ATLS 4320: Advanced Mobile Application Development
Week 8: iOS and Realm

Realm

https://realm.io/
Realm is a Danish startup founded in 2011. Realm is a cross-platform object-oriented database that has been open-sourced as of 2016, and is available free of charge to developers. Realm is available for iOS, Android, and JavaScript(React Native and Node.js). It's fast, easy and lightweight and doesn't have the steep learning curve of SQLite or Core Data on iOS. Their mobile platform now includes a server that handles data synchronization and an API to existing databases(Oracle, MongoDB, and others). We're going to use their database for local persistence.

Realm: Realm instances are the heart of the framework; it's your access point to the underlying database. You will create instances using the Realm() initializer.
https://realm.io/docs/swift/latest#realms

Object: Object is the Realm class used to define Realm model objects. The act of creating a model defines the schema of the database. To create a model you simply subclass Object and define the fields you want to persist as properties. https://realm.io/docs/swift/latest#models

Relationships: You create one-to-many relationships between objects by simply declaring a property of the type of the Object you want to refer to. You can create many-to-one and many-to-many relationships via a property of type List. https://realm.io/docs/swift/latest#model-inheritance and https://realm.io/docs/swift/latest#relationships

Write Transactions: Any operations in the database such as creating, editing, or deleting objects must be performed within writes which are done by calling write(_:) on Realm instances.
https://realm.io/docs/swift/latest#writes

Queries: To retrieve objects from the database you'll need to use queries. The simplest form of a query is calling objects() on a Realm instance, passing in the class of the Object you are looking for. If your data retrieval needs are more complex you can make use of predicates, chain your queries, and order your results as well. https://realm.io/docs/swift/latest#queries

Results: Results is an auto updating container type that you get back from object queries. They have a lot of similarities with regular Arrays, including the subscript syntax for grabbing an item at an index.

Realm and iOS

https://realm.io/docs/swift/latest/

Create a new single view universal app called groceryList.

In the terminal go into your project's directory and initialize cocoapods.
pod init

Open up the Podfile created in your project directory and add the RealmSwift pod
```
  # Pods for groceryList
```

pod 'RealmSwift'

(to upgrade Realm use `pod update RealmSwift`)
(to upgrade cocoapods use `pod update`)

now install the pod
pod install

When installation is finished open up your xcworkspace file and build the project to make sure it's successful.
If you get the error "no such module realmswift", the module needs to be built on the project:

- Product > Schemes > New Scheme...
- Select: RealmSwift and click OK
- Build the RealmSwift target (cmd + b)

In the AppDelegate import RealmSwift and if it gives you an error, build your project.

```
import RealmSwift
```

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 navigation item the title "Groceries".
For the table view cell make the style Basic and give it a reuse identifier "cell".
Select the table view and go into the connections inspector and make sure the dataSource and delegate are connected to the View Controller(Groceries).

Delete the ViewController.swift file.
Add a Cocoa touch classes to control this view called GroceryTableViewController and subclass TableViewController.
Back in the storyboard change the table view controller to use this class.

Create a data model class to represent your grocery items. Your class must inherit from Object, the Realm class for defining Realm model objects. Realm doesn't support Swift structs as models.
https://realm.io/docs/swift/latest#faq-swift-structs
Specific datatypes in Realm, such as strings, must be initialized with a value, so we use an empty string.

The act of creating a model defines the schema of the database, so your class properties represent the data being stored in the database. If you create multiple classes you will have multiple Realm models, and you can create relationships between the models by using a class type for a property in another class.

```
import RealmSwift
class Grocery: Object {
    @objc dynamic var name = ""
    @objc dynamic var bought = false
}
```

All property types(except for List and RealmOptional) must be declared as @objc dynamic var.

By applying the dynamic declaration modifier to a member of a class, you tell the compiler that dynamic dispatch should be used to access that member. Swift uses static and virtual dispatch whenever possible over dynamic dispatch, whereas Objective-C only uses dynamic dispatch. The dynamic declaration forces the use of dynamic dispatch, which is required by Realm. As dynamic dispatch is Objective-C you need to mark it with the objc attribute to make the declaration available in Objective-C.
Dynamic dispatch means that the Objective-C runtime decides at runtime which implementation of a particular method or function it needs to invoke. For example, if a subclass overrides a method of its superclass, dynamic dispatch figures out which implementation of the method needs to be invoked, that of the subclass or that of the parent class.

Now let's create a data model controller class called GroceryDataController.

```
import RealmSwift

class GroceryDataController {
    var myRealm : Realm!  //Realm database instance
    var groceryData: Results<Grocery> //collection of Objects
     {
        get {
            return myRealm.objects(Grocery.self) //returns all Grocery
objects from Realm
        }
    }
```

Results is an auto-updating container type in Realm returned from object queries. Since it's a generic type so you need to specify the type in < >

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

    func dbSetup(){
        //initialize the Realm database
        do {
            myRealm = try Realm()
        } catch let error {
            print(error.localizedDescription)
        }
        onDataUpdate?(Array(groceryData)) //converts collection of Objects
to an Array
        print(Realm.Configuration.defaultConfiguration.fileURL!) //prints
location of Realm database
    }

    func getGroceries()->[Grocery]{
        return Array(groceryData)
    }
}
```

In GroceryTableViewController we'll define an instance of our GroceryDataController class and an array to hold our list of Grocery items.

