Chapter X

Networking in iOS

IV.1. Networking in iOS

1. Using Webview

UIWeb View can load local html file and contents from url as in

e.g:

```
myWebView.scalesPageToFit = true

if let url = NSURL(string: "http://www.google.com") {

let urlRequest = NSURLRequest(URL: url)

myWebView.loadRequest(urlRequest)

}

// put some delegation to detect the event

myWebView.delegate = self

func webViewDidFinishLoad(webView: UIWebView) {

self.mySpinner.stopAnimating()
}
```

2. Downloading Image from web

```
let url = NSURL(string:

"https://picjumbo.imgix.net/HNCK2481.jpg?q=40&w=1650&sharp=30")

let task = NSURLSession.sharedSession().dataTaskWithURL(url!) { (data, response, error) ->
        Void in

if error != nil {
    print(error)
} else {
    if let validData = data {

        dispatch_async(dispatch_get_main_queue(), { () -> Void in
         self.yourimageView.image = UIImage(data: validData)
                })
        } else {
                print("invalid data")
                }

        } //if

} //task

task.resume()
```

3. Concurrency (MultiThreading)

In order to create a fluid and smooth experience for your users, asynchronous operations are required. To be responsive, even during long-running

operations such as network access or processing-intensive operations, three approaches to threading in iOS and OS X: **NSThread, Grand Central Dispatch, and NSOperationQueue** are used

1) Using Thread to run instantly:

```
Thread.detachNewThreadSelector("ThreadMethod" , toTarget: self, withObject: nil)
```

2) Using Thread to run with controllable properties:

```
var thread = Thread(target: myInstance, selector: "threadMethod:", object: nil)
 thread.stackSize = 16000
 thread.threadPriority = 0.75
thread.start()   //Check Note*
```

3) Using GCD to manage the thread

Typically, GCD is used when you want the main thread to continue while other tasks run in parallel. Your application submits tasks to a FIFO queue, managed by GCD, using a closure. GCD has three types of queues:

Main: Tasks run sequentially in FIFO order on the main thread of the application.

Concurrent: Tasks execute in FIFO order, but run in parallel and can finish in any order.

Serial: Tasks execute sequentially in FIFO order.

If you have a large number of tasks to complete, concurrent queues are the best option.

If the tasks must be executed in a designated order, a serial queue is the best option.

The main thread should be used for any user-interface updates.

One of the most common GCD patterns is to perform work on a global background queue and update the UI on the main queue as soon as the work is done

Queue has four prioritised value

* DISPATCH_QUEUE_PRIORITY_HIGH: .userInitiated
* DISPATCH_QUEUE_PRIORITY_DEFAULT: .default
* DISPATCH_QUEUE_PRIORITY_LOW: .utility
* DISPATCH_QUEUE_PRIORITY_BACKGROUND: .background
*
* To get main queue for user interface updates , use dispatch_get_main_query( )

e.g

```
label = UILabel(frame: CGRect(x: 0.0, y: 0.0,
        width: 200.0, height: 20.0))
```

```
        label.center = self.view.center
        label.text = "Loading..."
        self.view.addSubview(label)
        // queue a long running task
          DispatchQueue.global(qos: .background).async {
                                //Do something
                               self.longRunningTask( )
}
func longRunningTask() {
                    sleep(3)
                    DispatchQueue.main.async {
           self.label.text = "Complete."
       }
          }
```

e.g.. Run after a period

```
   let delay = DispatchTime.now() + 3


       DispatchQueue.main.asyncAfter(deadline: delay) {
            print("Run after three sec")
       }
```

4) Using Operations and OperationQuery

Having a collection of tasks to execute asynchronously, and those tasks have dependencies that must be resolved before they can execute in additional to to cancel, suspend, or re-use a task, Operation is in play.

eg.

```
    let oq1 = OP("one")
     let oq2 = OP("two")
     oq2.addDependency(oq2)
     oq2.queuePriority = .high

     let oq = OperationQueue()
      // oq.addOperation(oq2)
     oq.addOperation(oq1)
     oq.addOperation(oq2)
```

eg.

```
override func viewDidLoad() {
       super.viewDidLoad()

          let mainQ = OperationQueue.main
          let oq = OperationQueue()
         oq.maxConcurrentOperationCount   = 3
         oq.addOperation {
```

```
        for i in 0..<10000
        {
            print("Running for \(i)")
            mainQ.addOperation {
                //Do UI Update
            }
        }
}
```

Note:OperationsQueue are built on top of GCD. Apple recommends starting with this highest-level abstraction and then choosing lower levels of thread management if necessary. The reasons for this include better performance and more control over memory utilization.

