Task 6P – Spike Report : Data There

Goals:

The main goal of task 6P was to work with data stored online. It was required to learn about the retrieval and usage of data stored online.

The following list outlines the goal broken down into more specific knowledge gaps involved in the goal.

- Retrieving data from an online source
- Formatting the JSON object to retrieve specific data
- Identifying different modes of communication between Model, View and Controller
- Working with asynchronous tasks
- Use of completion handlers to handle asynchronous tasks

Tools and Resources Used

Tools used;

- XCode

Online Resources used;

Lynda course

- Table Views (https://www.lynda.com/iOS-tutorials/Start-career-iOS-development/751326/807111-4.html?org=swin.edu.au)
- iOS Network Development using URLSession and Alarmofire (https://www.lynda.com/iOS-tutorials/iOS-Network-Development-Using-URLSession-AlamoFire/645028-2.html)

API

- Zomato API - https://developers.zomato.com/documentation#!/restaurant/search
- RecipePug API - http://www.recipepuppy.com/api/

Online sources

- Use an API with an API key - https://stackoverflow.com/questions/52730997/how-do-i-use-an-api-with-an-api-key-in-xcode
- Guide to JSON Parsing - https://benscheirman.com/2017/06/swift-json/
- URL Request - https://developer.apple.com/documentation/foundation/urlrequest
- 3 methods of communicating between classes - https://medium.com/@gavin9/3-methods-for-communicating-between-classes-in-swift-4-using-the-model-view-controller-3673f14c70b2
- Three ways to parse data from model to controller - https://medium.com/@stasost/ios-three-ways-to-pass-data-from-model-to-controller-b47cc72a4336
- Pass Data with Delegation on Swift - https://medium.com/ios-os-x-development/pass-data-with-delegation-in-swift-86f6bc5d0894

- Difference between DispatchQueue.main.sync and async - https://www.reddit.com/r/iOSProgramming/comments/7n9e9f/what_is_the_difference_between/
- Display image from. URL - https://stackoverflow.com/questions/39813497/swift-3-display-image-from-url/39813761
- Fixing Thread 1 – SIGABRT - https://medium.com/@consbulaquena/fixing-thread-1-signal-sigabrt-error-in-ios-9a50a94368da
- Swift Completion Handler - https://grokswift.com/completion-handler-faqs/#data_out
- Completion Handlers in Swift - https://medium.com/@nimjea/completion-handler-in-swift-4-2-671f12d33178
- Swift waiting for server result - https://stackoverflow.com/questions/45077064/swift-3-wait-for-the-server-result

Knowledge Gaps and Solutions

Gap 1: Retrieving data from an online source

Problem: The main knowledge gap was to retrieve data from an online source.

Process followed to reach knowledge gap:

1. As a start, I looked for a few free APIs of my interest. The first API was the RecipePug which had lists of recipes. It did not require an API key so I thought it was free to access but once the URL was used in the application, data retrieval was restricted due to security policy restriction.
   error : Optional(Error Domain=NSURLErrorDomain Code=-1022 "The resource could not be loaded because the App Transport Security policy requires the use of a secure connection."
2. I found the API of Zomato very clear and useful therefore I used it to retrieve restaurant data.
3. I followed the Lynda tutorial and a few online resources to retrieve the data onto the application. For this I also had to use an API key.

Solution: I used the Zomato API to retrieve latest news of the food culture around Hawthorn. I created an API key with the Zomato API to be eligible to access the data. The following code snippet shows the retrieval of data using a dataTask. City_id 259 represented the Hawthorn city. The dta was then stored in a struct of type Codable.

```
let url =
"https://developers.zomato.com/api/v2.1/collections?city_id=259"
        if let URLToServer = URL.init(string: url)
        {
            var request = URLRequest(url: URLToServer)

request.addValue("13de75ce059551c4f98fe8fa577bd774",
forHTTPHeaderField: "user-key")
```

```
            let task = URLSession.shared.dataTask(with:
request, completionHandler:  { (data, response, error) in

                if error != nil || data == nil
                {
                    print("error : \(error)")
                }
                else
                {
                    guard let data = data else { return}

                    do{
                        let     collectionList     =     try!
JSONDecoder().decode(Collections.self, from: data)
…
```

Gap 2: Formatting the JSON object to retrieve specific data

Problem: The problem was to decode the complex JSON string into attributes defined in the Codable class.

Process followed to reach knowledge gap:

1. I used the online Zomato API to print out the structure of the JSON object. This was done to identify the nested arrays in the resultant object.
2. I went through online sources to identify how data can be retrieved from a complex JSON object
3. The best option was to use the Codable protocol. I followed a Lynda course which demonstrated retrieval of data from a simple JSON Object but I wanted to retrieve data from a complex nested JSON object.
4. I followed an online source to understand the creation of Codable classes/structs to replicate the nested levels in the JSON object.
5. Finally I used three structs to retrieve the three levels of data from the JSON object.

