18.4.2021

Weather_power_app

COMP.SE.110 Project work

Teemu Mökkönen, Jaakko Raina, Eero Eriksson, Miska Romppainen

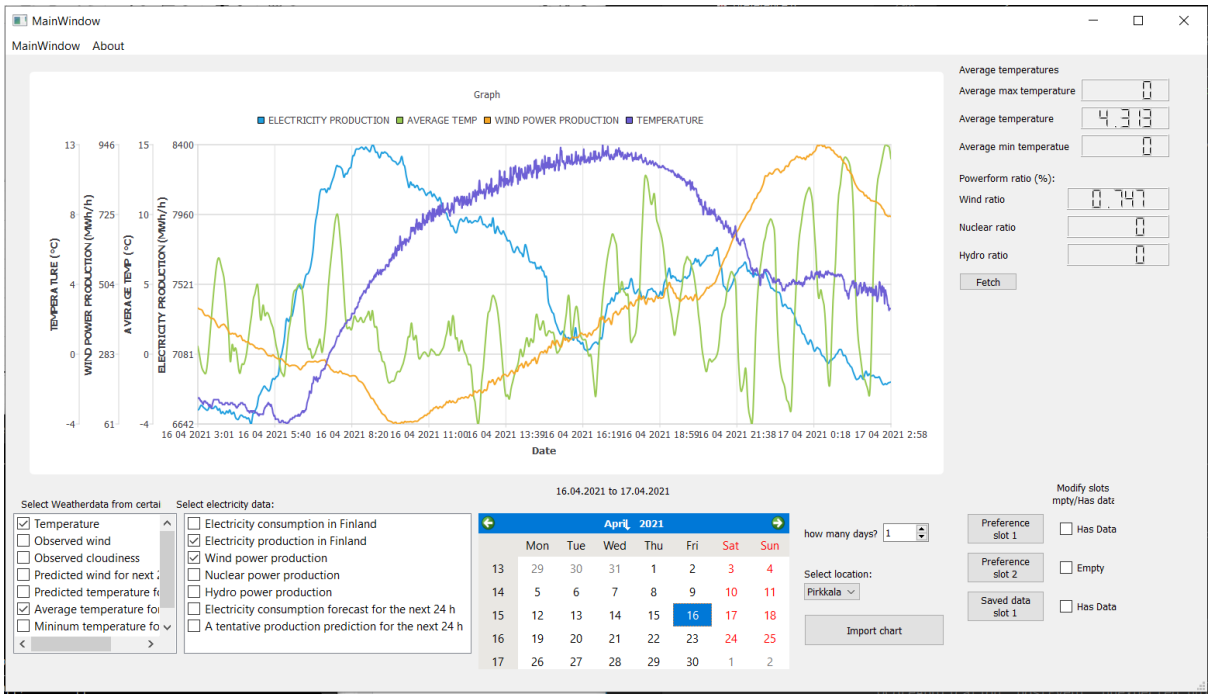
Table of contents

1. Introduction

This document is created in order to give the user specific instructions of how to use the program properly and present methods used to create the software – both the front- and the backend. General idea of the program is to monitor electricity consumption and compare it to the given weather forecast. This is done by fetching power and weather data from an API, presenting it to the user in an interface. User can choose to compare different data from multiple to a specific date – this is completely up-to the end-user. User can create graphs from chosen data. At this point of the development, we are not sincerely certain what we want to include to the program and what we want to present the user, but we are certain we want to present graphs for end-user from data given.

1.1. High level description of the application

Our application will monitor electricity consumption and compare it to weather forecast. Power data is fetched from Fingrid data API and weather data is fetched from FMI open data interfaces. This is done by creating the program a *Logic*-class for the application in which all the calculations and data fetching is done. By using &QNetworkAccessManager we allow the application to send requests and receive replies. This data is then presented to the user in the UI (user-interface) main window.



Picture 1: User interface 1.0

Our UI is demonstrated in picture 1. Upper side of the interface is dedicated for the presented graph. Middle lower side includes calendar from which the user can choose the date to fetch data from. Right side of the calendar user can choose how many days of data they need. When user hits a date all the data is shown as a graph. LCDnumbers in the right hand side of the window shows average temperatures of the selected calendar month and power form ratios. Below them there is saving opinions to save preferences and forecast data.

