Design document

Kalle Gustafsson

Tuomas Mäenpää

Aleksi Suonsivu

Table of Contents

1. Introduction

This design document describes functionality of a group project program and shows the steps implementing the program from prototype to final working program. We chose to implement the program with Python and the PyQT-framework.

2. Program functionality

2.1 Requirements

Use case 1):

- User can select which monitoring station's (one or more) data they are interested in. - User can select which greenhouse gas / data variable they are interested in (minimum options: CO2, SO2, NOx)
- User can set a time period from which data is shown - In addition to viewing raw data, user can ask for a minimum, maximum and average values of the selected data
    - For a given time period for a given station
    - For several stations for a given time
    - For several stations for a defined time period
- Not all measuring stations provide exact same data. However, the user interface should be user friendly in such a way that the user does not need to do "test-and-try" to see where the data exists. In other words: the application should only allow selecting data that is actually available.
- Error handling: you must efficiently prepare for errors in the data and handle null values.

Use case 2):

- The user can select a time range for viewing data from STATFI
- User selects one or more available datasets: CO2 (in tonnes), CO2 intensity, CO2 indexed, or CO2 intensity indexed (one or more).
- User is shown a visualization of the data

Use case 3):

- User is given a visualization of the CO2, SO2 and NOx values from SMEAR for a time period specified by the user, for the station(s) selected by the user.
- The user can select data available from STATFI on historical averages
- The user can specify the time range (e.g. year 2010, or years 2000-2009) they are interested in.
- The historical values are shown alongside real data in a comparable way.

Use case 4):

- The user selects a time range for viewing data from STATFI - User selects data from CO2 (in tonnes), CO2 intensity, CO2 indexed, or CO2 intensity indexed.
- User is shown a visualization of the data
- The user is able to choose real time data from SMEAR. NOTE: SMEAR data does not handle the same timespan as STATFI. You will need to check how long back data is retrievable from SMEAR and guide the user accordingly.
- Breakdown of the historical average to the selected year based on SMEAR data is visualized.

General:

- The user can save preferences (e.g. certain stations, certain time period in history, which greenhouse gas(es) from SMEAR, which statistics from STAT FI) and apply them when using the software later
- Design must be such that additional data sources could easily be added

2.2 UI-prototype

The program opens to a view resembling the following figure. The plot button has been pressed already in this figure but there won't be a plot shown at start in the final program.

The "historical" view is the same appearance-wise as "real-time" view. In "compare" view there are two charts to be compared to each other followingly.

The “averages” view also resembles “historical” view in terms of appearance.