Solution: The figure below displays a sample set of JSON data used to populate the list in my application.


```
"collections": [
  {
    "collection": {
      "collection_id": 1,
      "res_count": 30,
      "image_url": "https://b.zmtcdn.com/data/collections/08d501d476332b528603055bfd513e7a_1554422639.jpg",
      "url": "https://www.zomato.com/melbourne/top-restaurants?utm_source=api_basic_user&utm_medium=api&utm_campa
      "title": "Trending This Week",
      "description": "Most popular restaurants in town this week",
      "share_url": "http://www.zoma.to/c-259/1"
    }
  },
  {
    "collection": {
      "collection_id": 274852,
      "res_count": 293,
      "image_url": "https://b.zmtcdn.com/data/collections/07b4a7549754f07ee9b5f6997bade2e1_1543211964.jpg",
      "url": "https://www.zomato.com/melbourne/great-food-no-bull?utm_source=api_basic_user&utm_medium=api&utm_ca
      "title": "Best of Melbourne",
      "description": "The hunt for the highest-rated restaurants in your city ends here",
      "share_url": "http://www.zoma.to/c-259/274852"
```


I used three structs to replicate the three levels of data in the response Object. The following code snippet demonstrates the three levels;

```
struct Collection: Codable {
        let collection: CollectionData
    }

    struct CollectionData : Codable{
        let collection_id: Int
        let res_count: Int
        let image_url: String
        let url:String
        let title : String
        let description: String
        let share_url: String
    }

    struct Collections: Codable {
        let collections: [Collection]
        let has_more: Int
        let share_url: String
        let display_text: String
        let has_total: Int
    }
```

Once the structure was identified, decoding the JSON object was easily done in one line of code,

```
let collectionList = try! JSONDecoder().decode(Collections.self, from: data)
```

The code was decoded into Collections structure which had the other structures nested in it. Thereafter I successfully decoded the lists of data into the collectionList array. I then looped through the array to create objects of NewsItem class and populated them with data retrieved from the JSON object.

Gap 3: Identifying different modes of communication between Model, View and Controller

Problem: Task 6P required the application to hold the data retrieval process from the online source in the Data Model and not to use the Controller to perform any requests to the online source. The problem was to identify how the initial step of communication needed to be performed.

Process followed to reach knowledge gap:

1. I researched about the different ways of communication between the models and the controllers. I read about three methods in general; delegation, using notifications and using callbacks.
2. The delegation design pattern is said to be the mostly used method to communicate between MVC classes. (Sharder, 2018) This design pattern uses protocols to define methods.
3. I used delegates to perform the call to model class from controller.
4. A delegate protocol was created in the model class;
```
protocol ModelDelegate: class {
    func didReceiveData(_ data: [NewsItem])
}
```
5. The controller class uses the ModelDelegate protocol and creates an instane of the model.
6. Finally at the point where the view is loaded into the application, the delegate is used to retrieve data into the controller.

Solution: The delegate design pattern was used to send data from model to controller. The model handles all interactions with the online source and stores them in a class. The fetchData() function is called to get the data from the model. The following code snippet shows how the data retrieved from the online source is stored in a class and adds it to the delegate

```
var tempNews = NewsItem(title: item.collection.title, desc:
item.collection.description, resCount: item.collection.res_count,
imageURL: item.collection.image_url)
newsList.append(tempNews)

self.delegate?.didReceiveData(newsList)
```

This can then be retrieved from the instance created in the controller.

Gap 4: Working with asynchronous tasks

Problem: A problem occurred when displaying the data in the table view. Although the application executed with no errors, it did not load data into the table view.

Process followed to reach knowledge gap:

1. The data retrieval required a Data task to run to get data from the online source. This happened in a different queue. The table load happened in the main thread.
2. The main thread finished loading its tasks without waiting for the data retrieval since the queues run asynchronously.
3. I researched about the different ways of handling asynchronous tasks. I researched about completion handlers, closures and changing the current queues.

4. Since I used delegates to communicate between model and controller, I knew when the task of retrieving data ended, therefore I forced a table load to run once the data was completely retrieved.

Solutions : Once the data was completely retrieved from the model, I forced a table reload to happen on the current queue. This reloaded the table with the complete set of fetched data.

```
func didReceiveData(_ data: [NewsItem])
    {
        newsItems = data
        printData()
        DispatchQueue.main.async {
            self.tableView.reloadData()
        }

    }
```

Gap 5: Use of completion handlers to perform tasks asynchronously

Problem: The problem was with downloading an image from the URL to display on the details page. The details will load before the image download is completed.

Process followed to reach knowledge gap:

1. First I ran a download task to download the images on the controller. Although this worked, it made the controller work on data retrieval using the URL which was not ideal for MVC architecture.
2. I tried to use the same approach in the model view and call it from the controller but the main thread completed before downloading the image.
3. Then I researched on completion handlers,which was a block of code that would retrieve the response data and can be used to pass it to another function.

Solution: A class namely ImageProvider was used to perform the asynchronous task of downloading the image. A completion handler was used to handle the request of downloading an image. It notifies with the result once the request comes through.

```
func loadImages(from imageURL: String, completion: @escaping (_
image: UIImage) -> Void) {
        downloadQueue.async(execute: { () -> Void in
            let url = URL(string: imageURL)!

            do{

                let data = try Data(contentsOf: url)
                if let image = UIImage(data: data) {
                    DispatchQueue.main.async { completion(image) }
                } else { print("Could not decode image") }
```

```
            }catch { print("Could not load URL: \(url): \(error)")
}
        })
    }
```

The controller will not have any idea about downloading images, It will simply create an object from the imageProvider class and retrieves the image.

```
let imageProvider = ImageProvider()
            imageProvider.requestImage(from: newsItem!.imageURL) {
image in
                self.imageLabel.image = image
            }
```

Open Issues and Recommendations

An issue I came across with formatting the data in JSON objects was if it is necessary to retrieve all data in the JSON object into the application data model. There were some data that I didn't want in my application but caused errors if it was not retrieved. This issue needs further research.

I also want to learn more about the use of memory and performance factor of using online data especially with downloading images from online.

I want to learn more about the optimal ways of handling asynchronous tasks and also practice them in an application.