TA	COMP.SE.110. SOFTWARE DESIGN PROJECT	03/12/2023
Kristian Skogberg	Software Project	GROUP NAME: DESIGN DRIVEN DEVS Long Nguyen Minh Hoang

CONTENTS

LIST OF FIGURES

1. Introduction

Weather applications have become an integral part of our daily lives, helping individuals plan their schedules, decide the activities, and be cautious about weather conditions. This project seeks to create a friendly, with some functionality and solutions by integrating Application Programming Interfaces (APIs) into a graphical user interface (GUI) implementation to deliver necessary, accurate, and up-to-date weather data to users.

2. Scope and Objectives

The project is aimed at these purposes:

- Utilizing object-oriented design to create a Weather system.
- Integration of a GUI as a component of the program
- Applying the Model-View-Controller (MVC) software design pattern for development, classification of the project
- Incorporating external APIs, such as OpenWeatherInfo, into the implementation
- Providing and ensuring the high-quality GUI adapts to Desktop screens by applying heuristic, principles of design guidelines based on human technology interaction topics.

3. Overall Architecture:

The application can follow the MVC architectural pattern, which is suitable for JavaFX applications. This pattern helps maintain the separation of concerns and makes your application more maintainable.

4. GUI Implementation

4.1. Packages:

The project's packages will be divided into four parts, which will support us in minimizing the unnecessary teamwork Git workflow conflicts, and helpful for us in dividing the tasks during the process.

The packages will be separated based on each window, which can be demonstrated in Figure 1.

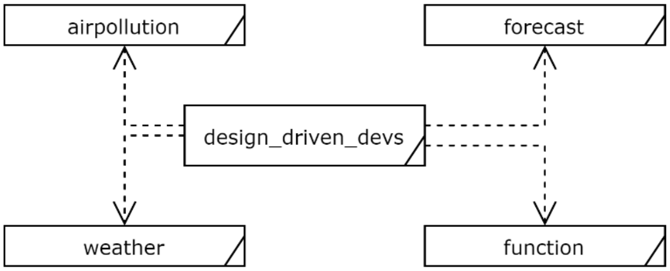


Figure 1. *Packages demonstration of the Project*

Discussing each package, the "airpollution" package handles the "Air Pollution" screen (Third Screen), the "forecast" package handles the "Forecast" screen, the "weather" package handles the "Weather" screen, the "function" package handles the necessary functions we need to implement and process, and the necessary APIs we would like to extract.

The relationship between the classes can be demonstrated in Figure 2.

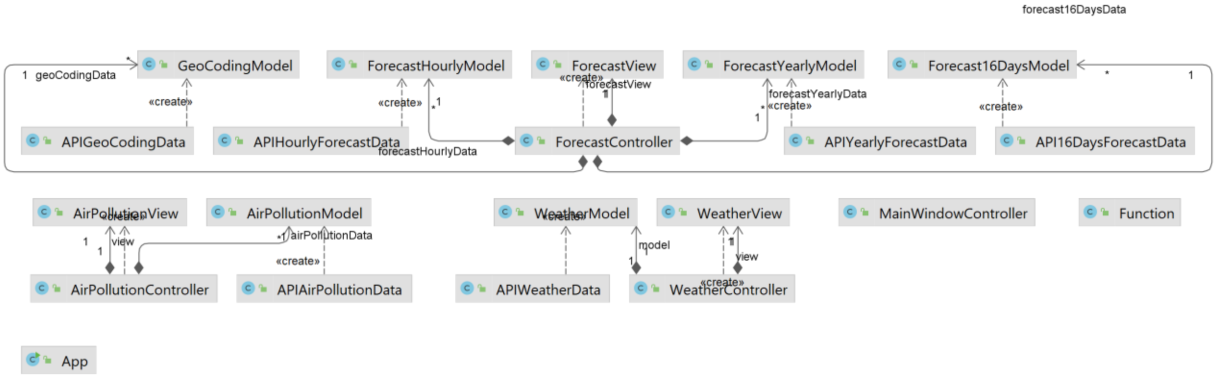


Figure 2. *Current classes' relationship*

The dependency of the project can be presented in Figure 3:

