Group 1

Aaron Hirvi

Miikka Venäläinen

Onni Merilä

Samundra Dhakal

Design document

2.12.2022

Application design

The application provides the user with the ability to examine road or weather data from selected road or point. The UI consists of five views.

1. Menu
    a. After starting the application, a menu is opened where the user can navigate in the program. From there the user can navigate to selected UI view
2. Traffic data view with coordinates
    a. Here the user can observe road condition and weather data from selected coordinates.
3. Traffic data view with road number
    a. Here the user can observe road condition and weather data by selecting a road number
4. Weather data view
    a. Here the user can observe weather data from selected coordinates.
5. Combined view
    a. Here the user can observe weather, road condition and road maintenance data from selected coordinates.

Components and responsibilities:

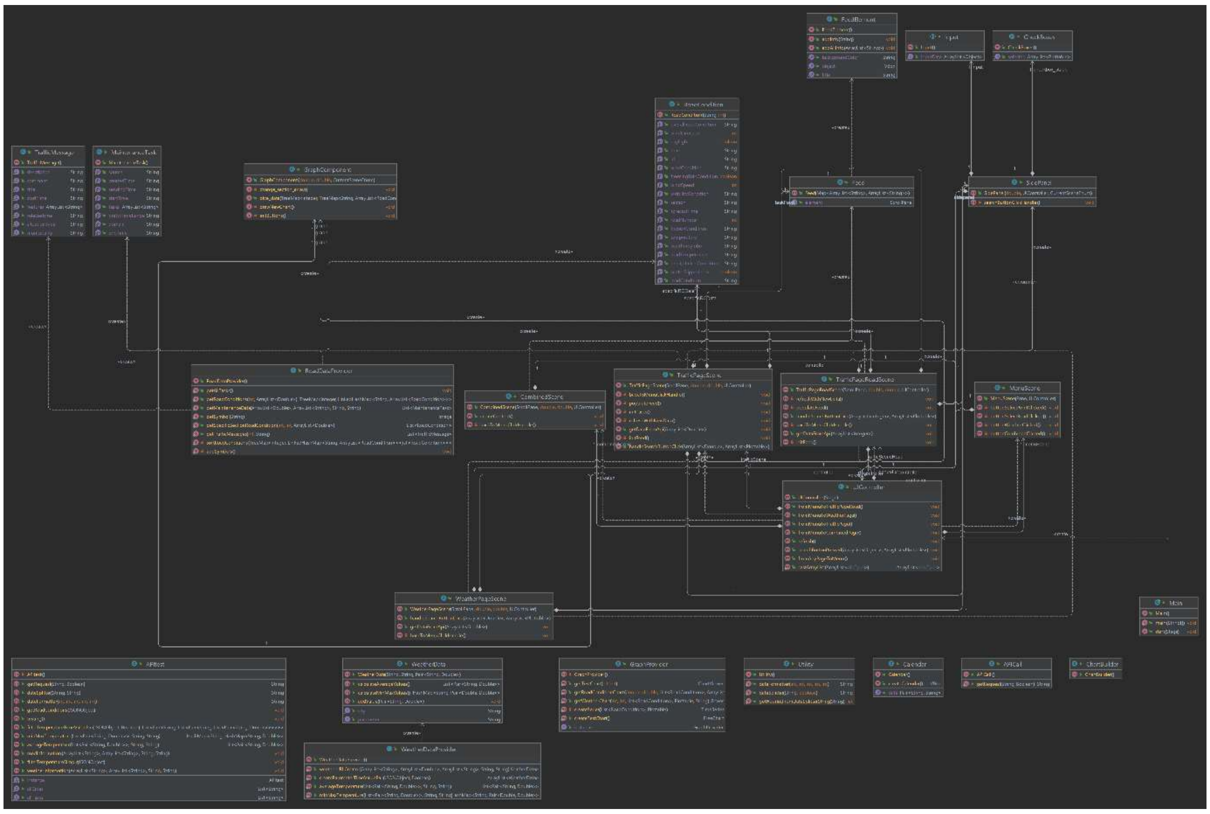
- API
    - APICall
        - Responsible for all Digitraffic and FMI API calls. Called every time new information is fetched from Api.
    - MaintenanceTask
        - Represents a single maintenance task. Saves data to private e.g. task names, start and ending times and domain of task. Created inside RoadDataProvider. Used by the UI class when processing data to visual elements.
    - RoadCondition
        - Represents a single road condition datapoint. Saves data to private e.g. road number and section, temperature and conditions. Created inside