We took this approach to the user-interface, so it is very self-explanatory, easy to access and easy to learn but same time application should give user free hands to combine different kinds of data and

different settings. Final UI is well balanced combination of user-friendliness and a wide range of different choices that user could make.

1.2. UI usage tips

Our first iteration provides all functionalities, but some of them are little bit complicate to use.

1. Introduction

This app is a simple graph app which get data from fmi-api and fingrid api and makes graphs from the

fetched data. This program can fetch multiple data from the apis at a time, but it is recommended that user selects only two

data sets at a time. Otherwise the app will get quite slow and the graphs will be crowded.

2. How to use this app

First user selects which data sets are wanted to be fetched.

User selects second from which place data is wanted(standard is pirkkala).

After data sets and place has been chosen the user selects the duration from which the data is wanted by selecting how many days is wanted to be proceeded from the date that will be chosen from the calender widget.

(NOTE 1: Weather data can only be for a period of 7 days max. This is due to fmi-api.)

(NOTE 2: Some places may not have all data sets available to the them).

(NOTE 3: When trying to fetch large amount of data (e.g 100m days) it is recommended that only one data set is selected. Otherwise the program can be quite slow.)

Some data sets will not be allowed to be called at the same time such as forecast data and observation data.

3. Saving preference settings:

-Check check box (right hand side of the preference button)

-Check all wanted weather data and electricity data checkboxes active

-Click preference slot

-Uncheck check box (right hand side of the preference button)

-Now your preferences are saved and clicking preference slot will make all saved boxes active

4. Saving data settings:

-Check check box (right hand side of the saved data button)

-Check either predicted wind or predicted temperature checkbox

-Fetch data by clicking the calendar

-Click saved data slot button

-Uncheck check box (right hand side of the saved data button)

-Now your data is saved and clicking saved data slot will draw the data

5. LCD numbers:

To update LCD numbers you must click fetch button every time when new data is drawn into calendar. If user does not click fetch button LCD numbers might show incorrect numbers.

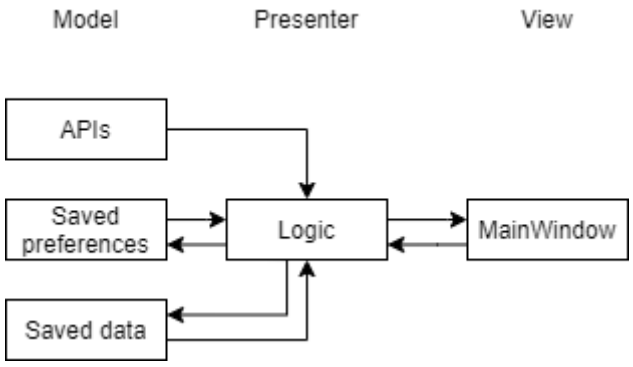
6. Nice to know:

-Clicking calendar when no check boxes active the graph window will be cleared

-If user make mistake during saving preferences or data, the file could be corrupted, and user must clear the whole file manually. Text files “debugpreferences” and “debugsaves” are located in build folder.

1.3. Components

We have split the program into smaller, “easier-to-handle" components. Our design pattern was MVC. The logical function in the program is the logic presenter itself. We use the Logic-file to create the graph from the fetched data points and to fetch the data. The UI is handled by the MainWindow component, all the visual and user-accessible data is handled and presented in the MainWindow.

