Software Design Plan Document

Project Title: Finnish Weather and Accident Correlation Analyzer

Group Members:

Vili Huotari

Kalle Rantalainen

Eetu Virtanen

Tariq Hussain

Table of Contents

[1. Introduction 2](#_Toc180873324)

[2. High-Level Overview 2](#_Toc180873325)

[3. Prototype Overview 2](#_Toc180873326)

[4. Design Approaches 2](#_Toc180873327)

[5. Software Components 3](#_Toc180873328)

[5.1. Model 3](#_Toc180873329)

[5.2. Controller 3](#_Toc180873330)

[5.3. View 3](#_Toc180873331)

[6. Interfaces 5](#_Toc180873332)

[7. Data Sources 5](#_Toc180873333)

[8. GUI Framework 5](#_Toc180873334)

[9. Future Considerations 6](#_Toc180873335)

[10. Use of AI 6](#_Toc180873336)

# 

# 1. Introduction

This design document outlines the technical specifications and architecture of the "Finnish Weather and Accident Correlation Analyzer," an application that combines weather data from the Finnish Meteorological Institute's Open Data API with traffic accident data from Stat.fi API. The goal of this application is to analyze and visualize the correlation between weather conditions and traffic accidents in a selected city.

# 2. High-Level Overview

The application retrieves weather data and traffic accident data from the respective APIs and correlates them to provide meaningful insights. Users can select a specific city, adjust the timeframe, and view the correlation between weather conditions (e.g., temperature, precipitation) and accident rates. The user interface will display the data through interactive visualizations, including graphs that illustrate trends over time. Additionally, the application allows users to save and reload preferences, making it easy to view updated data based on previously defined settings.

# 3. Prototype Overview

The initial prototype demonstrates a Weather and Accident Correlation Dashboard, which includes:

- Weather Data: Maximum and minimum temperatures displayed across various days.

- Traffic Accident Data: Accident statistics by vehicle type (e.g., cars, bikes), correlated with the weather conditions.

- Filters: Users can filter the data by date range, vehicle type, and location to focus on specific timeframes and cities.

# 4. Design Approaches

The application is designed using the Model-View-Controller (MVC) architecture. This design pattern is ideal for separating concerns, ensuring each component (Model, View, Controller) performs its distinct responsibilities without overlap. The application follows the Separation of Concerns principle, which allows for better maintainability and potential future expansion.

# 5. Software Components

## 5.1. Model

## The Model will be responsible for:

## - Fetching weather data from the Finnish Meteorological Institute API and accident data from the Stat.fi API.

## - Processing and formatting the data to establish correlations between weather conditions and accident rates. - Storing user preferences for future use, including preferred city, date range, and filter settings.

## 5.2. Controller

The Controller will:

-act an intermediary between the Model and the View, managing data flow and coordinating business logic.

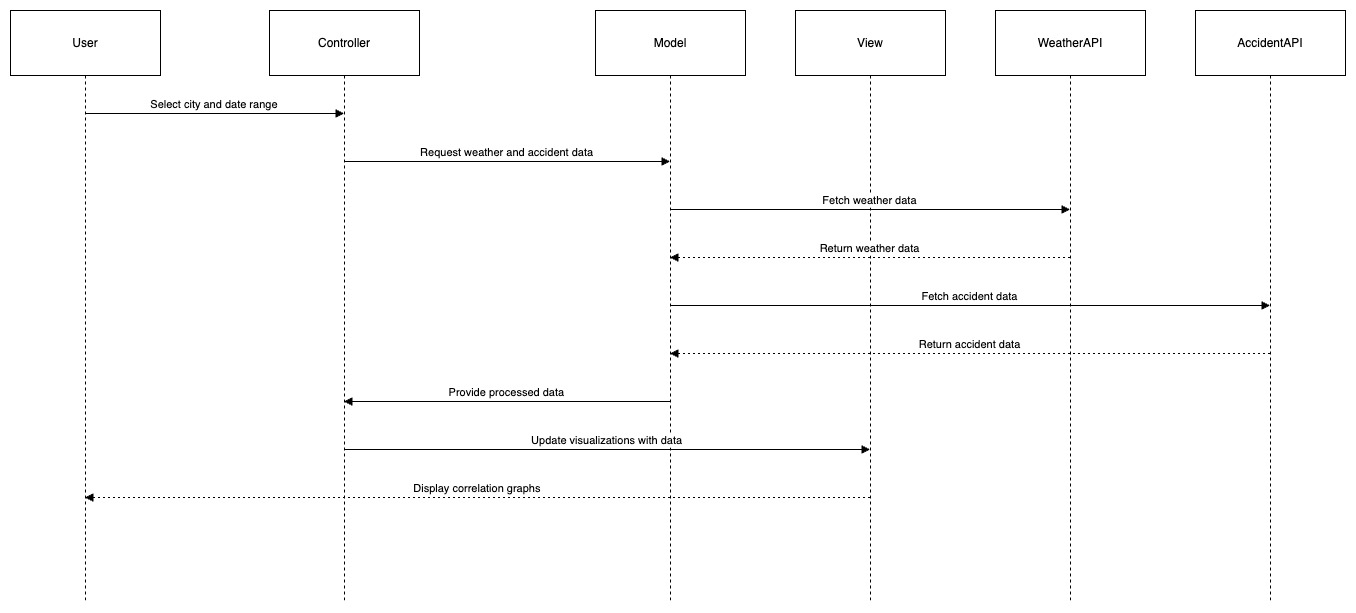
- Handle user interactions, such as selecting the city, adjusting filters, and saving preferences.