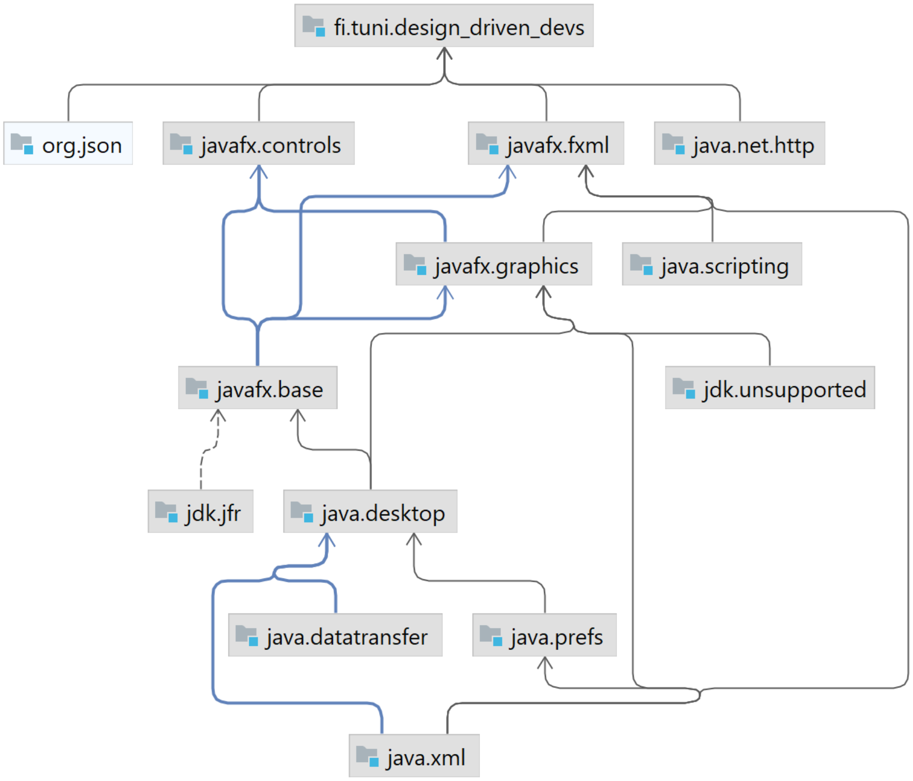


Figure 3. *Dependency of the project*

We have implemented the application by using JavaFX, JSON, and Apache HttpClient. JavaFX has been used for the implementation of the GUI, and Apache HttpClient and JSON have been used for reading the contents of the JSON files when we extract the APIs

4.2. Selection Screen

The GUI layout can be presented in Figure 4.

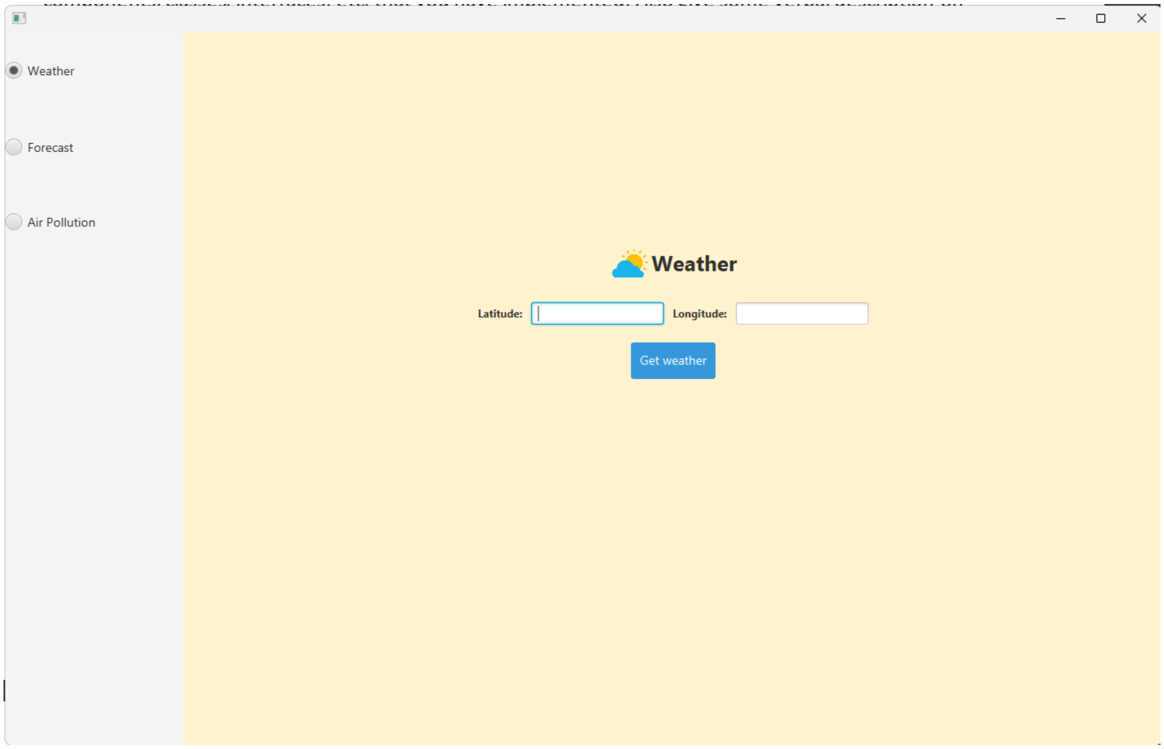


Figure 4. GUI layout for the first screen

From the beginning when we ran the project, the “Weather” screen has been set up as the default screen. However, in the Selection screen, there would be three options, which could be switched by using the Toggle button on the left side. Since UI design doesn’t relate to the project’s grading criteria, I have made the GUI to be easily implemented, leading to the asynchronization of the color background. Moreover, the UML diagram for the necessary classes can be illustrated in Figure 5.

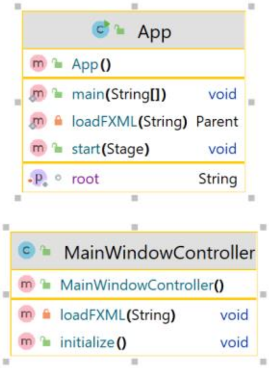


Figure 5. UML diagram for the selection screen

4.3. First Screen:

We would choose to apply OpenWeather API for this screen to demonstrate the up-to-date weather data.

API Links:

- https://openweathermap.org/api/hourly-forecast

Components:

- View: JavaFX UI with latitude and longitude input fields, a "Fetch" button, and a display area for weather data.
- Controller: Handles user input, communicates with the API, and updates the view.
- Model: Represents weather data received from the API.

To demonstrate our idea, we would use a UML diagram, as in Figure 6, and the GUI presentation in Figure 7.