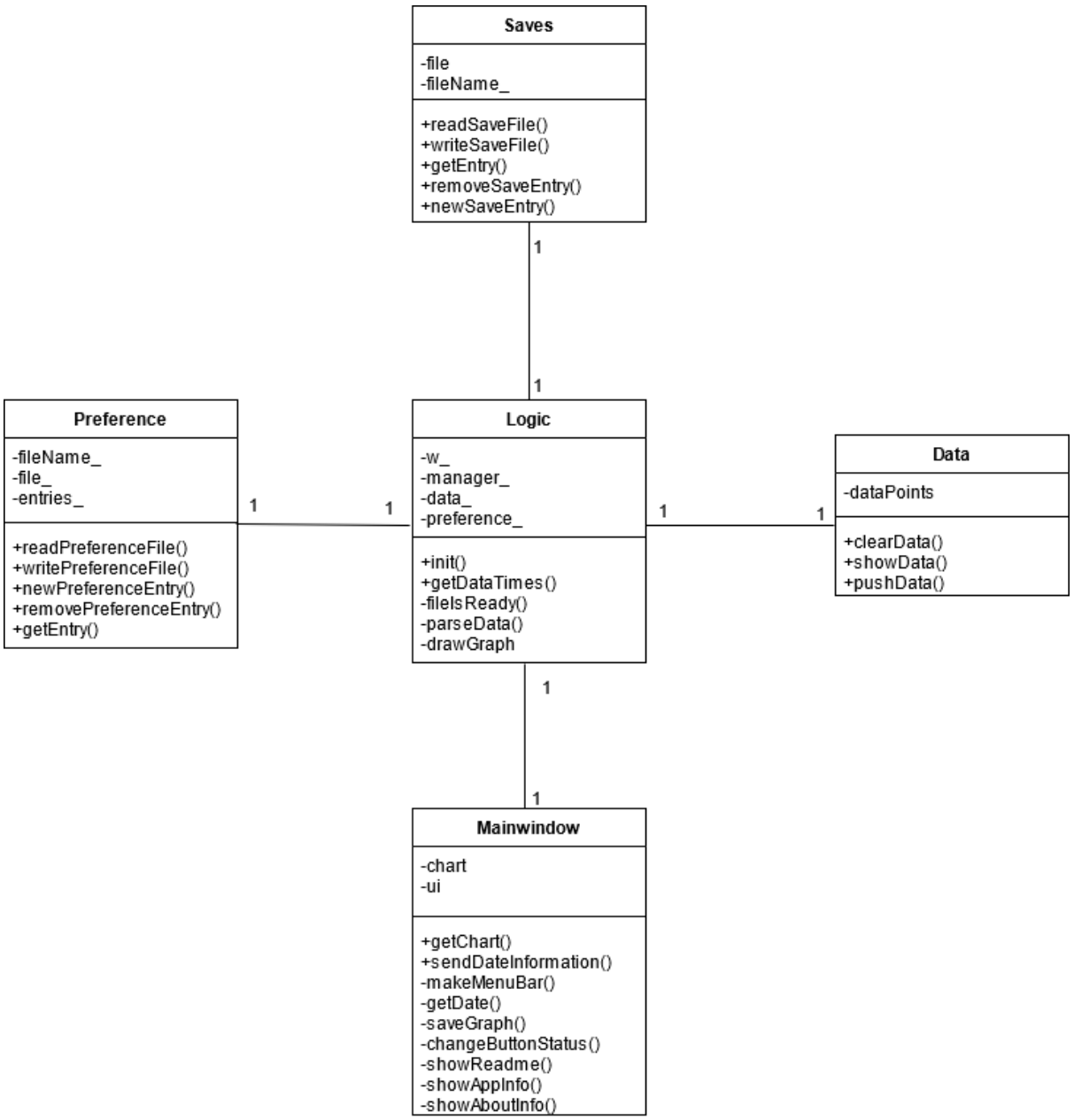


Picture 2: Class diagram 1.0

The class diagram in the picture 2 presents how the data is handled and how the data flows in the application. The Logic is the presenter itself. All the data flows through logic. Logic then converts the data in the desired form of output and passes the data onwards to the proper component. All visual - “viewed” - data is presented in the MainWindow.

1.4. Boundaries and interfaces

Our class provides following public interfaces. In picture 3 public interfaces can be found after + sign in the lowest boxes.



Picture 3: All public interfaces and their dependencies

MVC design pattern gave our program easy to understand structure. All our public functions, slots and signals are listed in the picture 3. Functions are easy to understand, and their dependencies are clear.

2. Modules and classes (Design decisions)

The modules we currently use in the project are Qt Core, Qt Charts, Qt Network and Qt Widget.

