Design document – Final submission

Henriikka Rikala, Matias Nurmi, Eetu Häkkinen ja Mikko Haavisto

Contents

Decision making process

Our group has decided to use C++ and Qt to do our software. Prototyping was done using Figma. Those were chosen mainly because we were familiar with them.

As we used Qt our design follows MVC-pattern. Some Qt elements already use MVC, and it felt natural to design our software to follow it.

For displaying the data as graphs, we used QtCharts-module. It was presented in this course and seemed like the best approach.

Qt doesn't provide the selection type we wanted so we decided to use opensource code from the internet to create multiple selection combobox. It was altered to fit our needs. [1]

We used SOLID principles as guidelines for our software. The whole software follows the single-responsibility principle. The open-closed principle was not followed that well in our software. If we would like to add new APIs we could not do that without changing some of the existing functions. Our software does not have so much inheritance and it is still a small software. That is why we did not really concentrate on the Liskov substitution-, interface segregation- or dependency inversion principle.

Idea for our software

This part is a short description of how our design works. The idea is also visualized in the prototype for the parts that are still not implemented.

- First user sees the starting page which shows no data yet. The header of the page shows the selections that the user needs to do to show data.
    - There are boxes for selecting data, measurement station and the time period
    - From plus-sign user can submit choices
    - From ⁝-sign user can take closer look on settings
- When user selects data and station on certain time and presses +-sign, data is shown. If more than one station is selected, those will be shown on the same chart. If more data is selected it is shown in its own chart. (There is also a possibility for multiple stations)
- From settings (⁝-button) user can see the current selections, there is also options for looking history and saved selections.
- In current selections
    - User can remove selections from trash cans

    - User can save current selections from save selections
    - User can clear all selections from clear selections
- In history selection history is shown
- In bookmarks
    - User can check saved selections
    - User can delete saved selections
    - User can upload selections to show the charts

After midterm submission we also added some error handling to our software. Now users cannot do such actions that could cause failure. Also, error messages are given in case of error.

Components

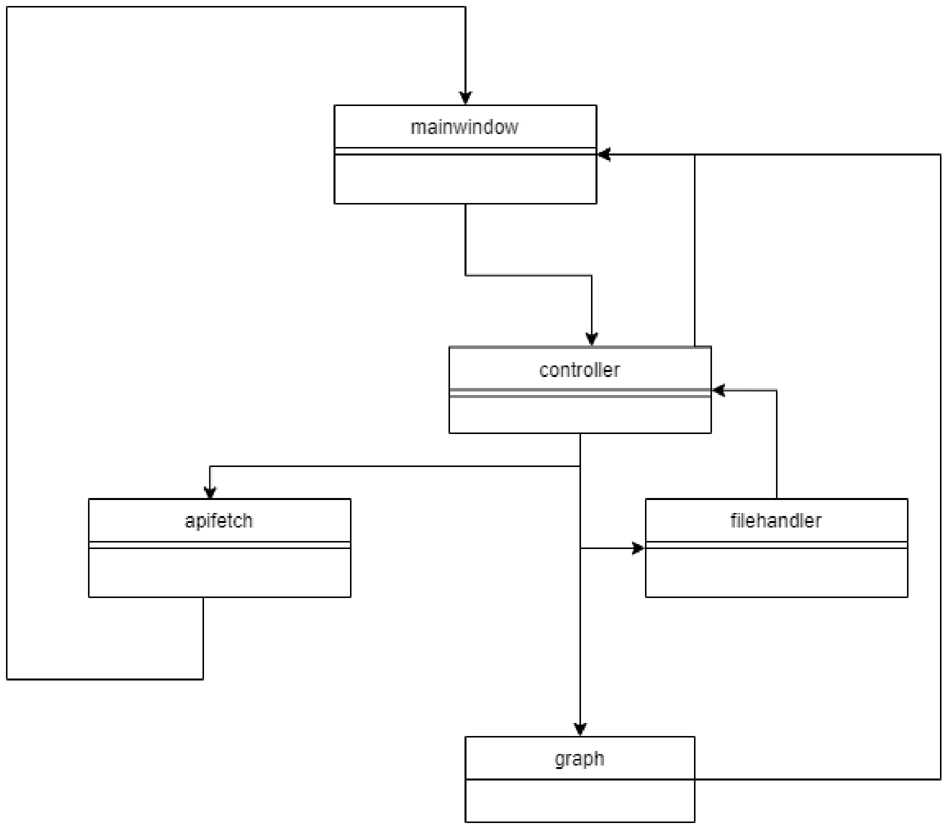


Figure 1 Class diagram

Figure 1 presents a simple class diagram of our software. The attachment shows a more detailed version of the class diagram. Functions have short descriptions on header files.

MainWindow

Mainwindow is the view of our design. It communicates with controller, and it shows wanted UI.

