checkMate

iPhone Software Engineering
Assignment 2
2017

Miracle John Octavio Jr (s3638336)
Annie Vuong (s3579678)

Table of Contents

Introduction

The purpose of Assignment 2 was to extend and complete the proposed iOS application in Assignment 1. From the previous assignment, we introduced our application 'checkMate', which is an application to help users be more productive and organised in their everyday lives. Our application at that point had functionalities that included the ability to create reminders, set a due date, and turn on alarm notifications.

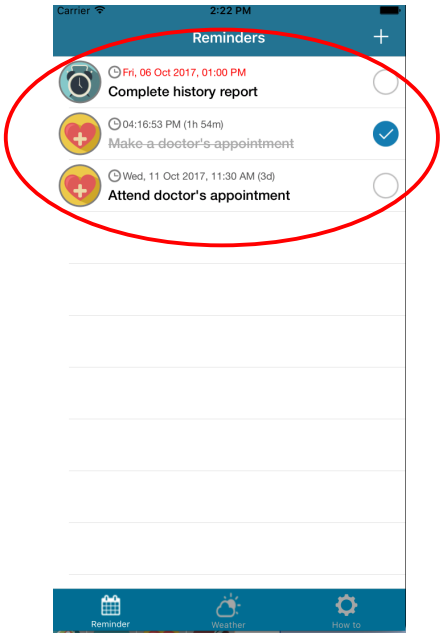
In this assignment, we have added a couple more feature into our application. These features are the ability to take and store photos as part of a reminder and to check the current weather of any specified city. An example scenario of where these features may be useful is if you needed to go out to buy groceries and your housemate asked you to buy a particular brand of shampoo for them, which you were not familiar with. Using checkMate, you could create a reminder to buy your housemate's shampoo and include a picture of that particular brand. You would also be able to check the current weather of your city before leaving your house, and prepare for any adverse conditions.

Design Principles

Contrast

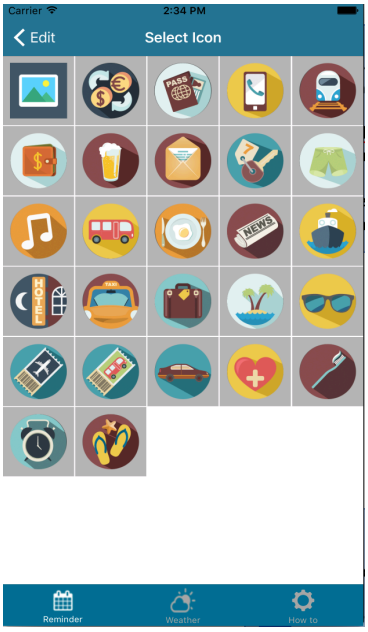
Contrast has been used in our application to help users differentiate the different stages of a reminder. For example, black and bold text indicates that a reminder is active and still within the due, grey and striked out text indicates that the reminder is not active and has been completed, and red text indicates that the reminder is overdue and should be completed soon.

It has also been ensured that sufficient contrast has been applied throughout the application so that users do not find content hard to read or view.



Repetition

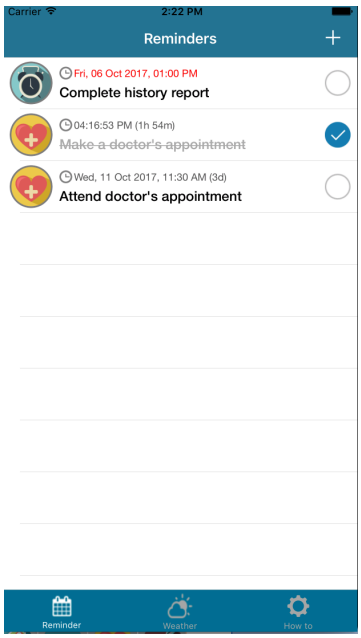
To maintain the design principle of repetition in our application, colours, text and shape have been kept consistent throughout each scene. An example of this is the use of the circle shape for check boxes and icon.



