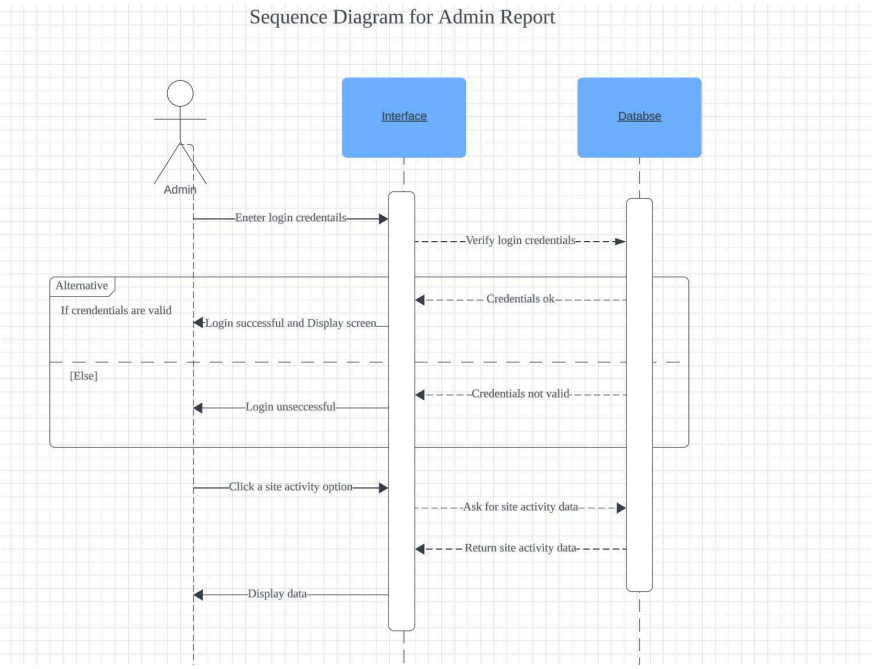
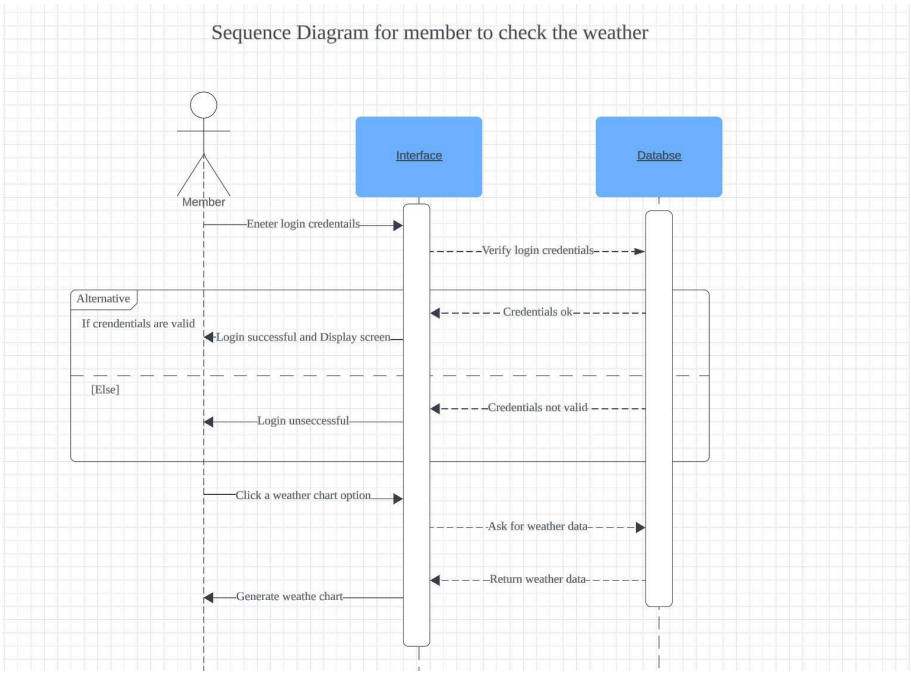
M3: Formal Analysis and Architecture

Sequence Diagrams:

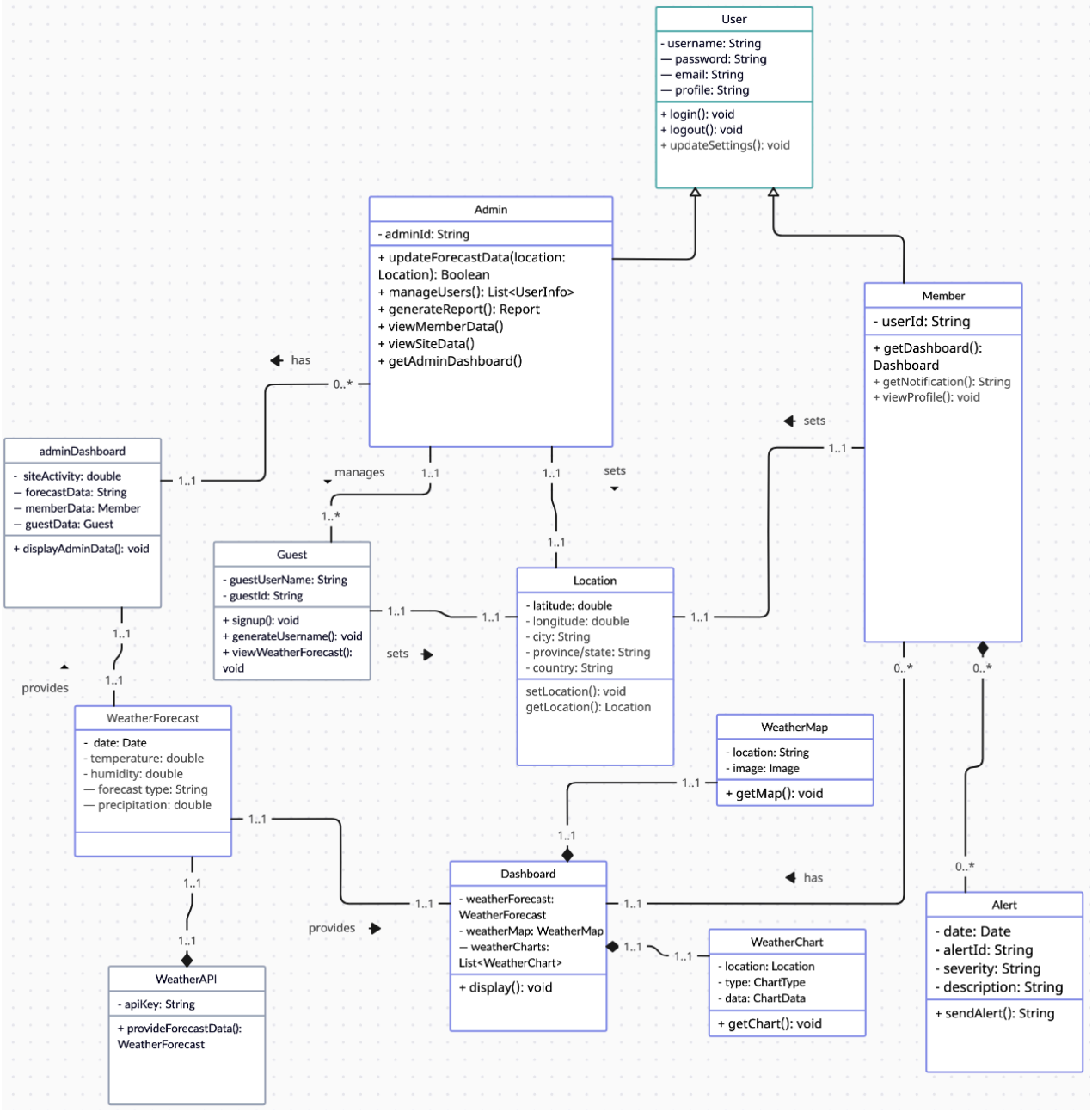
1. Admin Report



2. Check Weather



Class Diagram:



Test Plan:

- **Testing Objective:** Ensure the functionality of the weather forecast software is operational and provides accurate, reliable weather information.
- **Testing Phases:**
    - Functional Testing
    - Compatibility Testing
    - Performance Testing

- **Testing Scope:**
  - User Interface
  - Accuracy of Weather Data
  - Precision of Weather Forecasts
  - Software Performance (loading speed, response time, etc.)
  - Software Compatibility (across different devices, operating systems)
- **Functional Testing:**
  1. Validate the usability and user-friendliness of the user interface.
  2. Confirm the software can accurately retrieve and display real-time weather information.
  3. Test the weather forecasting functionality to ensure accurate future weather predictions.
  4. Validate the search functionality to ensure users can retrieve weather information by city or region.
  5. Test notification functionality to confirm users receive timely notifications of weather changes.

