FORECAST HUB - Desktop Edition

Project Team

Ayman Asad Khan

Rafin Akther Utshaw

Abdullah Mohammad Ashraf

Kasper Matias Kivistö

Phuong Nguyen

CONTENTS

Section 1 - Project Description

Section 2 - Project Overview
- 2.1 Purpose
- 2.2 Scope
- 2.3 Requirements
  - 2.3.1 Key Features Traceability Matrix
- 2.4 Key Features

Section 3 - System Components w.r.t Responsibilities

Section 4 - System Architecture

Section 5 – Design Patterns

Section 6 - Services and Methods

Section 7 - User Interface Design
- 7.1 User Interface Design Overview
- 7.2 User Interface Navigation Flow

Section 8 - Tools and Techniques

Section 9 - Future Enhancement in Design Features

Section 10 - Self Evaluation

Section 11 - Division of Work

Section 1 - Project Description

Weather ForecastHub is a desktop software solution designed to deliver precise meteorological predictions tailored to user-specified geographic coordinates and time intervals. This application delivers a user-friendly and efficient interface, allowing users the ability to retrieve current and future weather data for defined timeframes spanning from individual days to multiple weeks or 16 days.

Section 2 - Project Overview

2.1 Purpose

Module Focus:

This module of the overall weather forecast project is primarily focused on the development of a real-time weather data visualization component w.r.t location. Its main purpose is to collect, process, and present weather data in an accessible and informative manner to end-users.

Intended Audience:

The intended audience for this module includes anyone who relies on accurate weather forecasts and seeks an easy-to-use, desktop-based solution for accessing weather information.

2.2 Scope

The scope of the project to be produced encompasses the following key aspects:

1. Data Retrieval:

This module will be responsible for connecting to external meteorological data sources, current local time zones, fetching cities/countries and precise locations through APIs to collect real-time weather data for specified locations in specified time.

2. Data Processing and Analysis:

It will process the raw weather data, converting it into a user-friendly format. This includes interpreting and aggregating data points, calculating summaries, and providing forecasts.

3. Data Visualization:

The module will transform processed weather data into visually informative representations, offering users data visualization in the form of line graphs and tabular charts. Users can customize the presentation based on their preferences.

4. User Interaction:

Users will be able to input location and the module will provide real-time and forecasted weather data based on this input with time intervals.

5. User Preferences:

It will offer options for users to customize units of measurements, enabling users to save and manage multiple locations for quick access to weather information allowing a more personalized experience.

2.3 Requirements

2.3.1 Key Features Traceability Matrix

Cross referencing the software system requirements with implemented modules:

SSD Requirement	SSD Module
The data visualization should include graphs, plots, maps, or other self-developed advanced visualization. Tables can be used, but they cannot be the only way of showing information.	**2.4.4 Weather Data Visualization**

The user must be able to select what data is shown (e.g. customize results). In practice, this also means that the application must be able to show multiple types of data. Showing simple data (e.g. only a weather forecast) will not be accepted.	**2.4.5 Daily Summaries** **2.4.6 Weekly and Monthly Forecasts**
The user can save preferences for producing visualizations (e.g., date and location), and fetching those preferences will produce a visualization using the most recent data with the given parameters.	**2.4.7 Save Locations**
The design must be such that further data sources (e.g. another third-party API), or additional data from existing sources could be easily added.	**3.3 Data Acquisition via API Integration**

2.4 Key Features

2.4.1 Intuitive User Interface

The application features a user interface characterized by an intuitive design and aesthetically pleasing elements, enhancing the ease of user interaction and navigation within the weather data.

2.4.2 Location Selection

Users can input their desired location through a search bar or choose from a list of user saved preferences of locations. The application will be employing an API to automatically detect the user's current location by IP.

2.4.3 Time Intervals

Users can view the weather forecasts over the time intervals on different view panels. Options including: Current Day, Hourly, Weekly and Next 16 days weather forecast.

2.4.4 Weather Data Visualization

The application renders comprehensive meteorological data, encompassing parameters such as temperature, humidity, precipitation, wind speed and direction, UV index and additional metrics. Users have the option to visualize this data through line charts and tabular formats.

2.4.5 Daily Summaries

In the context of daily forecasts, users can retrieve a succinct overview of the day's weather conditions, encompassing temperature highs and lows, precipitation probabilities, and noteworthy weather events.

2.4.6 Weekly and Monthly Forecasts

For extended forecasts, the application provides users with a weekly and 16 days overview of the weather conditions. This information helps users plan activities and events well in advance.

2.4.7 Save Locations

The application allows users to save their frequently used locations for quick access to weather forecasts.

Section 3 - System Components w.r.t Responsibilities

1. User Interface (UI):

- Responsibility: The user interface component is responsible for presenting the application's graphical interface to users.
- Features:
  - Accept user input for location and time intervals.
  - Display weather data in a visually comprehensible format.
  - Facilitate user interaction with the application.

2. Location and Time Input Handler:

- Responsibility: This component manages user inputs for location and time intervals.
- Features:

- Receives and validates location information.
- Parses and interprets time interval selections.

