University of Regina

CS 455/855
Mobile Computing

Basic Networking

Dr. Orland Hoeber
orland.hoeber@uregina.ca
http://www.cs.uregina.ca/~hoeber/cs455/2018F

Readings

- UIWebView
- URL Session Programming Guide

Networking & Mobile Computing

- One of the things that makes mobile computing so useful is the ability to do networking
  - if we take networking away from our mobile devices, smart phones become a lot less smart
  - networking is critically important in many of the things we do with our mobile devices
    - email
    - web browsing
    - Facebook/Twitter/etc.
    - maps and GPS
    - Skype/FaceTime
    - Google Translate, Siri, Shazam
    - accessing third-party services via RESTful APIs
    - many games
  - we have come to rely on our mobile devices to not only be small and powerful, but to have full access to network resources at all times (even as we move with the device)

Getting Started with Networking

- There are a wide range of networking APIs in the iOS SDK
  - Cocoa Touch layer
    - UIKit framework – UIWebView
  - Core Services layer
    - Foundation frameworks – Swift abstractions for various APIs in the CoreFoundation frameworks, e.g. URL, URLConnection, URLRequest, URLSession, etc.
    - GameKit framework – P2P networking
    - CFNetwork framework (in C) – CFHost, CFHTTPMessage, CFHTTPStream, CFFTPStream, CFNetServices
    - Core Foundation framework (in C) – CFSocket
  - Core OS layer – BSD socket (in C)
- These APIs provide transparent access to underlying hardware
  - For the most part, don't get to choose between WiFi and cellular

Top Down

- Before you start, decide:
  - how much abstraction or direct control you want?
  - what language you prefer to use (C, Objective-C, or Swift)?
  - what is the networking model (client-server or peer-to-peer)?
- For example
  - Download and display web pages:
    - UIWebView
  - download content using URLs
    - CFURL, NSURL, URL, URLSession
  - Interacting with web or FTP servers
    - CFHTTPStream, CFFTPStream, URLSession
  - Communicating using known or custom protocols
    - CFSocket
  - Platform-independency
    - BSD socket

Networking Overhead

- Scenario
  - App sends an HTTP request and receives an HTTP response
- Actions taken by your device
  - Wake up antenna
  - Send DNS query to translate hostname of Web server to its IP address
  - Receive DNS response
  - Send ARP query to translate IP of gateway to its MAC address
  - Receive ARP response
  - Send TCP SYN segment to Web server
  - Receive TCP SYN-ACK
  - Send HTTP query as payload of upstream TCP segment
  - Receive TCP ACK from server
  - Receive HTTP response as pay of downstream TCP segment
  - Send TCP ACK to server
  - Send TCP FIN to request to close TCP connection
  - Receive TCP FIN ACK
  - Receive TCP FIN request from server
  - Send TCP FIN ACK to server

General Principles of Networking

- Accessing the network is one of the most power-hungry operations on mobile devices
- You should:
  - connect to an external network only when needed (do not poll the server)
  - transmit the smallest amount of data needed to do the job
  - transmit data in bursts, rather than spreading packets out over time
  - use Wi-Fi rather than baseband radio when possible
  - if you are going to use the location service, enable it when you need it and disable it again as soon as you can

UIWebView

- UIWebView is essentially the same core web browser as in Mobile Safari (WebKit)
  - encapsulates a number of web technologies all in one convenient location
    - URL resolution
    - HTTP requests and responses generation
    - TCP/IP communication
    - HTML/CSS rendering
    - JavaScript run-time engine
    - management of browsing history (back and forward lists)
  - some of these things you have control over and can manipulate; others just happen for you

Programming UIWebView

- The primary method for loading a web page into the UIWebView is to send the object the loadRequest message
  - requires that we first construct two supporting objects:
    - URL
    - URLRequest

```
let urlPath = "https://www.google.ca/"
let url = URL(string: urlPath)!
let request = URLRequest(url: url)

self.webView.loadRequest(request)
```