Alignment

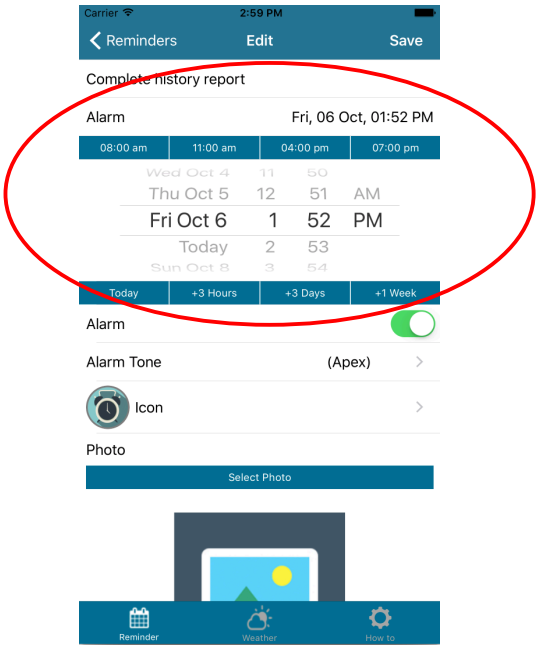
As a principle of our application is to help users to be more productive and organised in their lives, alignment is an important aspect in the design of our application. Alignment provides a more ordered design, which is preferred over a messy design. How can users become more productive and organised if our application was messy? Alignment also helps to create an invisible connection between elements.

Hence, we have applied this principle by aligning elements such as icons, text and checkboxes such that it represents a connection to a particular reminder.



Proximity

Proximity creates and suggests that there is a relationship between certain elements. An example of this can be seen in the date time picker when creating a reminder. The proximity of all the elements around the date time picker element suggests that they are all related. When the user changes the date/time on the picker, it is reflected in the text field above it. Also, the elements around the date/time picker provide an efficient way for users to quickly select a date and time.

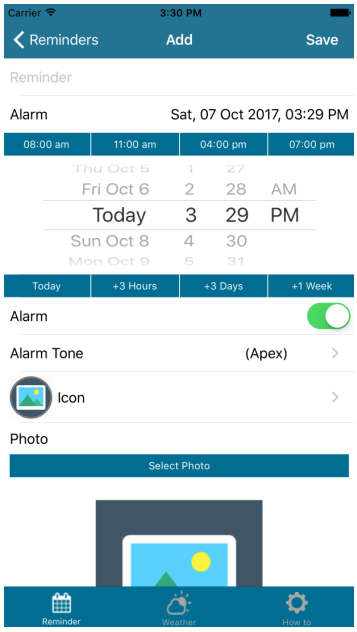


Deference

According to the principle of deference, the UI should never compete with the content of an application. An example of this can be seen in the weather section of our application. Here, it can be clearly seen that the temperature element is the most important aspect of this view. It represents the main content and aim of this scene.

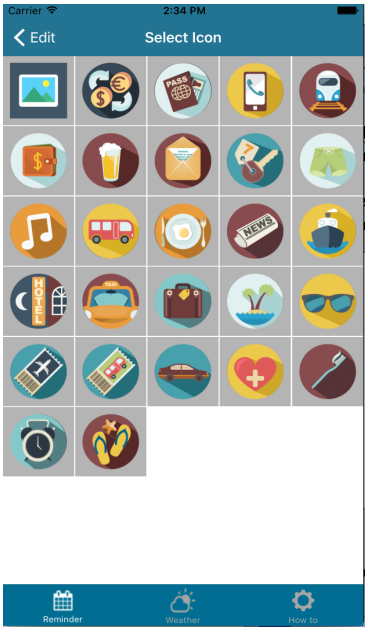
Clarity

Clarity was achieved in our application by clearly labelling each section with meaningful texts and providing prompt text where user input is required. At the top of each scene, meaningful titles are applied so that users can clearly see what the scene is attempting to achieve.



Depth

Depth was created in our app by the use of transitions to different screens when a user tapped on a particular section, whilst also allowing the user to go back to the previous screen.



Design Patterns

The design patterns (apart from MVC and Singleton) implemented in our code are:

- The Decorator pattern utilising Extensions
- The Delegation pattern

Extensions were used used in the weatherViewController.swift file in which the method downloadImage() is used to download weather icons from the APIXU API resources. Below are snippets of code that show this implementation.

```
                    DispatchQueue.main.async {
                        if self._blnExists {
                            self.isHideControls(blnHide: false)

                            // Set data into fields on view
                            self.lblCityName.text = self._strCity
                            self.imgWeatherIcon.downloadImage(from: self._strImgURL!)
                            self.lblWeatherCondition.text = self._strCondition
                            self.lblTemperature.text = "\(self._intDegree.description)°C"

                            // Stop Activity Indicator
                            self.aivFetching.stopAnimating()

                        } else {
                            // City not found
                            self.isHideControls(blnHide: true)
                            self.lblCityName.isHidden = false
                            self.lblCityName.text = "No matching city found"
                        }
                    }
```

