# Design document

COMP.SE. 110 – Software Design

Akseli Epäilys 50005819, Miro Anttila 50255148,

Noora Sassali H299753, Ville Hietanen VH72245

# Contents

# 1 Introduction

This document contains a design plan for the application implemented as a group assignment on course COMP.SE.110 Software Design. Section 2 briefly presents the designed application and its planned features. Section 3 views the planned architectural design, followed by details over the APIs used in Section 4. Finally, we conclude the paper in Section 5 with a self-evaluation, where we reflect on the decisions made and design solutions chosen for the application, and how well they were implementable in practice.

# 2 Environmental data logger

The initial idea for the application was generated by using generative language model, OpenAI's ChatGPT-3.5 [1]. Among the ideas suggested by the model, we decided to choose "Environmental data logger" for the project topic. After innovating the idea with few ChatGPT prompts, the idea was further refined and discussed among the group, until a common agreement of the core features was reached.

The initial draft for the application can be shortly described as follows: Environmental data logger retrieves real-time data from different APIs and visualizes the data in the form of charts and maps. The user can choose a timespan for the data and what data is shown. The data retrieved from the APIs includes temperature, humidity, and air quality. In addition, the user can save some of the view settings for the next session when the application is used. The application is implemented in Java, using JavaFX application platform [2].

Based on this short description, prototyping (Figure 1) and design phase of the implementation was started.

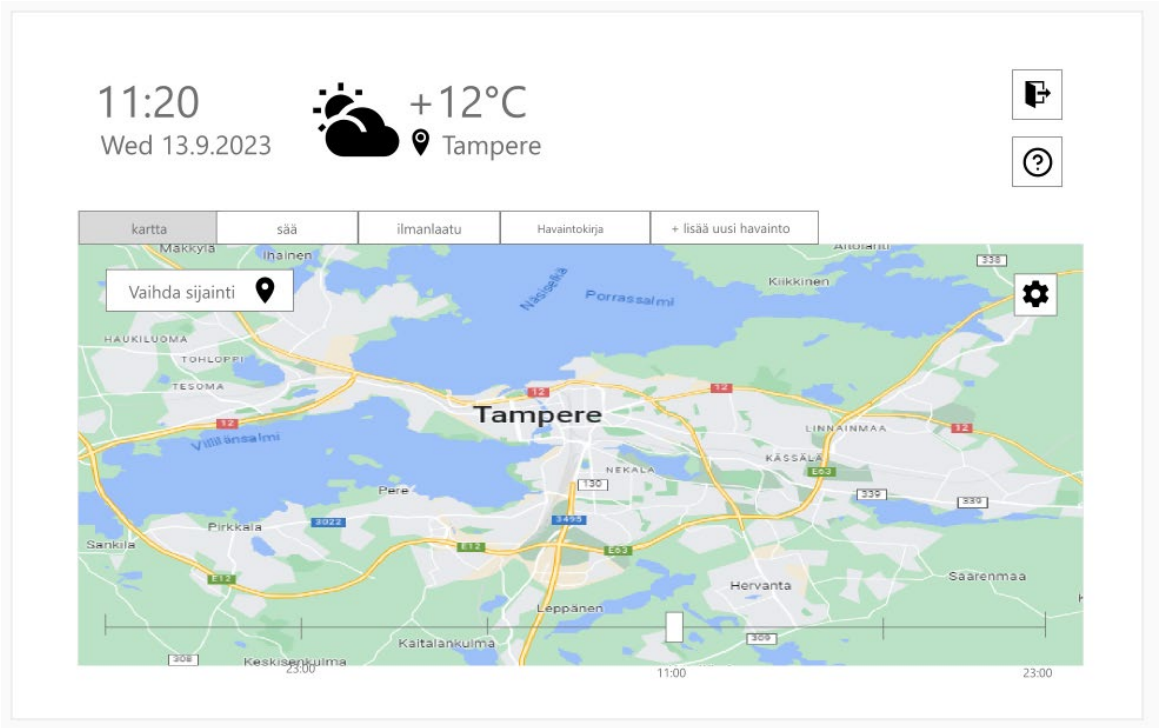


*Figure 1: Initial prototyped GUI for the application*

## 2.1 Minimum requirements

A minimum version of the application was defined by listing and prioritizing the wanted features. This was to help focus on the core features of the application first. Table 1 showcases the minimum features required from the logger.