Secure vs. Insecure Connections

- If you attempt to connect to http rather than https, you will get the following warning:

```
App Transport Security has blocked a cleartext HTTP (http://) resource load since it is insecure. Temporary exceptions can be configured via your app's Info.plist file.
```

- Either use https, or make an entry into the info.plist file for your app

UIWebView Methods

- Useful methods
  - load a web page:
    - loadRequest (URLRequest)
    - load (data, mimeType, textEncodingName, baseURL)
    - loadHTMLString (string, baseURL)
  - browser control:
    - reload()
    - stopLoading()
    - goBack()
    - goForward()
  - interaction
    - stringByEvaluatingJavaScript (from)

UIWebKit Properties

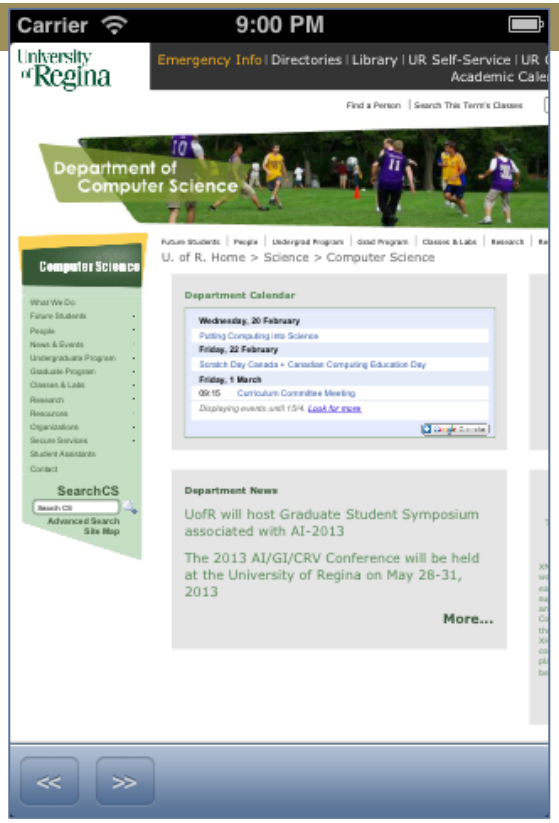
- Useful properties of the object
  - loading status:
    - URLRequest? request
    - Bool isLoading
  - properties of the browsing history (read-only)
    - Bool canGoBack
    - Bool canGoForward
  - rendering properties
    - UIDataDetectorTypes dataDetectorTypes
    - BOOL scalesPageToFit

UIWebViewDelegate

- You can make the view controller conform to the UIWebViewDelegate protocol in order to capture important events in the UIWebView process
  - webView (webView, shouldStartLoadWithRequest, navigationType)
    - monitor the types interaction being done with the browser (e.g., links, form submissions, etc.)
  - webViewDidStartLoad (webView)
    - start an activity indicator
  - webViewDidFinishLoad (webView)
    - stop the activity indicator
  - webView (webView, didFailLoadWithError)
    - capture network or resource errors

Example: Mini Web Browser

- Simple web browser consisting of:
  - UIWebView
  - UIToolBar
  - UIBarButtonItem x 2
  - UIActivityIndicatorView



- Modify the following in the view controller
  - viewDidLoad
  - IBActions for the back and forward buttons
  - method for the UIWebViewDelegate protocol

Making HTTP Requests

- Sometimes we do not want to automatically load the results of an HTTP request in a UIWebView
  - download data for use in other controls (e.g., images)
  - access a web resource and parse out the information needed
  - access a web service (e.g., using a RESTful API)

  - create the URL and URLRequest in the same way
  - but instead of passing it to the UIWebView, we use it with an URLSession

URLSession

- The URLSession object is a multi-purpose class that supports many different URL schemes (ftp, file, http, https)
- It is inherently asynchronous
  - use a completion handler block, which is executed when the transfer is complete
  - set a delegate class, which can capture the state of the connection
- Tasks are done within the context of the session
  - session.dataTask (with, completionHandler)
  - session.downloadTask (with, completionHandler)
  - session.streamTask (withHostName, port)

