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

19.3.2022

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. Various libraries inside Qt are used for specific tasks. These include QChart, QLineSeries, and QValueAxis for drawing graphs, QDate and QValidator for interpreting user inputs as dates, and QtNetwork for doing network calls. QJsonObject and QJsonArray are used for parsing JSON data.

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.

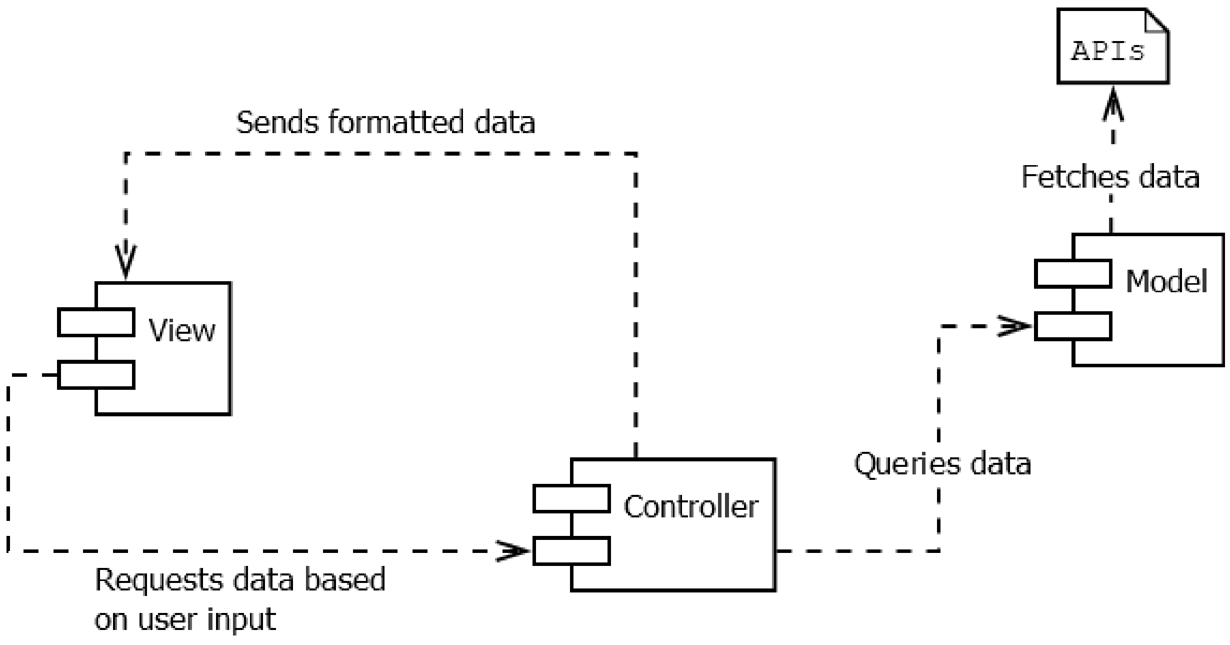


Image 1: Application structure as a simple component diagram

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. Parsing the JSON data and arranging it into suitable container(s)
3. Storing the data and offering a simple interface for the controller to access it

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 are implemented for both. These classes implement a common data fetcher base class and follow the strategy design pattern. A JSON parser class is responsible for parsing the fetched data into data structures that fit our needs. Likewise, different parsing strategies are needed for SMEAR and STATFI data. Finally, there is a data storage class, the only purpose of which is to store and offer access to the parsed data.

A key difference between STATFI and SMEAR data handling is that since the STATFI data is much simpler and there is much less of it, all of it is fetched and parsed at the launch of the application. However, SMEAR data is only fetched and parsed when the user has made specific selections in the GUI and wants to update a graph. Some SMEAR data is fetched on launch to determine limit values for the data. These limits are used to prevent the user from making invalid selections in the GUI, such as trying to query data from years where no data points exist.

3.2 View

The view is responsible for the graphical user interface (GUI) of the application. The view 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 GUI is mostly implemented inside a chart window class, but there is also a separate time window class that is used when the user inputs start and end dates for the data they seek.