*Table 1. Minimum version of the product*

| | Feature | Notes |
|---|---|---|
| GUI | Application has an initial view | Basic navigation within the application |
| | The user can navigate between the views | |
| | The user can close the application | |
| | The user can choose a location | Initially, the location should be within Finland. |
| Weather data | The user can visualize weather data | The data is visualized in e.g., charts. Visualization can be combined with air quality data. |
| | The user can choose a timespan for the data (hour, day, month) | The time span depends on the API. |
| | The user can save the timespan settings | The settings are saved for the next session |
| Air Quality data | The user can visualize air quality data | The data is visualized in e.g., charts. Visualization can be combined with weather data. |
| | The user can choose a timespan for the data (hour, day, month) | The time span depends on the API. |
| | The user can save the timespan settings | The settings are saved for the next session |

Additional ideas for features are shown in Table 2. These features would add value to the application but could be left out from the process without causing any harm to the main purpose of the application. Some of the features are more practical, such as adding GUI elements, but there are also more abstract features such as the user being able to create insights of the visualized data.

*Table 2. Ideas for additional features*

| Additional features | Notes |
|---|---|
| User can open the application settings | The settings could include elements e.g., a button to reset the application |
| User can open About -section | About section contains the application version and details of the used APIs |
| The user can reset the application to initial state | Any saved preferences can be removed |
| The user can see the location on a map | Additional map API |
| The user receives clear error messages, when needed | This includes, e.g., handling error when http requests fail |
| The user can visualize data parameters one by one | E.g., temperature, humidity. |
| The user can visualize multiple data parameters at a time | The parameters can be combined from both weather and air quality data |
| The user can create insights of weather's impact to air quality | Visualization helps making deductions |
| The user can choose between historical, current, and forecast data | |
| The user can read short descriptions of the data parameters | Explanations can be included to e.g., air quality parameters or the considered good limits for the parameter |
| The user can create insights of the data visualizations without any verbal explanations | The data visualization doesn't require excessive explaining of the topic |
| The user can save the wanted visualization | |
| The user can delete the wanted visualization | |
| The user can save observations into a log | E.g., text, the data chart created |
| The user can delete the wanted visualization | |
| The user can mark locations as favorites | |

Considering these features and the prototype, an architecture for the application was drafted.

# 3 Application architecture

The application follows the MVP (Model-View-Presenter) architecture, ensuring a clear separation between the business logic, user interface, and data presentation. By adhering to this design pattern, our solution offers better maintainability and scalability. The diagram below (Figure 2) demonstrates the structure of the MVP architecture in the designed application.