- Request data from the Model based on user inputs and pass the processed data to the View for visualization.

- Trigger updates in the View when data is fetched or filters are applied.

## 5.3. View

The View will:  
- Render visualizations such as line charts and bar graphs to display the correlation between weather and accident rates.  
- Provide interactive controls that allow users to filter data by city, date, and vehicle type.



**Participants**

* **User**: The person interacting with the application.
* **Controller**: Manages the flow of data between the Model and the View.
* **Model**: Handles data fetching and processing.
* **View**: Responsible for displaying the user interface and visualizations.
* **WeatherAPI**: The external API providing weather data.
* **AccidentAPI**: The external API providing traffic accident data.

**Sequence of Interactions**

1. **User Selects City and Date Range**:
   * The user chooses a city and a specific date range for which they want to analyze weather and accident data.
2. **Controller Requests Data**:
   * The Controller receives the user input and sends a request to the Model for the relevant weather and accident data.
3. **Model Fetches Weather Data**:
   * The Model makes a call to the WeatherAPI to retrieve weather data based on the user's selected criteria.
4. **WeatherAPI Returns Data**:
   * The WeatherAPI sends back the requested weather data to the Model.
5. **Model Fetches Accident Data**:
   * The Model then calls the AccidentAPI to get the corresponding traffic accident data.
6. **AccidentAPI Returns Data**:
   * The AccidentAPI sends back the traffic accident data to the Model.
7. **Model Provides Processed Data**:
   * After gathering both datasets, the Model processes the data to establish correlations and sends it back to the Controller.
8. **Controller Updates View**:
   * The Controller instructs the View to update the visualizations using the processed data.
9. **View Displays Correlation Graphs**:
   * Finally, the View presents the correlation graphs to the User, illustrating the relationship between weather conditions and traffic accidents.

# 6. Interfaces

The interfaces between the Model, Controller, and View will be well-defined to ensure smooth data flow:

- Model-Controller Interface: The Model provides methods like getWeatherData() and getAccidentData() to retrieve and process API data. The Controller interacts with these methods to gather relevant data and correlate weather conditions with traffic accidents.

- Controller-View Interface: The Controller sends the processed data to the View, which renders the correlation visualizations. User input from the View, such as changing the city or applying filters, is handled by the Controller to fetch new data and update the display.

# 7. Data Sources

- Finnish Meteorological Institute's Open Data API: Provides real-time weather data, including temperature, precipitation, and wind conditions.

- Data format: XML

- Key methods: Fetch weather data by location, date, and time.

- Stat.fi API: Provides traffic accident data, including accidents by vehicle type, accident severity, and location.

- Data format: JSON

- Key methods: Fetch traffic accident data filtered by location and time range.

# 8. GUI Framework

The graphical user interface (GUI) will be implemented using JavaFX. JavaFX provides an intuitive and responsive interface for creating complex visualizations such as interactive charts and graphs. Its flexibility and integration with Java backend code make it the ideal choice for this project. The View will display graphs that show the correlation between weather conditions (temperature, precipitation) and accident data (accident rates by vehicle type and severity).

# 9. Future Considerations

Potential future developments include:

- Deeper analysis: Implementing machine learning models to predict accident risk based on weather conditions.

- Additional data sources: Adding other relevant APIs, such as road condition data or live traffic feeds, to improve the accuracy of the correlation analysis.

- User Alerts: Implementing a feature to alert users when weather conditions that correlate with high accident rates are forecasted for a specific area.

- Mobile Support: Developing a mobile version of the application to provide users with access to the data while on the go.

# 10. Use of AI

We focused on using AI to help us get ideas and guidance for our project, especially in choosing design patterns. We have used ChatGPT to give us ideas and to suggest the best design pattern for our project. ChatGPT recommended the MVC (Model-View-Controller) pattern, and after reviewing it ourselves, we decided that it’s the best choice for our project.