In addition to that, there are user selection classes that keep track of user actions in the application. The data inside the user selection classes is formatted as such that it can conveniently be sent to the controller and used to filter the data the user wants from the model. Different user selection objects exist for STATFI and SMEAR data. Both inherit a common user selection base class. There is also a utility class called Date, that is used to represent and easily compare different points in time. This is mainly useful when filtering data from SMEAR.

3.3 Controller

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

1. Reacting to data requests from the view
2. Accessing data from the model
3. Operating on the data based on the user input
4. Offering the processed data back to the view

The controller is implemented as a set of static methods that are called by the view as needed. The controller itself doesn’t store any data and no instances of the controller need to be created. Separate methods exist for processing STATFI and SMEAR data.

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.

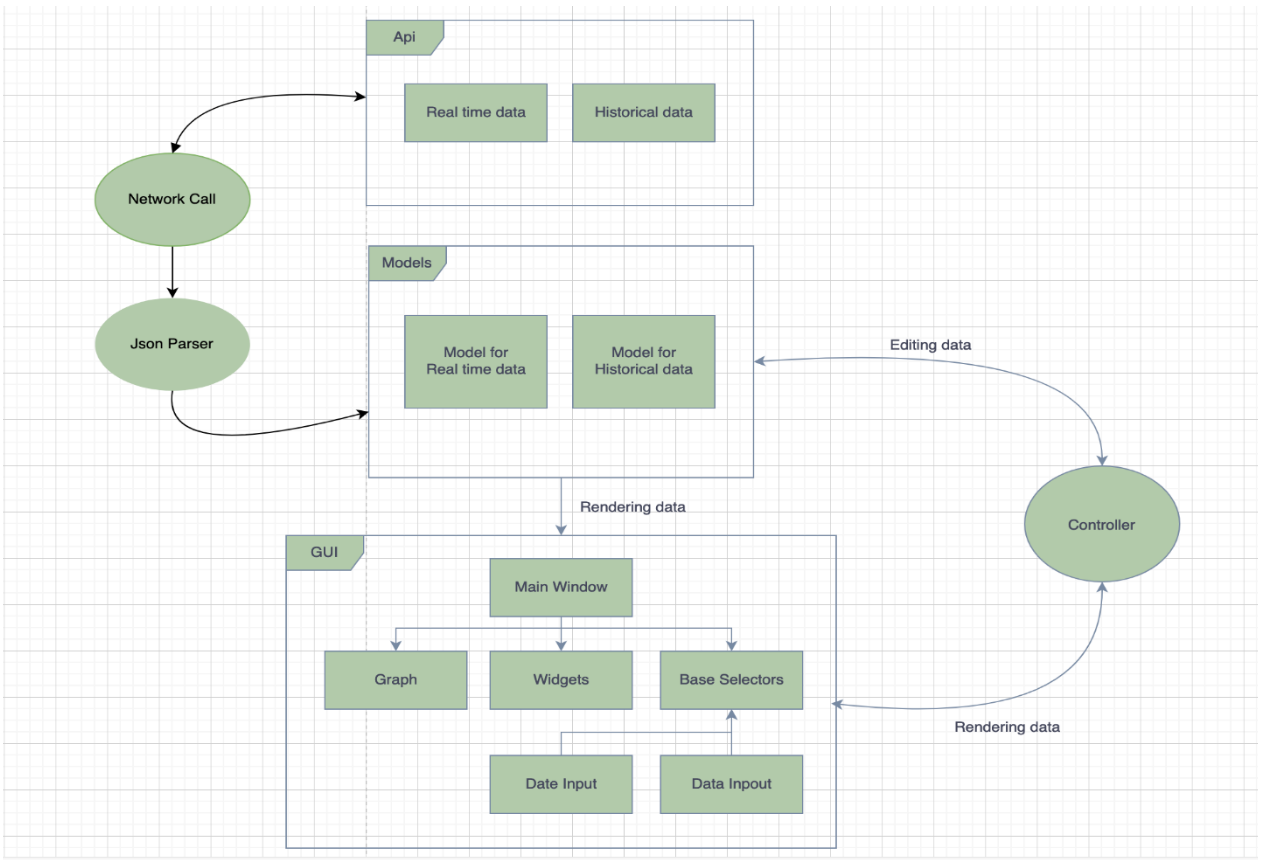


Image 2: Application components interaction in component diagram

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. A network call brings the data and through a JSON parser data satisfy the models. 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. A controller has been established to filter data according to user selection from models. After filtering the data, controller send the data to be rendered in GUI.