        - RoadDataProvider. Used by UI class when processing data to visual elements.
    - WeatherData
        - Represents a single weather data datapoint. Saves weather related data to object. Used by the UI.
    - TrafficMessage
        - Represents a single traffic message
    - RoadDataProvider
        - Responsible for providing and parsing of Digitraffic road data. Fetches data from APICall class and creates objects based on received data.
    - WeatherDataProvider
        - Responsible for providing and parsing of FMI weather data. Fetched data from APICall class and creates objects based on received data.
    - Utility
        - Contains utility methods used in the Api class
    -
- Graph
    - GraphProvider
        - Provides Graph element fitted with wanted data. Inputs raw data and output ready ChartViewer element.
- UIView
    - Scenes
        - MenuScene
            - Represents an UI view where the user can select which scene they want to view. All scenes have a back button where they can return to the menu.
        - TrafficPageScene
            - Represents an UI view where the user can inspect road condition and weather data from selected coordinate area. The user can specify which parameters they want to plot into a graph. A feed element displays relevant road condition data. Coordinated can be edited from the side panel.
        - TrafficPageRoadScene
            - Represents an UI view where the user can inspect road condition and weather data from selected road. The user can specify which parameters they want to plot into a graph. A feed element displays relevant road condition data. The road can be switched with arrow buttons or by doing a new search from the side panel.
        - WeatherPageScene
            - Represents an UI view where the user can inspect weather data from selected coordinate area. The user can specify which parameters they want to plot into a graph. A feed element displays relevant road condition data. Coordinated can be edited from the side panel.
        - CombinedScene
            - Represents an UI view where the user can inspect weather, road condition and road maintenance data from selected coordinate

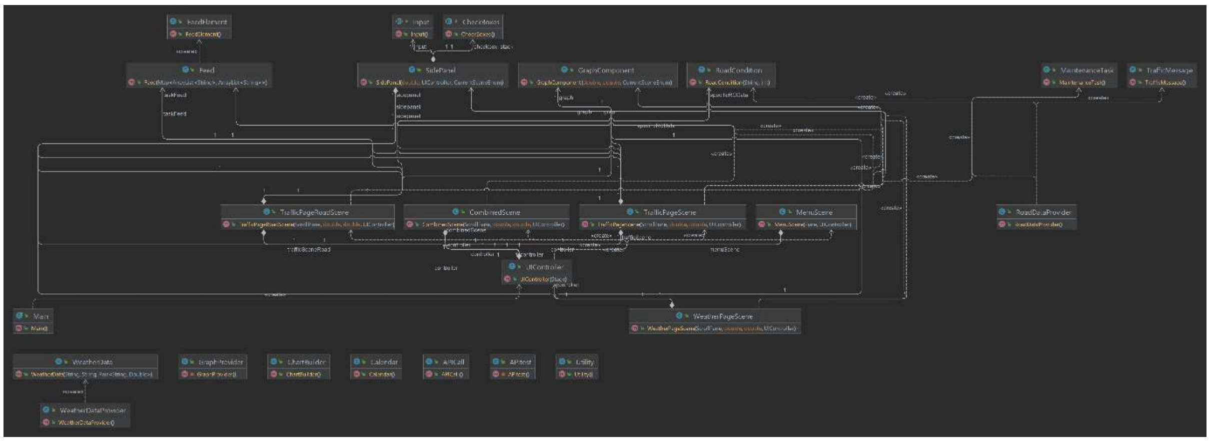
        area. The user can specify which parameters they want to plot into a graph. A feed element displays relevant road condition data. Coordinated can be edited from the side panel.

