COMP.SE.110 Group Project

Ville Venetjoki, Ilari Miettunen, Viivi Mustonen ja Kasperi Kouri

Table of contents

Introduction

The purpose of this program is to present its user with data from the Finnish meteorological institute as well as DigiTraffic in an intuitive and accessible manner. The user can customize what data and how it is shown. The user can also save some preferred views to be accessed later.

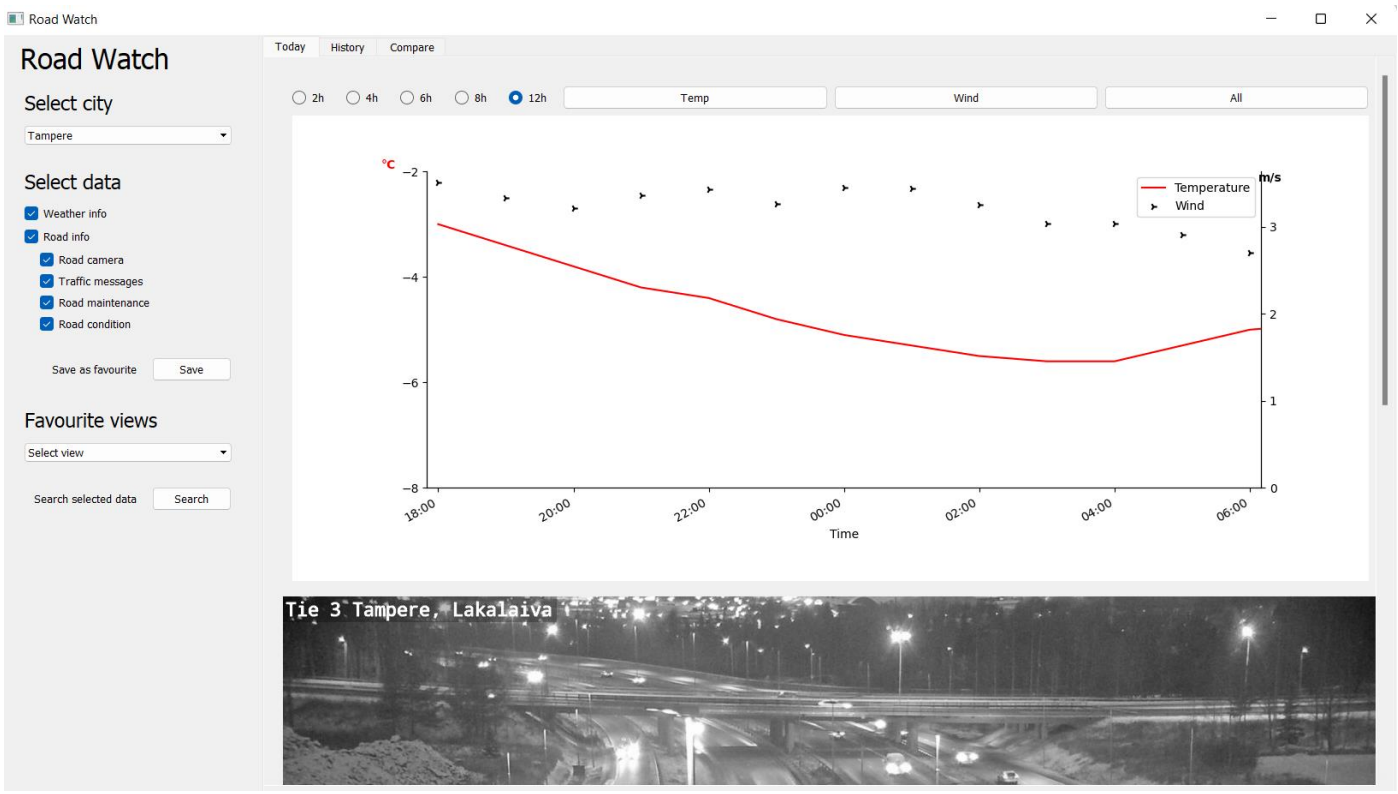
We chose to implement the program using the Python programming language as all the group members had previously used it and because it felt best among the accepted languages. Python has many useful built-in libraries as well as an easy syntax which allows us to spend more time planning than struggling with coding. The drawback with Python for this course is that it doesn't have a direct way to implement interfaces.

