ATLS 4320: Advanced Mobile Application Development

Week 6: JSON

JSON

JavaScript Object Notation (JSON) is a language independent, human-readable data format used for transporting data between two systems. http://json.org/

Supported by every major modern programming language including JavaScript, Swift, and Java. JSON has a limited number of data types: string, boolean, array, object/dictionary, null and number.

JSON is built on two structures:

- A collection of name/value pairs stored as an object, record, struct, dictionary, hash table, keyed list, or associative array in various languages
  - An object in in curly brackets { }
  - Format = "name: value"
  - Name/value pairs are separated by a comma ,
- An ordered list of values are stored as an array, vector, list, or sequence in various languages
  - An array is in square brackets [ ]
  - Values are separated by a comma

JSON:
https://developer.nps.gov/api/v1/campgrounds?stateCode=co&api_key=KuzYWrSKHbuz6CBO8oc0pX35CeljxNSfgxane4lH
DOCS: https://www.nps.gov/subjects/developer/api-documentation.htm
Note: To display the JSON formatted in Chrome you'll need to install an extension.

iOS Networking

There are three main classes you need to know about in order to handle networking in iOS:

1. URLRequest encapsulates information about a URL request.
   a. Used by URLSession to send the request
2. URLSession coordinates the set of requests and responses that make up a HTTP session
   https://developer.apple.com/documentation/foundation/urlsession
- URLSession has a singleton shared session for basic requests
  https://developer.apple.com/documentation/foundation/urlsession/1409000-shared
- You can create a URLSessionTask to retrieve the contents of a URL using URLSession.shared.dataTask(with:completionHandler:)

https://developer.apple.com/documentation/foundation/urlsession/1407613-datatask

   a. Requests a URLRequest
   b. Completion handler to call when the request is complete and successful
      i. Data – holds the downloaded data if successful
      ii. Response - An object that provides response metadata, such as HTTP headers and status code. If you are making an HTTP or HTTPS request, the returned object is actually an HTTPURLResponse object.

1. The HTTP status code is stored in the HTTPURLResponse statusCode property
2. HTTP status codes
    a. https://en.wikipedia.org/wiki/List_of_HTTP_status_codes
    b. 200 is OK

iii. Error – an error object if the request fails, data will be nil

c. After you create the task, you must start it by calling its resume method
d. Returns the new session data task.

3. URLSessionTask performs the actual transfer of data through the URLSession instance. It has different subclasses for different types of tasks. URLSessionDataTask is used for the contents of an URL.

So, making an HTTP request in iOS boils down to:

- Create and configure an instance of URLSession, to make one or more HTTP requests
- Optionally create and configure an instance URLRequest for your requests. Needed only if you need to send additional information in the header of the request.
- Start a URLSessionDataTask through the URLSession instance.

JSON in iOS

Once the JSON has been downloaded successfully we need to parse the data and save it in our data model

- The DispatchQueue class manages the execution of work items. Each work item submitted to a queue is processed on a pool of threads managed by the system.
- The DispatchQueue.main.async{} method will submit requests to be run asynchronously on other threads. You should use this to parse the JSON asynchronously. It's really important to only use the main queue for the UI, otherwise the other tasks make the app unresponsive and slow as it's waiting on the other tasks — **there is no telling how long an HTTP request and subsequent parsing could take!**
- Instead of the PropertyListDecoder we've been using for plists, we'll use the JSONDecoder which is the JSON equivalent. Just as with property lists, we need the property names of our struct to match the key name in the JSON file.

API key: KuzYWrSKHbuz6CBO8oc0pX35CeljxNSfgxane4IH
NPS News
File -> New Project
Single View App

Setup

Go to Main.storyboard and **embed** the view controller into a navigation controller (select view controller, go to Editor->Embed In->Navigation Controller in top menu)

Drag a label, picker, and button on to the view controller.
Change the label text to "Select a state" or something
Set the datasource and delegate for the picker to the View Controller

Change the button text to “Search”.

Create Model

Look at JSON data from the /campgrounds endpoint
Look at the items in the data array, they are all objects representing a single
We’re going to use a struct to represent a petition. We’ll pick out the data items in the value of results that we’re interested in.
File -> New -> File -> Swift File
Name the file *Campsite*

Add struct

```
struct Campsite: Decodable {
    let name: String
    let description: String
    let directionsoverview: String
}
```

To use a JSONDecoder we need a data structure with a property named data and the value an array of campsites.

```
struct CampsiteData: Decodable {
    var data = [Campsite]()
}
```

Data Model Controller

Create a file for our class
File -> New -> File -> Swift File
Name file *CampsiteDataController*

Create an enum for our errors

```
enum JsonError: Error {
    case BadURL
    case BadResponse
    case CouldNotParse
}
```