Example

```
let sessionConfig = URLSessionConfiguration.default
let session = URLSession(configuration: sessionConfig)
let task = session.dataTask(with: url){(data, response, error) in
    // completion handler block
    if (error != nil) {
        print(error!)
    } else {
        print(data!.description)
        if let data = data {
            if let html = String(data: data, encoding:
                 String.Encoding.utf8) {
                print (html)
            }
        }
    }
}
task.resume()
```

Don't Block the Main Thread

- Non-trivial apps may have long processing times or blocked IO
  - parsing the data
  - network connection problems
- When this happens, it is vitally important to not block the main thread
  - the UI still needs to respond to user interactions
  - this won't be possible if the main thread is blocked
- Options
  - synchronous requests
    - do not use these!
  - asynchronous requests
    - allow the response to be processed later
      - completion handler block
      - using a delegate object
  - multi-threaded programming
    - the tasks from the URLSession are already pushed off the main thread

Parsing the Response

- How you deal with the data that comes back will depend on the data type and what needs to be done with the data
  - general process:
    - convert the Data response to an appropriate data type
    - do what is necessary with that data
  - examples:
    - simple text, HTML
      - covert to String
      - parse string manually or using a regular expression (RegularExpression)
    - XML
      - load into XMLParser
      - parse using XMLParser methods
    - JSON
      - process the Data using JSONSerialization
      - produces a data structure of nested Array and/or Dictionary objects
    - image
      - convert directly to Image class instance

Response Formats

- Generally, what you do will depend greatly on whether you have control over the response formats:
  - no control
    - do what is necessary to process the data
    - may need to modify your app if the response formats change
    - make sure you document your code
  - full control
    - use structured data formats such as JSON (or XML)
    - keep in mind that you are essentially creating a data communication protocol upon HTTP
      - requests using REST formats
        - stateless
        - use verbs in the URL to describe the action (get, put, etc.)
      - responses using JSON

Building Your Own REST API

- There are many different frameworks available for building your own REST API
  - Spark Framework
    - Java
    - http://sparkjava.com
  - Restify
    - Server-Side JavaScript (node.js)
    - http://restify.com
  - In simple cases, your data may be static and hard-coded into the API or read from a file
  - In more complex cases, you should use a database that can store the data in native JSON format (e.g., MongoDB)

Using Delegates

- Rather than writing completion handler blocks as part of the asynchronous requests, a better approach to managing the network communication is to use the delegate protocols
  - URLSessionDelegate
  - URLSessionTaskDelegate
  - URLSessionDataDelegate

- Why?
  - separates the request from the handling of the request
  - allows for more detailed tracking of what is happening (useful for debugging)
    - check the documentation for the list of methods that can be implemented here and what they are used for

Example: Parsing JSON Data

- This example will:
  - make an asynchronous request
  - use a delegate protocol
  - show the raw data (JSON object)
  - parse the JSON object to extract specific data

- It will use a fake REST API
  - http://jsonplaceholder.typicode.com

Key Take-Aways

- When creating the URLSession object instance, you have to specify the delegate class (usually, self)
- Creating the task looks much simpler, since there is no completion handler block
- The URLSessionDataDelegate protocol methods will automatically be running on a new thread
- If you want to access the interface, you need to run the code back on the main thread:

```
DispatchQueue.main.async {
    self.rawText.text = dataString
}
```

Networking and MVC

- Do the two examples provided conform to the MVC design pattern?
  - MiniWebBrowser
    - UIWebView is a view
    - the code in the controller is to configure the view and to support interaction between views (buttons and web view)
    - this is ok
  - REST-JSON-Tester
    - the URLSessionDataDelegate methods of the controller are actually data processing methods
    - should be encapsulated in a model class
    - the controller would then coordinate the data transfer between this object and the view objects

Homework

- Catch up on your reading

- Project Milestone 2: Project Update
  - due Nov 16
- Short Paper #2 (CS 855)
  - due Nov 23