COMP47390: Swift Programming for App Development

Individual Assignment 4: MetUCD

September 13, 2023

Table of Contents

Objectives

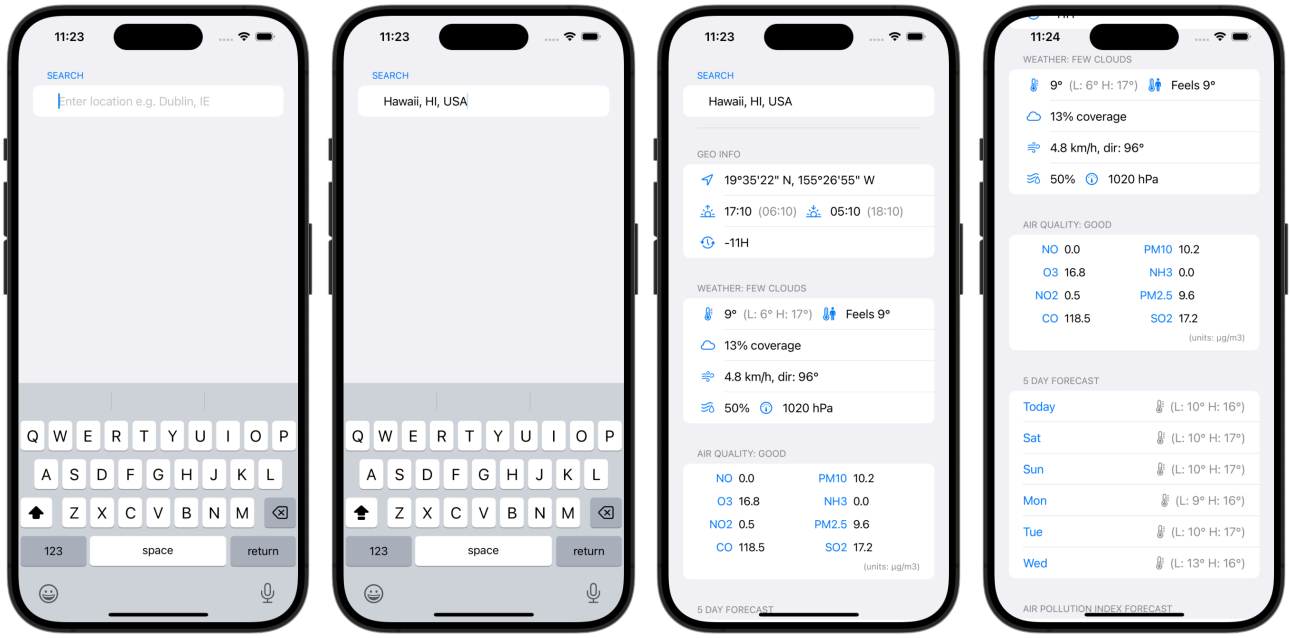
In this SwiftUI project assignment, you will create a mobile app called `MetUCD` that allows users to check the current weather for a location of their choice by dropping a pin on a map. The primary objectives of this assignment are to:

- Implement a user-friendly interface that enables users to interact with a map and drop pins on locations they want to check the weather for.
- Use the OpenWeather JSON API to fetch real-time weather data for the selected location, including temperature, humidity, wind speed and weather conditions.
- Implement the MVVM design pattern, ensuring model remains UI agnostic.
- Display the retrieved weather data in an informative and visually engaging manner.
- Ensure that the app provides an appealing user experience by presenting weather data in an easy-to-understand format, possibly including graphical elements like icons to represent weather conditions[1].
- Allow users to view additional information for the selected location, such as a 5-day 3-hour forecast and the air quality index as a graph.
- Implement error handling to gracefully handle cases where the API requests fail or the user provides invalid location data.