Declare the class and **add** methods to fetch raw data, parse Json also **add** class variables to store a notification closure and the parsed data

```
class CampsiteDataController {
    //stores all of the campsites from the most recent response
    var currentCampsites = CampsiteData()
    //closure to notify the view controller when the json has been loaded and parsed
```

```
    var onDataUpdate: ((_ data: [Campsite]) -> Void)?

    //makes the http request based on stateCode parameter
    func loadJson(stateCode: String) throws {
        //construct URL by interpolating the state code into
        let urlPath =
"https://developer.nps.gov/api/v1/campgrounds?stateCode=\(stateCode)&api_key=KuzYWrSKHbuz
6CBO8oc0pX35CeljxNSfgxane4IH"

        //use a guard statement with conditional unwrapping to make sure the url is valid
        guard let url = URL(string: urlPath) else {
            throw JsonError.BadURL
        }
        let group = DispatchGroup()
        group.enter()


        //valid url so make the request and give it a completetion handler closure
        let session = URLSession.shared.dataTask(with: url, completionHandler: {(data,
response, error) in
            //downcase to URLResponse since we made and https request
            let httpResponse = response as! HTTPURLResponse

            //get the status code
            let statusCode = httpResponse.statusCode

            //make sure we got a good response
            guard statusCode == 200 else {
                print("file download error")
                return
            }
            //download successful
            print("download complete")
            //parse json asynch
            DispatchQueue.main.async {self.parseJson(rawData: data!)}
        })

        //must call resume to run session and execute request
        session.resume()
    }

    //parses the raw http response and notifies the view controller
    func parseJson(rawData: Data)  {
        do {
            //try to decode the response
            let parsedData = try JSONDecoder().decode(CampsiteData.self, from: rawData)
            //clear out old data
            currentCampsites.data.removeAll()
            //add all the campsite entries to our class property that stores the current
```

```
campsites
            for campsite in parsedData.data {
                currentCampsites.data.append(campsite)
            }
        } catch {
            //something went wrong parsing the data -- throw error!
            print("json error")
            print(error.localizedDescription)
        }
        print("parsejson done")

        //pass data back to requesting object
        onDataUpdate?(currentCampsites.data)
    }
}
```

Setup View Controller

Go to `ViewController.swift` and **add** conformance to `UIPickerViewDelegate` and `UIPickerViewDataSource` protocols

```
class ViewController: UIViewController, UIPickerViewDelegate,
UIPickerViewDataSource {
```

Be sure to **add** the required stubs based on the error

Go to Main.storyboard and **create** an action connection for the search button called *searchCampsites* by **ctrl-click and dragging** from the *button* to `ViewController.swift` and **selecting** *action* from the connection drop down

```
@IBAction func searchCampsites(_ sender: Any) {}
```

Add the following variables to the top

```
    //list of states for picker
    let stateOptions = ["CO", "CA", "FL", "NM", "UT"]
    //holds the currently selected state
    var selectedState = String()
    //instance of data controller
    var campsiteDC = CampsiteDataController()
    //local copy of data
    var data = [Campsite]()
```

Add the following lines to viewDidLoad() to give the selected state an initial value and tell the data controller which function to call when it gets data successfully

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

    //set the initial state
    selectedState = stateOptions[0]

    //set the function to notify when response is complete
```

```
    campsiteDC.onDataUpdate = {[weak self] (data:[Campsite]) in
self?.searchDone(campsites: data)}
}
```

The [weak self] portion of the closure is called a capturing list.

weak: Strong references are the default when you assign a class instance to a variable. Closures also create strong references when they are assigned to a property. Strong references are not deallocated until all the objects with a reference are destroyed. We don't need a strong reference to the closure since we already have one at the class level.

self: This captures the context of self at the level where the closure or function is defined (refers to our ViewController class outside of the closure) and passes that in to the body. Since we want access to the class methods inside the body we need to do this.

Implement the `searchCampsites()` method to make the request to load data based on selectedState

```
//executes the search
@IBAction func searchCampsites(_ sender: Any) {
    do {
        //start loading the data
        try campsiteDC.loadJson(stateCode: selectedState)
        //block user events and show spinner while fetching the campsites
        let alert = UIAlertController(title: nil, message: "Searching in
\(selectedState)...", preferredStyle: .alert)

        let loadingIndicator = UIActivityIndicatorView(frame: CGRect(x: 0, y: 5, width:
50, height: 50))
        loadingIndicator.hidesWhenStopped = true
        loadingIndicator.style = UIActivityIndicatorView.Style.medium
        loadingIndicator.startAnimating();

        alert.view.addSubview(loadingIndicator)
        present(alert, animated: true, completion: nil)
    } catch {
        print(error)
    }
}
```

Add the method to be notified when data is done loading/parsing

```
//called when the json data has been parsed
//trigger segue and set local data
func searchDone(campsites: [Campsite]) {
    //dismiss the loading alery
    dismiss(animated: true, completion: nil)

    //set data property
    data = campsites
}
```

Configure data source and delegate methods for the picker