Code found in the weatherViewController.swift

```
// Used to download icon image from API
extension UIImageView {

    /*

     Note: Some of the icons will not be downloaded due to
     security issues. To resolve this, you need to
     add the following lines on your info.plist

     > App Transport Security Settings
        > Allow Arbitrary Loads = YES

     */

    func downloadImage(from url: String) {
        let urlRequest = URLRequest(url: URL(string: url)!)

        let dataTask = URLSession.shared.dataTask(with: urlRequest) { (data, response, error) in
            if error == nil {
                DispatchQueue.main.async {
                    self.image = UIImage(data: data!)
                    print("Successfully Loaded Icon!")
                }
            } else {
                print("Error Loading Icon!\(error.debugDescription)")
            }
        }
        dataTask.resume()

    }

}
```

Code found in the weatherViewController.swift

Another instance where Extensions was used in the MyGlobals.swift file. The method hideKeyboardWhenTappedAround() is used to allow the user to tap anywhere around the keyboard to hide the keyboard. This method was implemented in the entryTableViewController.swift and weatherViewController.swift files. Below are snippets of code that show this implementation.

```swift

// This will hide the keyboard when top is detected
/*

 Usage:

 override func viewDidLoad() {
 super.viewDidLoad()

 // This will hide the keyboard when top is detected (See extenxion Below)
 self.hideKeyboardWhenTappedAround()

 }

 */

extension UIViewController {
    func hideKeyboardWhenTappedAround() {
        let tap: UITapGestureRecognizer = UITapGestureRecognizer(target: self,
                                                                 action: #selector(UIViewController.dismissKeyboard))
        tap.cancelsTouchesInView = false
        view.addGestureRecognizer(tap)
    }

    func dismissKeyboard() {
        view.endEditing(true)
    }
}

```

Code found in MyGlobals.swift

```swift

    override func viewDidLoad() {
        super.viewDidLoad()

        // Assign Delegate
        sbrCityName.delegate = self
        // sbrCityName.showsCancelButton = true    // This will display Cancel Button

        // Insert the Background
        view.insertSubview(imgBackground!, at: 0)

        // This will hide the keyboard when top is detected (See extension from MyGlobals)
        self.hideKeyboardWhenTappedAround()

    }

```

Code found in the entryTableViewController.swift

```swift

    override func viewDidLoad() {
        super.viewDidLoad()

        // Assign Delegate
        sbrCityName.delegate = self
        // sbrCityName.showsCancelButton = true    // This will display Cancel Button

        // Insert the Background
        view.insertSubview(imgBackground!, at: 0)

        // This will hide the keyboard when top is detected (See extension from MyGlobals)
        self.hideKeyboardWhenTappedAround()

    }

```

Code found in the weatherViewController.swift

The Delegation pattern is implemented in the toneTableViewController.swift and entryTableViewController.swift files. The main function of the delegate for this particular process is to pass the value of the selected Tone. Assuming the user click the “Alarm Tone”, which brings him to the selection of Tones displayed by the UITableViewController. Once the user decided to select any of the tones, it will tell the caller of that UITableViewController that a Tone was been selected. It will also send what specific tone was been chosen. So, basically the delegate allows the called UITableViewController (UITVC2) to communicate with the caller UITableViewContoller (UITVC1). Below are snippets of code that show this implementation.

```swift

// Custom Delegation Step # 1
protocol ToneSelectedDelegate: class {
    func toneSelected(strToneSelected: String)
}

```

```swift

    // Custom Delegation Step # 2
    weak var delegate:ToneSelectedDelegate?

```

```
74
75      // Reset all checkbox
76      override func tableView(_ tableView: UITableView, didSelectRowAt indexPath: IndexPath) {
77
78          // Get the selected tone
79          MyGlobals.shared.selectedTone = MyGlobals.shared.arrTone[indexPath.row]
80
81          // Custom Delegation Step # 3
82          self.delegate?.toneSelected(strToneSelected: MyGlobals.shared.arrTone[indexPath.row])
83
84          // Back to Root View Controller
85          // _ = self.navigationController?.popToRootViewController(animated: true) // Root
86          _ = self.navigationController?.popViewController(animated: true)         // Previous
87
88      }
89
```

```
10
11  // Custom Delegation Step # 4
12  class entryTableViewController: UITableViewController, UITextFieldDelegate,
13      UIImagePickerControllerDelegate, UINavigationControllerDelegate, ToneSelectedDelegate {
14
```

