not a valid mileage: \(milesTextField.text!)")
        }
    }
}
```

Back in `RunTableViewController.swift` 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.

Detail View

Go to storyboard and drag out a new View Controller
Ctrl-click and drag from the TableViewCell to the new controller to create segue
Select *Show* from the popup list
Set the title of the nav bar to *Run*

Drag 6 labels onto the new view controller
Make three of them titles for "Date", "Miles", "Notes" and layout according to this structure.
Use stack views and auto layout to make it look decent

Create a new Cocoa Touch Class called DetailViewController and make sure it subclasses UIViewController
Go back to storyboard and **set** it as the class for our detail view
Make outlet connections for each of the content labels called "*milesLabel*", "*dateLabel*", and "*notesLabel*"

Make the following changes to DetailViewController.swift

```
class DetailViewController: UIViewController {

    //outlet connections
    @IBOutlet weak var dateLabel: UILabel!
    @IBOutlet weak var milesLabel: UILabel!
    @IBOutlet weak var notesLabel: UILabel!

    //run object
    var run: Run?

    override func viewWillAppear(_ animated: Bool) {
        //check to make sure we have the run
        if let myRun = run {
            dateLabel.text = myRun.getDate()
            milesLabel.text = String(myRun.miles)
            notesLabel.text = String(myRun.notes)
        }
```

```
    }
```

Add the prepare() method to `RunTableViewController.swift`

```
//pass data before the segue
override func prepare(for segue: UIStoryboardSegue, sender: Any?) {
    if segue.identifier == "DetailSegue" {
        //downcast destination
        let vc = segue.destination as! DetailViewController
        //get index of run to view
        let idx = tableView.indexPath(for: (sender as! UITableViewCell))
        //pass the run along
        vc.run = runData[idx!.row]

    }
}
```

Run the app and verify that the segue works

Firebase Google Sign In

At this point our app doesn't have many use cases since there is no sign in and all the data would be pushed to all the users. Firebase comes with a variety of different sign in methods to implement authentication including:

- Email and password
- Apple ID
- Google
- Facebook
- and others

We'll be using Google Auth but it's relatively easy to implement other sign in methods once we've figured out how to do this one.

Database changes

We need to think about how this will change our data structure so that we can access the right data based on the authenticated user. There's no single right solution for structure, but some have advantages over others so it's certainly worth spending some time thinking about. For our purposes, I've decided to change the database to have a collection of users and a subcollection of runs for each user.

Our new structure looks like this:

- users : collection of user ids
    - runs : collection of run documents
        - Miles : number
        - Date : timestamp
        - Notes : string

Create this dummy structure in the firebase application

Enable Auth for Firebase project

To start, we need to enable authentication in our Firebase project.

Go to the Firebase console -> **select** Authentication from the sidebar on the right -> **click** the sign in method tab near the top -> and **click** "*Google*" from the list

Change the switch in the top right of the card to "*Enable*", **enter** your email in the "*Project support email*" field, and **click** "*Save*"

Add URL Scheme to Xcode

Since we will be authenticating from an external URL, we need to add a URL scheme to our project. This just declares which URLs are allowed to redirect to our app.

Go to your `GoogleService-Info.plist` file and **copy** the value in the "REVERSED_CLIENT_ID" field

Navigate to your main app target ("*Miles*" for me) in the navigator pain and select the "*Info*" tab from the options at the top.
Scroll down to the URL Types entry in the accordion and **expand**. **Click** the plus sign to add a new scheme and **paste** the *REVERSED_CLIENT_ID* URL into the *URL Schemes* field.

Add Cocoapod Dependencies

We'll be using a library called **FirebaseUI** to make the authentication workflow easier. It provides a view controller with the standard buttons for each supported service and also has a **delegate protocol** to help with handling **success/error cases** during auth and also allows **customization** of the auth view controller.

Close Xcode

We need to **add** the necessary dependencies to our Podfile:

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

Be sure to run `pod install` after making the changes to the Podfile
Reopen Xcode

Implement Authentication

Sign In View Controller

Create a new Cocoa Touch Class called *SignInViewController* and be sure that you subclass UIViewController

Go to `Main.storyboard` and **drag** out a new view controller

Set its class to our new *SignInViewController*
Set it as the Initial View Controller
Add a label and a button and **give** them appropriate titles (“Welcome” and “Sign In”)
Create an Action Connection for the button to *SignInViewController* called *logIn*

Select the navigation controller and **give** it a Storyboard ID of “rootNav”

Modify Data Controller

We need to make some changes to our data controller class to accommodate the structure change to our database.

Luckily it’s pretty easy to do using the Firebase SDK for Swift, we just need to get the id of the authenticate user and read/write from their specific document in our users collection.

Go to `RunDataController.swift`
Make the following changes in bold to the `loadData` and `writeData` methods