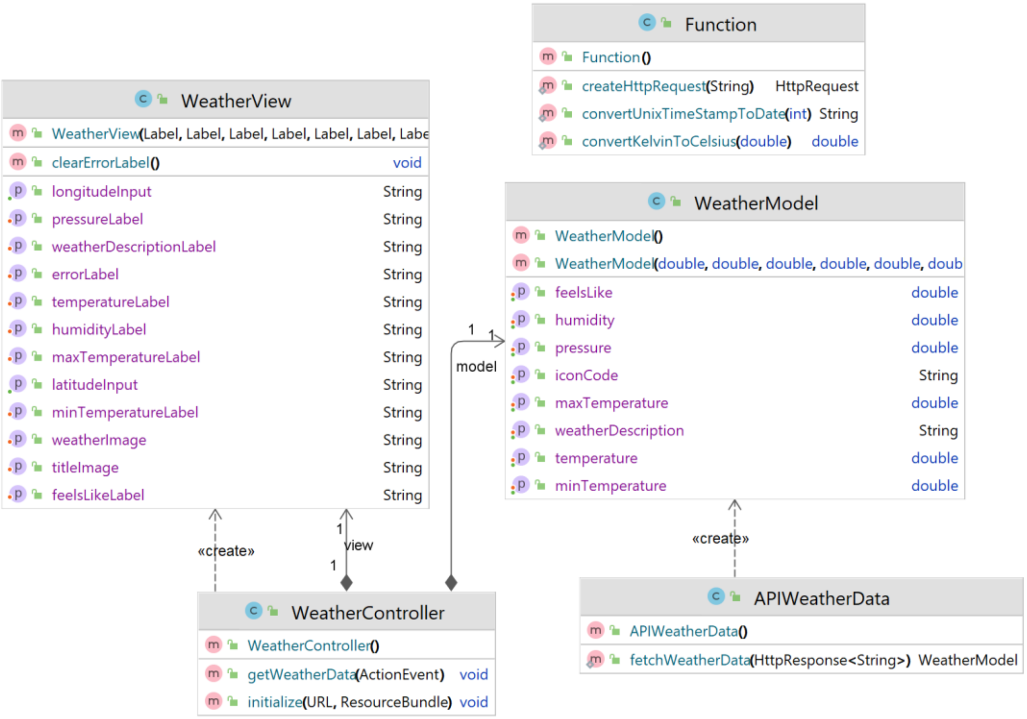
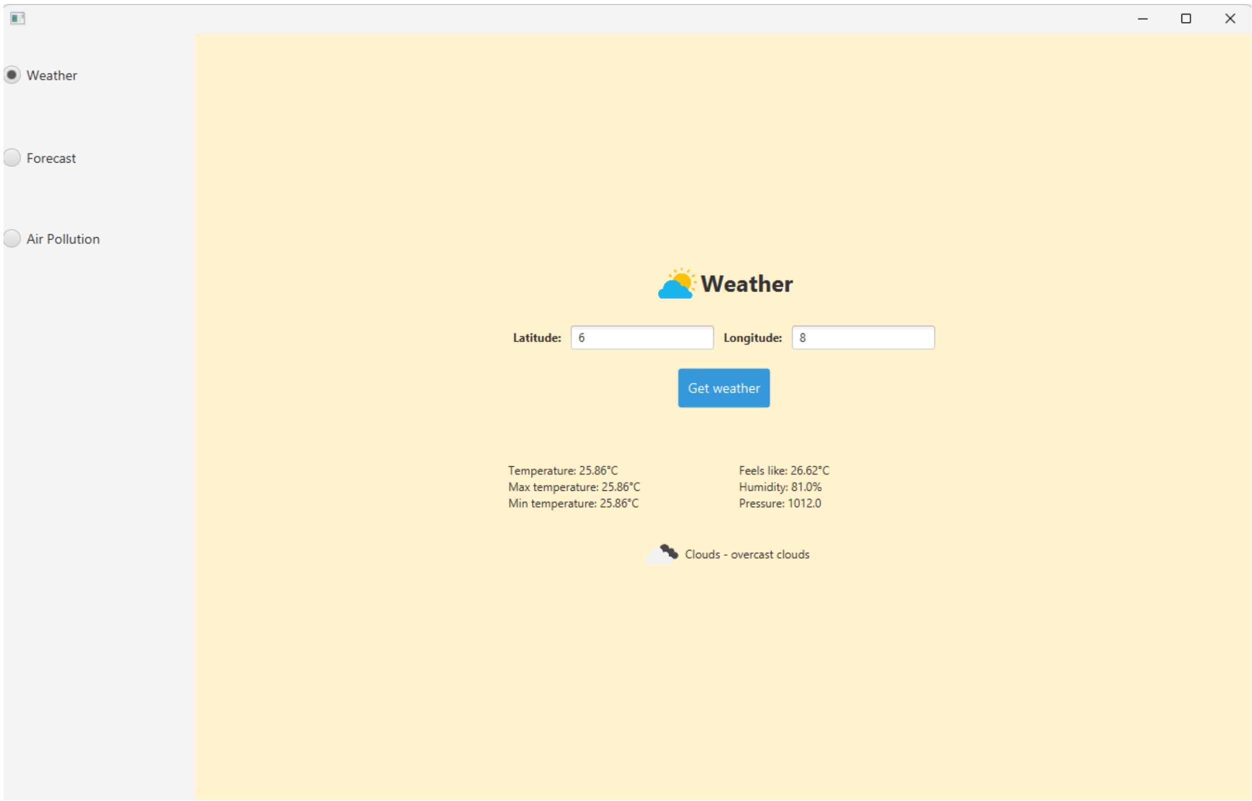


Figure 6. UML presentation for the First Screen necessary components



Latitude: 6 Longitude: 8

Temperature: 25.86°C
Max temperature: 25.86°C
Min temperature: 25.86°C

Feels like: 26.62°C
Humidity: 81.0%
Pressure: 1012.0

Clouds - overcast clouds

Figure 7. GUI layout for the first screen when fetching the data.