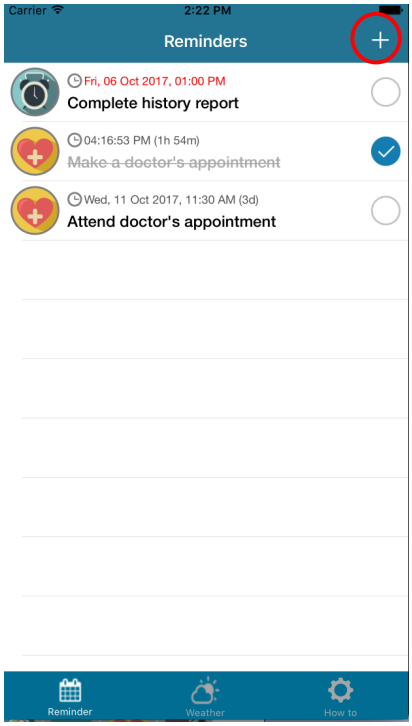
```
274
275     // Custom Delegation Step # 5
276     func toneSelected(strToneSelected: String) {
277         _selectedTone = strToneSelected
278     }
279
```

```
282
283     // In a storyboard-based application, you will often want to do a little preparation before navigation
284     override func prepare(for segue: UIStoryboardSegue, sender: Any?) {
285
286         // Pass the current value to Global Lookup
287         MyGlobals.shared.arrTask_Lookup.Task = txtTask.text!
288         MyGlobals.shared.arrTask_Lookup.DateTime = dtpTask.date
289         if (swtIsAlarmMessageOn.isOn == true) {
290             MyGlobals.shared.arrTask_Lookup.IsAlarmMessageOn = 1
291         } else {
292             MyGlobals.shared.arrTask_Lookup.IsAlarmMessageOn = 0
293         }
294         if (MyGlobals.shared.arrTask_Lookup.IconFile != "") {
295             imgIcon.image = UIImage(named: MyGlobals.shared.arrTask_Lookup.IconFile)
296         }
297
298         if (segue.identifier == "segueToneView") {
299             let tvc: toneTableViewController = segue.destination as! toneTableViewController
300             tvc._intIndex = _intIndex
301             // Custom Delegation Step # 6
302             tvc.delegate = self
303         } else if (segue.identifier == "segueIconView") {
304
305         }
306     }
307
```

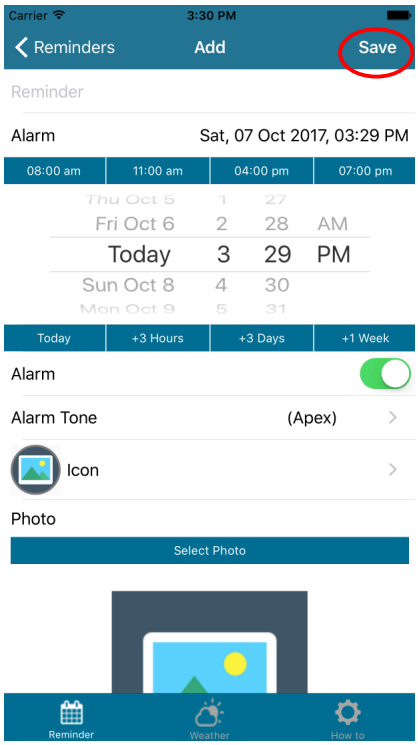
Persistence

Core Data

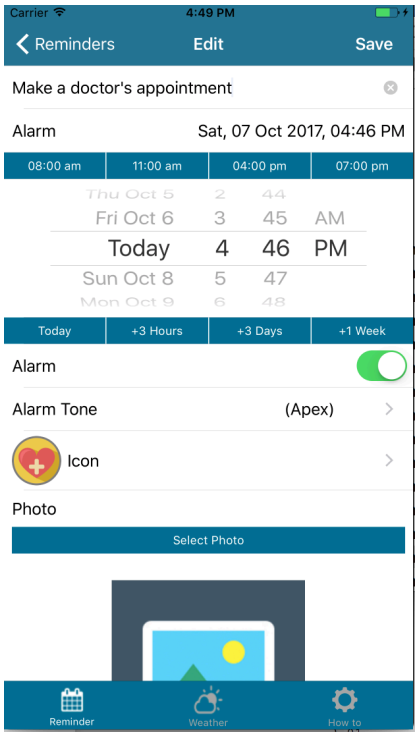
Our application implements Core Data to store data. The following scenes support CRUD operations.