5 GRAPHICAL USER INTERFACE

On start-up the program opens the main view as can be seen in image 3. 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 color-coded 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.

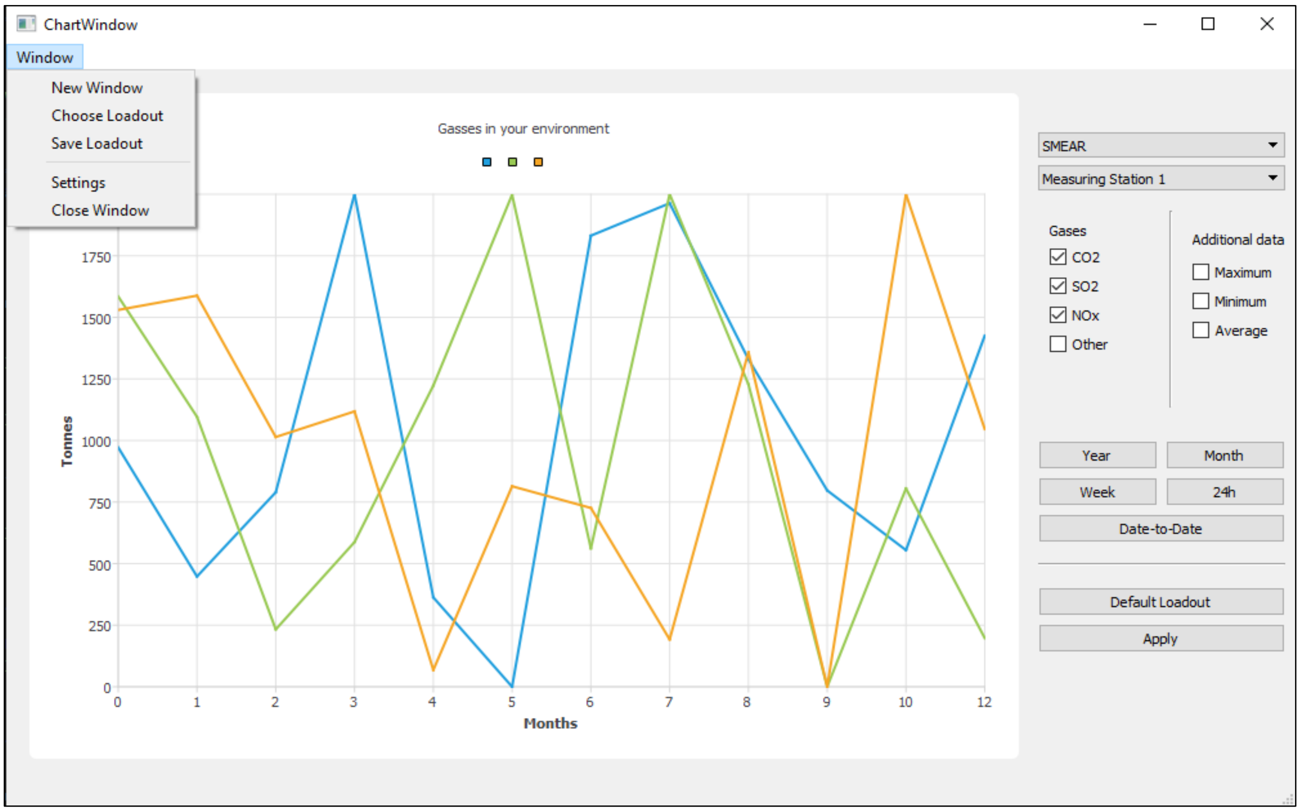


Image 3: GUI When SMEAR is the chosen database

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 any data can be shown.