4.4. Second Screen:

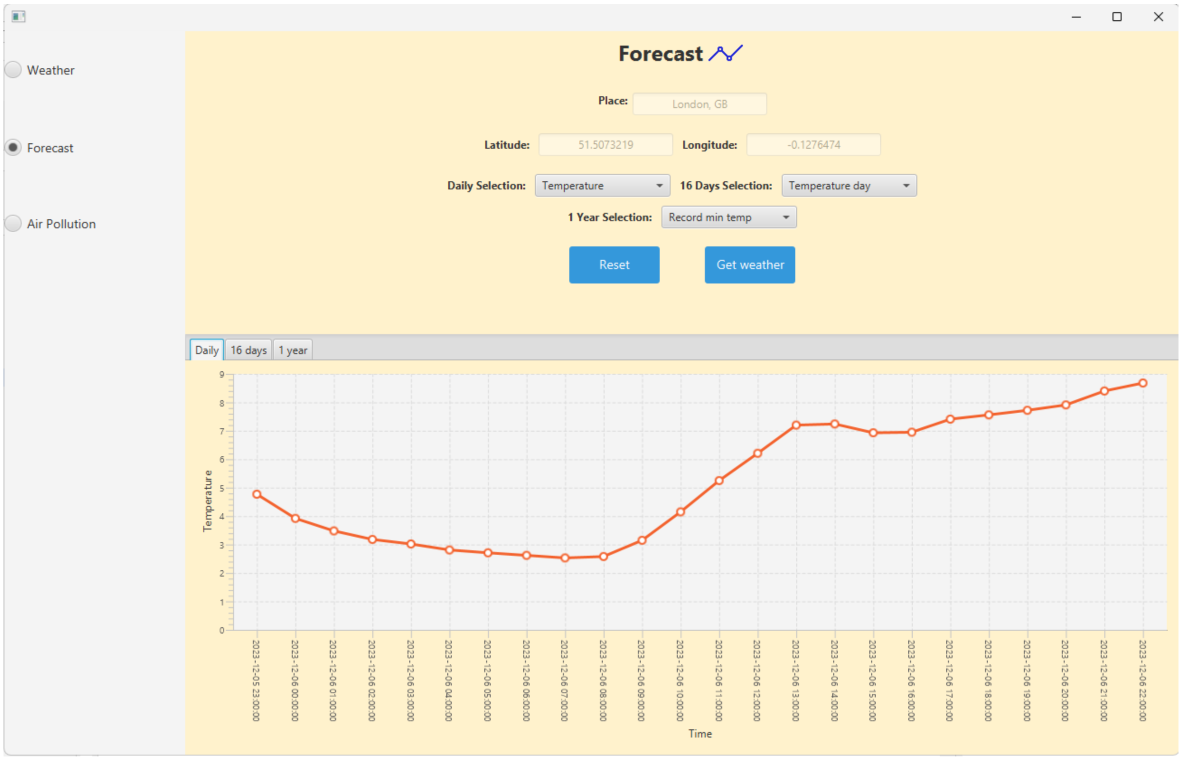
API Links:

- https://openweathermap.org/api/hourly-forecast
- https://openweathermap.org/forecast16
- https://openweathermap.org/api/geocoding-api
- https://openweathermap.org/api/statistics-api

Components:

- View: JavaFX UI with options to select either 4 days, 16 days, or 365 days forecast, a tab pane to display forecast data.
- Controller: Handles user input, communicates with the API, and updates the view.
- Model: Represents forecast data received from the API.

The second screen is presented in Figures 8, 9 and 10 and the UML presentation is presented in Figure 11