ApiFetch

ApiFetch communicates with controller. It does the wanted API calls to get the right data and then handles the data. It returns the data to mainwindow.

Controller

The controller communicates with mainwindow, ApiFetch and Graph. It uses ApiFetch as data handler and Graph to create graphs.

Graph

Creates graphs when the controller asks. Sends data to mainwindow

FileHandler

Communicates with controller and is responsible for operations towards files.

Self-evaluation

Final submission

How well have we been able to stick to our original design and how well have we been able to implement features based on our original plan

We were able to do the software well according to the original plan. Our design was very well thought out, and the prototype was easy to implement in software. Of course, some UI things changed during the implementation depending on what we thought looked and worked well. Overall, we think we were able to stick to the original design throughout the entire project.

What changes we needed to make to the original design in order to implement all the features

As mentioned in the midterm submissions self-evaluation, the biggest change we made to our design was to drop the option for changing selections on the settings tab. Otherwise the software works a lot like we planned.

Midterm submission

How our design has supported implementation

The implementation phase has been easier because everyone has been on the same page about how things are going to work. We also think that prototyping has helped us to implement the software since we had a concrete model of what we were trying to do.

How well have we been able to stick to our original design

We have been able to stick to our design very well. We thought the ideas very thoroughly while doing the prototype, so we didn't have to do it anymore in the implementation phase. We also discussed how things are possible to do while designing the prototype, so we were able to follow the prototype.

How well have we been able to implement features based on our original plan

Since we already discussed possible implementation ways for some of the main parts in prototyping phase our prototype was easy to implement. We did run into some changes during implementation so not everything is exactly like we thought before.

For example, we prototyped the selections menu in a way that selections could be changed in the settings section. When implementing it we thought that it would make the software unnecessarily complicated and doing it would have been harder. We thought it would be enough to show selections and to be able to remove them one by one.

We also added pages to the implementation. Now our graphs can be shown on multiple pages instead of all showing at once.

How does our design correspond to quality

We have tried to make UI easy to use and clear to read. We thought of those things already when prototyping and in implementation we have evaluated how well the prototyped features function. One important part we focused on was to prevent user from doing things that would cause errors.

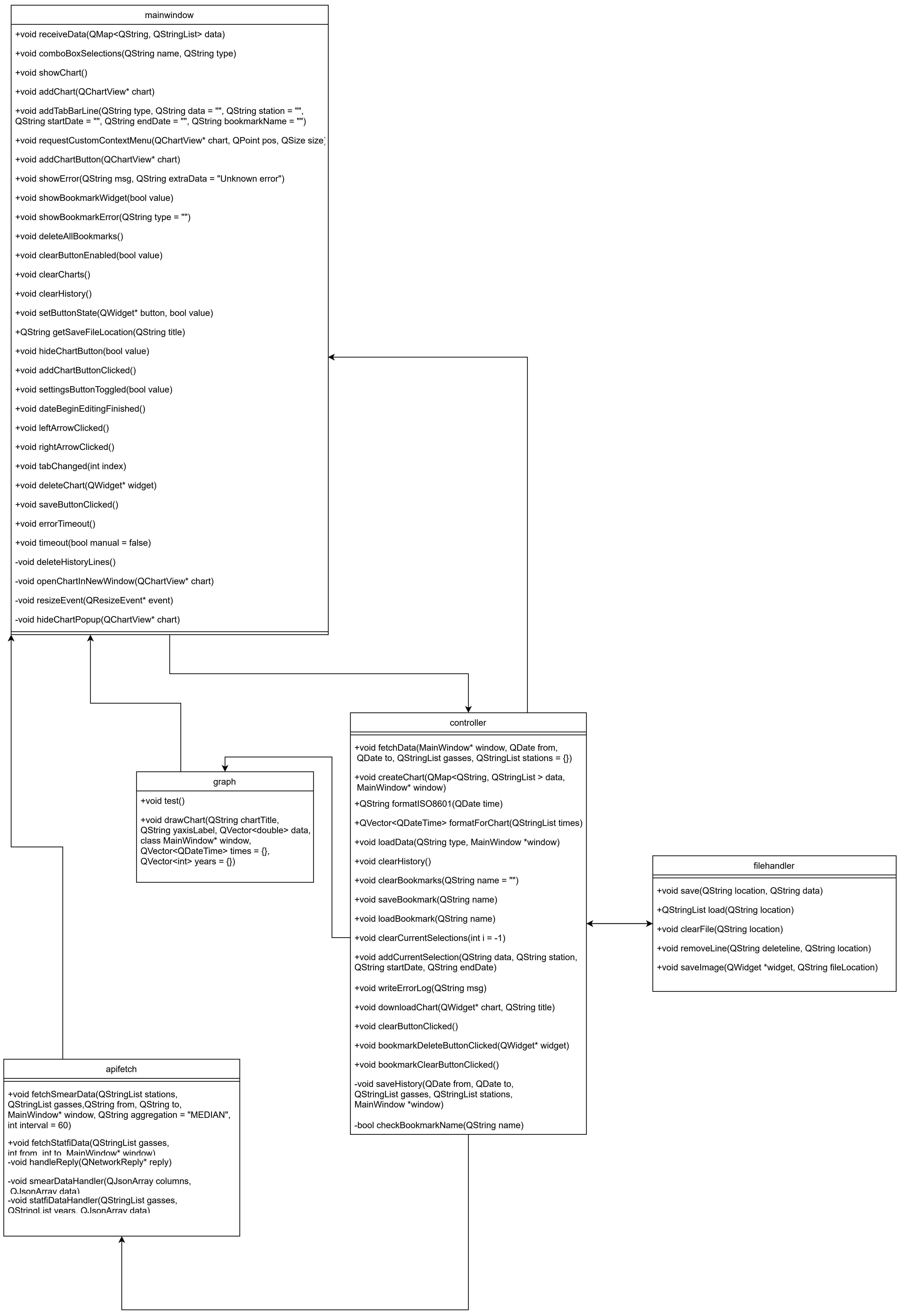
What changes are we anticipating we need to make to be able to implement the remaining features