3. Program structure

3.1 Description of the program structure

3.1.1 External Interface

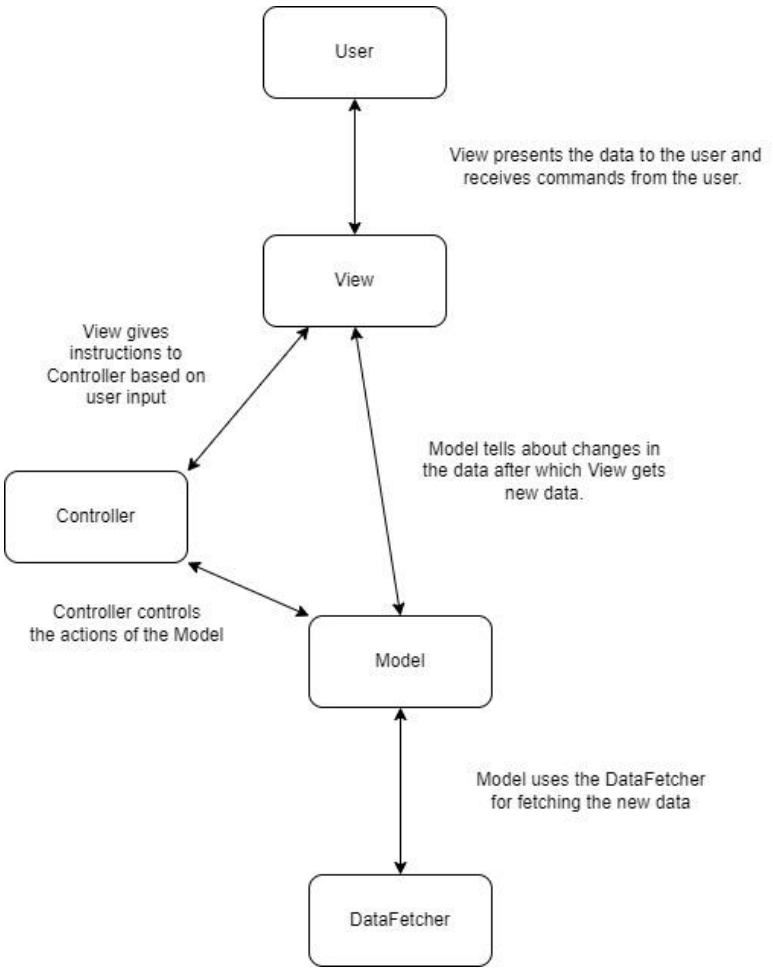
The program reads the data from the database using an external interface. This interface will be implemented inside the DataFetcher class. This class is used to fetch data and forward it to the user in correct form.

3.1.2 Internal Interfaces

The internal interfaces of the program will be implemented using MVC architecture. (Model, View and Controller)

- Model works as a middleman between the external interface and the internal interfaces. Model handles the data received from the DataFetcher class and forwards it to the other classes.
- View fetches the data from the Model. Model will also contain functions that will be used based on the user input.
- Controller interface will include methods that communicate with View and Model.

4. MVC



4.1 Model

- Receives the data from the DataFetcher and processes the data to correct form.

- Contains the backend logic of the program.

- Sends information about changes in the data to the MainWindow class.

Modules: DataFetcher, pandas

Public methods:

- fetch_realtime(self, start_date, end_date, table_variables, interval, aggregation)
- fetch_historical(self, years, categories)
- clean_data(self, df: pd.DataFrame)

Private methods:

- __init__(self, datafetcher: DataFetcher)

4.1.1 View (MainWindow)

- Created with PyQt5

- Plots the data. Plotting is done with Matplotlib library.

- Updates the UI based on the user input.

Modules: tracemalloc, PyQt5, PyQt5.QtGui, matplotlib.figure, matplotlib.backends.backend_qt5agg, matplotlib, pathlib, json, Controller

Classes:

class Ui_MainWindow(object)

Public methods:

- setupUi(self, MainWindow)
- retranslateUi(self, MainWindow)
- setup_controller(self, controller: Controller)
- plot_realtime(self)
- plot_historical(self)
- save_realtime_plot(self)
- save_historical_plot(self)
- save_compare_plot(self)

class MplCanvas(Canvas)

Private methods:

- __init__(self)

class MplWidget(QtWidgets.QWidget)

Private methods:

- __init__(self, parent=None)

class MplCanvasDouble(Canvas)

Private methods:

- __init__(self)

class MplWidgetDouble(QtWidgets.QWidget)

Private methods:

- __init__(self, parent=None)

class CheckableComboBox(QtWidgets.QComboBox)

Private methods:

- __init__(self, *args, **kwargs):

Public methods:

- loadGases(self, event)
- resizeEvent(self, event)
- eventFilter(self, object, event)
- showPopup(self)
- hidePopup(self)
- timerEvent(self, event)
- updateText(self)
- addItem(self, text, data=None)
- addItems(self, texts, datalist=None)
- currentData(self)

Subclass: class Delegate(QtWidgets.QStyledItemDelegate):

Public methods:

- def sizeHint(self, option, index):

4.1.2Controller

- Listens to View's instructions.

- Tells the Model to behave based on user input.