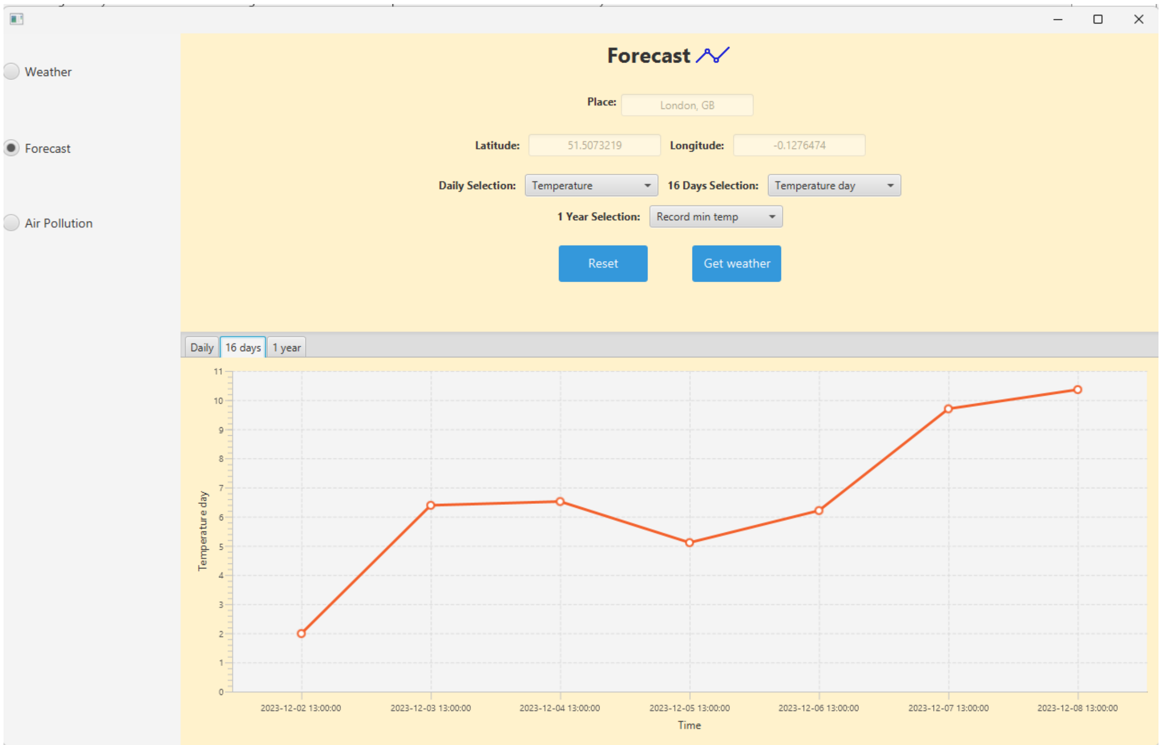
Place: London, GB

Latitude: 51.5073219 Longitude: -0.1276474

Daily Selection: Temperature 16 Days Selection: Temperature day

1 Year Selection: Record min temp

Figure 8. *Sketching of the second screen in the Daily tab*



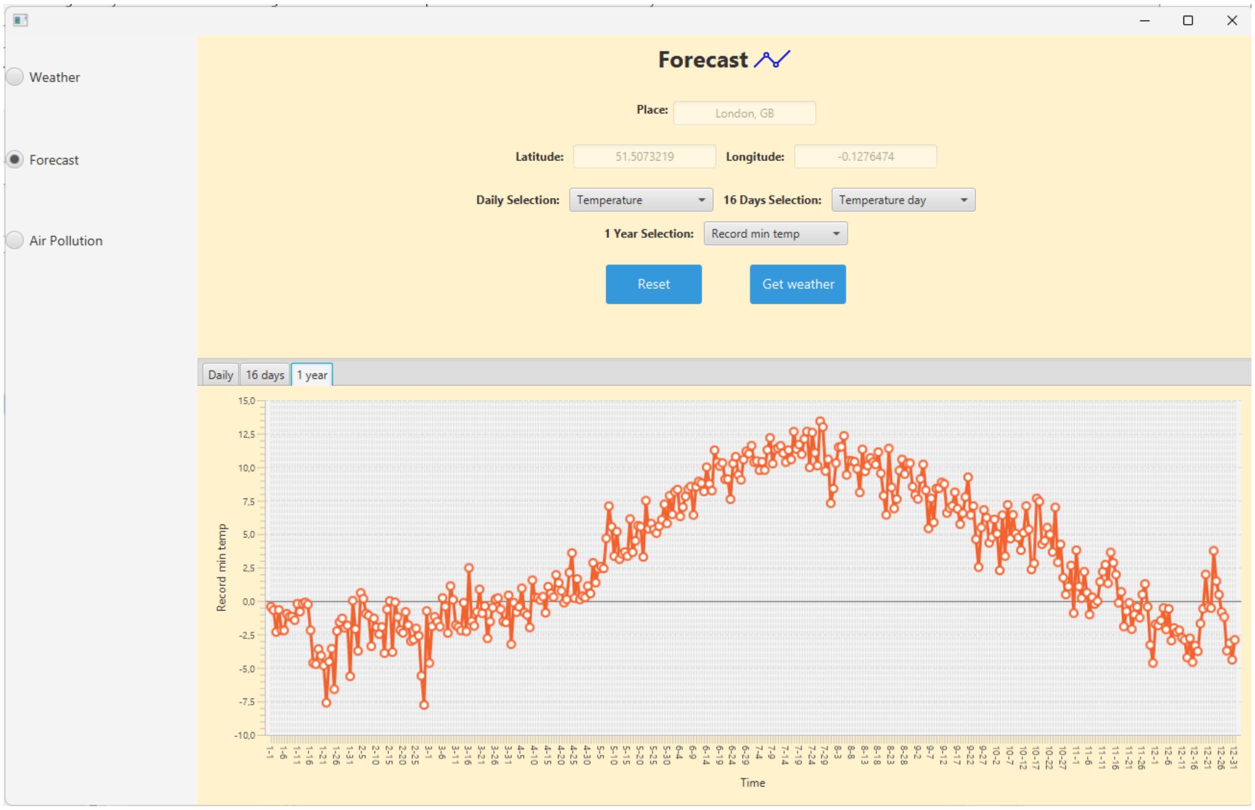
Place: London, GB

Latitude: 51.5073219 Longitude: -0.1276474

Daily Selection: Temperature 16 Days Selection: Temperature day

1 Year Selection: Record min temp

Figure 9. *Sketching of the second screen in the 16 days tab*



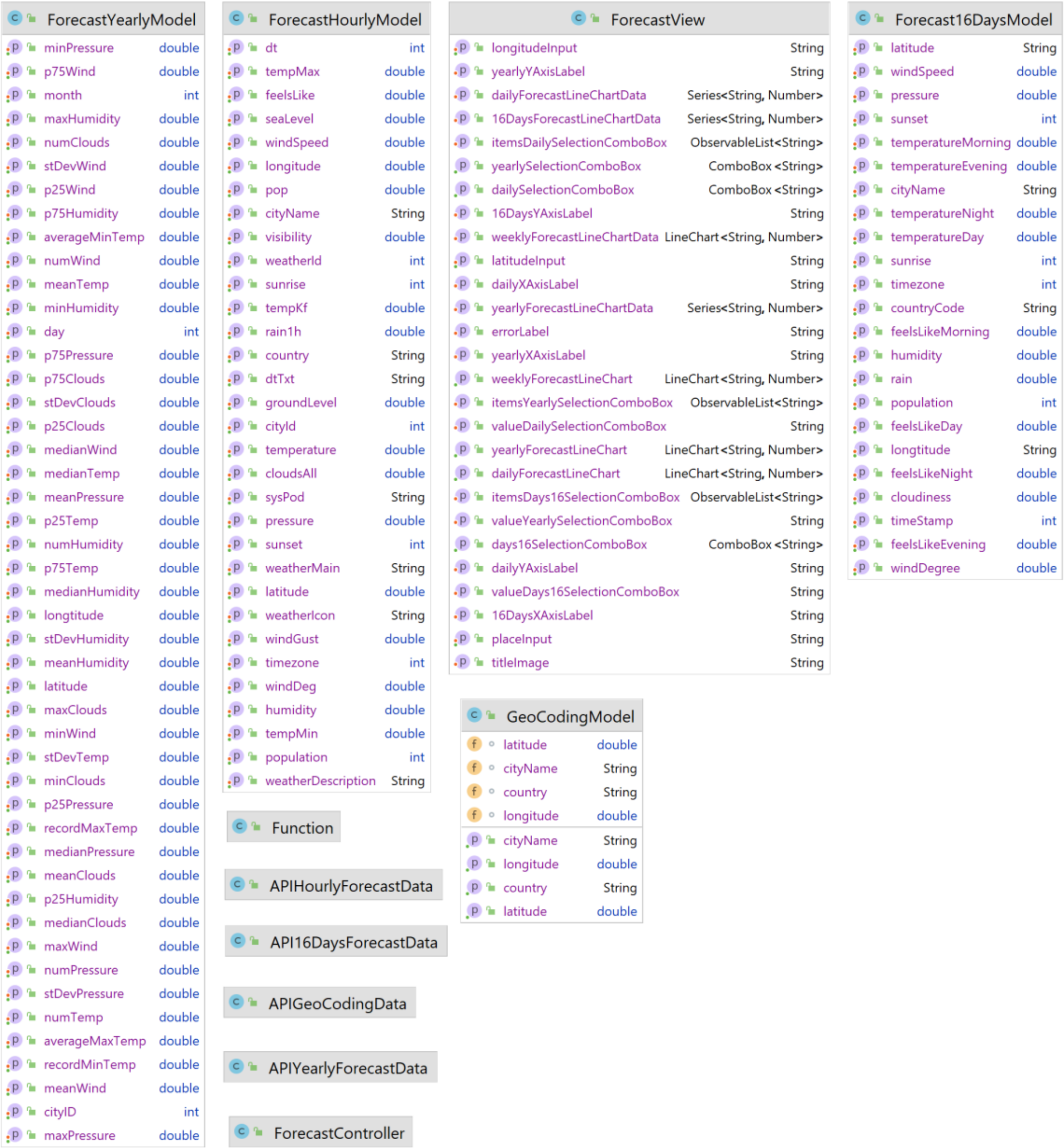
Place: London, GB

Latitude: 51.5073219 Longitude: -0.1276474

Daily Selection: Temperature 16 Days Selection: Temperature day

1 Year Selection: Record min temp

Figure 10. *Sketching of the second screen in the 1 year tab*



ForecastYearlyModel	
minPressure	double
p75Wind	double
month	int
maxHumidity	double
numClouds	double
stDevWind	double
p25Wind	double
p75Humidity	double
averageMinTemp	double
numWind	double
meanTemp	double
minHumidity	double
day	int
p75Pressure	double
p75Clouds	double
stDevClouds	double
p25Clouds	double
medianWind	double
medianTemp	double
meanPressure	double
p25Temp	double
numHumidity	double
p75Temp	double
medianHumidity	double
longtitude	double
stDevHumidity	double
meanHumidity	double
latitude	double
maxClouds	double
minWind	double
stDevTemp	double
minClouds	double
p25Pressure	double
recordMaxTemp	double
medianPressure	double
meanClouds	double
p25Humidity	double
medianClouds	double
maxWind	double
numPressure	double
stDevPressure	double
numTemp	double
averageMaxTemp	double