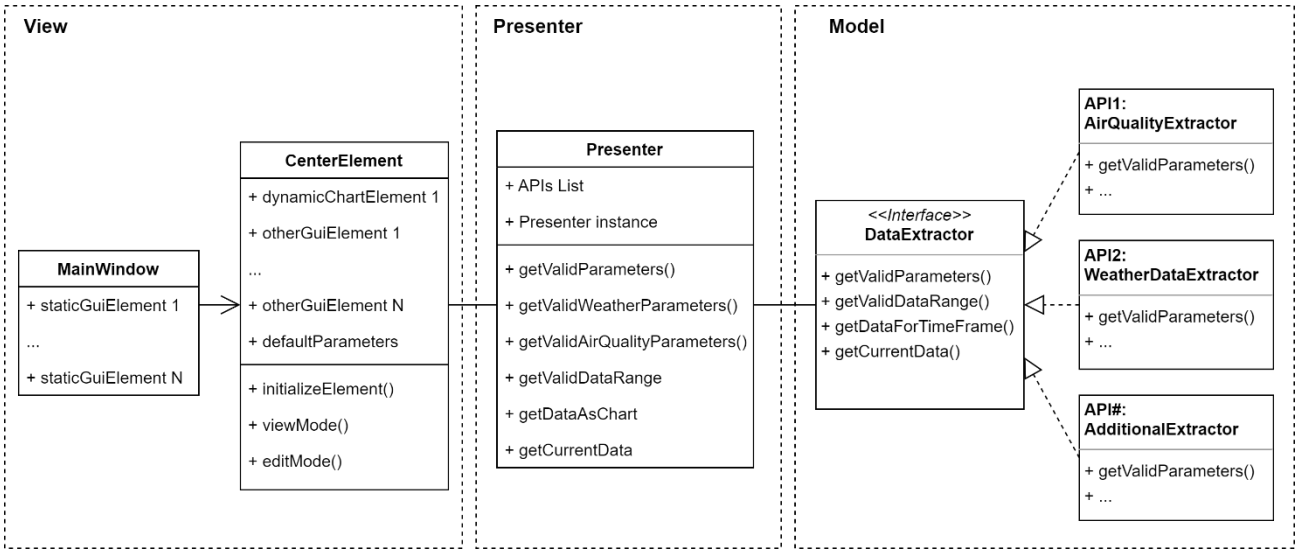


*Figure 2: UML diagram of the application core architecture*

Presenter is a class which acts as a link between the user interface and the used weather and air quality APIs. The presenter is not limited to these two APIs but can have additional APIs besides the mentioned ones. It provides necessary logic for selecting an API based on the requested data type. Presenter can forward the data to View component as a meaningfully presented chart, via public get method.

View component consists of two core elements: a main window and a center element. The main window works as a static application template, while the center element implements the dynamic part of the GUI. The center element can consist of multiple elements inherited from JavaFX class. The center element utilizes tab windows to show different representations of weather and air quality data, as shown in the initial prototype (Figure 1).

Model consists of DataExtractor interface and two specialized classes implementing the interface. DataExtractor interface standardizes the method of data extraction, ensuring uniformity in how data is sourced, regardless of the underlying data type or source. The implementation promotes code reusability and adaptability. Additionally, to ensure only a single instance of the extractor classes exists throughout the application's lifecycle, Singleton pattern was utilized. This design choice optimizes resource usage, avoiding redundant API calls and data processing.

# 4 Chosen APIs

The data for the Environmental data logger is gathered from respective APIs through synchronous HTTP requests. Once retrieved, the raw data is parsed and mapped to the application's internal data structures, ready for presentation. Following sections describe the APIs that were chosen for the application and general information of the data parameters fetched.

It is worth mentioning that by using interfaces and standardized methods of extraction, the application can seamlessly integrate additional data sources in the future without major codebase alterations. Similarly, currently used data sources can be switched to other resources, if needed.

## 4.1 AirQualityDataExtractor

Air quality is defined by measuring the different compounds of air. Typically, the measured values include concentrations of atmospheric gases such as sulfur dioxide (SO2), nitrogen dioxide (NO2), ozone (O3), carbon monoxide (CO). Besides this, it is common to measure particulate matter for both particles of aerodynamic diameter under 10µm (PM10) and fine particles under 2.5µm (PM2.5). The air quality index itself is based on measured air quality data and characterization of the air quality. [3]

The air data is collected worldwide to make useful forecasts and mindful decisions, prevent harmful effects, and create visualizations of the current air quality. For example, the European Union has defined air quality standards and objectives for the member countries [4]. Emission reduction in the EU is primarily based on EU directives implemented by national legislation [5]. Because the EU member states must report on air quality [6], the data is often openly distributed in APIs maintained by the member state governments.

Few different APIs were considered for the environmental data logger application; the data from Finnish Meteorological Institute (FMI) [7], Open-Meteo API on air quality [8], World Air Quality Index project [9] and Google's AIR quality API [10]. From these the Open-Meteo API was chosen, with access to data ranging from past year to plus four days forecast. The API offers data for a larger geographical area for free, limited to 10.000 daily API calls. The decision holds a trade-off: with FMI API we could get more data on a national scale but miss the international scale. Open-Meteo API ensures that we can visualize data outside the Finnish country borders as well.

The data extractor class for air quality is designed to follow the DataExtractor interface presented in section 3. Additionally, the application includes a util enumeration class for air quality parameters separately, to set API-specific query words for parameters more easily.

## 4.2 WeatherDataExtractor

Reliable weather data is crucial in various industries such as transportation, agriculture, and the energy sector. Weather affects us on an individual level as well, shaping our everyday lives. Typical weather data consists of information such as temperature, humidity, and wind conditions. Much like air quality data, weather data is collected by various entities worldwide.

There are plenty of different weather APIs offering data for the common weather data. The data for Environmental data logger is fetched from Visual Crossing API [11]. The API was chosen due to its usability and limited but free use. The API offers 1000 records per day and gives global coverage, including both historical data and 15-day forecast.

The weather data extractor follows the interface class DataExtractor, providing necessary methods to fetch data from the API. Our data logger focuses on visualizing two most common parameters: temperature and humidity.

## 4.3 OpenWeather geocoding API

The Location class makes use of OpenWeather's geocoding API to determine full location information based on either the coordinates of the location, or the name of the location. The API is free to use with a limit of 60 calls/minute and 1 000 000 calls/month. This API enables creating an additional feature for using map to showcase user's preferred location.

