Advanced Mobile Application Development
Week 8: iOS and Realm

Realm

https://realm.io/

Realm was created at a Danish startup founded in 2011 and was acquired by MongoDB in 2019. 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 Realm for local persistence as it also provides the opportunity to look at how outside frameworks can be integrated into Xcode.

Fundamentals:
https://docs.mongodb.com/realm/sdk/ios/fundamentals/realms/

- Realms are the core data structure used to organize data in Realm Database. A realm is a collection of the objects that you use in your application, called Realm objects.
- Object: Object is the Realm class used to define the properties and relationships for Realm model objects. To create your data model you simply subclass Object and define the properties and methods for your class. The act of creating a model defines the schema of the database.
- All Realm objects are live objects**,** which means they automatically update whenever they're modified so you never have to refresh objects.
- 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 filter and sort your results as well.
- Results: Results is an auto updating container type that you get back from object reads or queries. They have a lot of similarities with regular Arrays, including the subscript syntax for grabbing an item at an index.
- Write Transactions: Any operations in the database such as creating, updating, or deleting objects must be performed within a write transaction which are done by calling write(_:) on Realm instances.
- Relationships: Realm allows you to define explicit relationships between objects. A relationship is an object property that references another Realm object rather than one of the primitive data types. You define relationships by setting the type of an object property to reference other objects of the same or another object type.
    - Realm Database supports to-one, to-many, and inverse relationships.

You can also use MongoDBRealm Sync to keep data synced with MongoDB in the cloud but we're just going to look at using Realm for local data persistence today.

CocoaPods

CocoaPods is a popular dependency manager for iOS projects. A dependency manager makes it easy to add, remove, update and manage the third-party dependencies your app uses. Thousands of libraries use it and it's the most popular way to integrate third-party libraries in an iOS project. A pod is a general term that is used for either a library or framework that's added to a project using cocoapods.
Cocoapods uses Ruby which comes on MacOS so you should only need to install cocoapods.

Go into the terminal and install (or upgrade) cocoapods (if you haven't already)
```
sudo gem install cocoapods
```

You can also use brew instead of gem to install cocoapods (If you don't have permission add sudo at the beginning).
```
brew install cocoapods
```

Troubleshooting (MacOS 10.15.7 Catalina):
pod update gave me the error
```
/System/Library/Frameworks/Ruby.framework/Versions/2.3/usr/bin/ruby: bad
interpreter: No such file or directory
```

Installing cocoapods gave me the error
```
ERROR: Failed to build gem native extension.
Could not create Makefile due to some reason, probably lack of necessary
libraries and/or headers.
You have to install development tools first.
```

In the log file the error said
```
package configuration for libffi is not found
```
I needed to update my version of ruby.
I installed the Ruby Version Manager (rvm)
```
curl -L https://get.rvm.io | bash -s stable
```

Then installed the latest version of ruby.
```
rvm install ruby-2.7.2
```

Then installing cocoapods worked.
```
sudo gem install cocoapods
```

(source: https://stackoverflow.com/questions/62700347/unable-to-install-cocoa-pods)

Install Realm
https://docs.mongodb.com/realm/sdk/swift/install/
Create a new app called GroceryRealm.
Exit out of the project in Xcode.
In the terminal go into your project's directory and initialize cocoapods.
```
pod init
```

Open up the Podfile created in your project directory and uncomment the line with platform and update it to 15.2. (I don't think this version really matters).
```
platform :ios, '15.2'
```

Add the RealmSwift pod
```
  # Pods for GroceryRealm
pod 'RealmSwift', '~>10'
```

Save the file and close it.
now install the pod
```
pod install
```

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

When installation is finished open up your **xcworkspace** file and wait for the index/processing of files finishes.
Then build the project to make sure it builds (Project | Build)
If you get a warning about updating to recommended settings so the deployment target is iOS12 go ahead and do that.

Note: The app in my GitHub repo does not have the pods included due to their large file size. If you download it you will need to run `pod install` before opening and running the project.

SwiftPM
Instead of using CocoaPods you can use SwiftPM, Swift package manager.
https://docs.mongodb.com/realm/sdk/swift/install/
After creating your Xcode project go into File | Add Package
In the search field search for `https://github.com/realm/realm-swift.git`
In the options for the `realm-swift` package which comes up, leave the default `Dependency Rule` of `Up to Next Major`, then click `Add Package`.
When prompted select both `Realm` and `RealmSwift`, then click `Add Package`.

