MetUCD Report Part 1

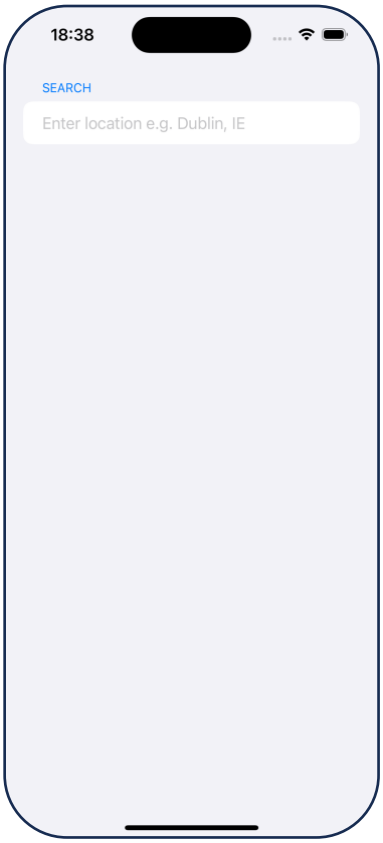
James Clackett
15732749
21st November 2023

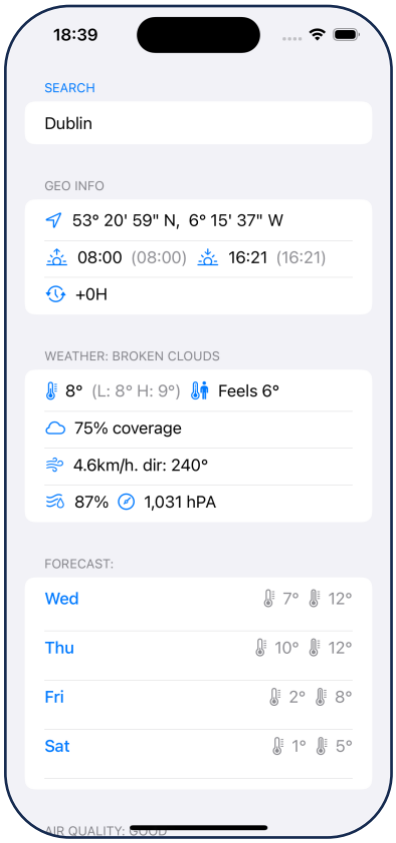
The objective of Part 1 was to build a weather forecast application that can search for locations using a geo coder and display weather information about that location to the user. See code for more detailed notes on functionality.

Summary of actions taken:

1) Created WeatherViewModel view model.
2) Built the API request function fetchData and a companion function getResult.
3) Created the model.
4) Skipped the part 1 version of 5-day weather forecast and implemented the version from part 2 instead. Refactored later to create the expected part 1 forecast.
5) Created the UI (SwiftUI Views).
6) Refactored and Documented code.

Application UI:





Important code:

Fetch Data function:

```
static func fetchData<T: Decodable> (url: String) async -> Result<T, Error> {

    guard let url = URL(string: url)
    else {
        return .failure(URLError(.badURL, userInfo: [NSLocalizedDescriptionKey: "Invalid URL"]))
    }

    do {
        let (data, _) = try await URLSession.shared.data(from: url)
        let decoder = JSONDecoder()
        let response = try decoder.decode(T.self, from: data)
        return .success(response)

    } catch {
        print("Invalid Response")
        return .failure(error)
    }
}
```

Weather View:

```
List {
    Section(header: Text("SEARCH").foregroundColor(.blue)) {
        SearchLocationField(forecastDisplayed: $forecastDisplayed)
    }

    if forecastDisplayed {

        Section(header: Text("GEO INFO")) {
            GeoInfoView()
        }

        Section(header: Text("WEATHER: \(weatherDesc)")) {
            CurrentWeatherView()
        }

        Section(header: Text("FORECAST:")) {
            WeatherForecastView()
        }

        Section(header: Text("AIR QUALITY: \(airQuality)")) {
            PollutionView()
        }

        Section(header: Text("AIR POLLUTION INDEX FORECAST")) {
            PollutionForecastView()
        }
    }
}
.listStyle(DefaultListStyle())
```

Issues:

1) When using some device simulators, a *UIKIT* constraint warning occurs. It appears to be related to the TextField but I can't consistently reproduce it. It is more likely on extra large or small devices.

2) Long-Press and Autocorrect had to be turned off to prevent a number of small warnings that occur (see SearchLocationField). These are known bugs which are discussed here: https://developer.apple.com/forums/thread/738726.
3) I had trouble understanding how to achieve a 5-day forecast. The API returns 5 full days of data. However if you make the call in the middle of the day, you will receive 4 full days and 2 half days (at the beginning and end). To ensure consistent results for the view to deal with, I simply discarded any non-full days (see API Structs/WeatherForecast).

Discussion:

MetUCD part 1 creates a Weather View that displays weather information to users.
On loading of the application, the WeatherView is passed a WeatherViewModel environment variable. An environment variable was used so that sub views can easily access the same view model object.

The view model is simply an Observable class that provides a representation of the model to WeatherView. To adhere to MVVM, views should be model agnostic. WeatherViewModel is responsible for requesting the services and data of the model. Unlike our class demos, WeatherModel is itself a type of interface. Rather than containing data itself, it is a mechanism for communicating with openWeatherAPI. However, as far as the view model is concerned, it could be anything. Several *codable* structs were created to represent the JSON API responses and were the core of the model's functionality. They each contained user-friendly methods and properties to simplify the use of their data.

The SearchLocationField within WeatherView was responsible for telling the view model to update the location. The view model would then update its state based on a new location request to the model. The WeatherView, which is listening to any changes to the view model, refreshes itself and builds its sub views with the updated weather forecast.

One part where I made a mistake was including the following code in SearchLocationField:

```
.onSubmit {
    Task {
        let location = try await WeatherModel.getLocation(location: userInput)
        if let coords = location.first {
            withAnimation{
                weatherViewModel.newLocation(lat: coords.lat, lon: coords.lon)
                forecastDisplayed = true
            }
        }
    }
}
```

I realised this while writing my report, namely that I was directly calling the model in my View. In response, I created a new function in the view model that takes a String location as input.