recordMinTemp	double
meanWind	double
cityID	int
maxPressure	double

ForecastHourlyModel	
dt	int
tempMax	double
feelsLike	double
seaLevel	double
windSpeed	double
longitude	double
pop	double
cityName	String
visibility	double
weatherId	int
sunrise	int
tempKf	double
rain1h	double
country	String
dtTxt	String
groundLevel	double
cityId	int
temperature	double
cloudsAll	double
sysPod	String
pressure	double
sunset	int
weatherMain	String
latitude	double
weatherIcon	String
windGust	double
timezone	int
windDeg	double
humidity	double
tempMin	double
population	int
weatherDescription	String

ForecastView	
longitudeInput	String
yearlyYAxisLabel	String
dailyForecastLineChartData	Series<String, Number>
16DaysForecastLineChartData	Series<String, Number>
itemsDailySelectionComboBox	ObservableList<String>
yearlySelectionComboBox	ComboBox<String>
dailySelectionComboBox	ComboBox<String>
16DaysYAxisLabel	String
weeklyForecastLineChartData	LineChart<String, Number>
latitudeInput	String
dailyXAxisLabel	String
yearlyForecastLineChartData	Series<String, Number>
errorLabel	String
yearlyXAxisLabel	String
weeklyForecastLineChart	LineChart<String, Number>
itemsYearlySelectionComboBox	ObservableList<String>
valueDailySelectionComboBox	String
yearlyForecastLineChart	LineChart<String, Number>
dailyForecastLineChart	LineChart<String, Number>
itemsDays16SelectionComboBox	ObservableList<String>
valueYearlySelectionComboBox	String
days16SelectionComboBox	ComboBox<String>
dailyYAxisLabel	String
valueDays16SelectionComboBox	String
16DaysXAxisLabel	String
placeInput	String
titleImage	String