4. Handling JSON

Using JSONSerialization.JSONObjectWithData(rawdata, options: []) and JSONSerialization.dataWithJSONObject, we can decode and encode to NSDictionary vice versa.

Third Party handling use: https://github.com/SwiftyJSON/SwiftyJSON

5. Understanding & Testing HTTP Verbs

1. HTTP Verbs Basic

The primary or most-commonly-used HTTP verbs (or methods, as they are properly called) are **POST, GET, PUT, PATCH, and DELETE.** These correspond to create, read, update, and delete (or CRUD) operations, respectively. Others are less frequent used and OPTIONS and HEAD are used more often than those.

To Test Networking, Set NSAppTransportSecurity key in app's plist file. And set NSThirdPartyExceptionAllowsInsecureHTTPLoads to YES if need, under website name key.

2. Making Request: Async

Using async URL request of NSURLSession, we can perform request as:

public func dataTaskWithRequest(request: NSURLRequest, completionHandler: (NSData?, NSURLResponse?, NSError?) -> Void) -> NSURLSessionDataTask

GET e.g.

GET cont'd

```
let postEndpoint: String = "http://jsonplaceholder.typicode.com/posts/1"
guard let url = NSURL(string: postEndpoint) else {
  print("Error: cannot create URL")
return
}
let urlRequest = NSURLRequest(URL: url)
let config = NSURLSessionConfiguration.defaultSessionConfiguration()
let session = NSURLSession(configuration: config)
```

```
let task = session.dataTaskWithRequest(urlRequest, completionHandler: { (data: NSData?, response:
NSURLResponse?, error: NSError?) in
// this is where the completion handler code goes
 guard let responseData = data else {
          print("Error: did not receive data")
          return }
guard error == nil else {
          print("error calling GET on /posts/1") print(error)
          return }
print("Now Process Data")
 do   {
let results = try NSJSONSerialization.JSONObjectWithData(data!, options: .AllowFragments) as!
NSDictionary
              print(results)
              print("\(results["id"])")
          }
})
task.resume()
```

POST

```
        let config = NSURLSessionConfiguration.defaultSessionConfiguration()
        let session = NSURLSession(configuration: config)
        let endPointURL = NSURL(string: "http://jsonplaceholder.typicode.com/posts")!
        let myRequest = NSMutableURLRequest(URL: endPointURL)
        myRequest.HTTPMethod = "POST"
        let myPost = ["userId":1,"title":"Test Post","body":"Testing My New Post "]
do
      {
         //flat the data!
         try myRequest.HTTPBody = NSJSONSerialization.dataWithJSONObject(myPost,
        options: [])
```

eg.

```
         //Prepare the task
         let task = session.dataTaskWithRequest(myRequest, completionHandler: {  (data,
        respone, error) in
        guard error == nil else {
                  print("error calling POST on /posts") ;print(error) ; return }
         guard data != nil else {
                   print("Data nil on endpoint") ;print(error) ; return }
           print("Process Data")
            do { let results = try NSJSONSerialization.JSONObjectWithData(data!,
                  options: .AllowFragments) as! NSDictionary
                  print(results)
                  print("\(results["id"])") }
            catch
            { print("Error converting json") }
         })
         //Let ignite!
         task.resume()
      }
      catch { }
```

PUT

Note: Set the URL Endpoint which is accepting updating with parameter HTTPMethod = "PUT". The data want to update is as in POST example

DELETE

Note: Set the URL Endpoint which is accepting deleting objects with parameter HTTPMethod = "DELETE"

Note:use Thirdparty Swift Networking Framework: Alamofire to make efficient coding practise

3. Authentication

Terms to Study: Authentication , Authorisation, Password-based authentication , Multifactor authentication (MFA) , Anonymous authorisation, Federated identity provider , Authentication Protocol: OAuth, OpenId , SAML

6. Third Party Frame Work and Service

Popular third party framework for authentication: Simple Username & password method to Facebook, twitter, Google+ , Github etc. And backend service: Firebase, Open sourced Parse server, Kinvey, etc.Each framework has its own documentation and as they are evolving overtime , please read their user guide carefully in term of supported OS, language and version.

7. Exercise

1. Create an app that able to download image from internet and set to Image view.And then cache on the disk.
2. Create Login Screen and then use Firebase as a backend as a service to authenticate user and store user data.