```
func numberOfComponents(in pickerView: UIPickerView) -> Int {
    return 1
}

func pickerView(_ pickerView: UIPickerView, numberOfRowsInComponent component: Int) ->
Int {
    return stateOptions.count
}

func pickerView(_ pickerView: UIPickerView, titleForRow row: Int, forComponent component:
Int) -> String? {
    return stateOptions[row]
}

func pickerView(_ pickerView: UIPickerView, didSelectRow row: Int, inComponent component:
Int) {
    selectedState = stateOptions[row]
}
```

At this point we could **run** the app and if we **add** print statements we can **verify** that our data is getting loaded and parsed properly

Results Controller

Next, we'll use a table view to display the campsite names in a list.
Create new file for VC
File->New->File->Cocoa Touch Class
Name it *ResultsViewController* and be sure to **subclass** from *UITableViewController*
Save
Add a class property (array of Campsite) to store the campsites

```
class ResultsViewController: UITableViewController {

    var results = [Campsite]()
}
```

Drag a new Table View View Controller in storyboard
Change Cell reuse to "*CampsiteCell*"
Change cell type to *Basic*
Change accessory to "*Disclosure Indicator*"
Set class for new `ResultsViewController.swift` in Identity Inspector
Add segue by selecting the view controller (circle with square) and ctrl-click drag to table view controller. **Use** the "*Show*" segue
Change Identifier to "*Search Results*"

Add to `searchDone()` in root `ViewController.swift`

```
performSegue(withIdentifier: "SearchResults", sender: nil)
```

Add `prepare()` method to pass data before segueing

```
override func prepare(for segue: UIStoryboardSegue, sender: Any?) {
    //check id of segue
    if segue.identifier == "SearchResults" {
        //downcast destination vc
        let resultsVC = segue.destination as! ResultsViewController
        //set the title
        resultsVC.title = "\(selectedState) Campsites"
        //pass the data
        resultsVC.results = data
    }
}
```

In ResultsViewController.swift **implement** the TableView DataSource methods to display the results

```
override func numberOfSections(in tableView: UITableView) -> Int {
    // #warning Incomplete implementation, return the number of sections
    return 1
}

override func tableView(_ tableView: UITableView, numberOfRowsInSection section: Int) ->
Int {
    // #warning Incomplete implementation, return the number of rows
    return results.count
}


override func tableView(_ tableView: UITableView, cellForRowAt indexPath: IndexPath) ->
UITableViewCell {
    let cell = tableView.dequeueReusableCell(withIdentifier: "CampsiteCell", for:
indexPath)

    //set the title of cell label
    cell.textLabel!.text = results[indexPath.row].name

    return cell
}
```

Run the app and checkout our segue and list

Detail View Controller

The last step is to add a detail view controller to display the campsite description and directions.
Create a new file for the VC
File ->New->File->Cocoa Touch Class
Name it *DetailViewController* and be sure to **subclass** *UIViewController*

Drag a new regular View Controller onto the storyboard
Set the class in Identity Inspector to *DetailViewController*
Create a segue by selecting the cell in the TableView and **ctrl-click drag** to the new view controller.
Give it the segue identifier "*DetailSegue*"
Choose a "*Show*" segue from the popup list

Drag 4 labels onto the new view
Change two of them to titles (Description and Directions). Leave the other two
Add auto-layout constraints. **Pin** to *top* and *leading* for the title labels. **Pin** to *each side* and **set** the width to less than or equal to for the text labels.
Change Number of Lines to 0 for the text labels in the Identity Inspector and **set** Line Break to *Word Wrap*

Add outlet connections for the text labels to `DetailViewController.swift`. I called mine **descriptionLabel** and **directionsLabel**

In DetailViewController.swift **add** class variables to store the data for the directions and description and set the label text in the `viewDidAppear()` method

```
class DetailViewController: UIViewController {

    //data variables
    var siteDescription = String()
    var siteDirections = String()

    //connections
    @IBOutlet weak var descriptionLabel: UILabel!
    @IBOutlet weak var directionsLabel: UILabel!

    //set the label text
    override func viewWillAppear(_ animated: Bool) {
        descriptionLabel.text = siteDescription
        directionsLabel.text = siteDirections
    }
}
```

Add the following method to ResultsViewController.swift to pass along the appropriate data before segueing

```
override func prepare(for segue: UIStoryboardSegue, sender: Any?) {
    if segue.identifier == "DetailSegue" {
        //get the selected campsite
        let idx = tableView.indexPath(for: sender as! UITableViewCell)!.row
        let selectedCampsite = results[idx]
        //get reference to detail vc
        let detailVC = segue.destination as! DetailViewController

        //set detail vc properties
        detailVC.title = selectedCampsite.name
        detailVC.siteDirections = selectedCampsite.directionsoverview
        detailVC.siteDescription = selectedCampsite.description
    }
}
```

Run the app and make sure it all works