Weatherize

Design document

Etkot software

Table of Contents

Design document version history

Version 0.1	Create the document	19.2.2021
Version 0.2	Reformatting	21.3.2021
Version 0.3	Add chapters: Design, High level description, Boundaries and interfaces & Self-evaluation	21.3.2021
Version 0.4	Updated design decisions, high level description, boundaries and interfaces, and self evaluation	18.4.2021

Introduction

This product will be a desktop application that will graph and compare weather data such as temperature with electricity consumption and production from different sources. The user can specify the time period they want to observe.

The data for power consumption and production will be from Fingrid and the weather data will be from the Finnish Meteorological Institute.

We will be writing the application using C++ and using Qt for the user interface and other things such as graphing the data onto the application screen.

Components and interfaces

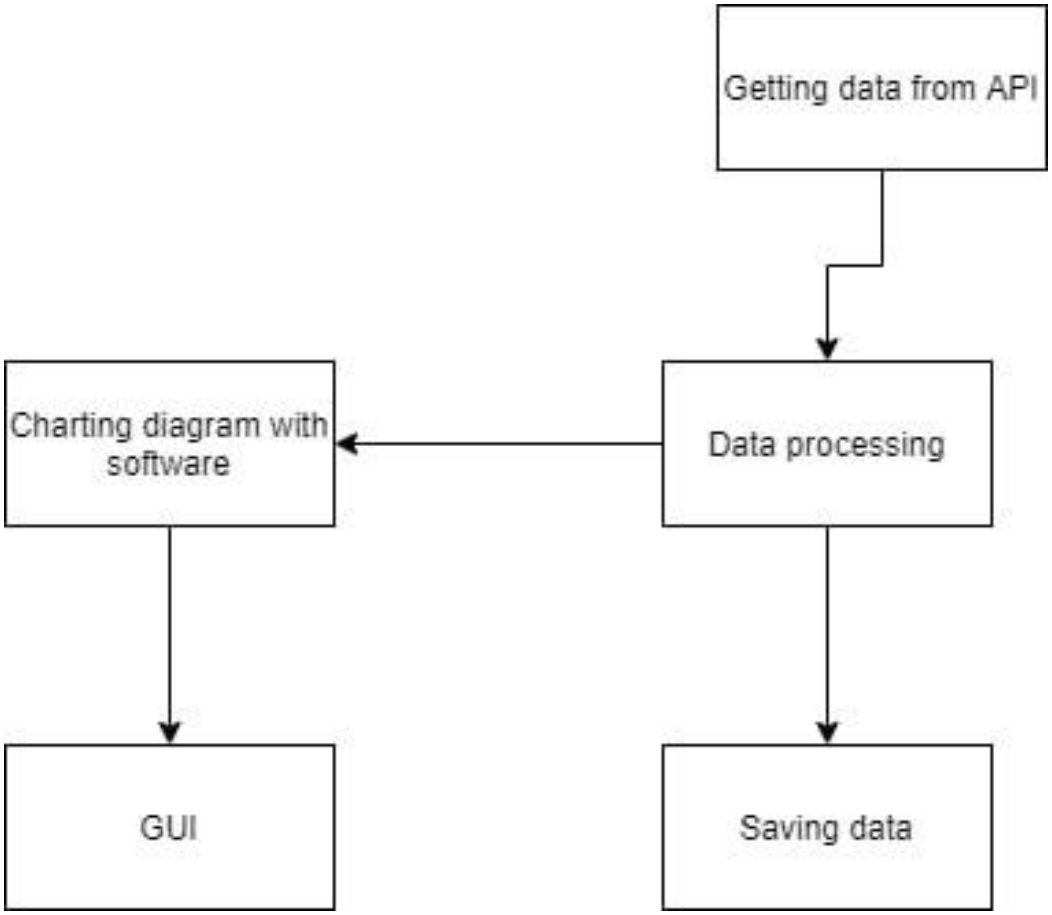

Figure 1: Main components

The component for getting API data simply deals with getting the data and passing it forward. It is then processed and filtered for the relevant data by the processing component and sent forward to the charting component and the data saving component if the user wants to save the data.