We still need to implement some features. History, bookmarks and downloading a graph are a few features we are planning to do. Also, some UI improvements like axels on graphs and some buttons. Those changes are probably possible to implement as prototyped. Some changes might occur if something won't work as well as we thought.

Sources

[1] https://github.com/ThisIsClark/Qt-MultiSelectComboBox

Attachments


mainwindow
+void receiveData(QMap<QString, QStringList> data)
+void comboBoxSelections(QString name, QString type)
+void showChart()
+void addChart(QChartView* chart)
+void addTabBarLine(QString type, QString data = "", QString station = "", QString startDate = "", QString endDate = "", QString bookmarkName = "")
+void requestCustomContextMenu(QChartView* chart, QPoint pos, QSize size
+void addChartButton(QChartView* chart)
+void showError(QString msg, QString extraData = "Unknown error")
+void showBookmarkWidget(bool value)
+void showBookmarkError(QString type = "")
+void deleteAllBookmarks()
+void clearButtonEnabled(bool value)
+void clearCharts()
+void clearHistory()
+void setButtonState(QWidget* button, bool value)
+QString getSaveFileLocation(QString title)
+void hideChartButton(bool value)
+void addChartButtonClicked()
+void settingsButtonToggled(bool value)
+void dateBeginEditingFinished()
+void leftArrowClicked()
+void rightArrowClicked()
+void tabChanged(int index)
+void deleteChart(QWidget* widget)
+void saveButtonClicked()
+void errorTimeout()
+void timeout(bool manual = false)
-void deleteHistoryLines()
-void openChartInNewWindow(QChartView* chart)
-void resizeEvent(QResizeEvent* event)
-void hideChartPopup(QChartView* chart)
controller
+void fetchData(MainWindow* window, QDate from, QDate to, QStringList gasses, QStringList stations = {})
+void createChart(QMap<QString, QStringList > data, MainWindow* window)
+QString formatISO8601(QDate time)
+QVector<QDateTime> formatForChart(QStringList times)
+void loadData(QString type, MainWindow *window)
+void clearHistory()
+void clearBookmarks(QString name = "")
+void saveBookmark(QString name)
+void loadBookmark(QString name)
+void clearCurrentSelections(int i = -1)
+void addCurrentSelection(QString data, QString station, QString startDate, QString endDate)
+void writeErrorLog(QString msg)
+void downloadChart(QWidget* chart, QString title)
+void clearButtonClicked()
+void bookmarkDeleteButtonClicked(QWidget* widget)
+void bookmarkClearButtonClicked()
-void saveHistory(QDate from, QDate to, QStringList gasses, QStringList stations, MainWindow *window)
-bool checkBookmarkName(QString name)
graph
+void test()
+void drawChart(QString chartTitle, QString yaxisLabel, QVector<double> data, class MainWindow* window, QVector<QDateTime> times = {}, QVector<int> years = {})
filehandler
+void save(QString location, QString data)
+QStringList load(QString location)
+void clearFile(QString location)
+void removeLine(QString deleteline, QString location)
+void saveImage(QWidget *widget, QString fileLocation)
apifetch
+void fetchSmearData(QStringList stations, QStringList gasses,QString from, QString to, MainWindow* window, QString aggregation = "MEDIAN", int interval = 60)
+void fetchStatfiData(QStringList gasses, int from, int to, MainWindow* window)
-void handleReply(QNetworkReply* reply)
-void smearDataHandler(QJsonArray columns, QJsonArray data)
-void statfiDataHandler(QStringList gasses, QStringList years, QJsonArray data)