3. Data Retrieval and API Integration:

Following are the third party open source APIs that collectively contribute to the accurate and location-specific weather forecast functionality by providing weather data, location information, IP-based geolocation, and time zone synchronization.

1. Open-Meteo Weather API:

- Responsibility: This API is responsible for providing weather data, including temperature, precipitation, humidity, wind speed, and more, for a specified location and time.
- Features:

It offers current weather conditions and forecasts, historical weather data, and can support various weather parameters such as pressure, UV index, and cloud cover, providing data in a standardized format like JSON.

2. GeoNames API:

- Responsibility: The GeoNames API assists in geolocation and place name lookup, helping to convert place names or coordinates into specific location information, such as city names, countries, and administrative divisions.
- Features:

It offers a vast geographical database with details on millions of locations worldwide. It supports reverse geocoding, allowing you to find location information based on coordinates, and forward geocoding to discover coordinates from place names.

3. ipinfo.io:

- Responsibility: This API is used to determine the geographical location and IP-related information of users accessing the application. It aids in identifying the user's approximate location based on their IP address.
- Features:

ipinfo.io provides data on the user's IP address, location (city, region, country), and network information. This data can be useful for customizing weather information based on the user's location.

4. World Time Api:

- Responsibility: The world time api API serves to fetch accurate time zone and time-related data for different locations around the globe, ensuring that weather information is presented in the local time of the specified location.

- Features:

It offers time zone information, current time, and conversion between time zones. This is valuable for displaying weather forecasts in the context of the user's local time, accounting for differences in time zones.

4. Data Processing and Analysis:

- Responsibility: Analyzes and processes the raw weather data to make it usable for presentation and decision-making.
- Features:
  - Converts raw data into a format suitable for visualization.
  - Calculates summaries, statistics, and historical comparisons.
  - Filters and sorts data for user preferences.

5. Data Visualization:

- Responsibility: Transforms processed data into visually informative representations.
- Features:
  - Generates graphs, charts, tables, and maps to display weather data.
  - Customizes data presentation based on user preferences.
  - Updates visualizations in real-time or as per user requests.

6. User Preferences:

- Responsibility: Manages user-specific settings and preferences.
- Features:
  - Allows users to customize units of measurement, themes, and notification preferences.
  - Saves user profiles for recurring usage.

7. Location Management:

- Responsibility: Enables users to save and organize multiple locations.
- Features:

- Allows users to save frequently used locations.
- Manages user's location preferences for quick access.

Section 4 - System Architecture

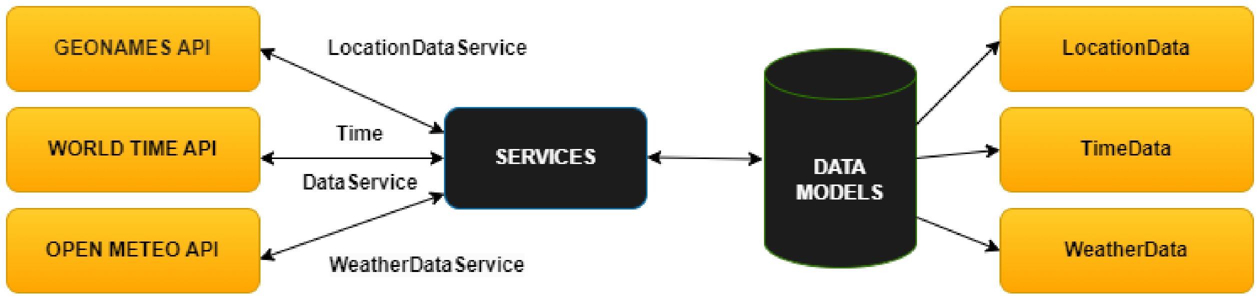
For developing a JavaFx weather forecast, **Model-View-Controller (MVC) architecture** was decided as a suitable choice. Here's an architectural overview of how we implemented packaging in MVC architecture for our project:

- **Models**: We define our data models here. These models represent the data used in the application, such as weather data, location data, time data, user and preferences data.
- **Views**: The ***JavaFX FXML views*** were placed in this package. These views represent the user interface components of your application.
- **Controllers**: We implemented the ***JavaFX controllers*** that handle user input and interact with the models. Each view has a corresponding controller.
- **Services:** Consequently we implemented service classes responsible for handling data retrieval, processing, and external API integrations. These services act as intermediaries between the controllers and the models.
- **Utilities**: The utility classes for tasks such as date formatting, unit conversion, and general helper methods that are to be consumed across the application are placed here.

Why MVC?