Forecast16DaysModel	
latitude	String
windSpeed	double
pressure	double
sunset	int
temperatureMorning	double
temperatureEvening	double
cityName	String
temperatureNight	double
temperatureDay	double
sunrise	int
timezone	int
countryCode	String
feelsLikeMorning	double
humidity	double
rain	double
population	int
feelsLikeDay	double
longtitude	String
feelsLikeNight	double
cloudiness	double
timeStamp	int
feelsLikeEvening	double
windDegree	double

GeoCodingModel	
latitude	double
cityName	String
country	String
longitude	double
cityName	String
longitude	double
country	String
latitude	double

Function

APIHourlyForecastData

API16DaysForecastData

APIGeoCodingData

APIYearlyForecastData

ForecastController

Figure 11. *UML representation (excluding methods) related to the second screen*

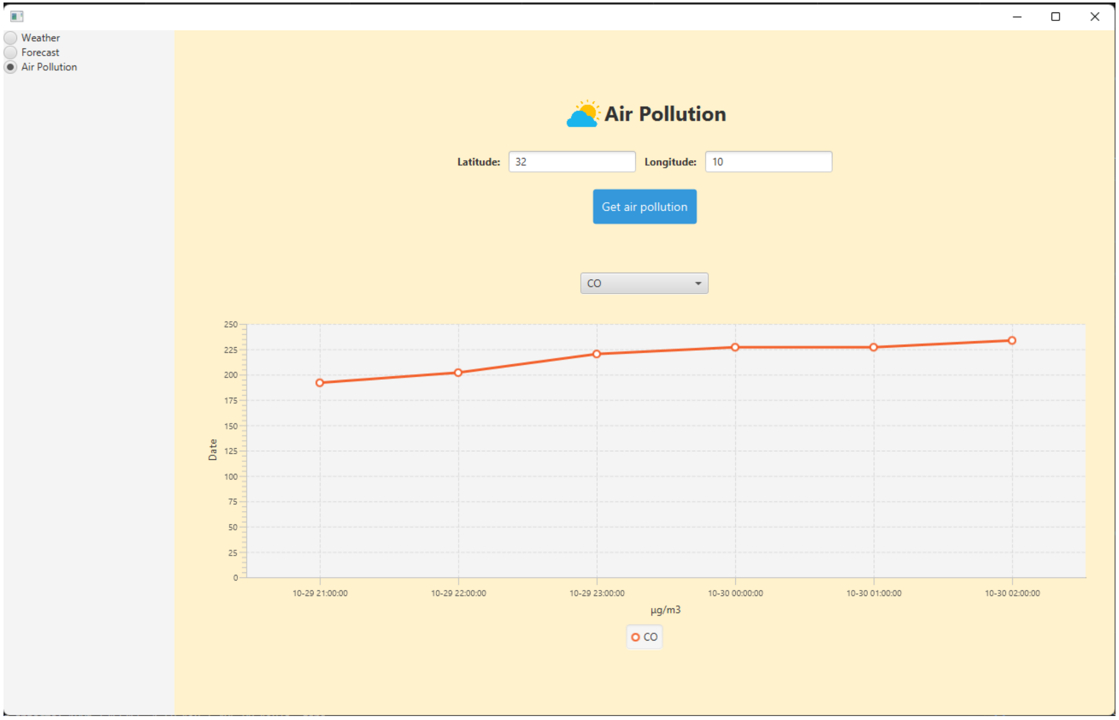
4.5. Third Screen:

API Link: https://openweathermap.org/api/air-pollution

Components:

- View: JavaFX UI with latitude and longitude input fields and a display area for air pollution forecast data.
- Controller: Handles user input, communicates with the API, and updates the view.
- Model: Represents air pollution data received from the API.

We would use the sketch and the UML diagram to demonstrate our idea precisely.



