Software design document

COMP.SE.110 Software Design

Weather and traffic data manipulation

Written by: Veikka Peltomäki, Niklas Kanetele, Eemil Lehto, (Samuel Ahopelto)

Introduction

This document works as a design document on our groups project in the course COMP.SE.110 Software Design. The purpose of this document is to provide a description of our system which is comprehensive enough to work as a source of details for how the software is to be built.

The software project will be implemented using programming language C++20 and the Qt framework. The project is to be implemented using agile methods where the group will continuously evaluate the gathered requirements and iterate on those.

System Overview

The goal of this software project is to create a software which allows its user to study weather conditions and its history as well as road conditions and its history and their correlation by studying graphs, tables, and other visualizations in the software UI. These visualizations will be based on data gathered from Digitraffic (road condition data) and FMI (Finnish Meteorological Institute, weather data). The data is gathered in real time using public APIs provided by Digitraffic and FMI.

The software will be based on a UI where the user is able to select if they wish to view weather data, road conditions or the correlation between these (weathers effect on road conditions). The user will be able to input the coordinates of the area which he wishes to study as well as select a timeline of the data involved in the visualizations. The user should also be able to compare the data between different timelines and coordinate areas.

Requirements

These requirements will be iterated on continuously during the development process as the software development group identifies more of them.

The software system shall be able to gather real-time as well as historical data on weather and road conditions using APIs provided by Digitraffic and FMI

The software system shall have a GUI which allows the user to select which of the provided data to study

The data shall be visualized in various forms such as tables, graphs, and plots.

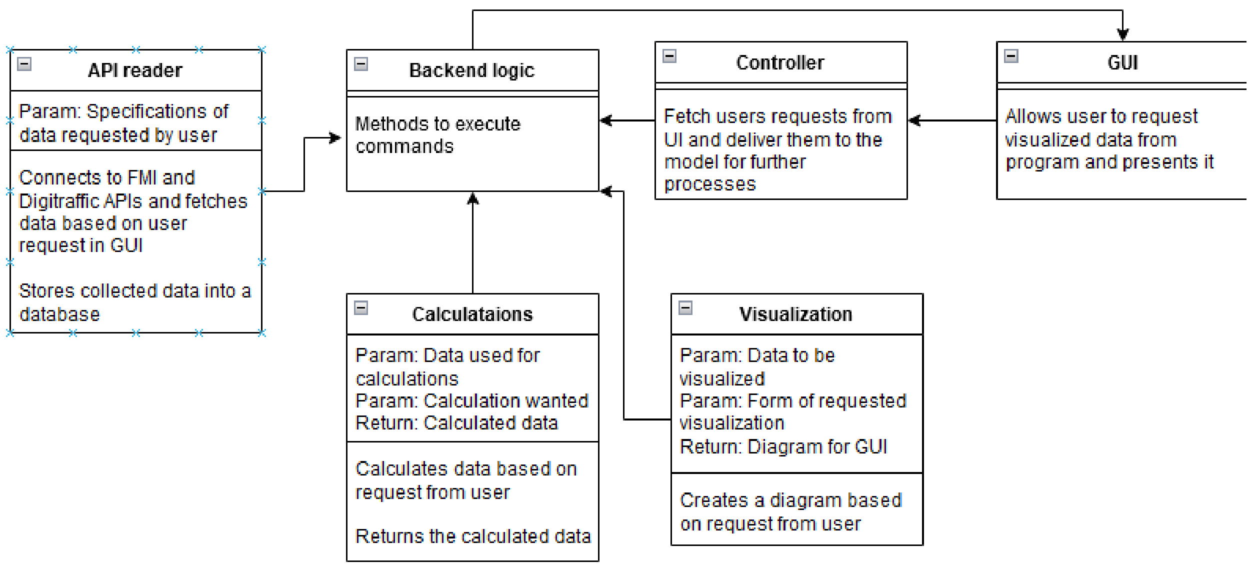
The user should be able to select the timeline of the data they wish to see in these visualizations

The user should be able to specify coordinates of the area of where the visualized data will be based on