The user interface is implemented using PYQT5 / QML. QML follows the Qt approach on event-based programming, which means that control is loosely integrated into the view. This means that as the view gets inputs from the user, it calls the model for required data which gets updated when it is received.

Program operation and additional features

When the program is started, the user can choose between Tampere, Helsinki, Turku, Oulu and Lappeenranta for information about the city's weather and road conditions. Weather data consists of temperature and wind speed. The user can choose whether he/she wants to see both or just one part of the data. The viewing interval can be selected from above the graph.

Road data consists of road cameras, traffic messages, road maintenance and road maintenance. The information selected from the menu is presented in the tab below. It is possible to save the city and the settings selected for it as favorite. Favorite views can be opened from the sidebar.



On the History tab, the user can select one day and retrieve weather and road data from this past day. The dataset of the timeline could then be saved in JSON format for later use. It was intended to be adjustable but unfortunately, we didn't have time to implement that feature.

On the Compare tab, the user should be able to load two saved files side by side in the same view and compare the data in them. Files can be changed by clicking the load buttons in the menu. Unfortunately, the implementation of this tab was not fully operational or implemented due to scheduling challenges and bugs in integrating many libraries and pieces of code together.

High level description

We decided to use the MVC model in our project. We chose the Model-View-Controller model because it is the most well-known design pattern / architecture style to separate the UI and other parts of the software. A software (or component) is made of three parts, each having their own role. The implementation is somewhat loose since we're not strictly following all parts of the model.

• View (data_visualization): Presents all the data to the user within the application using resources given to it by the controller.

• Controller(main_window.py): Main logical application of the software. It calls other components and creates the application window using the side_panel.py and view_panel.py components. It is also used to save sets of data and settings into JSON-files for later use.

• Model (apirequests.py): Contains functions which are used to fetch and parse data from the Finnish meteorological institute as well as DigiTraffic according to given parameters. The fetched data is parsed from JSON or xml into Python dictionaries and passed back to the requesting application.