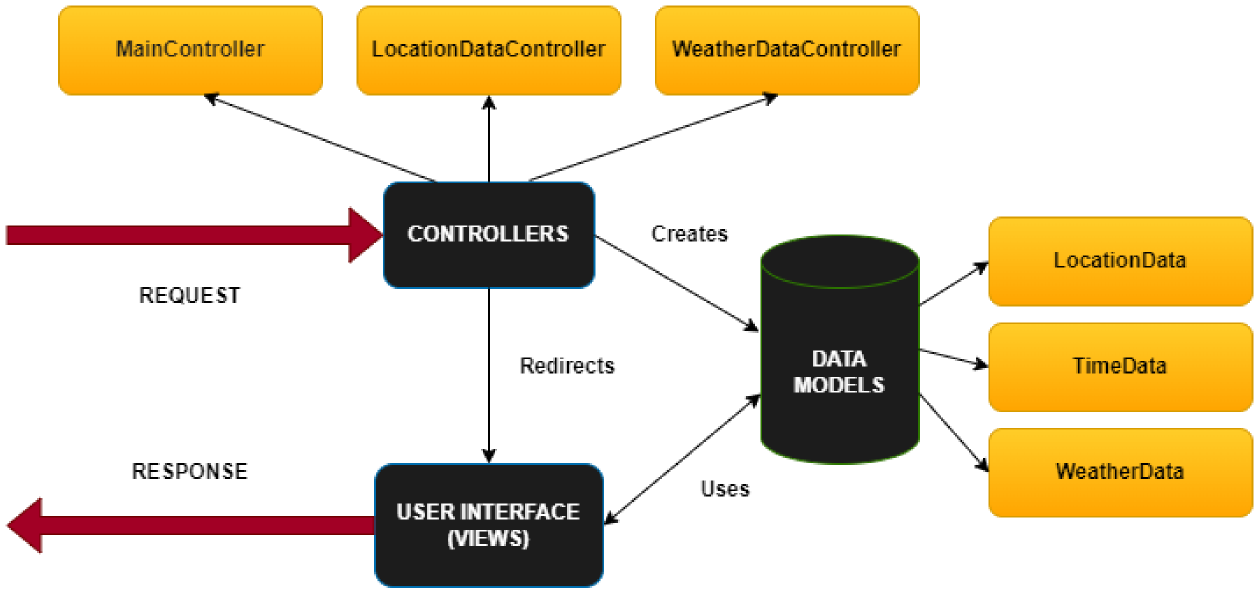
MVC enforced a clear separation of concerns within the application, which made working in a collaborative codebase more organized and maintainable. Each component (Model, View, Controller) had a distinct role, reducing code entanglement. The team members worked on different services, data models, backend and user interfaces simultaneously. The flexibility of working with user interface (views) without altering the underlying logic (models and controllers) was crucial.

Here is a bird-eye view of the Model-View-Controller (MVC) architectural pattern as it has been implemented within our project and how components interact with each other. For Instance,



- The `*WeatherDataController*` communicates with the `*WeatherDataService*` to fetch weather data based on user input.
- The `*MainViewController*` switches between different views and manages overall navigation.
- The controllers interact with the models to retrieve and update data. For example, the `*WeatherDataController*` uses the `*WeatherData*` Model to store and display weather data.

- Services, such as `*WeatherDataService*` and `*LocationService*`, interact with external data sources (APIs or databases) and provide structured data to controllers and views.



Section 5 – Design Patterns

For the design patterns we concluded on ***The Singleton Design Pattern.***

The Singleton design pattern was picked to have a central component or service that needs to be instanced only once and accessed globally. With this structured approach, we implemented the Singleton design pattern in our data services where necessary. In this context, a singleton for components like ***`WeatherDataService*** and ***`LocationDataService*** have only one instance of the service throughout the application's lifecycle.

The primary goal of using the Singleton pattern was to guarantee that only a single instance of the classes exist at any given time to avoid inconsistent data handling. This is vital when managing services responsible for fetching and handling data. The Singleton pattern allowing global access to the Data Service's instance facilitated a consistent data retrieval process across the application. We also ensured that the instance is properly initialized in a multithreaded environment by double-checked locking (***synchronized*** block).

Section 6 - Services and Methods

`WeatherDataService

Methods:

- getInstance() : ***WeatherDataService***
- getCurrentWeather(String location, double latitude, double longitude) : WeatherData
- getHourlyForecast(String location, double latitude, double longitude) : List<WeatherData>
- getWeeklyForecast(String location, double latitude, double longitude) : List<WeatherData>
- get16DaysForecast(String location, double latitude, double longitude) : List<WeatherData>

`TimeDataService

Methods:

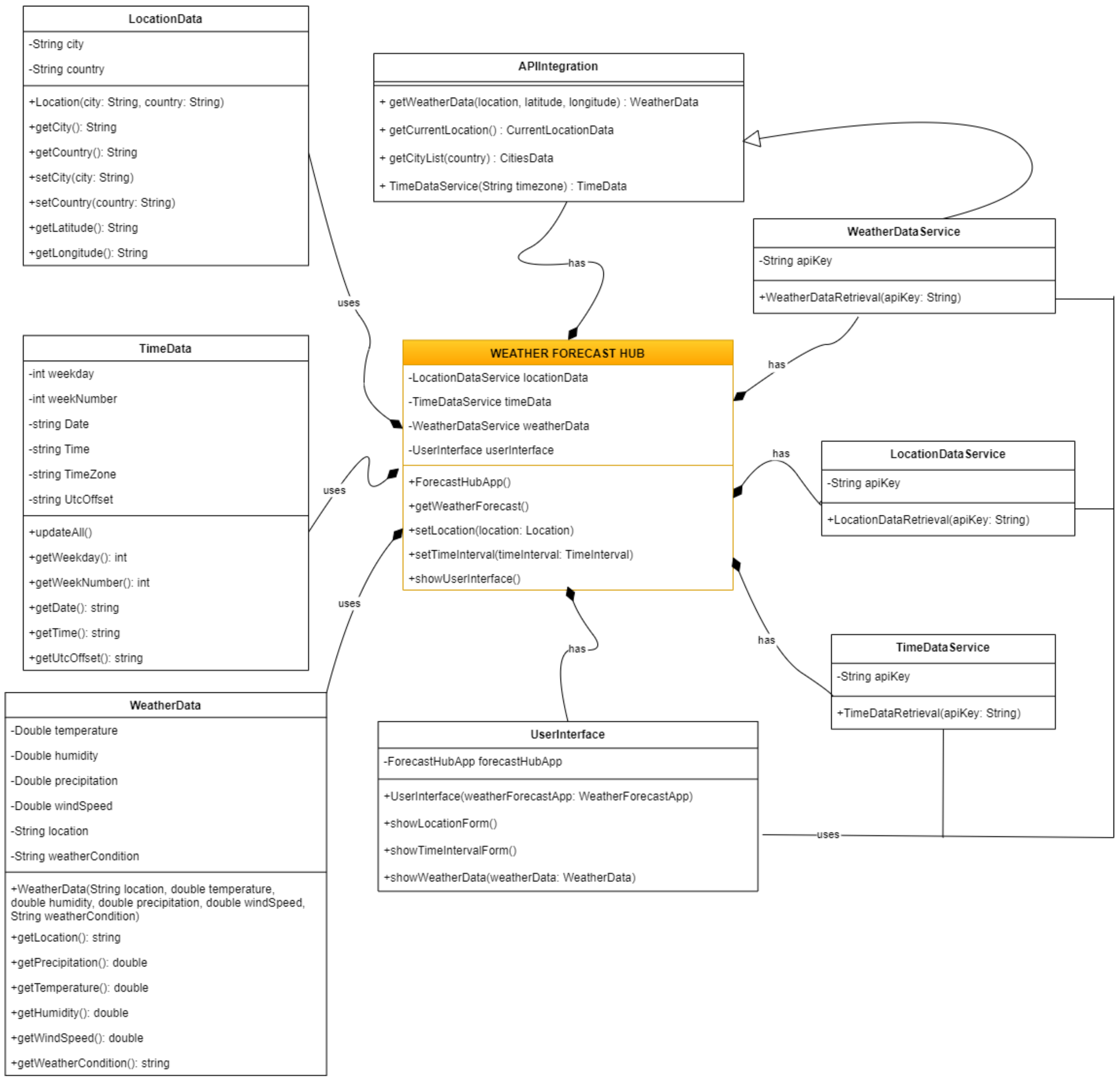
- TimeDataService(String timezone)
- getData() : JsonObject

`LocationDataService

Methods:

- getInstance() : ***LocationDataService***
- getObservableActiveLocation() : SimpleObjectProperty<Location>
- getCurrentLocation() : LocationData
- getCityList(String country) : JSONObject

GENERAL VISUALIZATION - Class Diagram


LocationData
-String city
-String country
+Location(city: String, country: String)
+getCity(): String
+getCountry(): String
+setCity(city: String)
+setCountry(country: String)
+getLatitude(): String
+getLongitude(): String
APIIntegration
+ getWeatherData(location, latitude, longitude) : WeatherData
+ getCurrentLocation() : CurrentLocationData
+ getCityList(country) : CitiesData
+ TimeDataService(String timezone) : TimeData
WeatherDataService
-String apiKey
+WeatherDataRetrieval(apiKey: String)
TimeData
-int weekday
-int weekNumber
-string Date
-string Time
-string TimeZone
-string UtcOffset
+updateAll()
+getWeekday(): int
+getWeekNumber(): int
+getDate(): string
+getTime(): string
+getUtcOffset(): string
WEATHER FORECAST HUB
-LocationDataService locationData
-TimeDataService timeData
-WeatherDataService weatherData
-UserInterface userInterface
+ForecastHubApp()
+getWeatherForecast()
+setLocation(location: Location)
+setTimeInterval(timeInterval: TimeInterval)
+showUserInterface()
LocationDataService
-String apiKey
+LocationDataRetrieval(apiKey: String)
TimeDataService
-String apiKey
+TimeDataRetrieval(apiKey: String)
WeatherData
-Double temperature
-Double humidity
-Double precipitation
-Double windSpeed
-String location
-String weatherCondition
+WeatherData(String location, double temperature, double humidity, double precipitation, double windSpeed, String weatherCondition)
+getLocation(): string
+getPrecipitation(): double
+getTemperature(): double
+getHumidity(): double
+getWindSpeed(): double
+getWeatherCondition(): string
UserInterface
-ForecastHubApp forecastHubApp
+UserInterface(weatherForecastApp: WeatherForecastApp)
+showLocationForm()
+showTimeIntervalForm()
+showWeatherData(weatherData: WeatherData)
uses
uses
uses
has
has
has
has
has
uses