```
var groceryData = GroceryDataController()
var groceryList = [Grocery]()
```

We'll create a method that we want to be called whenever the data changes.

```
    func render(){
        groceryList=groceryData.getGroceries()
        //reload the table data
        tableView.reloadData()
    }
```

In viewDidLoad() we'll set the callback closure to call render() and we'll call our Realm set up method.

```
groceryData.onDataUpdate = {[weak self] (data:[Grocery]) in self?.render()}
groceryData.dbSetup()
```

Update the tableview delegate and data source methods.

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

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

    override func tableView(_ tableView: UITableView, cellForRowAt
indexPath: IndexPath) -> UITableViewCell {
        let cell = tableView.dequeueReusableCell(withIdentifier: "cell",
for: indexPath)
        let item = groceryList[indexPath.row]
        cell.textLabel!.text = item.name
         cell.accessoryType = item.bought ? .checkmark : .none //set
checkmark if bought
        return cell
    }
```

<u>Add Items</u>

Let's add a method to our GroceryDataController class to add an item.

```
    func addItem(newItem:Grocery){
        do {
            try self.myRealm.write({
                self.myRealm.add(newItem) //add to realm database
            })
        } catch let error{
            print(error.localizedDescription)
        }
        onDataUpdate?(Array(groceryData))
    }
```

In the storyboard add a bar button item to the navigation bar of the GroceryTableViewController and change it to Add. Connect it as an action called addGroceryItem.

```
    @IBAction func addGroceryItem(_ sender: UIBarButtonItem) {
        let addalert = UIAlertController(title: "New Item", message: "Add a
new item to your grocery list", preferredStyle: .alert)
        //add textfield to the alert
        addalert.addTextField(configurationHandler: {(UITextField) in
        })
        let cancelAction = UIAlertAction(title: "Cancel", style: .cancel,
handler: nil)
        addalert.addAction(cancelAction)
        let addItemAction = UIAlertAction(title: "Add", style: .default,
handler: {(UIAlertAction)in
            // adds new item
            let newitem = addalert.textFields![0] //gets textfield
            let newGroceryItem = Grocery() //create new Grocery instance
            newGroceryItem.name = newitem.text! //set name with textfield
text
            newGroceryItem.bought = false
            self.groceryData.addItem(newItem: newGroceryItem)
        })
        addalert.addAction(addItemAction)
        present(addalert, animated: true, completion: nil)

    }
```

To test, run the app, exit the app, and start it again, the data should still be there.

Updating Items

When the user taps a cell we want to mark the item bought, update Realm, and make the change visually.

Let's add a method to our GroceryDataController class to indicate an item has been bought.

```
    func boughtItem(item: Grocery){
        try! self.myRealm.write {
            item.bought = !item.bought
        }
        onDataUpdate?(Array(groceryData))
    }
```

Then in GroceryTableViewController uncomment/update this method to return true.

```
    override func tableView(_ tableView: UITableView, canEditRowAt
indexPath: IndexPath) -> Bool {
        // Return false if you do not want the specified item to be editable.
        return true
    }
```

And update this method to call our boughtItem(item:) method.

```
    override func tableView(_ tableView: UITableView, didSelectRowAt
indexPath: IndexPath) {
```

```
        let boughtItem = groceryList[indexPath.row]
        groceryData.boughtItem(item: boughtItem)
    }
```

Deleting Items

Let's add a method to our GroceryDataController class to delete an item.

```
    func deleteItem(item: Grocery){
        try! self.myRealm.write {
            self.myRealm.delete(item) //delete from realm database
        }
        onDataUpdate?(Array(groceryData))
    }
```

Then in GroceryTableViewController uncomment/update this method to delete an item.

```
    override func tableView(_ tableView: UITableView, commit editingStyle:
UITableViewCell.EditingStyle, forRowAt indexPath: IndexPath) {
        if editingStyle == .delete {
            let item = groceryList[indexPath.row]
            groceryData.deleteItem(item: item)
        } else if editingStyle == .insert {
            // Create a new instance of the appropriate class, insert it
into the array, and add a new row to the table view
        }
    }
```

Now you should be able to check off an item or delete it and the data will persist between app launches.

Realm Browser

Realm Browser allows you view and edit Realm databases from your computer and is great for debugging (available on the App store for Mac only https://itunes.apple.com/app/realm-browser/id1007457278). It's really useful while developing as the Realm database format is proprietary and not easily human-readable. Realm Studio is available for Mac, Windows, and Linux https://realm.io/products/realm-studio

To help you find where your Realm database is you can print out the path in viewDidLoad()

```
print(Realm.Configuration.defaultConfiguration.fileURL!)
```

Navigate there and double click on default.realm and it will open in Realm Browser.

The easiest way to go to the database location is to open Finder, press Cmd-Shift-G and paste in the path. Leave off the file:/// and the file name (Users/aileen/Library/Developer/CoreSimulator/Devices/AA7ED426-2F2C-45F8-B895-2DB133897F1D/data/Containers/Data/Application/61EA4CC0-E521-408F-9589-6D45C14E7878/Documents)

More info on finding your Realm database https://stackoverflow.com/questions/28465706/how-to-find-my-realm-file