Important Notes:

- Before starting the project, sign up for an OpenWeather API account to obtain an API key (`https://home.openweathermap.org/users/sign_up`). You will need this API key to make requests to the OpenWeather API.
- Familiarise yourself with the free tier of the OpenWeather API by studying and testing the following endpoints:
  - Current Weather API: `https://api.openweathermap.org/data/2.5/weather`
  - 5-Day 3-Hour Forecast API: `https://api.openweathermap.org/data/2.5/forecast`
  - Air Pollution API: `https://api.openweathermap.org/data/2.5/air_pollution/forecast`

[1] Install SF Symbols (`https://developer.apple.com/sf-symbols/`)
Usage: pick some SVG symbol e.g. thermometer and in Swift `Image(systemName: "thermometer")`



(a) (b) (c) (d)

Figure 1: MetCD App – Screenshots of Part 1

- Geocoding API: `https://api.openweathermap.org/geo/1.0/direct`

It's important to understand how to construct API requests, handle responses, and parse the data you receive from these endpoints.

– Pay attention to the design and user experience aspects of the app, as creating an engaging and visually appealing interface is a key requirement.

– While implementing the app, consider error handling and responsiveness to ensure a seamless user experience using (async) when appropriate.

Part 1

The objective of this first part is to create a functional data model and a basic user interface. Users should be able to input a location in text format (e.g. `"Dublin, IE"`), and the app should fetch weather data from the OpenWeather API based on this input. The fetched weather data, including information such as temperature, humidity, wind speed, and weather conditions, should be presented in a visually appealing manner within a scrollable table view divided into multiple sections (Figure 1). Additionally, error handling should be implemented to handle API request failures and invalid inputs, ensuring a smooth and user-friendly experience. This phase lays the foundation for the subsequent development of the app's features and user interactions.

Here are the rough steps involved

1. Create a New SwiftUI Project:
   – Open Xcode.
   – Create a new SwiftUI project named `MetUCD`.
2. Define the Data Model:
   – Create Swift structs to represent the weather, pollution and forecast data models. Include suitable properties such as location, temperature, humidity, wind speed, and weather conditions. Ensure to embrace the `Codable` protocol.

3. Design the User Interface:
   - Use SwiftUI views such as `Form` and `VStack` to design the user interface.
   - Include a TextField to allow users to input the location (e.g., `"Dublin, IE"`).
   - Trigger the weather data fetching based on the entered location when user's tap on `Return` key of the keyboard.
   - Tapping on the search TextField, should show the keyboard and automatically clear text input and view.
   - Design the layout to have multiple sections for the user interface to separate different elements (e.g., input, geocoding info, weather data, pollution...).
4. Weather Data Fetching and Error Handling:
   - Implement the logic to fetch weather data from the OpenWeather API based on the user's input location.
   - Use Swift's asynchronous networking capabilities (e.g., URLSession) to make an API request.
   - Handle the API response and parse the JSON data into your defined data model.
   - Implement error handling to gracefully handle cases where the API requests fail or the user enters invalid location data.
   - Display informative error messages to the user when necessary.
5. Display Data:
   - Use a SwiftUI List, Section, ForEach, VStack and HStack to create a scrollable table view.
   - Organise the weather and pollution data into suitable sections within the table view, such as current weather, forecast, and additional information sections such as air quality and geolocalisation information (see Figure 1)
6. Refinement and User Experience:
   - Refine the user interface to make it more engaging and visually appealing using SF Symbols and colouring appropriately.
   - Provide a clear and user-friendly way to view forecast data, possibly with icons or visual representations of weather conditions.
   - Ensure that the user experience is smooth and intuitive.
7. Documentation:
   - Add appropriate comments to your code and document your report with code snippets to explain functionality and purpose.
   - Organise your project and code structure for clarity. Use 3 files called `WeatherDataModel`, `WeatherViewModel` and `WeatherView` so that one can identify the MVVM design pattern immediatly.

Build and Test

For simplicity, use the iOS Simulator. Your app should build and run with no warnings with a UI similar to Figure 1. Test the app by entering various locations and ensuring that weather data is fetched and displayed correctly. Debug any issues or errors in the code or user interface.

Part 2

In this second part, you will improve the application by incorporating an interactive map as the user's method for selecting the desired location to retrieve weather data. Tapping on a home location button should reset the map view point to the current user location using a location manager (cf. Figure 2a–b). An optional search textfield may be used to center the map to some location (cf. Figure 2c–d). Tapping on the map should drop a marker displaying the temperature at the selected location along with an overlay panel at the bottom with some summary of the weather conditions (cf. Figure 2d). Tapping on the weather panel will slide a modal table view displaying geocoding information, weather data, pollution conditions, as well as a forecast and air quality graph (cf. Figure 2e–f). You will use MapKit, CoreLocation, and Swift Charts to design an engaging UI. The app should always open at the current user location.