- UIController
    - Controls the flow of the UI. Handles all scenes and transferring between them. This class is responsible for showing the current scene.
- Components
    - Calendar
        - Represents UI element of a calendar widget.
    - Feed
        - Represents UI element of an information feed. Can be fitted with any kind of string-based title + text data
    - FeedElement
        - Represents Feed elements single row. Displays information.
    - GraphComponent
        - Represents a Graph which plots wanted data.
    - SidePanel
        - Represents UI element of a side panel which the user can use to give parameters to the program.

Class diagram



Interface diagram



Design decisions

We chose to just use JavaFX since that is the framework our team has the most experience on. JavaFX is very simple and easy to use and has plenty of support online. We choose org.json as our JSON parser since it also supports XML formats which are required for the fetching weather data. We chose not to use JavaFXML since none of us had any experience on it.

We choose JFreeChart as an external library to help us visualize data. JFreeChart offers plenty of easy-to-use charts and ways of plotting any sort of data.

We decided to use Java's object-oriented manner to help us process data by creating objects. This decision simplified the data flow of the application drastically. By creating objects from road and weather data, the code readability and ease of updating also increased.

Dividing responsibilities

Our team decided to split responsibilities, so each member didn't have to know every aspect of the program.

Rough estimate on our division of responsibilities:

- Miikka
  - API class calls
  - API data parsing
  - Creation of RoadData and WeatherData classes
  - Calendar component
- Onni
  - API data to UI elements
  - Graph elements
  - Refactoring
  - Modularity
- Aaron
  - UI Elements and functionality
  - CSS integration to JavaFX elements
  - Final documentation

This division worked great for us.

The team organised meetings to keep track of progress and to discuss new features and implementations. Outside of meetings, Discord and Telegram were used to discuss progress.

Dependencies

- JavaFX
  - Used for all UI elements
- org.json
  - Used for parsing Api response JSON and XML data
- JFreeChart
  - Used for plotting data to UI
- APIs
  - Fetch needed data
  - https://www.digitraffic.fi/en/road-traffic/
  - https://tie.digitraffic.fi/swagger/
  - https://en.ilmatieteenlaitos.fi/open-data-manual

Self-evaluation

Sticking to original plan

Our original plan was stuck to for the most part. The design regarding the windows and navigation between them was kept as planned. Biggest changes involved the flow of data and overall hierarchy. UIController class was added after the mid-term submission which turned out to be a successful change regarding simplicity and ease of use.

Best part of the design was the use of Figma to sketch out wireframe designs of the UI. The UI designs were also modified after careful inspection of the assignment requirements.

Overall success

The project lacked proper design. Before starting any coding of the application, the requirements should have been inspected more carefully. After thoroughly understanding the requirements, we should have made a clear and complete design of the whole software. After starting to build the application, we quickly realised some parts of the design were flawed and insufficient in terms of requirements. The outcome of the project would have certainly been better if we had proper design.

One factor, which made the project a lot harder, was the requirements which were hard to understand. Were the requirements a bit less complex and trivial, the app would have been more enjoyable to design.

Other factor which made our workload a lot bigger was the absence of one team member. This project was mostly made by three members which increased the work needed to fulfil all requirements. After realising that one member would not be contributing to this project, it was too late to increase other members workload efficiently without causing a rush before the deadline. Some requirements are left incomplete due to this.

On a positive note – this project taught us what to be cautious of when designing larger software. Next time we will all consider the individual responsibilities more carefully and emphasize the importance of proper design.

What was done well?

1. Use of Figma for designing the UI wireframe with the team
2. Decision making as a team
3. Organization of meetings despite of busy schedules

What could have been done differently?

1. More thorough inspection of the requirements
2. More time spent on software architecture design
3. Better use of inheritance and object-orientation
4. Use of branching and better use of version control