iOS Dev Accelerator Week3 Day2

- Custom App URL Schemes
- OAuth
- NSUserDefaults
- Singleton Pattern

Homework Review

Custom URL Scheme

Registering a custom URL scheme

- Go to your info.plist and choose Add Row
- Select URL types from the drop down (its an array with one dictionary)
- In the item 0 dictionary, add an entry for the key ‘URL Identifier’
- This entry will be the name of the custom url scheme you are defining. It is recommended to ensure uniqueness that you use reverse DNS ‘com.yourCompany.yourApp’
- Add another entry from the drop down called ‘URL Schemes’ which is an array
- For Item 0, give it the string you want to be your URL Scheme. For example, if you type MyApp. The custom URL for your app will be MyApp://

Try it out!

- Launch your app after editing your plist with your URL scheme.
- Now launch Safari from your simulator, and enter in your app's URL into the URL bar.
- Your app should now open.

Parsing information that is passed with URL's

- Often times, and especially with OAuth, your app's url will be called with extra parameters. Typically this is a token or flag.
- There is a method you can implement in your app delegate to intercept these URL calls:

```
- (BOOL)application:(UIApplication *)application
  openURL:(NSURL *)url
  sourceApplication:(NSString *)sourceApplication
  annotation:(id)annotation
```

- We will pass the URL they passed back to us to our network controller for parsing

OAuth

OAuth

- OAuth is is an authentication protocol that allows two apps to interact and share resources.
- There are 3 actors in the OAuth workflow:
  - The service provider: Ex: GitHub
  - The consumer: Ex: Our app
  - The user
- The main benefit is that the user never shares his account and password with the consumer app.

OAUTH Workflow

- Step 1: The user shows intent by attempting an action from the consumer app to the service provider (aka GitHub)
- Step 2: Our app (The consumer) redirects the user to the service provider for authentication
- Step 3:The user gives permission to the service provider for all the actions the consumer should be able to do on his behalf (ex: posting to his timeline, accessing his twitter photos, etc)
- Step 4:The service provider returns the user to the consumer app, with a request token
- Step 5:The consumer now sends the request token, together with its secret key to the service provider in exchange for an authentication token
- Step 6: The user performs actions and passes the authentication token with each call to prove who he is

Callback URL

- The callback URL is just the callback entry point of your app.
- The service provider performs an HTTP redirect to get the user back to the consumer app.
- In addition to the URL of your app, the callback url will have the authorization code appended to it. It is up to your app to parse this out and use it in completing the OAuth workflow.
- All apps can be launched from either another app or from the browser itself.

Demo

NSUserDefaults

- “NSUserDefaults allows an app to customize its behavior based on user preferences”
- Think of it as an automatically persisting dictionary or plist that is easily modified in code.
- Use the standardUserDefaults class method to return the shared defaults object.
- Setting values inside of it is as easy as these methods:
  - setBool:ForKey:
  - setObject:ForKey:
  - setInteger:ForKey:
- Call synchronize() after saving key in defaults.

Demo

Singleton Pattern

There can only be one

Singleton Pattern

- “In software engineering, the singleton pattern is a design pattern that restricts the instantiation of a class to one object”
- Apple uses the singleton patter a lot in Cocoa programming
- Some examples are UIApplication.sharedApplication(), and NSBundle.mainBundle()
- In our app, it makes sense to have a singleton of our network controller, so we don’t have to worry about passing a reference to it around, and also so we know there will only ever be one we will use

Singleton in swift

- There still isn't an established best practice way in Swift to write singletons, but this way seems to work just fine, and was approved by an apple dev in their forums

```
class Singleton {
    class var sharedInstance : Singleton {
        struct Static {
            static let instance : Singleton = Singleton()
        }
        return Static.instance
    }
}
```

Computed type property

Nested Struct

Static Properties

Returns the computed property

Holy Crap

Computed Properties

- The properties you have implemented so far have all been stored properties
- That is, they have data they store in memory
- Classes, structs, and enumerations can also define computed properties, which do not actually store a value
- Instead, they provide a getter and an optional setter to retrieve and set other properties and values indirectly

Computed Properties

```
struct Point {
    var x = 0.0, y = 0.0
}
struct Size {
    var width = 0.0, height = 0.0
}
struct Rect {
    var origin = Point()
    var size = Size()
    var center: Point {
        get {
            let centerX = origin.x + (size.width / 2)
            let centerY = origin.y + (size.height / 2)
            return Point(x: centerX, y: centerY)
        }
        set(newCenter) {
            origin.x = newCenter.x - (size.width / 2)
            origin.y = newCenter.y - (size.height / 2)
        }
    }
}
```

Read/Write Computed Property

```
struct Cuboid {
    var width = 0.0, height = 0.0, depth = 0.0
    var volume: Double {
        return width * height * depth
    }
}
```

Read only Computed Property

Demo