○ Weather
○ Forecast
● Air Pollution

Latitude: 32 Longitude: 10

CO

Figure 12. Sketching of the third screen

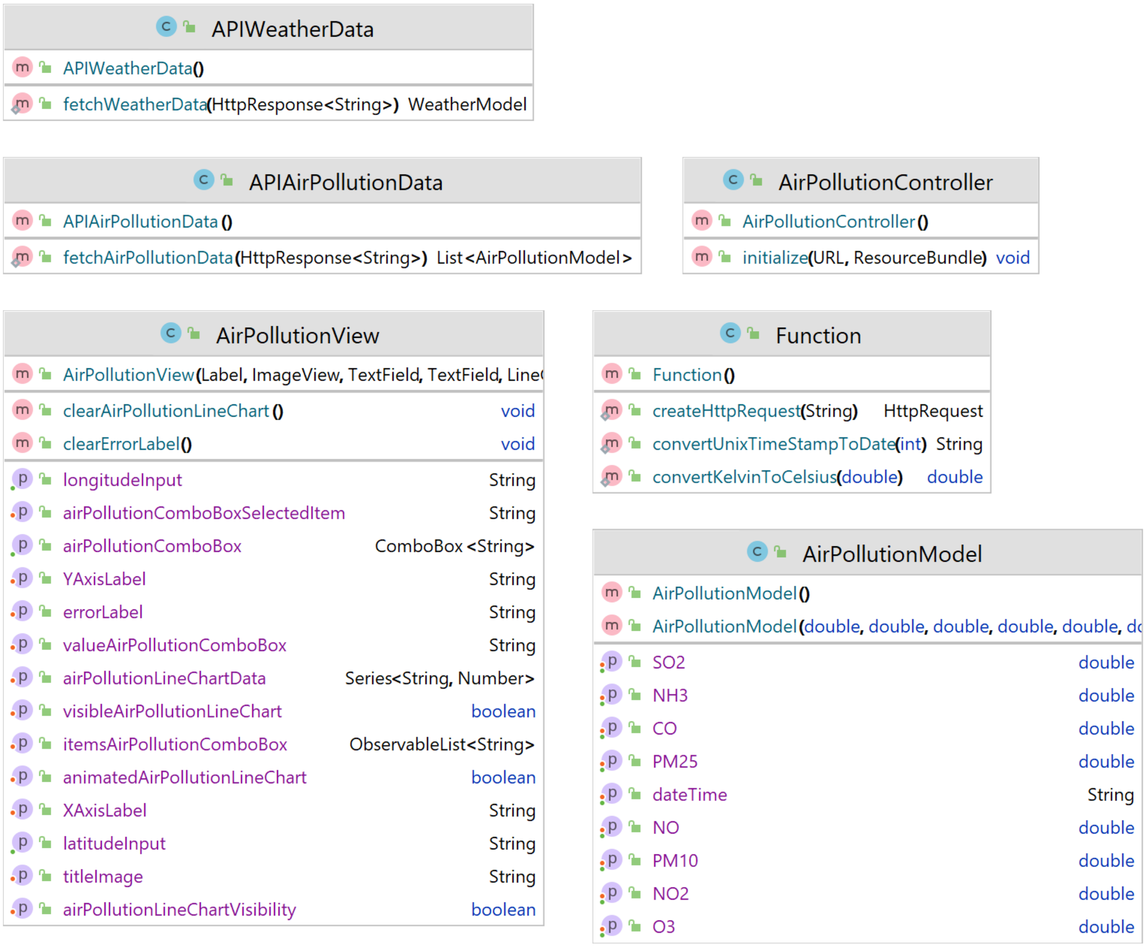

APIWeatherData
APIWeatherData()
fetchWeatherData(HttpResponse<String>) WeatherModel
APIAirPollutionData
APIAirPollutionData ()
fetchAirPollutionData (HttpResponse<String>) List<AirPollutionModel>
AirPollutionController
AirPollutionController()
initialize(URL, ResourceBundle) void
AirPollutionView
AirPollutionView(Label, ImageView, TextField, TextField, Line
clearAirPollutionLineChart () void
clearErrorLabel() void
longitudeInput String
airPollutionComboBoxSelectedItem String
airPollutionComboBox ComboBox<String>
YAxisLabel String
errorLabel String
valueAirPollutionComboBox String
airPollutionLineChartData Series<String, Number>
visibleAirPollutionLineChart boolean
itemsAirPollutionComboBox ObservableList<String>
animatedAirPollutionLineChart boolean
XAxisLabel String
latitudeInput String
titleImage String
airPollutionLineChartVisibility boolean
Function
Function()
createHttpRequest(String) HttpRequest
convertUnixTimeStampToDate(int) String
convertKelvinToCelsius(double) double
AirPollutionModel
AirPollutionModel()
AirPollutionModel(double, double, double, double, double, dc
SO2 double
NH3 double
CO double
PM25 double
dateTime String
NO double
PM10 double
NO2 double
O3 double

Figure 13. *UML representation related to the third screen.*

5. API Integration and Usage

Discussing API Integration, we would create separate classes that implement the data provider. These classes will interact with the 6 OpenWeather APIs and return the data required on each screen. Furthermore, we would like to apply the interaction between these five APIs, which can be supported for the missing deficiency of each APIs and demonstrate the convenience we would like to bring for the users.

Discussing the usage of the API, we would demonstrate the main idea for each screen as follows:

- For the First Screen, you will use the "Current Weather Data" API to fetch current weather data based on latitude and longitude.
- For the Second Screen, you will use the "Hourly Forecast 4 days", "Daily Forecast 16 days", the GeoCoding, and the Statistics APIs. Therefore, based on the forecast, we could use it to demonstrate the graphs as a function of a range of times.
- For the Third Screen, you will use the "Air Pollution" API to fetch air pollution data based on latitude and longitude.

6. Design Patterns:

We have used the singleton pattern for the project, which can be found as Funtion.java. Moreover, with the SOLID principles, we have applied the SOI principle. The S principle can be caught in our MVC model when we split each model in each file. The O principle can be used in Function.java when we choose public static for the extend. Moreover, we could experience the O principle in the private variables in necessary classes. The I principle has been inherited and used in different API extraction in the "function" folder, as different classes with different interfaces. For the L and D principles, our selection of the class structure does not need them, leading to our selection not to follow them.

Discussing the reason why we have used this implementation since we thought that with a simple system like this, a class for extracting the API and using MVC for all the other classes would be easier for us to handle the data since we could observe that the ways to extract the APIs are different. Moreover, the code is easier to maintain since the MVC model has split the classes into satisfied classes. You can observe that when we observe our packages and GUI Implementation above.

7. Self-Evaluation

For the self-evaluation part, we think that we have implemented it well as a group of 2. The design pattern for the project is simple and does not need to be complicated since the simple structure of the project. However, there are some things we can optimize, for example using the Factory Pattern for the API extraction, The integration between each API in each screen, and the UI design of the website. However, due to the workload and the limitation of group member size, we could not implement it.

Moreover, with the original design, we think that we have implemented the same. There are no changes if we remember correctly. Therefore, there are no conflicts between our group work.

8. `The use of AI

To be supported for the project, we have used GitHub Copilot for the coding implementation. Fundamentally, we would write the comment, showing the functionality of the code, and then wait for the Copilot's response, and confirm the result. However, we would ensure that all the ideas of the implementation have been inherited from our ideas.

Therefore, we have no understanding about the knowledge of using AI in software design except to save lots of time for the code implementation.