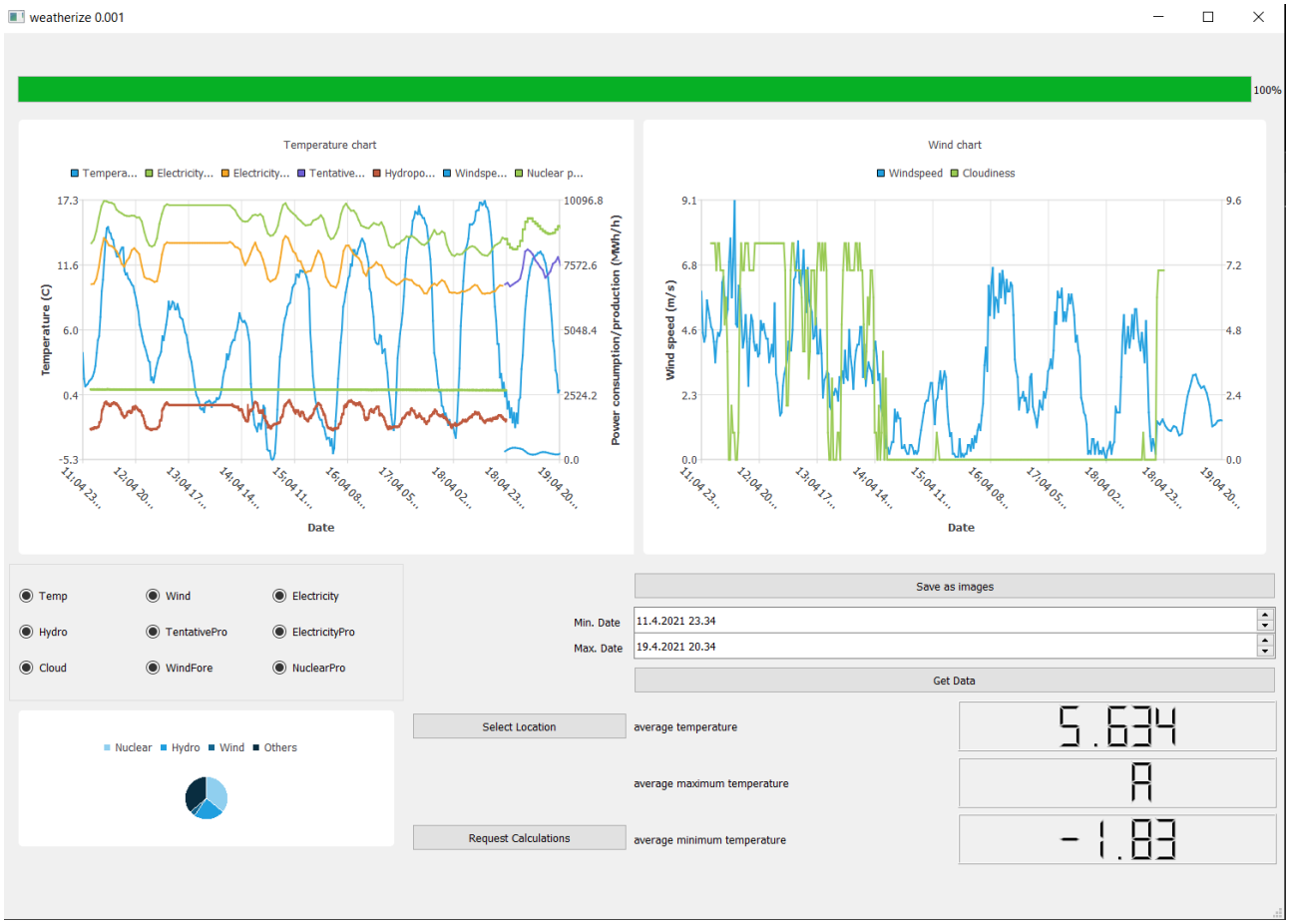

Figure 2: Screenshot of the program

The completed QtGraphs will be sent to the component responsible for the user interface and will be displayed to the user there.

