

COMP.SE.110 Group project

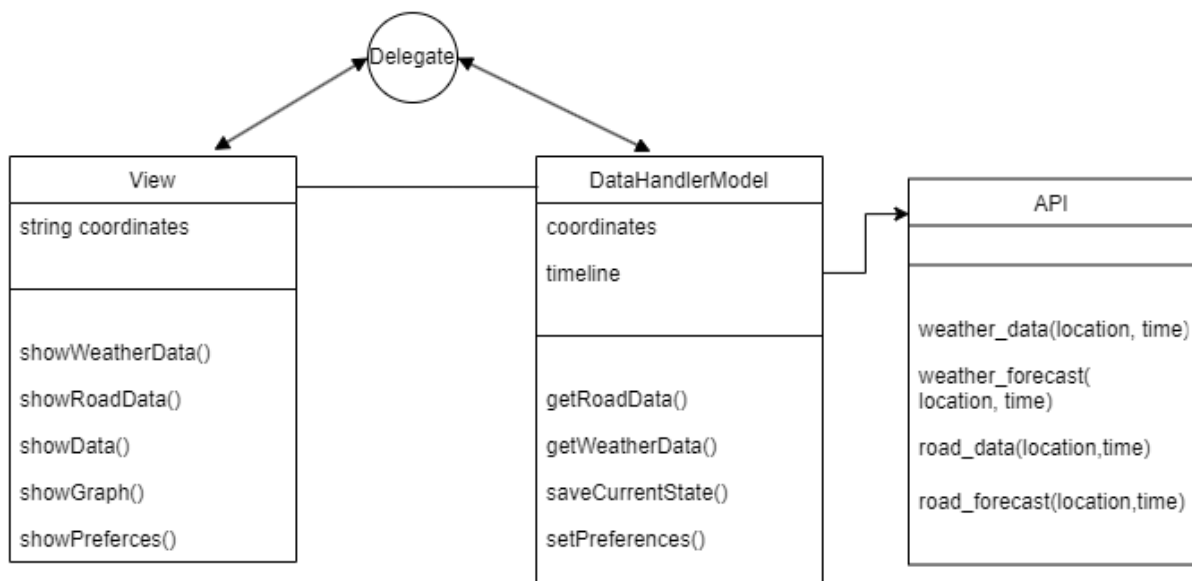
Ville Venetjoki, Ilari Miettunen, Viivi Mustonen ja Kasper Kouri

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 must be able to customize what and how the data is shown. The user must also be able to save some preferred views that can 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 will be implemented using PYQT / QML. QML follows the Qt approach on event-based programming, which means that control is 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.

Class diagram:



Due to Python not having an easy way of creating interfaces, our program is going to have classes 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 model 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.

The view will consist of methods that show some visualization of data received from the model. When a button is pressed and a method is called for, it's going to make a function call towards the model and make changes to the window according to data received from the model.

The model is going to act as the main logic and storage component in the project. It's going to ask the API for information in the form of python dictionaries, store and pass necessary data from said objects to the view. Additionally, the model is going to have some way of saving information in a way that can be accessed later.

The API's purpose is to make http requests according to parameters received from the model. Received data is parsed and passed on to the model for further processing. The implementation of the API must be multi-purpose since there are many types of requests (forecast, observations, etc.) from both data sources, which return different types of data (XML, JSON).

UI is designed to be implemented in a way that there is one class View which has side functions to create needed components and implement their logic. For example, side panel has a function that creates it and some more functions to get buttons behave as they should when clicked. UI is created using PyQt5

- Possible third-party libraries
 - python "requests"-library
 - Used because it is much more intuitive and easier to use compared to python's built in urllib-library
 - "pip install requests" in the python terminal to install
 - Pandas –library
 - Data analysis library will be used to parse XML into easy-to-use Pandas dataframes
 - PyQt5/QML
 - Used to build the user interface
 - Folium or Google maps (Optional)
 - We might try integrating an interactive map into the project to visualize traffic messages and road maintenance messages
 - Matplotlib
 - Used in Graph-class to create visualizations of data
 - Numpy
 - Used for mathematical functionality

Self-evaluation:

Designing UI had some difficulties due to lack of existing knowledge. However, we tried to learn some usable design patterns before implementing anything.

While studying and implementing the API fetching functions it came to light that the responses of the open interfaces aren't too complicated. This caused a change of plans to give up the plan of using inheritance to implement objects to store data in. Instead, it would make more sense to parse the responses into python dictionaries and return pass them on as they are. In hindsight dictionaries are not too far off from just very stripped-down classes in python so this change has little effect on usability but makes things much simpler to implement and possibly modify later.

In conclusion, beginning the implementation of the project has brought new perspectives into our design. Some of the things that seemed good in the designing phase proved to be unnecessary and/or unpractical, which is why making changes was deemed appropriate. However, with necessary changes having been made we're now well on our way to having a working project implemented.