SOFTWARE DESIGN PROJECT

COMP.SE.110 Software Design (spring 2022)

Aatu Laurikainen, aatu.laurikainen@tuni.fi, 283348

Aleksi Sirviö, aleksi.sirvio@tuni.fi, K444293

Mohammad Hassan, mohammad.hassan@tuni.fi, 50457535

Sakari Klemola, sakari.klemola@tuni.fi, 273086

Course assignment

18.2.2022

TABLE OF CONTENTS

1 INTRODUCTION

The goal of the project is to create and deploy a software program that visualizes data from stations that monitor greenhouse gases. Two types of data will be presented in the application: one is real time data provided by SMEAR[1] and another one is historical data provided by STATFI[2].

The application will compare the current situation based on the historical data. It will break down averages based on database provided by STATFI. The user will be able to select the time period and one specific or more than one monitoring station to visualize the data. This software project is for a university course named Software Design and the project is developed by a group of students. The purpose of this document is to record the development of said software.

2 TECHNOLOGIES

The exercise group working on this project was formed based on the preference of working with C++. Thus, C++ is the chosen programming language for the implementation of the application. Qt[3] is the group's graphical user interface library of choice, as the group members have experience working with Qt in previous courses. Using a robust graphical user interface library like Qt speeds up the project and allows the team to focus on the business logic of the application. Even more complicated feats such as drawing line graphs can be conveniently performed with Qt.

3 STRUCTURE

The application will be structured according to the model-view-controller (MVC) design pattern. The MVC pattern has a number of benefits, such as helping make the code base clearer, more structured, and easier to iterate on and expand in the future. The division of responsibilities inside the program naturally translates to a division of responsibilities among the developers. Traditionally Qt uses an MVC variant called the model/view architecture, which utilizes a delegate doing similar work as the controller, but in a smaller capacity. The plan is to create a controller with more responsibility than the traditional Qt delegate. In this document, the terms controller and delegate are used interchangeably. The three main components and their responsibilities will be discussed next. Image 1 describes the structure of the application with a very simple component diagram.

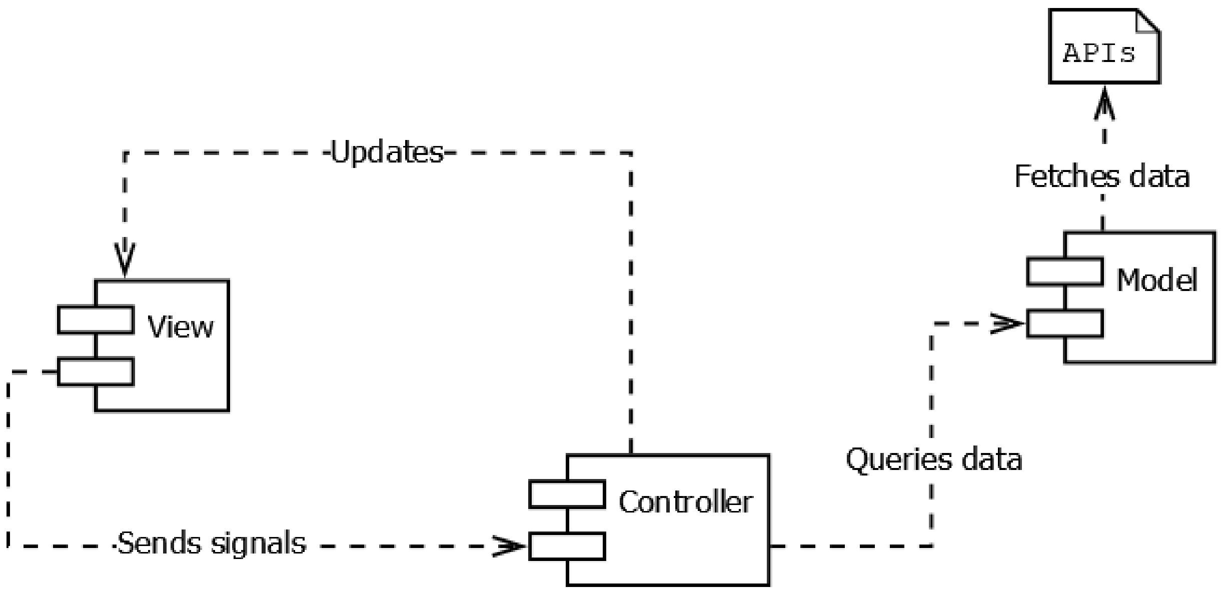

Image 1: Application structure as a simple component diagram

3.1 Model

The model is a representation of the data used in the application. The responsibilities of the model include:

1. Fetching data via the APIs of the given databases
2. Processing the data and arranging it into suitable container(s)
3. Offering an interface for other components to access the data in predetermined ways

To follow the single responsibility principle (SRP), the model must be divided into multiple classes. For example, one set of classes will be entirely responsible for accessing the databases and fetching the data, while another class will maintain the containers for the data. Since fetching data from SMEAR and STATFI is different, a separate fetcher classes will be created for both. These classes will implement a common data fetcher interface/abstract class.