Here are the rough steps involved

1. ViewModel and Location Manager:

   Modify your WeatherViewModel so that to manage the interaction between the user interface and the data. Integrate a location manager to track the user's current location and visible location of the map.

   Notes:

   - Study the documentation of `CoreLocation`[2].
   - Make your observable `WeatherViewModel` a subclass of `NSObject` and conforms to the protocol `CLLocationManagerDelegate` to be notified of location updates.
   - Add a location manager to report the user's current location.
   - Ensure the ViewModel updates its weather data, pollution and forecast appropriately based on the map's position.
   - You may also need to create some additional computed properties to expose some suitable and simple data structures to your SwiftUI Views (in particular for rendering forecast data).
   - Some privacy permissions must be set for CoreLocation access. Add the required privacy key `"location when in use"` to the target app to request user permission for accessing location data.

2. Design of the Map Interface:

   Redesign the user interface to include a map that is initially centered on the user's current location. Allow users to drop a marker pin on the map by tapping. The pin should display the current temperature at that location. Overlay a small view at the bottom of the map with essential location details and basic weather information, such as the current temperature and minimum/maximum temperature forecast for the next 24 hours (cf. Figure 2d). Don't forget to handle the safe-area of the map view so that user map interaction is disabled under the widget view.

   - Use the SwiftUI view `Map` to display a map (cf. Apple documentation[3]).
   - Show/update the overlay widget when the user drops a pin or changes position.
   - Create a dynamic marker that updates its position based on the map's current position.
   - Enhance the map interface to include a location button to center the map on the current user location.
   - Optionally add a search button that expands to a textfield and used to the map region from the the specified text (cf. Figure 2c).

[2] `https://developer.apple.com/documentation/corelocation/`
[3] `https://developer.apple.com/documentation/mapkit/map`
`https://developer.apple.com/videos/play/wwdc2023/10043`