Section 7 - User Interface Design

7.1 User Interface - FIGMA Prototypes

The app shall have 3 user interfaces

1. The Dashboard
2. The Chart Screen
3. The Tabular Chart Screen

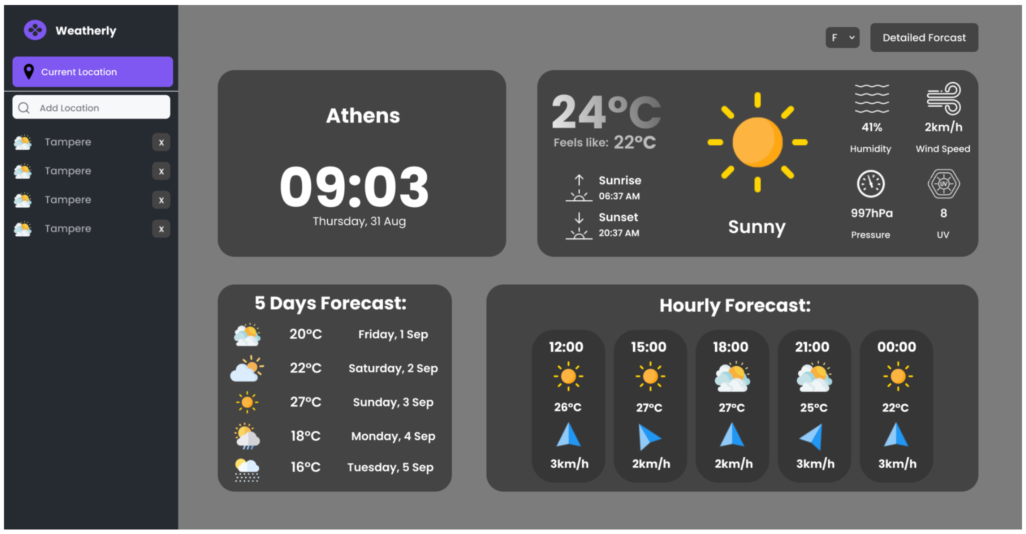


Fig 7.1: Dashboard Wireframe

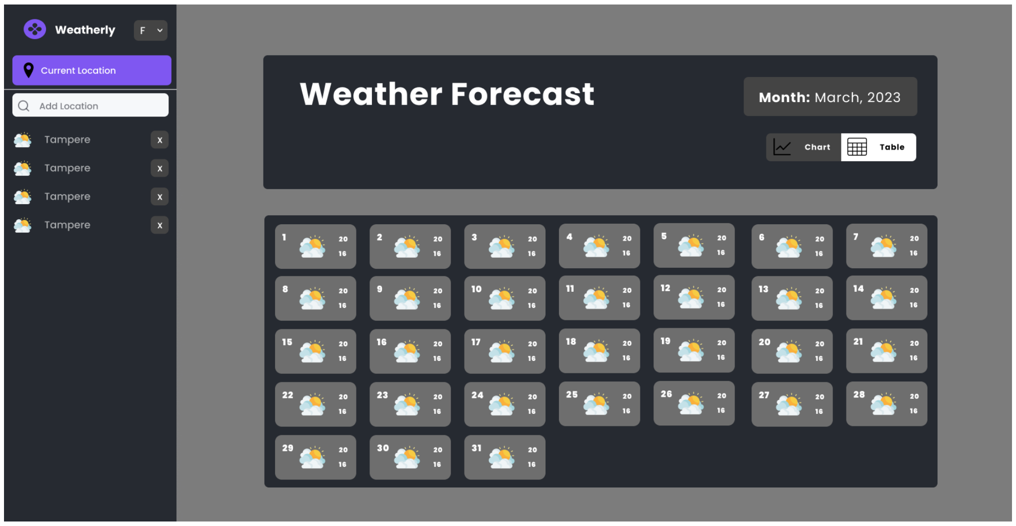


Fig 7.2: The Tabular Chart Screen Wireframe

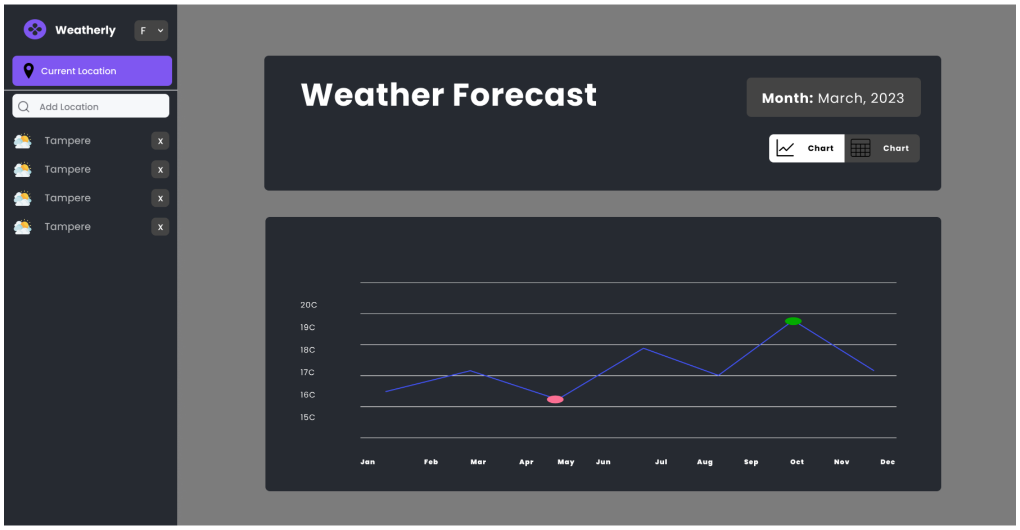


Fig 7.3: The Chart Screen Wireframe

UI Screens - Adherence to the Original design

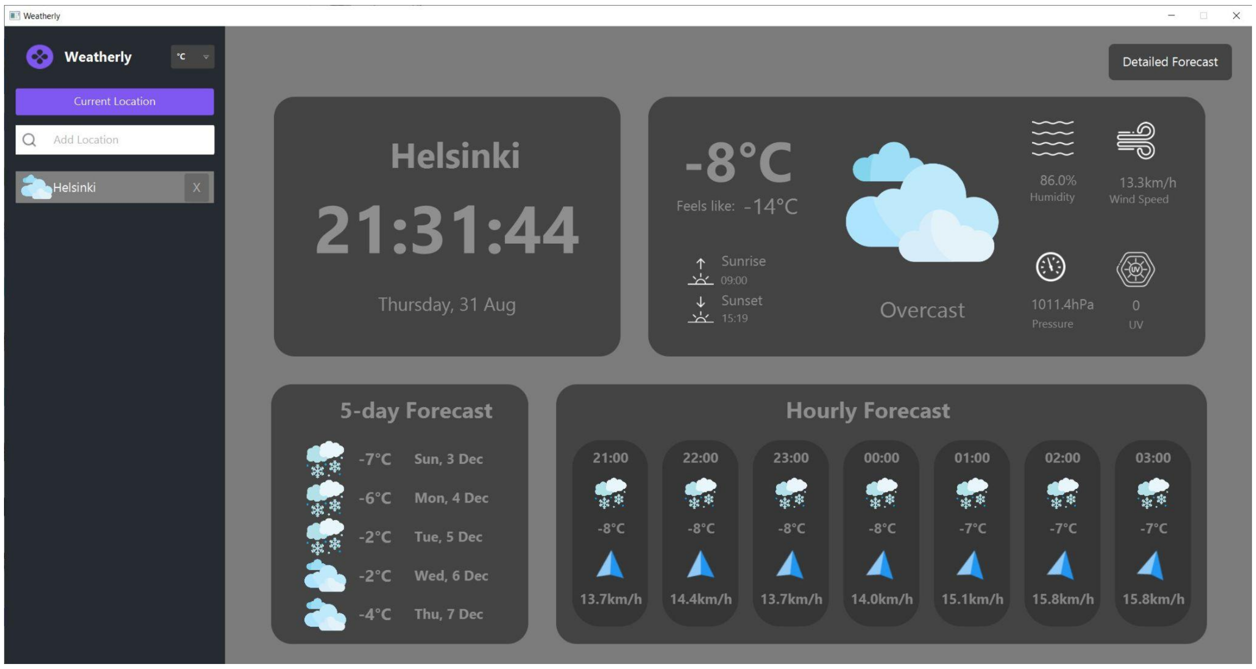


Fig 7.4: Dashboard