  - **Compatibility Testing:**
    1. Test the software's operation on different operating systems (Windows, Mac, Linux, iOS, Android).
    2. Ensure the software displays properly on devices with different resolutions and screen sizes.
    3. Test performance under different network conditions (3G, 4G, Wi-Fi).
  - **Performance Testing:**
    1. Test the software's loading speed to ensure users can quickly open the application.
    2. Test response time, including search and weather information refresh times.
    3. Test performance under heavy data loads to ensure stability and smooth operation.
- **Testing Environment:**
  - Testing Devices: PC, Mac, iOS devices, Android devices
  - Testing Tools: Various browsers, emulators, real devices
  - Testing Networks: Wi-Fi, mobile networks
- **Recording and Feedback of Test Results:**
  - Use test reports to record the results of each test case and any issues encountered.
  - Categorize issues by priority and provide timely feedback to the development team.
  - Track the progress of issue resolution to ensure fixes are implemented and verified.
- **Testing Cycle:**

    - Initial Testing: Conduct comprehensive testing after development completion.
    - Subsequent Updates: Perform regression testing after each software update to ensure new features do not affect the normal operation of existing features.
- **Testing Completion Criteria:**
    - All functionalities have been validated and tested, meeting expectations.
    - All known issues have been resolved or documented in the issue tracking system.
- **Testing Team:**
    - Test Manager Xiya Zi
    - Test Engineers: Seth Ojo, Baizen Li
    - User Experience Designer: Sev Nielsen

Design Patterns:

1. Observer Pattern (Behavioural)

The observer pattern is a powerful design tool that will allow us to connect to multiple users and provide them with updates, notifications, and alerts in weather information. This system will allow us to have a one-to-many dependency so that when one object changes state all dependencies are notified. For example, users will be able to receive frequent updates on current weather information and forecast data if they choose to. Furthermore, they may choose to watch for specific variables and ask to be notified if it is predicted to snow, rain, lighting, wind levels, and other filters.

Problem:

- **Intent:** We would like to create a one-to-many relationship that allows us to notify users of changes in weather in data and application updates.
- **Motivation:** For users to be able to subscribe and personalize their experience.
- **Applicability:** This will be used to provide robustness to scalability, allowing us to update all of our objects.

Solution:

- **Structure:** The subject will be us, the weather data, and the customers will be the observers. They will be able to customize their own filters.
- **Participation:** we the subject will maintain a list of observers and the observers will use the update interface.
- **Collaborations:** Observers will be connected to the subject upon initialization, and will be updated on state changes.
- **Implementation:** Because we are the subject we will use the subject interface on the weather data, and the observer interface into the customer that wants to be notified.

Consequence:

- **Warnings:** Can be straining on system memory if not kept track of number of users properly.

- **Known uses:** Best known uses such as instagram, facebook, yahoo finance in sending notification.
- **Related pattern:** The iterator pattern can be used to traverse the list of customers.

2. Singleton Pattern (Creational)

The singleton pattern is used when you must only create one instance of a class, and is important when you require a central data system. The singleton pattern can be utilized in our situation to centralize weather data and customer profiles, and other areas where you can only have one instance. For example, this will ensure only one account per customer is created, and can also be utilized to ensure consistent weather data that is the same across all platforms.

Problem:

- **Intent:** To ensure only one customer class is created and that all customers are connected to one centralized weather data set with a global point of access.
- **Motivation:** To enhance security, accuracy, and reliability for customers and ensure they are all receiving the same data and are not being replicated.
- **Applicability:** This will be used to access our database of weather, and customer data.

Solution:

- **Structure:** The class will contain a method that returns an instance of itself. To ensure only one instance is created the constructor class is private.
- **Participation:** Itself
- **Collaborations:** the class would collaborate with all objects that require access such as multiple customers accessing one data set.
- **Implementation:** The instance is declared as a private static variable and in our case could be the customer and the weather data set. A public static method will check to see if the instance exists, if not create a new one then only return the new one, and if so return the already created one.

Consequence:

- **Warnings:** Must be handled carefully when dealing with multi-threaded applications where multiple instances could easily be created if not.
- **Known uses:** Often used for connecting to databases, API connections, and logging data.
- **Related pattern:** The abstract factory pattern allows for the creation of new objects without specifying the exact class, can be used alongside the singleton pattern to ensure the factory itself is a single instance, and all the families it crewates have a global access point to it.