# 5 Self-evaluation

This section contains the self-evaluation over the project, from first prototype submission to midterm evaluation.

## 5.1 Thoughts over the design and its implementation

The original design for the application was rather minimal. Soon after defining the initial classes and interfaces we met up to discuss the architecture and interfaces of the application together. The tasks were divided roughly into four separate parts for each group member. As the meeting was concluded in an audio call, it was easy to change views over the structure and define initial class methods.

We were especially happy with the DataExtractor interface designed in the beginning of the process. After designing the interface together, it was easy to split up the workload and work on the API classes. The interface was defined well enough to support developing the code independently and expand the necessary API classes.

Because the initial design was not too strictly defined in the prototyping phase, we haven't made any drastic design-related changes in the application after it. The application has naturally expanded to include some additional classes for both new features and applying modularity to the code structure by separating e.g., GUI element handlers to their own classes.

One of the difficulties our group faced was related to the Weather API provided by Visual Crossing. The free subscription of Visual Crossing's service only allows quite a low number of daily calls. If one wants to receive all the possible weather data for a given period, they can only make 3-5 daily requests until the quota gets full.

The solution we found was to limit the size of the response object and only get the data we absolutely need. By that way it should now be possible to call the API couple of hundred times a day which should be enough for this project. However, the design of our application makes it rather smooth to switch the WeatherDataExtractor class to utilize another API providing weather data. API calls are the classes internal logic and switching the API does not affect functioning of the other classes.

Similarly, the original API plans for the air quality extractor changed on the way. Initially we were to use an API provided by Finnish Meteorological Institute but ended up using Open-Meteo API. This was due to trade-off mentioned previously; we didn't want to restrict the application usage in Finland only. As our architecture allows multiple APIs, we consider adding the FMI API to our application as well, to get more specific data within Finland.

## 5.2 Teamwork and practicalities

The teamwork between the group members has worked out well. One of the success factors in communication was deciding a static weekly meeting day. During the meetings it has been easy to discuss the application status and decide on the next steps. As the topics and individual goals for the next meeting were decided beforehand, it was easy to keep the workflow steady and organized. A proactive attitude has helped to keep stress levels low for the project.

Initially we used GitLab for creating the issues and listing each of the feature requirements there. However, over the course the tool hasn't been actively used. For us it seems that using GitLab added

unnecessary work and bureaucracy for the project. Similarly, as the responsibilities have been divided clearly, we didn’t see any need for separating changes to own git branches in our workflow.

For now, we expect the midterm evaluation to get possible tips and guidance in developing the application further. Should some conflicts with the design arise, we will plan changes accordingly during our next weekly meeting. Apart from that, as the defined minimum requirements are nearly satisfied in the application, we will probably shift on fine-tuning the existing features, organizing the code better on the workspace, and developing the additional features.

# 6 References


[1] OpenAI. (2023). ChatGPT-3.5 (Sept 25 version) [Large language model]. https://chat.openai.com

[2] JavaFX, application platform. Accessed 27 October 2023, https://openjfx.io/

[3] Air quality index. Finnish Meteorological Institute. Accessed 19 October 2023. https://en.ilmatieteenlaitos.fi/air-quality-index

[4] EU air quality standards. European Commission accessed 19 October 2023. https://environment.ec.europa.eu/topics/air/air-quality/eu-air-quality-standards_en

[5] Air pollution control. Finnish Ministry of the Environment. Accessed 19 October 2023. https://ym.fi/en/air-pollution-control

[6] Directive 2008/50/EC of the European Parliament and of the Council of 21 May 2008 on ambient air quality and cleaner air for Europe. OJ L 152, 11.6.2008, p. 1–44. Accessed 19 October 2023. https://eur-lex.europa.eu/eli/dir/2008/50/oj

[7] Open data, Finnish Meteorological Institute. Accessed 19 October 2023. https://en.ilmatieteenlaitos.fi/open-data

[8] Air quality API, Open-Meteo.com. Accessed 19 October 2023. https://open-meteo.com/en/docs/air-quality-api

[9] Air Quality Programmatic APIs, World Air Quality Index project. Accessed 19 October 2023. https://aqicn.org/api/

[10] Air Quality API, Google Maps Platform. Accessed 19 October 2023. https://developers.google.com/maps/documentation/air-quality

[11] Easy Global Weather API, Visual Crossing. Accessed 19 October 2023 https://www.visualcrossing.com/weather-api