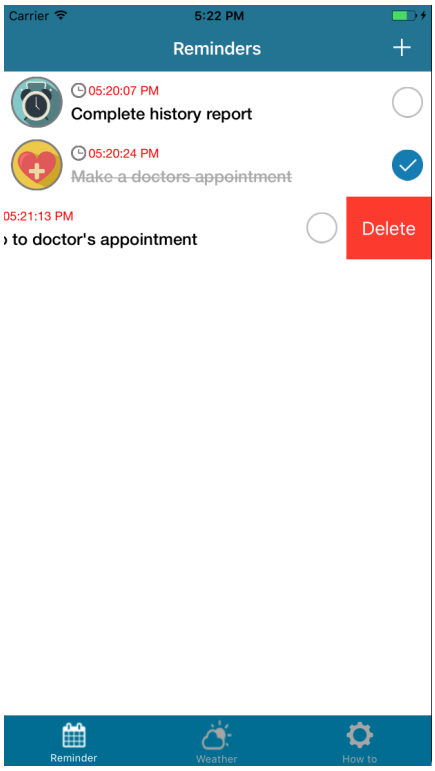
To add a reminder, tap the '+' button



Reminders can be created by tapping the 'Save' button. All data saved will be reflected in the list of reminders scene.



Reminders can also be edited by tapping on the reminder in the list view. By pressing 'Save', the data is stored to the Core Data database and reflected in the list view scene.



Reminders can be deleted by swiping left to reveal the delete button.

REST Based API

APIXU

The REST based API implemented in our application to provide the weather functionality is APIXU. APIXU provides weather related information via a JSON/XML restful API. The documentation for this API is provided in the references.

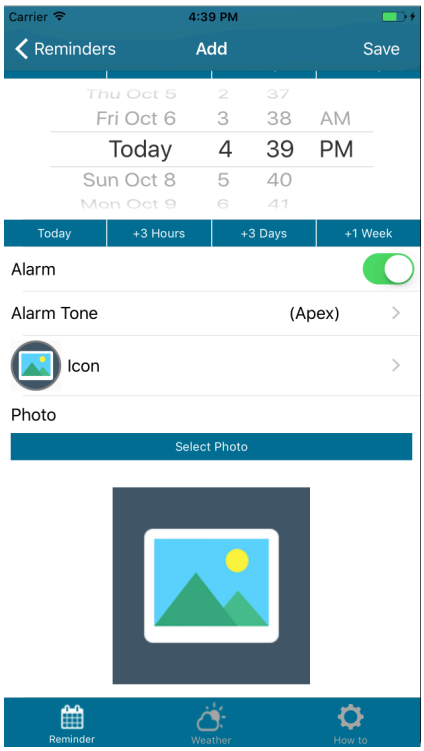
To use the information provided by APIXU in our application, a user needs to tap the weather icon in tab controller. They would then be redirected to scene where they can use a search bar to search for a particular city they wish to get weather information for. If the city is valid, then the APIXU API returns the current weather for that city. The city name label, weather image, condition label and temperature label utilise APIXU resources.