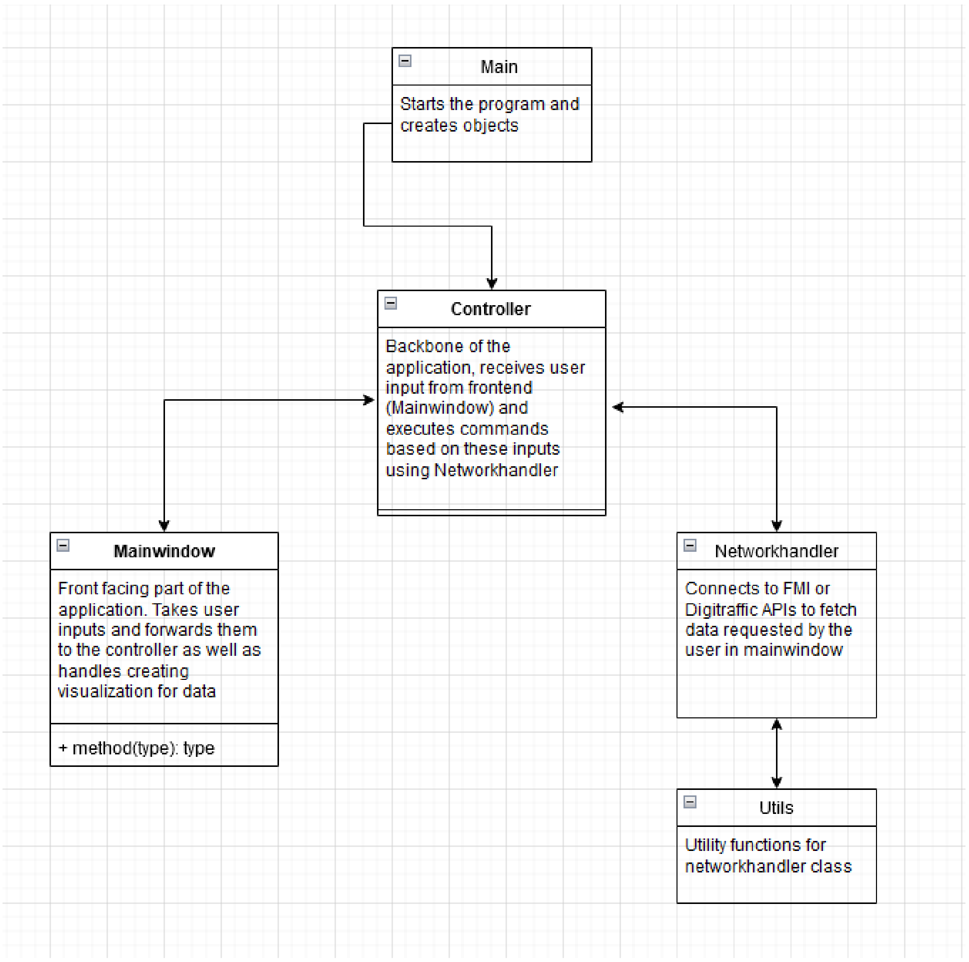
The system shall be able to process the data gathered from the API and do calculations based on it to provide the user with relevant information based on the calculations

Architecture

The software system is implemented using C++20 with a MVC architecture.


API reader
Param: Specifications of data requested by user
Connects to FMI and Digitraffic APIs and fetches data based on user request in GUI
Stores collected data into a database
Backend logic
Methods to execute commands
Controller
Fetch users requests from UI and deliver them to the model for further processes
GUI
Allows user to request visualized data from program and presents it
Calculataions
Param: Data used for calculations
Param: Calculation wanted
Return: Calculated data
Calculates data based on request from user
Returns the calculated data
Visualization
Param: Data to be visualized
Param: Form of requested visualization
Return: Diagram for GUI
Creates a diagram based on request from user

1 Planned diagram of software classes


Main
Starts the program and creates objects
Controller
Backbone of the application, receives user input from frontend (Mainwindow) and executes commands based on these inputs using Networkhandler
Mainwindow
Front facing part of the application. Takes user inputs and forwards them to the controller as well as handles creating visualization for data
+ method(type): type
Networkhandler
Connects to FMI or Digitraffic APIs to fetch data requested by the user in mainwindow
Utils
Utility functions for networkhandler class

I Actual diagram of application classes

- Controller
    - Works as a backbone runner for the application
    - Gets signals from frontend (Mainwindow / GUI) and backend (networkhandler) and executes functions in both of them according to the user’s request
- Mainwindow
    - Frontend of software
    - Based on prototype created in Figma with final implementation being iterated on that
    - Based on Qt GUI
    - Creates visualization based on the data collected from API
    - Takes user inputs and sends them in signal to Controller for further execution
- Networkhandler
    - Fetches data from APIs based on user inputs in Mainwindow
    - Receives data in JSON or XML format and utilizes utils class to parse data into presentable format
        - Sends data to controller for further commands using signal
- Utils
    - Includes utility tools for Networkhandler to keep the class size down
    - Most importantly parsing data from original file format to a form which is easier for visualization

The finished application utilizes multiple classes provided by using Qt as the environment. Largest ones are naturally Qt GUI combined with QChart for the whole frontend of the application and QNetwork class for API related functions. Also ctime is used for getting the current time to use in functions. QXmlStreamReader, QJsonObject and QJsonArray are used for parsing the XML and JSON documents into a readable form. For functionality C++ std is used in multiple places. Signals and slots are heavily utilized for communication between objects.

