

ATLS 4320/5320: Advanced Mobile Application Development

Week 6: JSON

JSON

JavaScript Object Notation(JSON) is a language independent data format used to store and exchange data. <http://json.org/>

Supported by every major modern programming language including JavaScript, Swift, and Java

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 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 ,

Show JSON example

XML: <https://api.whitehouse.gov/v1/petitions>

JSON: <https://api.whitehouse.gov/v1/petitions.json?limit=100>

JSON in iOS

<https://developer.apple.com/swift/blog/?id=37>

Retrieve the contents of a URL

- In iOS you can download a JSON file from a server using URLSession.shared.dataTask(with: URLRequest, completionHandler: @escaping (Data?, URLResponse?, Error?) -> Void)
<https://developer.apple.com/reference/foundation/urlsession>
<https://developer.apple.com/reference/foundation/urlsession/1407613-datatask>
- Completion block
 - Success: the data parameter will hold the data downloaded, the error parameter will be nil
 - Fail: the error parameter will hold the error, the data parameter will be nil
 - The response is a NSHTTPURLResponse object
 - The HTTP status code is stored in the NSHTTPURLResponse statusCode property
 - HTTP status codes https://en.wikipedia.org/wiki/List_of_HTTP_status_codes
 - 200 is OK
- After you create the task, you must start it by calling its resume method

Once the JSON has been downloaded successfully we are ready to parse the data

- We will use DispatchQueue.main.async{} to parse the JSON asynchronously in the main queue. 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.
 - Block will parse the JSON

You can use the Foundation framework's [JSONSerialization](#) class to convert JSON into Swift data types like Dictionary, Array, String, Number, and Bool.

<https://developer.apple.com/reference/foundation/jsonserialization>

The JSONSerialization class method jsonObject(with:options:) returns a value of type Any and throws an error if the data couldn't be parsed.

<https://developer.apple.com/reference/foundation/jsonserialization/1415493-jsonobject>

- data: JSON data object (Data)
- options:JSONSerialization.ReadingOptions
<https://developer.apple.com/reference/foundation/jsonserialization.readingoptions>
- This method could throw an error

Although valid JSON may contain only a single value, a response from a web application typically encodes an object or array as the top-level object. To get a Dictionary value from a JSON object type, conditionally cast it as [String: Any]. To get an Array value from a JSON array type, conditionally cast it as [Any] (or an array with a more specific element type, like [String]).

In Swift 4 we could use the Codable protocol but at this time it does not support the type Any which we will need in our example as the data types for the values are not always the same.

petitions

File | New Project
Master-Detail App
iPhone

Setup

Go into the Main storyboard and in the table view change the table view cell to the style subtitle. Change the title of the master scene to White House Petitions.

In MasterViewController.swift delete the insertNewObject() method entirely, delete everything from viewDidLoad() except the call to super.viewDidLoad(), delete or comment out the table view's commitEditingStyle and canEditRowAtIndexPath methods, and finally delete the as! NSDate text from the prepareForSegue() and cellForRowAtIndexPath methods – not the whole line, just the bit that says as! NSDate.

Make sure your app builds at this point.

Create Model

We're going to use a struct to represent a petition.

Classes and structures in Swift have many things in common. Both can:

- Define properties to store values
- Define methods to provide functionality
- Define subscripts to provide access to their values using subscript syntax
- Define initializers to set up their initial state
- Be extended to expand their functionality beyond a default implementation
- Conform to protocols to provide standard functionality of a certain kind

Classes have additional capabilities that structures do not:

- Inheritance enables one class to inherit the characteristics of another.
- Type casting enables you to check and interpret the type of a class instance at runtime.
- Deinitializers enable an instance of a class to free up any resources it has assigned.
- Reference counting allows more than one reference to a class instance.

File | New | File | Swift File
Petition

```
struct Petition: Decodable{
```

```

    let title: String
    let sigCount : Int
    let url : String
}

```

Load JSON file

Look at JSON data <https://api.whitehouse.gov/v1/petitions.json?limit=100>

It's easier to read in XML <https://api.whitehouse.gov/v1/petitions>

Look at the items in the result, they are key-value pairs

In MasterViewController xchange objects to store these key-value pairs using our Petition struct.

```
var petitions = [Petition]()
```

Create a method to download the json file.

```

func loadjson(){
    let urlPath =
"https://api.whitehouse.gov/v1/petitions.json?limit=50"
    guard let url = URL(string: urlPath)
    else {
        print("url error")
        return
    }

    let session = URLSession.shared.dataTask(with: url,
completionHandler: {(data, response, error) in
        let httpResponse = response as! HTTPURLResponse
        let statusCode = httpResponse.statusCode
        guard statusCode == 200
        else {
            print("file download error")
            return
        }
        //download successful
        print("download complete")
        //DispatchQueue.main.async {self.parsejson(data!)}
    })
    //must call resume to run session
    session.resume()
}

```

Update viewDidLoad() to call this method.

```

override func viewDidLoad() {
    super.viewDidLoad()
    loadjson()
}

```

If you run it now you should get the download successful message.