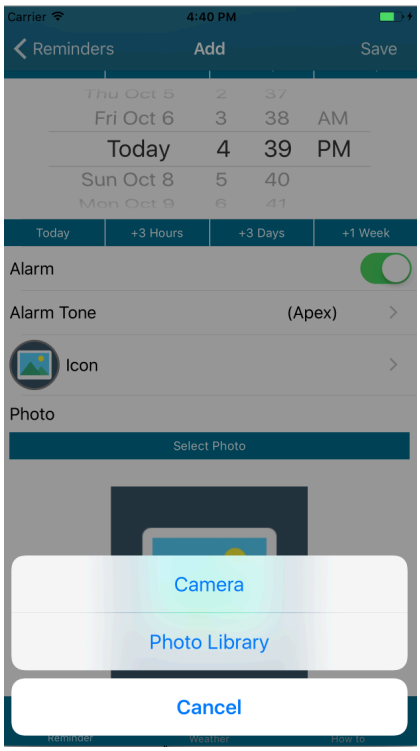
Cocoa Framework

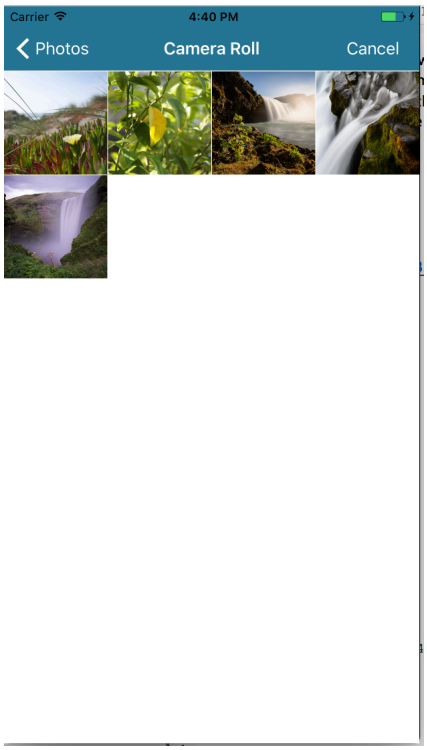
Camera and image picker

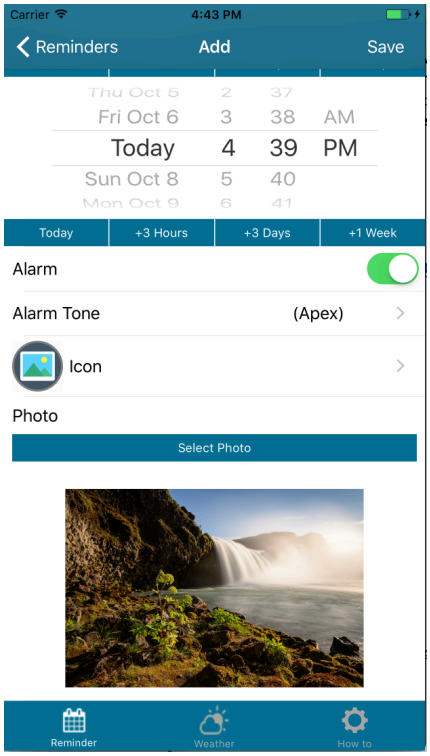
The Cocoa framework we decided to implement in our application is the use of iPhone's camera with included support for the image picker.

In the Add/Edit scenes of a reminder, users can choose to attach a photo to their reminders by tapping the 'Select Photo' button. They are they able to choose between using the device's camera to take a photo or choose an existing photo from their device's library. Once a photo has been chosen, it is then reflected in the Add/Edit reminder scene.









Unit Testing

Testing the file MyGlobals.swift

```swift
import XCTest
@testable import checkMate // This will allow us to access everything inside of our project

class testMyGlobals: XCTestCase {

    override func setUp() {
        super.setUp()
        // Put setup code here. This method is called before the invocation of each test method in the class.
    }

    override func tearDown() {
        // Put teardown code here. This method is called after the invocation of each test method in the class.
        super.tearDown()
    }

    func testmGetDateFormat() {
        let objMyGlobals = MyGlobals()

        XCTAssertEqual(objMyGlobals.mGetDateFormat(genmDateFormat.HrsMin), "hh:mm a") // "hh:mm a"
    }

    func testmDateToString() {
        let objMyGlobals = MyGlobals()

        let formatter = DateFormatter()
        formatter.dateFormat = "yyyy mm dd"
        let olddate = formatter.date(from: "2017 01 01")

        // Temp Test
        // print(objMyGlobals.mDateToString(olddate!,genmDateFormat.MonDayYrs))

        XCTAssertEqual(objMyGlobals.mDateToString(olddate!,genmDateFormat.MonDayYrs), "Jan 01 2017") // "Jan 01 2017"
    }

    func testmSecondsToHoursMinutesSeconds() {
        let objMyGlobals = MyGlobals()
        let strHoursMinutesSeconds: String = objMyGlobals.mSecondsToHoursMinutesSeconds(300000)

        // Temp Test
        // print(strHoursMinutesSeconds)

        XCTAssertEqual(strHoursMinutesSeconds, "83h 20m 0s")
    }

    func testmSecondsToDays() {
        let objMyGlobals = MyGlobals()
        let strToDays: String = objMyGlobals.mSecondsToDays(300000)

        // Temp Test
        // print(strToDays)

        XCTAssertEqual(strToDays, "3d")
    }

```

References

URL to the Documentation of APIXU API
https://www.apixu.com/doc/

Sample Request of APIXU Data
http://api.apixu.com/v1/current.json?key=349a650cde304863ac9132411172608&q=Melbourne

Design patterns guide
https://www.raywenderlich.com/160651/design-patterns-ios-using-swift-part-12

Unit Testing
https://cocoacasts.com/how-to-unit-test-private-methods-in-swift/