3.2 View

The view is responsible for the graphical user interface (GUI) of the application. The view passively maintains the UI components and waits for user input. The way the application is being designed, there can be 1 to n number of views, all of which represent the model differently. This makes comparing different representations of the data convenient for the user. The view will most likely be implemented in a single class, as its responsibilities are limited, and its implementation is quite simple thanks to the use of the Qt library.

3.3 Controller

The controller is the component that connects the view to the model. It contains most of the business logic of the application. The responsibilities of the controller include:

1. Interpreting user input events
2. Accessing data from the model
3. Operating on the data based on the user input
4. Updating a view to correspond to the processed data

The controller may be divided into multiple classes if deemed appropriate. Possible class division is not easy to picture at such an early state of the design process.

4 BOUNDARIES AND INTERFACES

Image 2 shows data flow of components throughout the application. A component diagram approach has been used to design this data flow.

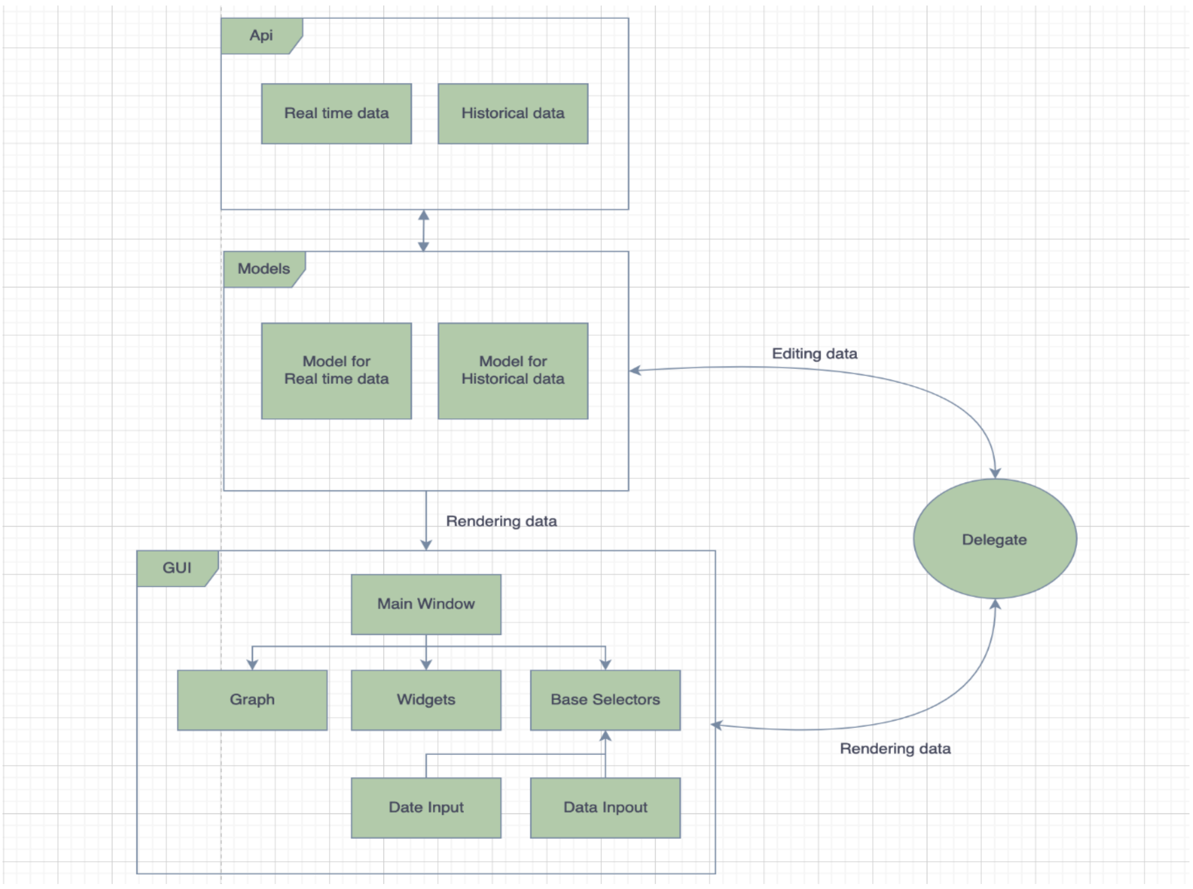

Image 2: Application components interaction in component diagram

Since two kinds of data have been provided from API one is real time data and historical data. Therefore, only two models are needed to synchronize the data in the application. In the GUI main window there are graphs to visualize desired data, widgets to select certain options for data and base selectors where date and data inputs are accepted for desired data. The approach of QT is that there is a delegate who acts as a controller to supply editing and rendering data between Models and GUI. If the project demands controller over delegate, it is possible to convert the delegate into a controller.

