Advanced Mobile Application Development
Week 10: Firebase

Firebase

Google's Firebase is a backend-as-a-service (BaaS). It's a real time database that syncs data across all devices/clients and supports iOS, Android, and Javascript through their SDKs.
Along with storage it provides user authentication, static hosting, and the ability to cache offline.
https://www.firebase.com/docs/ios/quickstart.html
Create a Firebase account using your Google login.
A brand new Firebase app will automatically be created for you with its own unique URL ending in firebaseio.com. We'll use this URL to store and sync data.
https://burning-inferno-6550.firebaseio.com
(skip the 5 min tutorial, it's aimed at JavaScript apps)

To get a look at what Firebase is all about, click Manage App in My First App. This area is known as Firebase Forge. It's worth taking the suggested tutorial which will guide you through creating keys, values, and children. To exit the brief Forge tutorial, click Dashboard in the upper-left corner of the screen.

To the left of My First App click the faded box to create a new app. For the app name, enter "Recipes". For the app url, enter "recipes-your-name", with "your-name" being your name or a userid. This field must be unique, as it is the actual url for your app. Click create new app when you're done.
Manage App will take you into the Forge screen where you can see the data for your app. You can manage your data here as well as instantly see it update as is does in our app. We can also enter app data directly into Forge. To get an idea of how our app will work, from a data perspective, we'll enter some data manually.

https://recipes-apierce.firebaseio.com

Click the + in the recipes-apierce row
Enter a recipe name into the name field (these must be unique)
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 now.

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
```

Xcode
Create a new single view universal app called **recipes**.

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 TableViewController.
Call the second AddViewController and subclass ViewController.

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

Connect the textfields as recipename and recipeurl respectively.

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

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

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.

CocoaPods Setup
To use CocoaPods for this app follow the instructions here
https://www.firebase.com/docs/ios/quickstart.html
Your Podfile should look like this:

```
#Uncomment this line to define a global platform for your project
platform :ios, '8.0'
#Uncomment this line if you're using Swift
use_frameworks!
pod 'Firebase', '>= 2.5.0'
target 'recipes' do

end
```

Close Xcode
Reopen project using the .xcworkspace file (instead of xcodeproj file)

Access Firebase
Let's get access to Firebase in our app.
In RecipeTableViewController you must import Firebase to use it.

```
import Firebase
```

Create a connection to your database using the Firebase URL for your app. This is called a reference because they refer to a location in Firebase.
All the JSON keys can also map to a Firebase URL.

```
let ref = Firebase(url: "https://recipes-apierce.firebaseio.com")
```

Reading Data
Create a class called Recipe for your data model.

```
class Recipe {
    var name: String
    var url: String

    init(newname: String, newurl: String){
        name = newname
        url = newurl
    }
}
```

In RecipeTableViewController create an array of recipes to hold our recipe data.

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

We will use viewWillAppear to define an event listener that is called every time data in your Firebase app changes.

```
    override func viewDidAppear(animated: Bool) {
        super.viewDidAppear(animated)
        //set up a listener for Firebase data change events
        //this event will fire with the initial data and then all data
changes
        ref.observeEventType(FEventType.Value, withBlock: {snapshot in
            self.recipes=[]
            //FDataSnapshot represents the Firebase data at a given time
            //gets all the child data nodes
            if let snapshots = snapshot.children.allObjects as?
[FDataSnapshot]{
                for item in snapshots {
                    guard let recipeName = item.value["name"] as? String,
                    let recipeURL = item.value["url"] as? String
                        else {
                            continue
                        }
                    //create new recipe object
```

```
                    let newRecipe = Recipe(newname: recipeName, newurl:
recipeURL)
                    //add recipe to recipes array
                    self.recipes.append(newRecipe)
                }
            }
            self.tableView.reloadData()
        })
    }
```

Update the table view delegate methods as usual.

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

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

    override func tableView(tableView: UITableView, cellForRowAtIndexPath
indexPath: NSIndexPath) -> UITableViewCell {
        let cell = tableView.dequeueReusableCellWithIdentifier("recipecell",
forIndexPath: 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.

Writing Data

Now let's save new recipes and write them to Firebase.

In AddViewController add variables for the recipe name and url and implement prepareForSegue.