Vital part of showing the data in the graph form is for the user to click the Apply button when they are done with their query. This activates the program to fetch the wanted data and put it into a visual form. This design decision is done to prevent the user from accidentally fetching any unnecessary data. If the view would update every time a check-box is changed, or radio button pressed the program would probably freeze for a few seconds due to the lack of multi-threaded programming.

As the database is changed from SMEAR to STATFI in the drop-down menu above, the possible selections will also change as seen in image 4. The checkboxes will change into radio buttons as the STATFI data can only be displayed one graph at the time. The drop-down menu for different measuring stations will also be absent when STATFI is selected as the database. This is because STATFI doesn't give out data from their individual measuring stations. As a final difference the date-to-date button is changed into Year-to-Year button. This will also affect the window that pops up as the buttons is pressed as the user input for this will be different.

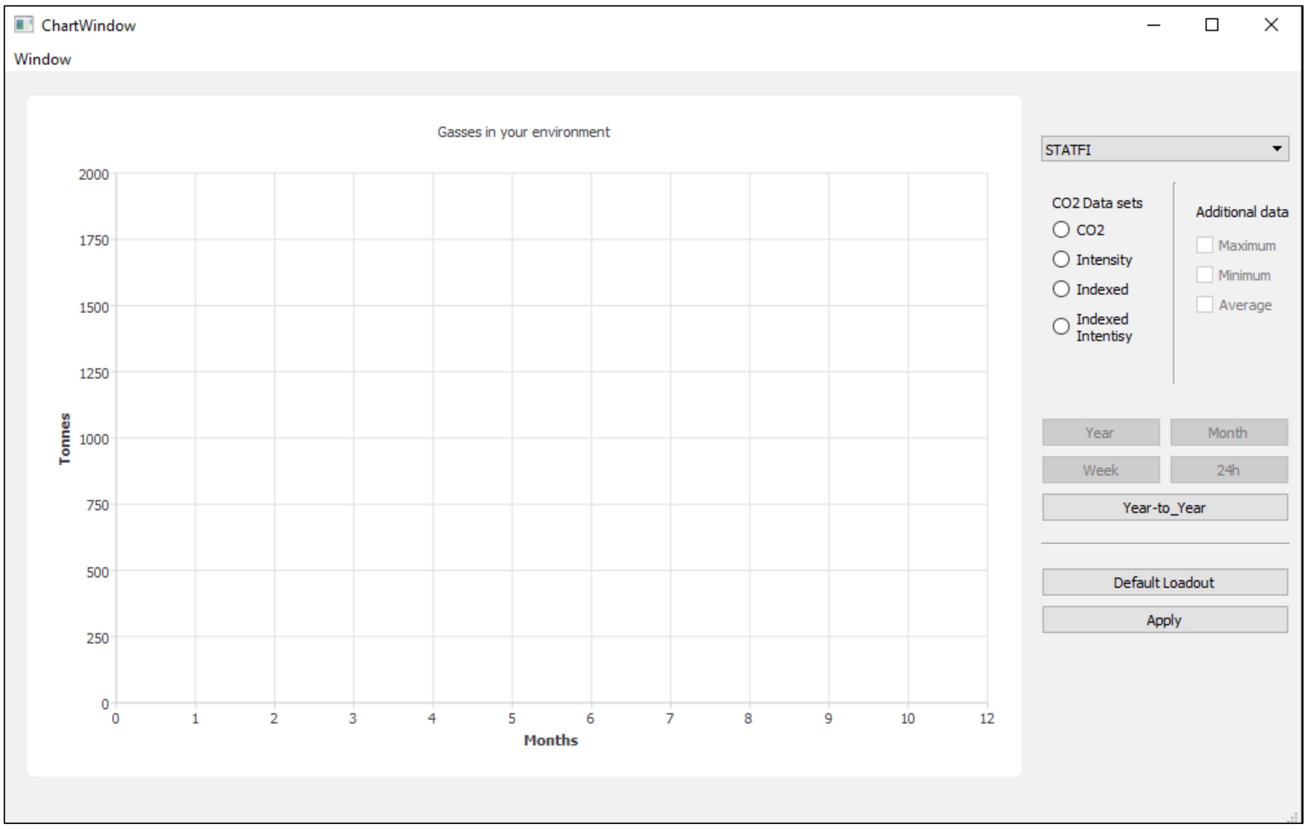
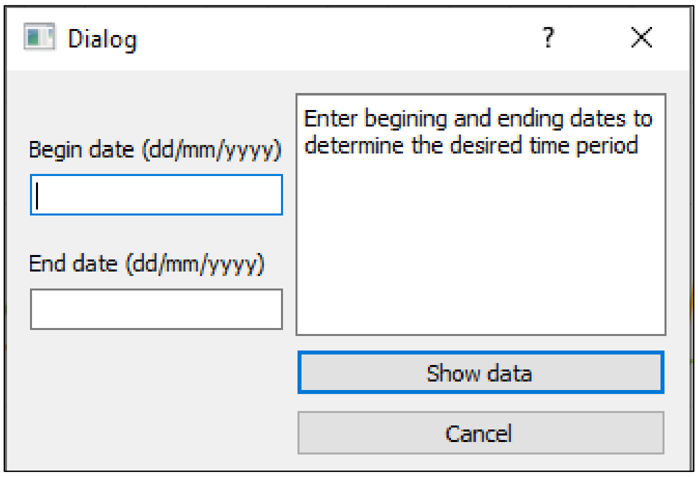


