In-Class Homework #14 – Web API integration on iOS

CUS 357 – Winter 2017
Due Date: End of class, Apr 17, 2017

Learning Objectives

- Learn how to integrate content retrieved from a web API into an iOS app.

Please work in pairs to get as much of this assignment done as possible by the end of today's class session.

Overview

Begin this exercise using the finished product of your In-Class Homework #11 solution. If you did not finish that homework, you are welcome to start with the instructor's solution, available on Blackboard.

Apply what you have learned in the Lecture 17 (Part 01) screencast to integrate with the Dark Sky weather API in your app. You will reproduce the same functionality demonstrated in the video, using a stubbed out version of the API. See Figure 1 below.

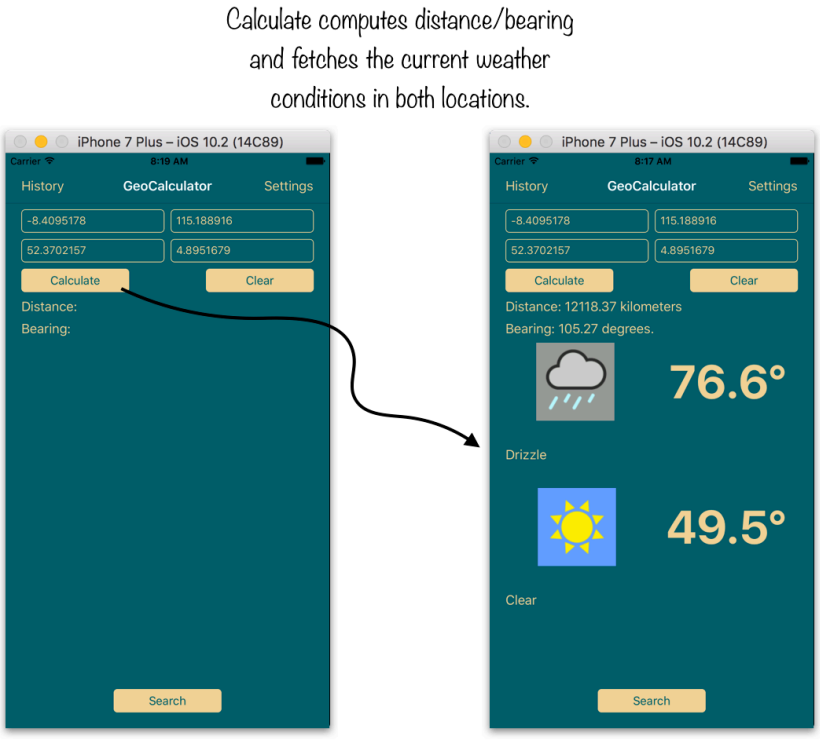

Figure 1. Searching for locations using Google Places Picker.

Integrating with the Dark Sky Weather API

Step 1. Register for a free API token on the Dark Sky Developer site: https://darksky.net/dev/

Step 2: Add the UIView's needed on your main calculator view controller (e.g. an image and two labels for each weather forecast). Don't worry about cosmetics for now, you can go back and clean that up later (e.g. use your customized UILabel class, and add the appropriate Auto Layout constraints.)

Step 3: Add outlets to your controller class for each of the views you added in the previous step.

Step 4: Unzip and drag a copy of the provided Weather.xcassets.zip folder into your Xcode project explorer.

Step 5: Introduce your weather model struct and the abstract interface that you will implement using the Dark Sky API:

```
struct Weather {
    var iconName : String
    var temperature : Double
    var summary : String

    init(iconName: String, temperature: Double, summary: String) {
        self.iconName = iconName
        self.temperature = temperature
        self.summary = summary
    }
}

protocol WeatherService {
    func getWeatherForDate(date: Date, forLocation location: (Double, Double),
                           completion: @escaping (Weather?) -> Void)
}
```

Step 6. Using the Dark Sky Dev site documentation, complete the following stubbed out Dark Sky API implementation (e.g. compute the URL endpoint, and provide the JSON parse logic.) We placed TODO comments where you need to provided additional code. There are essentially two tasks to complete here: 1) Figure out which API to use to get the weather conditions for a given lat/long and and current time, and concatenate the URL (using the token you received in step 1 above). 2) Study the JSON representation returned by the API and complete the partially implementation below to extract the three values you need (temperature, icon, and summary).

If you can't figure out how to do this from the docs, consult the code in the video as a last resort.

```
import Foundation

let sharedDarkSkyInstance = DarkSkyWeatherService()

class DarkSkyWeatherService: WeatherService {

    let API_BASE = "https://api.darksky.net/forecast/"
    var urlSession = URLSession.shared

    class func getInstance() -> DarkSkyWeatherService {
        return sharedDarkSkyInstance
    }

    func getWeatherForDate(date: Date, forLocation location: (Double, Double),
                           completion: @escaping (Weather?) -> Void)
    {

// TODO: concatentate the complete endpoint URL here.
        let urlStr = API_BASE +  “????”
        let url = URL(string: urlStr)

        let task = self.urlSession.dataTask(with: url!) {
            (data, response, error) in
            if let error = error {
                print(error.localizedDescription)
            } else if let _ = response {
                let parsedObj : Dictionary<String,AnyObject>!
                do {
                    parsedObj = try JSONSerialization.jsonObject(with: data!,
options:
                        .allowFragments) as? Dictionary<String,AnyObject>

                    guard let currently = parsedObj["currently"],

// TODO: extract the attributes you need for a Weather instance HERE

                        else {
                            completion(nil)
                            return
                    }

                    let weather = Weather(iconName: iconName, temperature:
temperature,
                                          summary: summary)
                    completion(weather)

                }  catch {
                    completion(nil)
                }
            }
        }

        task.resume()
    }
}
```

Step 7. Instantiate an instance of your API implementation in your main view controller as an instance variable:

```
let wAPI = DarkSkyWeatherService.getInstance()
```

Step 8: Add code to your geo calculation function that fetches the weather condition from each lat/long pair and binds the data to the user interface each time a bearing / distance is calculated. We've stubbed out the fetches below. Go ahead and replace the TODO comments with code that binds the fetched data to the outlets created in Step 3 above. Note that p1lt, t1ln, p2lt, p2ln are double values of what was entered in the four TextFields in the view. Your variable names may vary.

```
        wAPI.getWeatherForDate(date: Date(), forLocation: (p1lt, p1ln)) { (weather)
in
            if let w = weather {
                DispatchQueue.main.async {

                    // TODO: Bind the weather object attributes to the view here
                }
            }
        }

        wAPI.getWeatherForDate(date: Date(), forLocation: (p2lt, p2ln)) { (weather)
in
            if let w = weather {
                DispatchQueue.main.async {

                    // TODO: Bind the weather object attributes to the view here

                }
            }
        }
```

Step 9: If you have time left, add the necessary layout constraints so your interface looks like that in Figure 1 as much as possible.

If you finish before the end of the session, give a demo to the instructor. If you did not finish, you will have time during our next session.