Modules: pathlib, json, datetime, PyQt5, Model

Public methods:

- handle_realtime(self, start_date='', end_date='', table_variables=[], interval='', aggregation='', use_defaults=True, save_defaults=False)
- handle_historical(self, years=[], categories=[], use_defaults=True, save_defaults=False)
- save_historical_defaults(self, years: list, categories: list)
- save_realtime_defaults(self, start_date: str, end_date: str, table_variables: list, interval: str, aggregation: str)
- get_year(self, date: QtCore.QDate)
- datetime_to_ISO_string(self, date: QtCore.QDateTime)
- get_tablevariables(self, stations: list, variables: list)
- generate_filepath(self, source)

Private methods:

- __init__(self, model: Model)

4.1.3 DataFetcher

- Fetches data from STATFI and SMEAR API's

- Returns a Pandas dataframe

Modules: requests, json, pathlib, pandas, numpy, io

Public methods:

- get_realtime(self, start_date, end_date, table_variables, interval, aggregation)
- get_historical(self, years = [], categories = ["Khk_yht", "Khk_yht_index", "Khk_yht_las", "Khk_yht_las_index"])

Private methods:

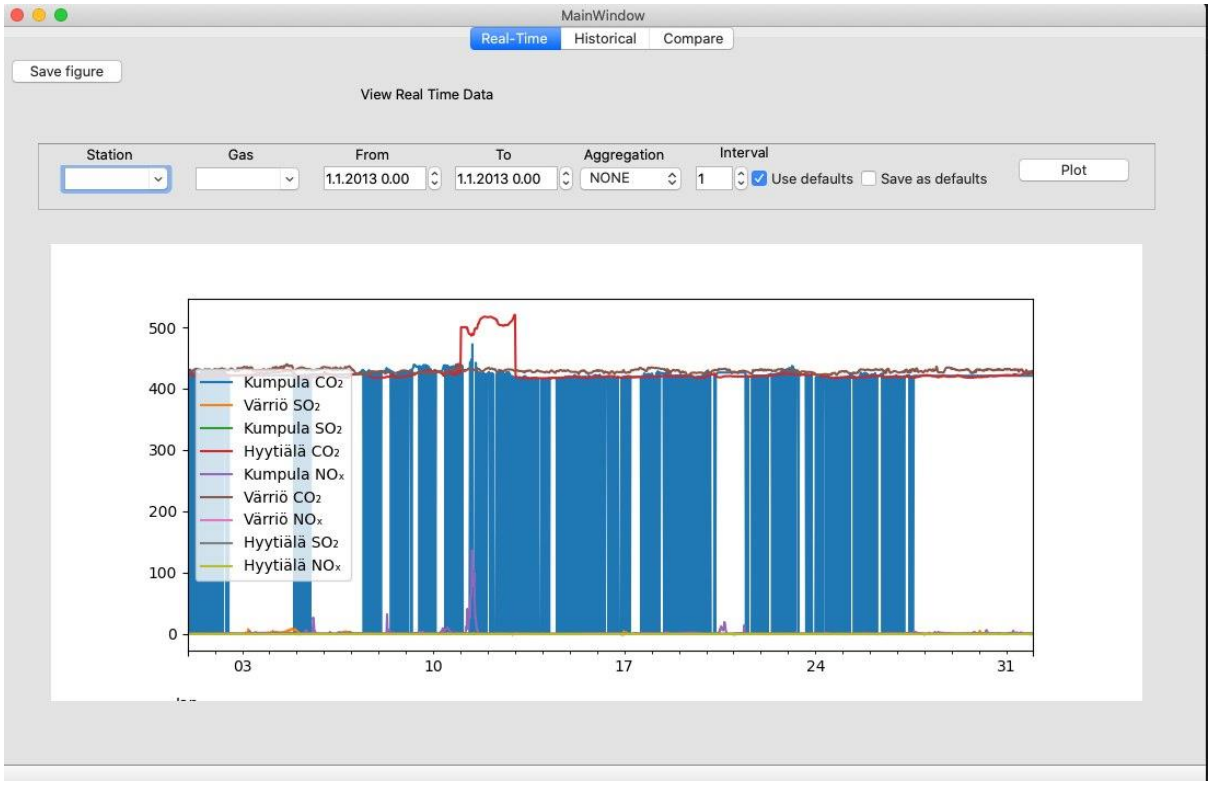
- __hist_to_dataframe(self, hist)