```
    var addedrecipe = String()
    var addedurl = String()

    override func prepareForSegue(segue: UIStoryboardSegue, sender:
AnyObject?) {
        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.sourceViewController as! AddViewController
            if source.addedrecipe.isEmpty == false {
                //create new recipe object
                let newRecipe = Recipe(newname: source.addedrecipe, newurl:
source.addedurl)
                //add recipe to recipes array
                recipes.append(newRecipe)
                //create Dictionary
                let newRecipeDict = ["name": source.addedrecipe, "url":
source.addedurl]
                //create a child reference in Firebase where the key value
is the recipe name
                let reciperef = ref.childByAppendingPath(source.addedrecipe)
                //write data to Firebase
                reciperef.setValue(newRecipeDict)
            }
        }
    }
```

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 uncomment `tableView(tableView: UITableView, canEditRowAtIndexPath indexPath: NSIndexPath) -> Bool {`

```
    return true
    }
```

Delete the recipe from Firebase

```
    override func tableView(tableView: UITableView, commitEditingStyle
editingStyle: UITableViewCellEditingStyle, forRowAtIndexPath indexPath:
NSIndexPath) {
        if editingStyle == .Delete {
            let recipe = recipes[indexPath.row]
           //create a child reference in Firebase where the key value is the
recipe name
            let reciperef = ref.childByAppendingPath(recipe.name)
            // Delete the row from Firebase
            reciperef.ref.removeValue()
        }
}
```

Offline

You can enable local persistence for the case where you don't have Internet access.
In the AppDelegate import Firebase and then override init()

```
    override init() {
        super.init()
        Firebase.defaultConfig().persistenceEnabled = 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.
Add needed 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 adopt the UIWebViewDelegate protocol for the class

```
class WebViewController: UIViewController, UIWebViewDelegate
```

Set up the web view's delegate in viewDidLoad()

```
webView.delegate=self
```

Write a method to load a web page.

```
    func loadWebPage(){
        //the urlString should be a propery formed url
        guard let weburl = webpage
            else {
                print("no web page found")
                return
            }
        //creates a NSURL object
        let url = NSURL(string: weburl)
        //create a NSURLRequest object
        let request = NSURLRequest(URL: url!)
        //load the NSURLRequest object in our web view
        webView.loadRequest(request)
    }
```

Implement the two delegate methods that are called when the web page starts and stops loading.

```
    //UIWebViewDelegate method that is called when a web page begins to load
    func webViewDidStartLoad(webView: UIWebView) {
        webSpinner.startAnimating()
    }

    //UIWebViewDelegate method that is called when a web page loads
successfully
    func webViewDidFinishLoad(webView: UIWebView) {
        webSpinner.stopAnimating()
    }
```

In RecipeTableViewController update prepareForSegue() to send the detail view the data it needs.

```
    override func prepareForSegue(segue: UIStoryboardSegue, sender:
AnyObject?) {
        if segue.identifier == "showdetail" {
            let detailVC = segue.destinationViewController as!
WebViewController
            let indexPath = tableView.indexPathForCell(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 new app transport security in iOS 9 to load web pages not available through https you need to add the following to your Info.plist

Key	Type	Value
▼ Information Property List	Dictionary	(16 items)
Localization native development r...	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 creator OS Type code	String	????
Bundle version	String	1
Application requires iPhone envir...	Boolean	YES
Launch screen interface file base...	String	LaunchScreen
Main storyboard file base name	String	Main
▶ Required device capabilities	Array	(1 item)
▶ Supported interface orientations	Array	(3 items)
▶ Supported interface orientations (...	Array	(4 items)
▼ App Transport Security Settings	Dictionary	(1 item)
▼ Exception Domains	Dictionary	(1 item)
▼ myrecipes.com	Dictionary	(3 items)
NSTemporaryExceptionMini...	String	TLSv1.1
NSTemporaryExceptionAllo...	Boolean	YES
NSIncludesSubdomains	Boolean	YES

Don’t forget the launch screen and app icons.