Realm and iOS
Open your project and wait until it's done indexing.
In the AppDelegate import RealmSwift and if it gives you an error, build your project. You must have this import in any file that uses Realm.

```
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".

Delete the ViewController.swift file (move to trash).
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.
Select the table view and go into the connections inspector and make sure the dataSource and delegate are connected to the View Controller (Groceries).

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.
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 {
    @Persisted(primaryKey: true) var _id = ObjectId.generate()
    @Persisted var name = ""
    @Persisted var bought = false
}
```

@Persisted is used to declare properties that you want to store to the Realm database.

It's recommended to have a primary key to ensure each item has a unique id. Once an object with a primary key is added to a Realm, the primary key cannot be changed.

ObjectIds are similar to a GUID or a UUID, and can be used to uniquely identify objects without a centralized ID generator. An ObjectID consists of:

1. A 4 byte timestamp measuring the creation time of the ObjectId in seconds since the Unix epoch.
2. A 5 byte random value
3. A 3 byte counter, initialized to a random value.

ObjectIds are intended to be fast to generate. Sorting by an ObjectId field will typically result in the objects being sorted in creation order.

You can either declare _id as type ObjectId or call the generate() method, both create a new randomly-initialized ObjectId.

Now let's create a data handler class called GroceryDataHandler.

```
import RealmSwift

class GroceryDataHandler {
    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
        }
    }

    func dbSetup(){
        //initialize the Realm database
        do {
            myRealm = try Realm()
        } catch let error {
            print(error.localizedDescription)
        }
```

```
        print(Realm.Configuration.defaultConfiguration.fileURL!) //prints
location of Realm database
    }

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

    func boughtItem(item: Grocery){
        do {
            try myRealm.write ({
                item.bought = !item.bought
            })
        }catch let error{
            print(error.localizedDescription)
        }
    }

    func deleteItem(item: Grocery){
        do {
            try myRealm.write ({
                myRealm.delete(item) //delete from realm database
            })
        } catch let error{
            print(error.localizedDescription)
        }
    }

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

Realm object instances are live, auto-updating views into the underlying data; you never have to refresh objects. Results is a container type in Realm returned from object queries. It's a generic type so you need to specify the type in < >
Results instances cannot be directly instantiated. Results are lazily evaluated the first time they are accessed; they only run queries when the result of the query is requested.
Results always reflect the current state of the Realm on the current thread, including during write transactions on the current thread.
When we access groceryData it fires its get method to return the Grocery objects from Realm.

We also define methods to initialize the Realm database, add an item, mark an item as bought, delete an item, and to return an array of Grocery objects.

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

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

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

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

In viewDidLoad() we'll call our Realm set up method and populate the array.

```
groceryData.dbSetup()
groceryList=groceryData.getGroceries()
```

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)
        var cellConfig = cell.defaultContentConfiguration()
        let item = groceryList[indexPath.row]
        cellConfig.text = item.name
        cell.contentConfiguration = cellConfig
        cell.accessoryType = item.bought ? .checkmark : .none //set
checkmark if bought
        return cell
    }
```

Add Items

In the storyboard add a bar button item to the navigation bar of the GroceryTableViewController and change the System Item 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)
           self.render()
        })
        addalert.addAction(addItemAction)
        present(addalert, animated: true, completion: nil)
    }
```

This is very similar to how we added an item in our last app, the only difference is that we're creating a Grocery instance, setting its name and bough properties, and then adding that to our array. And instead of directly updating our groceryList array, after updating Realm we're calling render() which is fetching the groceries from the GroceryDataHandler class and then reloading our table.

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.

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
    }
```

Add and implement 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)
      render()
    }
```

Deleting Items

If you want the Edit button visible in GroceryTableViewController 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
```

Uncomment the following if you haven't already:

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

Then uncomment/update this delegate 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)
           render()
        } 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 Data

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://github.com/realm/realm-studio

To help you find where your Realm database is you can print out the path

```
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

To delete all the objects in a Realm https://docs.mongodb.com/realm/sdk/ios/examples/read-and-write-data/ (bottom). That will still leave the structure.
If you change the structure by changing the data model I suggest deleting the app off the device so you remove the database. Otherwise you'll get errors unless you migrate the structures.