Example run case

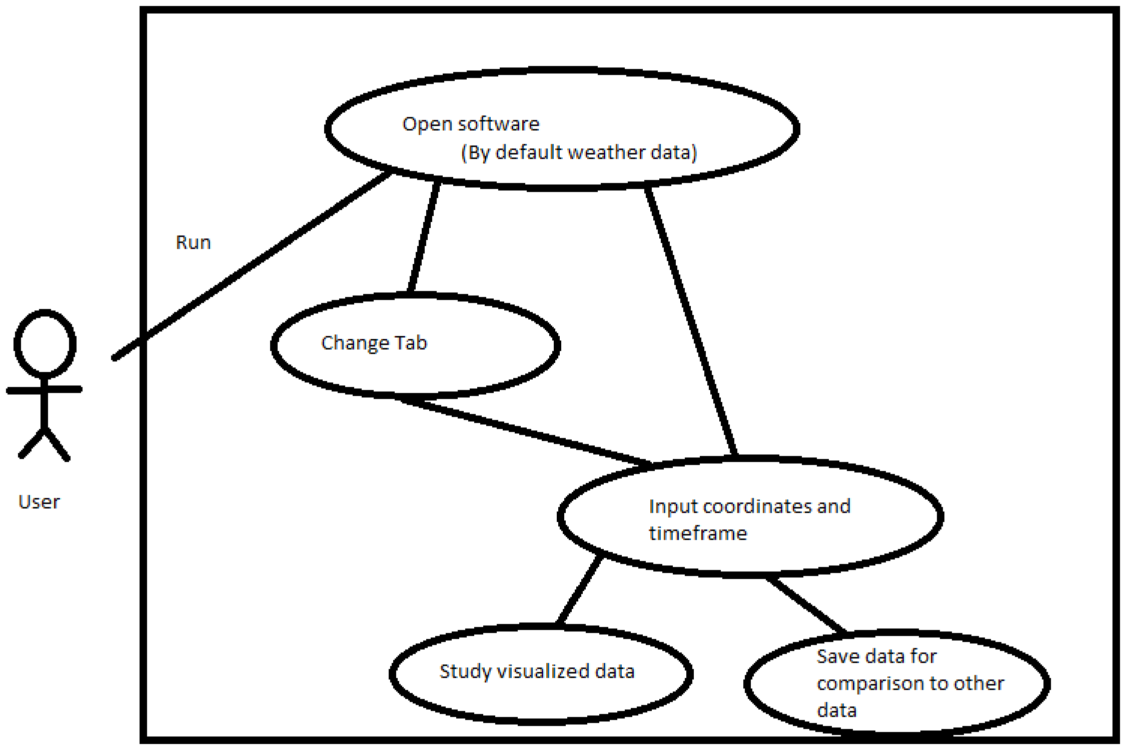

Open software
(By default weather data)
Run
Change Tab
User
Input coordinates and timeframe
Study visualized data
Save data for comparison to other data

Figure 2 Very simple use-case diagram

- Software is opened
- User selects the data source from the tabs
- User selects parameters for data
  - Location
    - Selection if most populated cities in Finland
  - Timeline
    - Timeline of data wanted
  - Data type
    - What kind of data user wants to be visualized
- After selecting inputs user clicks fetch data and the program opens the visualization in a new dialog window for user to view

Software project timeline

By 2.10.2022: Prototype of system UI and base design document done

By 30.10.2022: Somewhat working implementation of UI to be deployed with some functionalities. Working API calls to gather data from sources. Updated design document with self-evaluation based on the created prototype.

By 2.12.2022: Working final product

Self-evaluation as of 30.10.2022

As of the mid-term point of the project it has had a lot of ups and downs. Even though everyone in the group is mainly using the same IDE (QT Creator) with the same operating system QT Creator has proven a little problematic at times especially when trying to fetch data from the API due to OpenSSL having some issues requiring specific .dll files to be copied into the build folder for it to work. Due to figuring this problem out as well as the project group having pretty busy ends of 1st period, we haven't had the best of progresses with the backend side of the project.

As far as the design goes, we haven't had too many changes from our prototype. Our front end is very similar to what we envisioned in our Figma, and the biggest backend changes have been to our classes. Instead of combining visualization and calculations we concluded that these should be divided to their own classes. I believe our current design will work well in this project. It's simple in a way that we shouldn't end up with bloated classes with too much functionality or with too many classes each doing a small thing which a function in another class could do. I anticipate the larger changes in the future to be with our frontend as we dive deeper into the data the APIs are able to provide (datatypes, forms of diagrams, timelines and locations).

Self-evaluation for final submission

All in all the application project does not differ too largely from the original design. This is in part due to lack of time for major changes after starting the project and due to learning better design patterns from the course later on when there was no more chance for a larger overhaul. Left in this document we have the planned class diagram from around prototyping as well as a class diagram showing the finished product. As we can see the end product is a little more simple with most of the functionality combined to the main classes but overall the design is very similar. For certain reasons some of the planned features didn't end up making the final product due to busy schedules of everyone involved with the project and originally poor division of work where large parts of the development side were waiting for the more complicated parts, which naturally took longer to make. For example shifting to hardcoded "dummy data" to create the visualization instead of waiting for the backend to be completely done before working on the visualization part was important for the group to finish the project.

From prototype to first version to end product the frontend side kept very consistent with the group liking the original design and deciding on just iterating on in until the finished product.

All in all taking into account everything relating to the project the end product is passable. Design wise there are some major changes we would have made if we had the time for it but due to the nature of the project (starting early on the course and continuously making it while learning new things from the course) a lot of these things weren't able to be implemented.