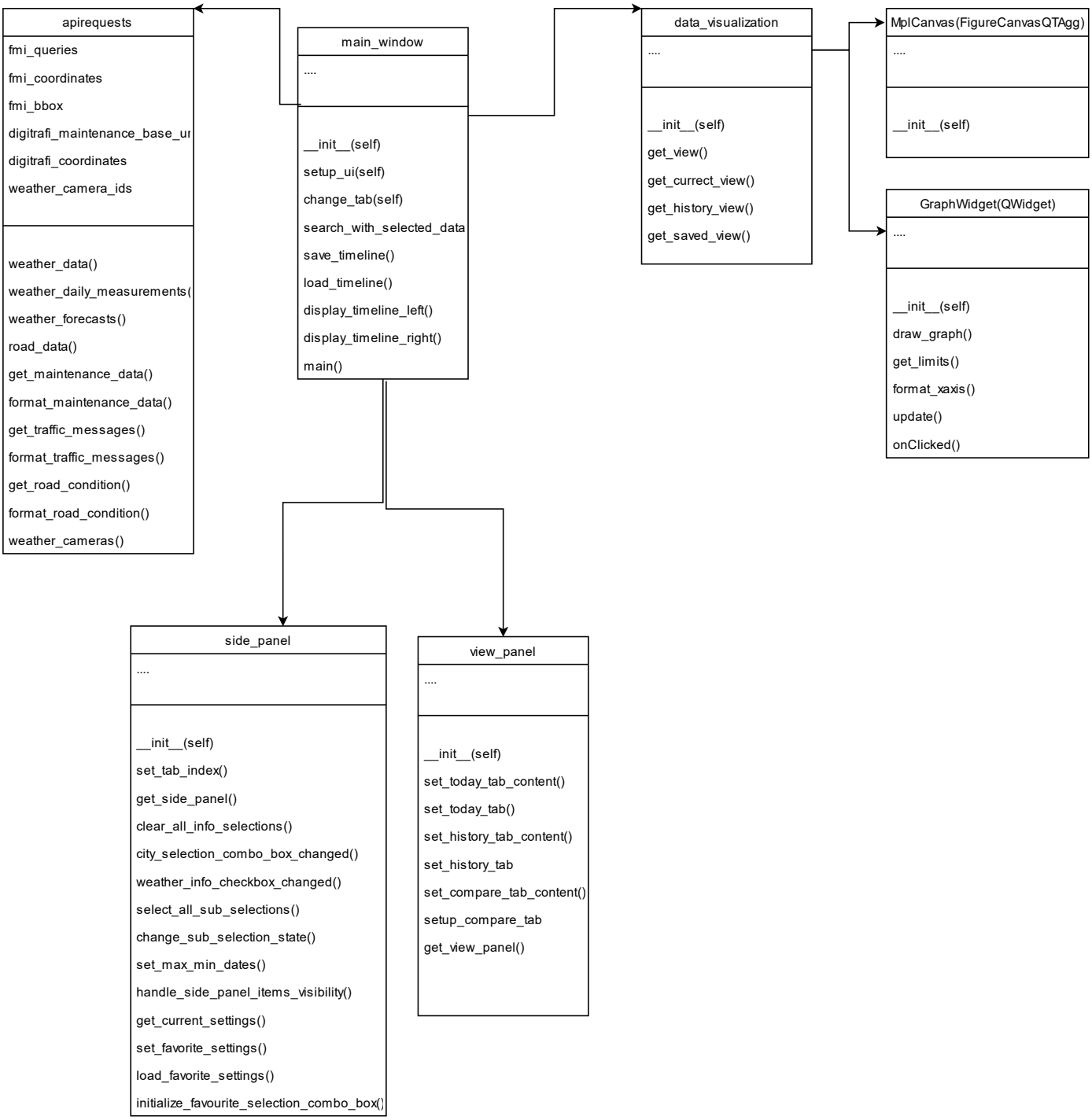
Due to Python not having an easy way of creating interfaces, our program is going have classes and files directly interacting between each other. The three main components are going to be the view, the model, and the API. During development some components may be split into smaller ones to avoid having "god classes". They will still serve the purpose of the same component, but just be split up into smaller files.

The main idea is that the controller will contain the actual logic and saved information. When the model needs some data, it doesn't already have, it's going to call the API program, which will make a request to either FMI or DigiTraffic according to given parameters. The API will store retrieved information into Python dictionaries and pass them onto the model.

Third-party libraries used

- Python "requests"-library
  - Used because it is much more intuitive and easier to use compared to pythons built in urllib-library
  - "pip install requests" in the python terminal to install
- fmiopendata
  - Used to parse weather data xml files into python dictionaries
- PyQt5/QML
  - Used to build the user interface
- Matplotlib
  - Used in Graph-class to create visualizations of data
- Numpy
  - Used for mathematical functionality

Class diagram:



Describing components

This includes more detailed description of controller, model and view.

Controller

The controller consists of main_window.py and folders components and saves. Components consists of side_panel.py and view_panel.py. As earlier said the controller acts as the main logic and storage component in the project. Main_window.py has access to view and model. It provides used with side menu for selecting wanted settings and tabview for navigating between different views. When user has provided settings, the main_window.py will call the model to fetch data from the internet. It will then call the view with the fetched data and in return it will get desired plots to display in open tab.

Components side_panel.py and view_panel.py implement the side and view panel components to the main_window.