5 GRAPHICAL USER INTERFACE

On start-up the program opens the main view as can be seen in image 2. Most of the available space is taken by the graph where the data is shown. On the x-axis is the given timeframe and on the y-axis is the emission amount. On said graph the data will be shown as points which are connected to each other by lines. On the same graph there can be multiple lines representing different gas emissions. These lines will be colorcoded so that the user can easily identify which line represents which gas. We decided to make the graph take a lot of space because doing so will ensure that a lot of data can be shown at once without it becoming hard to discern and that small changes are easier to notice.

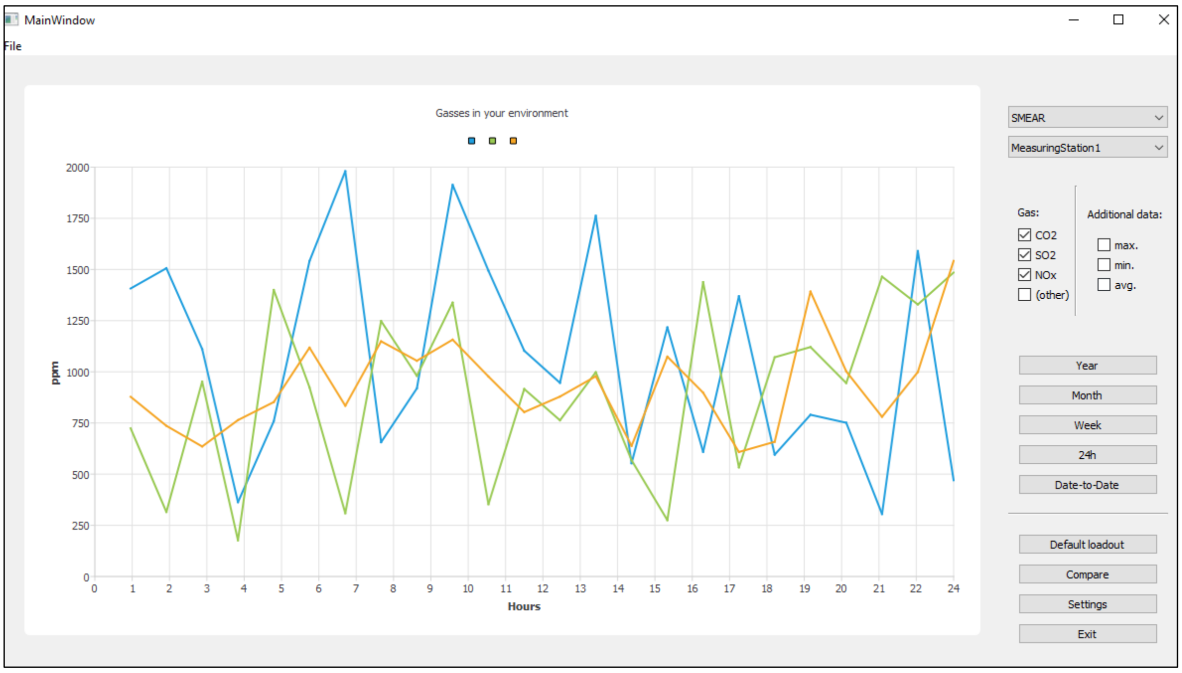

Image 3: GUI prototype

On the right side of the main view are different buttons and boxes with which the user chooses the data they want to see in the graph. At the top there is a drop-down menu from which the user chooses the database from where the data is stored. The next drop-down menu is for choosing the distinct measuring station from where the emission measurements have been taken. Under that are check-boxes for different gases and values. The user can choose as many gases as they please, though at least one must be chosen so that the value boxes can be checked. Next up is the date-to-date button which, when pressed, opens a small window to which the user can input the start- and end-dates as they please. Earlier selections define the available time period. Below that is the compare button which opens a carbon copy of the main view. These views can show different data and comparison between views is done manually by the user.

On the top of the main view there is the main navigation bar. There we will put other functionalities such as choosing a specific loadout or changing default settings. We chose to do this because we deem it important that only settings that immediately affect the view are easy to access and are visible at all times compared to settings which the user may or may not always need.

REFERENCES

1. SMEAR database and API. https://smear-backend.rahtiapp.fi/swagger-ui/ (Viewed 16.2.2022)
2. STATFI database and API. https://pxnet2.stat.fi/PXWeb/pxweb/en/ymp/ymp__taulukot/Kokodata.px/ (Viewed 16.2.2022)
3. Qt. https://www.qt.io/ (Viewed 16.2.2022)