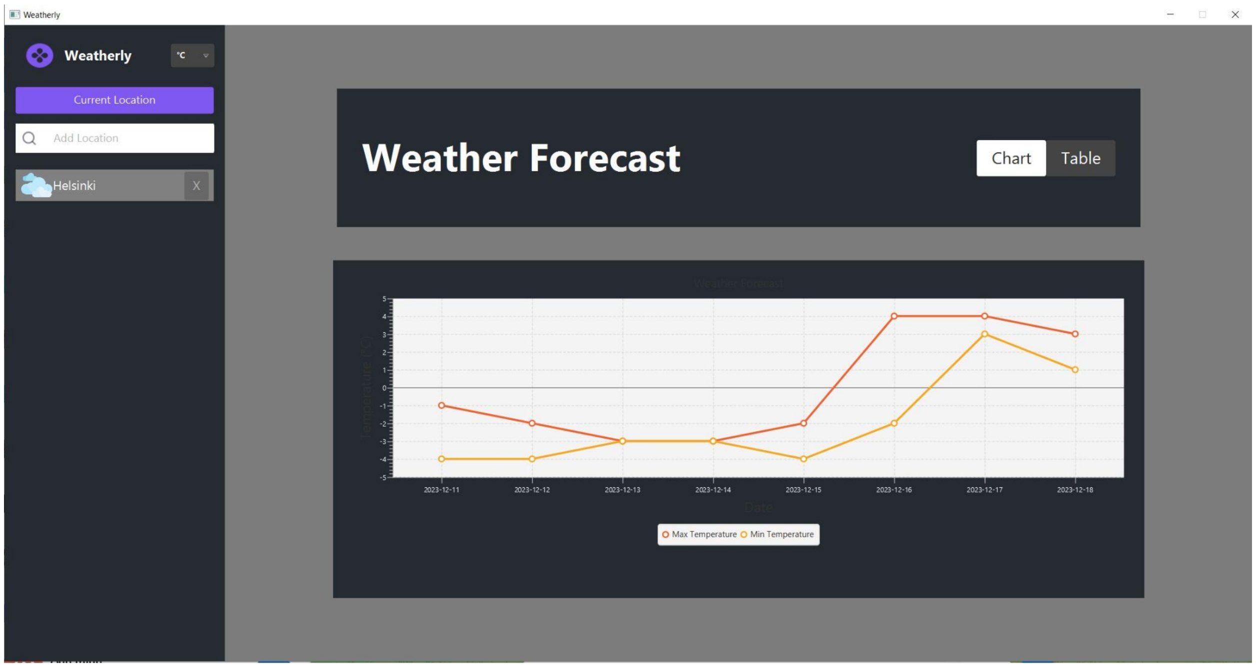
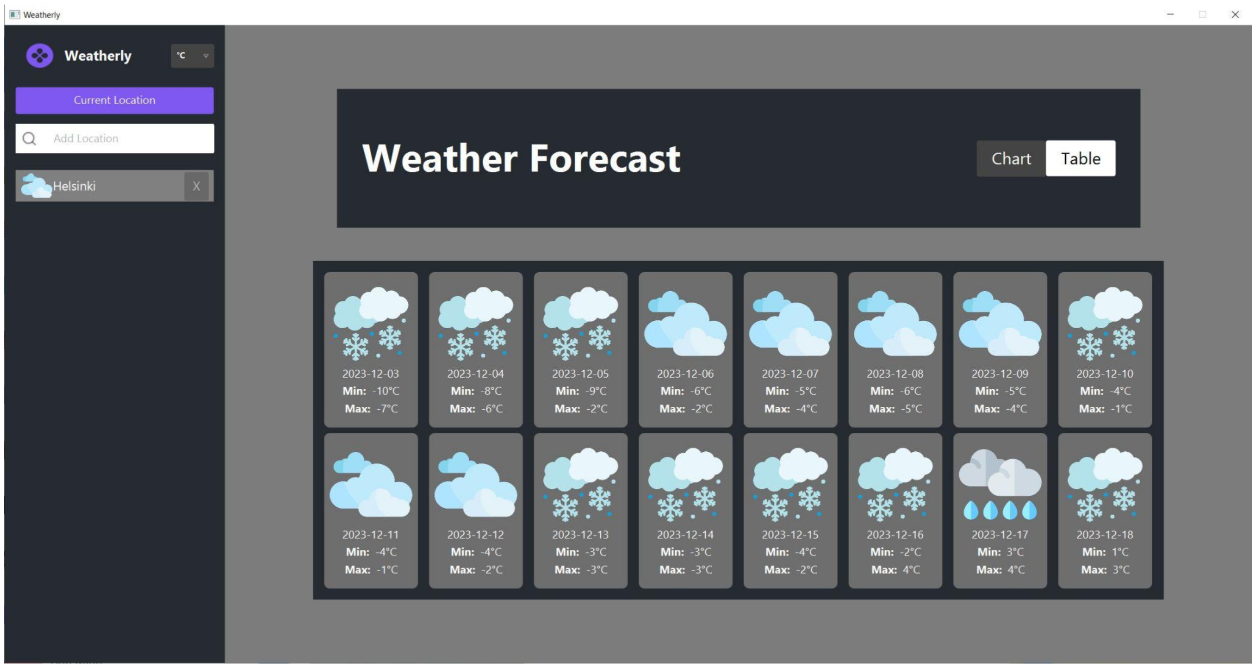


Fig 7.5: The Chart Screen



2023-12-03	**Min:** -10°C	**Max:** -7°C
2023-12-04	**Min:** -8°C	**Max:** -6°C
2023-12-05	**Min:** -9°C	**Max:** -2°C
2023-12-06	**Min:** -6°C	**Max:** -2°C
2023-12-07	**Min:** -5°C	**Max:** -4°C
2023-12-08	**Min:** -6°C	**Max:** -5°C
2023-12-09	**Min:** -5°C	**Max:** -4°C
2023-12-10	**Min:** -4°C	**Max:** -1°C
2023-12-11	**Min:** -4°C	**Max:** -1°C
2023-12-12	**Min:** -4°C	**Max:** -2°C
2023-12-13	**Min:** -3°C	**Max:** -3°C
2023-12-14	**Min:** -3°C	**Max:** -3°C
2023-12-15	**Min:** -4°C	**Max:** -2°C
2023-12-16	**Min:** -2°C	**Max:** 4°C
2023-12-17	**Min:** 3°C	**Max:** 4°C
2023-12-18	**Min:** 1°C	**Max:** 3°C

Fig 7.6: The Tabular Chart Screen

7.2 User Interface Navigation Flow

The app starts up with the dashboard screen. In the dashboard’s top right there is a button “Detailed forecast”. Pressing this button the user can navigate to the 16 Days tabular chart screen. In the chart screen the user can toggle between views of tabular and chart data.