5. SELF EVALUATION

5.1 Final product

5.1.1 UI

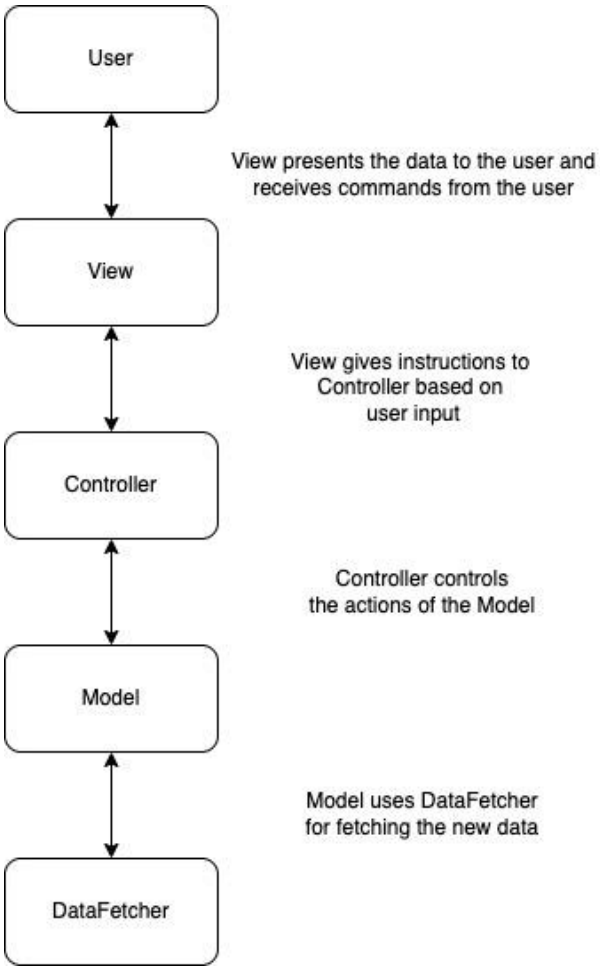
The basic functionality of the UI has stayed pretty much the same as in the prototype UI. Our UI is divided into a three-tab-window, which can be used to navigate between different views. This tab view makes it easy to implement different features while keeping the UI simple and easy to use. Biggest changes compared to the prototype are the removal of the ”Averages”-tab, changes in the color scheme and layout. The ”Averages”-tab was removed because we noticed that it doesnt need its own tab when we can show the averages same time with first three use cases.



This is the final version of the UI of our mainwindow. It now has three tabs instead of four and the selection of what data user wants to see has become more specific. Plotting is done with matplotlib.

5.1.2 Final Model architecture

For the final MVC architecture we ended up simplifying the structure. We removed the connection between the Model and View classes, as it caused a loop problem regarding the imports. The final structure is therefore simplier and the data flows linearly inside the program.



5.1.3 Conclusion

During the last phase of the project, we encountered some problems regarding the work amount because of our group size. We didn't have the fourth student with us, so the workload was quite heavy for rest of us.

During the coding, some of the plans we had were forgotten and we went ahead with more simple ways to implement the code. This resulted in MVC structure that was a little bit different than what we had planned before.

The whole project felt like we were behind the schedule and the final work did not meet all of the mandatory requirements. However, we feel like we did what we could and we are satisfied with the result.

The program itself runs a bit slow, as it downloads a huge amount of data when the application is running. The UI also could be better looking, but this simple UI provides the user the needed inputs and plots.