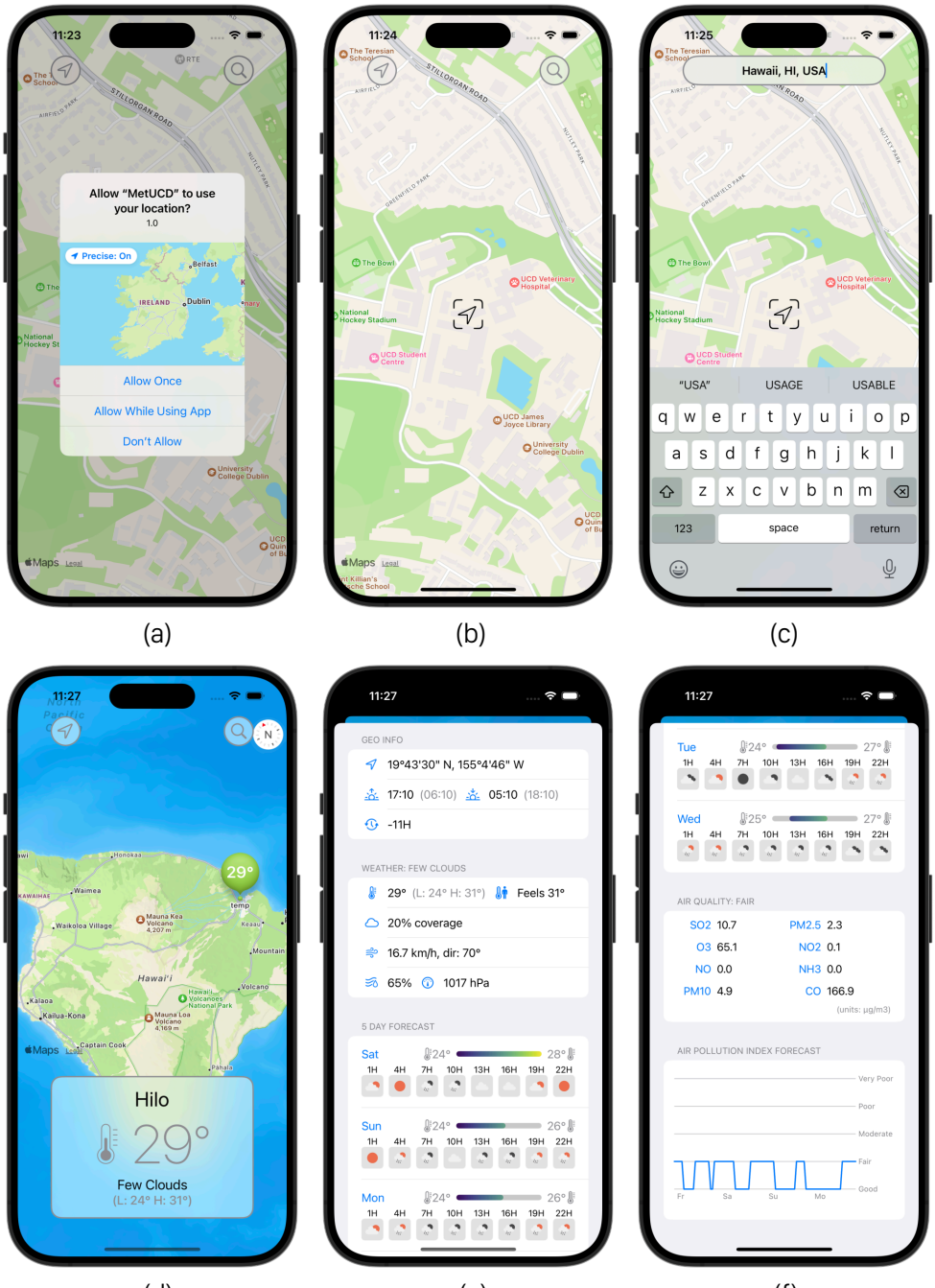


Figure 2: MetCD App – Screenshots of Part 2

3. Model Detailed Weather Sheet:

   Implement a feature where users can tap on the small widget displayed when they drop a pin on the map. This action should open a detailed weather view **presented modally as a sheet** (cf. Figures 12–f). The detailed view should be organised into sections, such as:

   - Geocoding and time information
   - Current weather conditions such as temperature, cloud coverage, wind, precipitation (rain/snow if applicable), humidity and air pressure.
   - Forecast for the next 5 days fetching asynchronously icons for weather conditions.
   - Pollution conditions with description and components concentration.
   - Plot of air quality index forecast (use Swift `Charts`[4])

These enhancements aim to create a more engaging and informative user experience by allowing users to interact with the map, drop pins, and access detailed weather information. The modal sheet presentation ensures that users can view comprehensive weather data while maintaining a user-friendly interface.

Add appropriate comments to your code, and document your report with code snippets to explain functionality and purpose.

Build and Test

For simplicity, use the iOS Simulator. Your app should build and run with no warnings with a UI similar to Figure 2. Thoroughly test the app to ensure that the ViewModel, Location Manager, and user interface elements work as expected. Debug any issues related to location tracking, data updates, or user interaction.

Submission

For this assignment, testing will be done by running the project on the iOS simulator. We will be looking at the following:

- Submit short reports called `readme-part<#>` with appropriate code segments and screenshots summarising your solution along with zip archives of your Xcode project called `<Firstname>_<Surname>_<StudentNumber>_part<#>.zip` for each parts.
- Your project should build without errors or warnings.
- Your project should run without crashing.
- Each of parts will be considered to verify that you've completed the assignment correctly.
- Your project should have a clean user interface, UI elements arranged logically, aligned neatly, etc.
- We will be verifying that all fundamental concepts are understood.
- Ensure your code is easy to read and not visually sloppy (indentation, etc...). Make good use of object-oriented design principles (avoid code duplication, use inheritance appropriately, etc...)
- Your solution should elegant and your code is easy for someone to read (right amount of comments, appropriate variable/method names, good solution structure, self documenting methods, etc...)

— END OF ASSIGNMENT 4 —

[4] `https://developer.apple.com/documentation/charts/`