Parse JSON file

Now we'll create a method to get the json data and parse it so we can use it.

```
func parsejson(_ data: Data){
```

```

        print(data)
    do {
        // get json data
        let json = try JSONSerialization.jsonObject(with: data, options:
JSONSerialization.ReadingOptions.allowFragments) as! [String:Any]
        //get all results
        let allresults = json["results"] as! [[String:Any]]
        print(allresults)

        //add results to objects
        for result in allresults {
            //check that data exists
            guard let newtitle = result["title"]! as? String,
                  let newsigCount = result["signatureCount"] as? Int,
                  let newurl = result["url"]! as? String
            else {
                continue
            }
            //create new object
            let newpetition = Petition(title: newtitle, sigCount:
newsigCount, url: newurl)
            //add object to array
            self.petitions.append(newpetition)
        }
        //handle thrown error
    } catch {
        print("Error with JSON: \(error)")
        return
    }
    //reload the table data after the json data has been downloaded
    tableView.reloadData()
}

```

Call this from loadjson() right after the print statement that the download was successful.

```
DispatchQueue.main.async {self.parsejson(data!)}
```

Load table data

Update the following method to use our petitions array.

```

    override func tableView(_ tableView: UITableView, numberOfRowsInSectionSection: Int) -> Int {
        return petitions.count
    }

```

Now we're ready to load the petition data in our table. Update the following table view delegate method

```

    override func tableView(_ tableView: UITableView, cellForRowAt indexPath: IndexPath) -> UITableViewCell {
        let cell = tableView.dequeueReusableCell(withIdentifier: "Cell",
for: indexPath)
        let petition = petitions[indexPath.row]
        cell.textLabel!.text = petition.title
        cell.detailTextLabel!.text = String(petition.sigCount) + "

```

signatures"

```
        return cell
    }
```

Detail View

In the main storyboard go into the detail view and remove the label (check that this removes its connection as well) and replace it with a webKit view that fills up the whole view.

Add an activity indicator on top of the webKit view. (it must be below the web view in the document hierarchy). In the attributes inspector check Hides When Stopped.

Connect the web view and activity indicator as webView and webSpinner.

Add needed constraints.

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 DetailViewController remove detailDescriptionLabel.

import WebKit and adopt the WKNavigationDelegate protocol for the class.

```
import WebKit
class DetailViewController: UIViewController, WKNavigationDelegate
```

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

```
webView.navigationDelegate = self
```

Write a method to load a web page.

```
func loadWebPage(_ urlString: String){
    //the urlString should be a properly formed url
    //creates a NSURL object
    let url = URL(string: urlString)
    //create a NSURLRequest object
    let request = URLRequest(url: url!)
    //load the NSURLRequest object in our web view
    webView.load(request)
}
```

Update detailItem and configureView() as follows:

```
var detailItem: String?

func configureView(){
    if let url = detailItem{
        if url != "null"{
            loadWebPage(url)
        }
    }
}
```

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

```
//WKNavigationDelegate method that is called when a web page begins to load
```

```
func webView(_ webView: WKWebView, didStartProvisionalNavigation
navigation: WKNavigation!) {
```

```

        webSpinner.startAnimating()
    }

    //WKNavigationDelegate method that is called when a web page loads
    successfully
    func webView(_ webView: WKWebView, didFinish navigation: WKNavigation!)
    {
        webSpinner.stopAnimating()
    }

```

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

```

    override func prepare(for segue: UIStoryboardSegue, sender: Any?) {
        if segue.identifier == "showDetail" {
            if let indexPath = self.tableView.indexPathForSelectedRow {
                let petition = objects[indexPath.row]
                let title = petition["title"]
                let url = petition["url"]
                let controller = (segue.destination as!
                UINavigationController).topViewController as! DetailViewController
                controller.detailItem = url
                controller.title = title
                controller.navigationItem.leftBarButtonItem =
                self.splitViewController?.displayModeButtonItem
                controller.navigationItem.leftItemsSupplementBackButton =
                true
            }
        }
    }

```

When you run the app if there's a space about the web view you need to update the top constraint to go all the way to the top of the view, not just to the bottom of the nav bar.

Don't forget the launch screen and app icons.

Source: <https://www.hackingwithswift.com/read/7/2/creating-the-basic-ui-uitabBarController>