```
//fetch data initially and add a listener for any new data
func loadData() {
    //get the user id of the currently authenticated user
    let authUserID = Auth.auth().currentUser?.uid
    if let userID = authUserID {
        //navigate to the user's run collection and add listener

db.collection("users").document(userID).collection("runs").addSnapshotListener {
querySnapshot, error in
            //make sure we got the collection
            guard let collection = querySnapshot else {
                print("Error fetching collection: \(error!)")
                return
            }

            //get the docs
            let docs = collection.documents

            //empty our data out
            self.runData.removeAll()

            //append to our list
            for doc in docs {
                //get the data dictionary from the document
                let data = doc.data()
                //get the data fields and downcast to appropriate types
                let date = (data["Date"] as! Timestamp).dateValue()
```

```
                let miles = data["Miles"] as! Double
                let notes = data["Notes"] as! String

                //get the id
                let id = doc.documentID
                //construct object
                let run = Run(date: date, miles: miles, notes: notes, id: id)


                self.runData.append(run)
            }
            //pass data to view controller
            self.onDataUpdate(self.runData)
        }
    } else {
        print("could not read data, no auth user")
    }
}

func writeData(date: Date, miles: Double, notes: String) {

    //try to get the authenticated user's ID
    let authUserID = Auth.auth().currentUser?.uid
    if let userID = authUserID {
        // Add another run document with a generated ID to the authenticated
user's collection of runs

db.collection("users").document(userID).collection("runs").addDocument(data: [
            "Date": Timestamp(date: date),
            "Miles": miles,
            "Notes": notes,
        ], completion: { err in
            if let err = err {
                print("Error adding document: \(err)")
            } else {
                print("new document added successfully!")
            }
        })

    } else {
        print("could not write, no auth user")
    }
}
```

Implement the View Controller

Don't forget to import FirebaseUI and conform to FUIAuthDelegate!

```
import FirebaseUI

class SignInViewController: UIViewController, FUIAuthDelegate {

    //create default auth UI instance
    let authUI = FUIAuth.defaultAuthUI()

    @IBAction func logIn(_ sender: Any) {

        print("trying to present view controller")
        //get the auth navigation controller
        let authViewController = authUI?.authViewController()
        //hand over control to FirebaseUI
        present(authViewController!, animated: true, completion: nil)
    }

    override func viewDidLoad() {
        super.viewDidLoad()

        //configure and present the auth UI view controller as the initial entry
point for our app
        authUI?.delegate = self

        //config the auth providers array
        let providers: [FUIAuthProvider] = [
          FUIGoogleAuth()
        ]
        //set the auth providers
        authUI?.providers = providers

    }

    //FUIAuthDelegat method which is called when authentication is completed
successfully or errors out
    func authUI(_ authUI: FUIAuth, didSignInWith user: User?, error: Error?) {
        //make sure we got the user object
        if user != nil {
            //get access to storyboard
            let storyboard = UIStoryboard(name: "Main", bundle: nil)
            //get instance of the root nav controller
            let vc = storyboard.instantiateViewController(withIdentifier:
```

```
"rootNav")
            //set it to full screen instead of popover
            vc.modalPresentationStyle = .fullScreen
            //present the root view controller
            present(vc, animated: true, completion: nil)
        } else {
            print(error!.localizedDescription)
        }
    }
}
```

Run it and verify that it works! Be sure to look at the database in the Firebase console to ensure that the client is reading/writing as expected