View_panel is responsible for the display of the tabs in the main window. The view panel displays graphs and plots of the data, messages and images. There are three tabs: Today, History and Compare. Tab today displays the timeline of the current day and the display consist of the weather forecast and the weather and road data of the current day. History tab displays the timeline between selected days. Display consists of the weather and road data between selected days. The compare tab displays two different saved timelines between selected days that have been loaded to the application. This display consists of the weather and road data between selected days.

Side_panel class implements the side panel component of the main_window. Side panel has multiple selections that are only displayed at the corresponding tab. User can select location, data types to show and timeline. Settings can be saved as favorite, and those favorites can be easily selected from the favorite views combo box. Timelines can be saved, and the saved files loaded back to the program in order to compare different timelines side by side.

Model

The model consists entirely of apirequests.py. This file fetches data from digitraffic and the fmi APIs. Like earlier said it works as a model for the application. For data, a start and end time are required. For a forecast, the end time of the forecast is required. Requests are parsed into python dict containers and returned.

Data retrieved from FMI consists of temperatures, windspeeds and cloudiness measurements. Data is also retrieved from daily measurements that consists of daily averages of temperatures, windspeeds and cloudiness measurements. And also weather forecasts which contains only temperatures and windspeeds.

Data retrieved from Digitraffic consists of different types of maintenance tasks and traffic messages in an area. Data is also retrieved about road conditions e.g., daylight, road temperature and overall condition in each area at current time and at 2h, 4h, 6h, .. to the future. We also retrieve one image of the area's road camera.

View

The view will consist of data_visualization.py and graph.py. View's job is to show some visualization of data received from the model. Data_visualization.py class visualizes the data for the user. It works as a view for the application. It only has access to the controller. Graph.py uses the weather and road data to draw visualization of the data as line and scatter graph.

Project planning and division of responsibilities within the group

Our team started planning the project on time. During the first weeks, we focused on reading the group assignment instructions and planning what kind of functionality our project will include. We brainstormed with the whiteboard every week, thinking of the best solutions for the needs of our project.

We decided to divide the responsibilities as follows. Kasperi took care of the project's user interface, Ilari took care of the graphic implementation, and Ville and Viivi took care of the FMI and Digitraffic APIs. We continued to work on our own areas independently, holding weekly joint meetings to review the current situation/progress. As the deadlines approached, we prepared the design document together, but Ville and Viivi had the main responsibility.

Open communication within the group and weekly meetings helped the progress of the project. Meetings became daily as the final deadline approached. The deadlines of other courses put pressure and partly also obstacles to the implementation of our project.

Self-evaluation

Overall, we achieved some of our goals with the project. The deadlines of other courses put pressure and partly also obstacles to the implementation of our project.

Partly because of the time challenges, we had to ditch the map view idea, that we thought about at the beginning. In the map view the user could freely choose the wanted area. In the middle of time restrictions, it was much easier or more sensible to implement a ready-made list of cities from different parts of Finland and find out the weather and road data of predetermined areas.

The first half of the project was spent figuring out the whole project and mapping out everything that needs to be implemented. A long-term investigation puts pressure on the implementation of the project and thus we did not have the opportunity to implement everything as we would have liked or as earlier than we would have liked.

All our team members have used Python before, but PyQt5 was totally new for all of us. It caused some headache as most of the application uses it. Learning PyQt5 while implementing the features made the progress slower than expected. PyQt5 is a Python translation of the Qt5 GUI-library for written in C++ so it has some flaws. We noticed that Python debugger wasn’t always able to report causes for crashes which made the debugging real pain at times. Maybe some other GUI-library would have been faster to use, but it’s hard to say with certainty.

During the whole implementation of the application we kept in mind the basic programming principles and user friendly design patterns for the GUI. Program logic is split into smaller functions that serve their own purposes. These functions are combined into classes that are separated into their own files for better structure. GUI is implemented in a way that there should be only relevant controls visible for the user. The controls are also programmed in a way that the user cannot make errors for example trying to fetch data from the future. Even though the application is unfinished, we are very happy with the code style it has.