Section 8 - Tools and Techniques

NetBeans for Java Development:

NetBeans is a feature-rich Integrated Development Environment (IDE) tailored specifically for Java and offers strong support for JavaFX. It simplified Java development by providing a powerful, user-friendly, and customizable environment.

JavaFX for User Interface:

JavaFX provides a native platform look and feel, ensuring that the application's UI is consistent and visually appealing across different operating systems. Its event handling system was crucial for real-time updates for weather data.

Figma for Prototyping:

Figma offered the ability to create interactive prototypes, allowing to simulate user interactions in a collaborative environment. It simplified the process of exporting design assets and specifications, making it easier for developers to implement the design as intended.

Builder.io AI Plugin:

Builder.io offers a plugin for Figma that allows users to embed dynamic content and components from Builder.io into Figma designs. This integration enabled us to design the initial wireframes from the project idea. It also facilitated interactivity within Figma, providing a more functional representation of the final product compared to static mockups.

ChatGPT 3.0 as AI Tool in Software Development Life Cycle:

- ☐ **Problem Solving**

Throughout the development of this project, ChatGPT primarily aided in breaking down requirements, solving coding issues and challenges, and offering solutions. Whether it's designing algorithms, explaining concepts, or providing code examples, clarifying prompts and integrating this AI tool was helpful.

- ☐ **Code Implementation and Review**

It supported implementing various backend and frontend features and functionalities, and offered guidance on how to structure and document our code effectively keeping in consideration the architecture and design pattern we implemented.

- ☐ **Documentation Assistant**

It initially helped in crafting the proposal with possibilities of must have, could have and should have features of our weather forecast application. ChatGPT also assisted in documenting the codebase properly.

- ☐ **Understanding APIs and Data Structures**

Explaining how to integrate APIs, get all the model-view-controller components of the project work in harmony, and work with different data structures is a part of the development process, and ChatGPT provided insights into these aspects

- ☐ **Continuous Learning**

It helped in both simple and complex scenarios, allowing for continuous learning and growth in software development.

Section 9 - Future Enhancement in Design Features

The future scalability of this project holds great promise, with the potential to enhance its capabilities and accommodate a broader user base including:

- The integration of offline access will enable users to access previously viewed weather data even without an internet connection, enhancing the application's usability in remote areas or during network disruptions.

- The implementation of data synchronization across multiple devices will provide a seamless experience for users who want to access their personalized preferences and saved locations from different platforms.

- Customizable units of measurement, allowing users to choose their preferred metrics, will add flexibility and accessibility for a wider international audience.

- The inclusion of historical weather data for tracking weather patterns will open up opportunities for research, analysis, and a deeper understanding of climate trends.

- Lastly, by expanding the weather alerts system, the application can offer more detailed and location-specific warnings, catering to the diverse needs of users and ensuring their safety during extreme weather conditions.

Section 10 - Self Evaluation

	Reflections By Team
● Adherence to the Original Design	Provided in the initial design document, it appears that we have followed the proposed design principles quite closely. The use of the Model-View-Controller (MVC) architecture has been maintained. We have implemented the Singleton design pattern for our services. We have effectively handled the integrations with API. A complete assessment of the adherence to the design would require a detailed review of the entire codebase, considering factors like error handling, user interactions, and application flow. We have created a JavaFX-based user interface, as indicated in the original plan. We believe as for the requisites defined we were

• Feature Implementation Progress	able to depict 3 views as illustrated in the prototype submitted. These were highlighted as the key features where we were able to implement data retrieval and visualization.
• Design Quality Assessment	The use of the Model-View-Controller (MVC) pattern promotes maintainability by separating concerns. The modular organization of the project facilitates scalability. It complements easy unit testing of individual components. While not explicitly demonstrated in code, the Services include error handling mechanisms to manage exceptions.
• Rationalizing Design	MVC Architecture | Separation of Concerns Models | Data Encapsulation Views | User Interface Separation Controllers | User Interaction Handling Services | Data Abstraction Singleton Design Pattern | Single Instance Structured Project Hierarchy | Organization and Maintainability Testability | Unit Testing

Section 11 - Division Of Work

Ayman Asad Khan	Rafin Akhter Utshaw	Phuong Nguyen	Kasper Matias Kivisto	Abdullah Muhammad Ashraf
Documentation Backend (Weather Data Service)	Prototyping Frontend (Location, Weather, Graphs)	Frontend (Weather Panels)	Backend (Time Data Service)	Backend (Location Data Service)