Image 4: GUI When STATFI is the chosen database



Enter begining and ending dates to determine the desired time period

Image 5: The Date-to-Date window

As the database is changed from SMEAR to STATFI in the drop-down menu above, the possible selections will also change as seen in image 4. The checkboxes will change into radio buttons as the STATFI data can only be displayed one graph at the time. The drop-down menu for different measuring stations will also be absent when STATFI is selected as the database. This is because STATFI doesn't give out data from their individual measuring stations. As a final difference the date-to-date button is changed into Year-to-Year button. This will also affect the window that pops up as the buttons is pressed as the user input for this will be different.

Next up is the date-to-date button which, when pressed, opens a small modal window to which the user can input the start- and end-dates as they please. Example if this window can be seen in image 5. Earlier selections define the available time period. 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. On the navigation bar there is also an option to open a new window for data comparison. This action opens a carbon copy of the main view. This copy works the same as the previous window and can display all the same data the first window could. These views can show different data and actual comparison between views is done manually by the user.

The actions in the navigation bar are there to save space for the actual data and graphs. 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.

6 SELF-EVALUATION

For the mid-submission, we designed the pipeline of STATFI data from the online database all the way to the graph quite carefully. All of this was worked out in a design session right after the prototype submission. All the way until the mid-term submission, we were able to follow the design very well. Right now, we feel that the STATFI data pipeline is efficient and easy to follow and update. One change that was made, was that for practical reasons we decided to implement the actual updating of the graph in the view instead of the controller. This way we eliminated one dependency, as the controller doesn't need to be aware of the view.

On the other hand, since most of our focus was on the STATFI data, we may have somewhat neglected the integration of other databases. Since the beginning, there were ideas of implementing STATFI and SMEAR data related functions in separate classes that would inherit a common base class, perhaps following the strategy design pattern, but none of this has been implemented in practice. Thus, we most likely have some refactoring and further design to do right after the mid-submission to be able to add SMEAR processes into the application in a clean way.

With that said, adding parent and sibling classes to the classes that currently operate on SMEAR data should not be much work, as these classes are very small in size thanks to their carefully defined singular responsibilities. We are happy with the STATFI data pipeline, and with the knowledge we accumulated while designing and developing it, we feel confident in being able to build the slightly more complicated SMEAR pipeline next.

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)