-widgets
QFlieDialog
QCalenderWidget

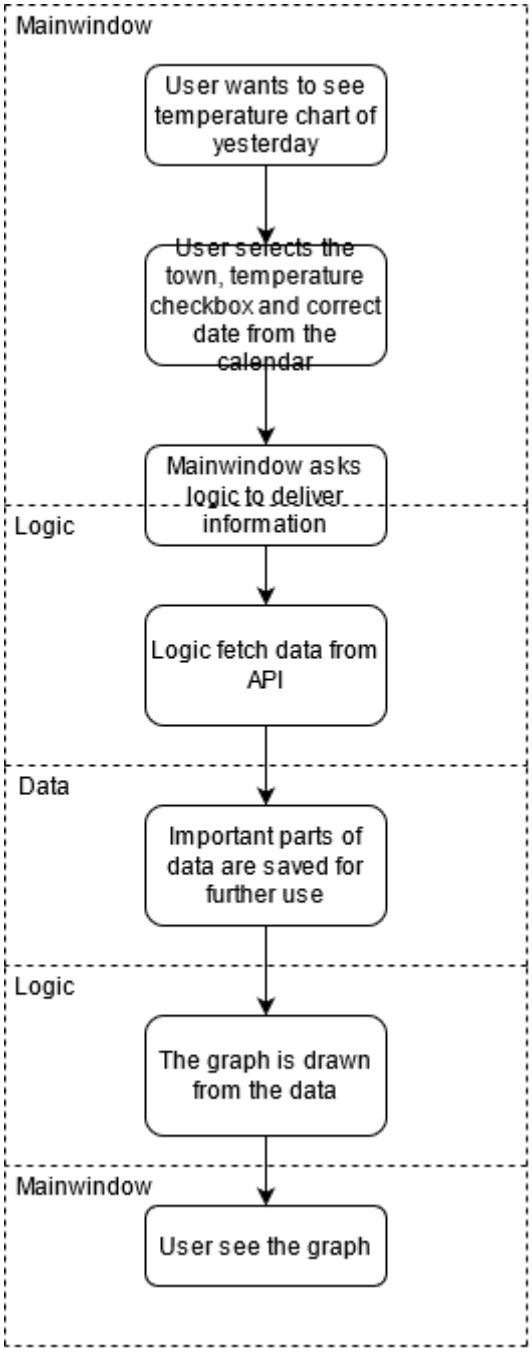
QMainWindow
QListWidget
QMessageBox
QFileDialog

-core
QTemporaryFile
QXmlStreamReader
QFile
QDir
QString
QTextStream

-network
QNetworkReply
QNetworkAccessManager

-QtCharts
QLineSeries

Used modules and classes are presented in above. Currently QNetworkReply and QNetworkAccessManager are some of the classes used from Qt Network – Qt Network module is implemented so we can access the API. We use Qt Widget in order to create the main window of the UI and all its widgets - including the Calendar widget. The fetched data from the API is handled by the QXmlStreamReader from Qt Core. We use Qt Core also in order to write an output file for the user, this is done with QFile class.



Picture 4: Activity diagram of drawing temperature chart

Picture 4 shows activity diagram of drawing temperature data to the user. As shown in the picture 4, MVC design pattern divides the activity diagram to clear parts. The classes communicate with each other in logical order.

3. Bonus requirements

We implemented visualization saving option as images. In picture 1 right side of the calendar is “Import chart” button and pressing the button saves a picture of the graph to the computer. Our program will ask which name user wants to give to the picture.

4. Self-evaluation

Our original design was fairly accurate and easy to follow. The original design was specifically designed in a way that different parts of the application could be changed without changing the other parts. Most of our current implementation are according to the original design of the application. In some places small changes have been made to things like STL containers to make individual parts work but this hasn't impacted the general design of the application.

There has been one structural change since we started the project. Change is in the way we are save the preferences the user sets. Originally it was supposed to be saved in a JSON file but that has been changed to a normal text file because of technical difficulties. Our original plan and design pattern were so easy to follow that we did not need to make any significant changes. Those factors made our project easy to develop, and allocating work was easybeasy.