Design decisions

The project uses the MVC pattern for separating the UI from the “back end” of the software. We have worked on implementing mainly the SOLID principles in the program as guideline. The single-responsibility principle was our main target and all except mainwindow were done using this principle. The open-closed principle was also used when fetching data. The Interface segregation principle was used in mainwindow and in locationdialog, because one button did one thing and one thing only. This approach felt the most natural to us so we decided to go with it.

High Level Description

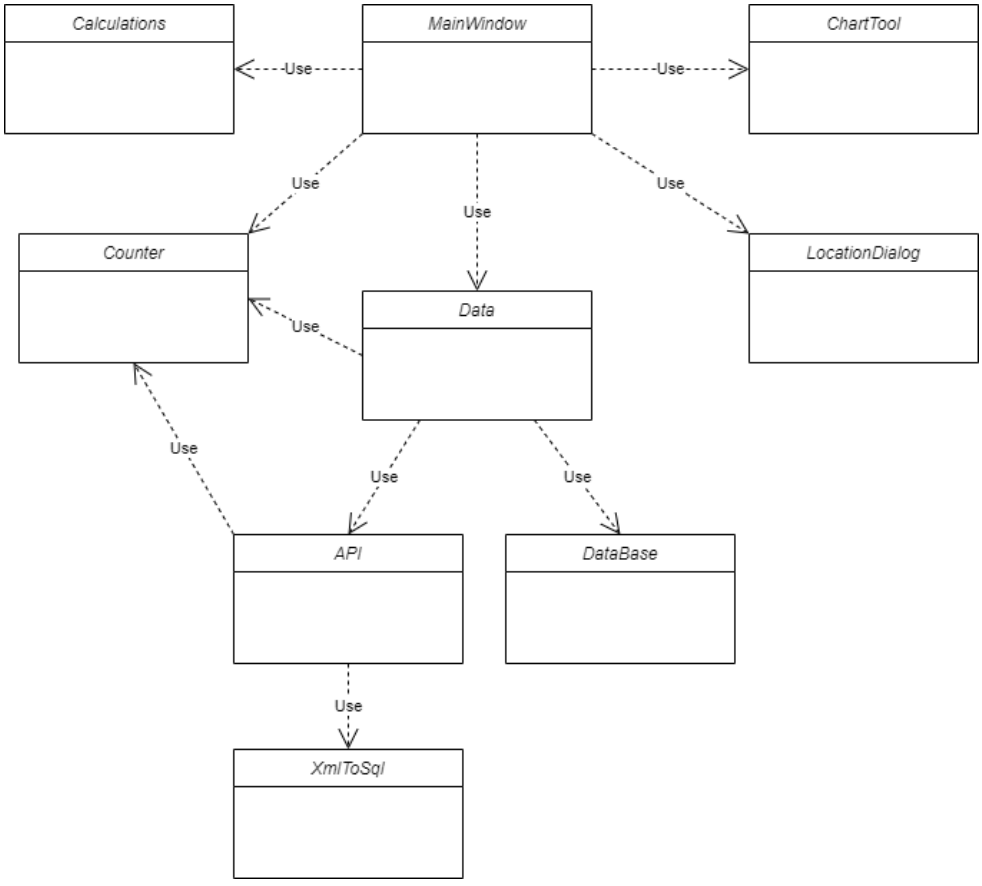

Figure 3: The class diagram

The classes can be divided into five categories: the GUI, fetching weather and electricity data from their sources, processing and charting said data, and saving the data to a local database.

The GUI has two charts for electricity and weather data and a pie chart to visualize the percentages of each source of electricity. The user is able to toggle charts that should appear on the screen with the radio buttons on the bottom left of the window. Beginning and end dates, as well as the geographical location of data can also be changed in the main window. A screenshot button saves the charts as PNG files. A Counter class updates the progress bar in the main window.

Feather and electricity data fetching is done by the API class. The Data class stores every data category and DataBase handles saving everything to an SQL database.

Data processing is done in Calculations, MainWindow, and XmlToSql. Calculations calculates the percentages of each energy source, and MainWindow selects the right data to chart from the database. XmlToSql parses the fetched data and converts it to SQL format.

The ChartTool class is used to visualize the data in the main window.

Database is implemented with SQLite. In database there is 2 tables, which are weather and electricity and in each primary key is time. Varchar have been changed to int.

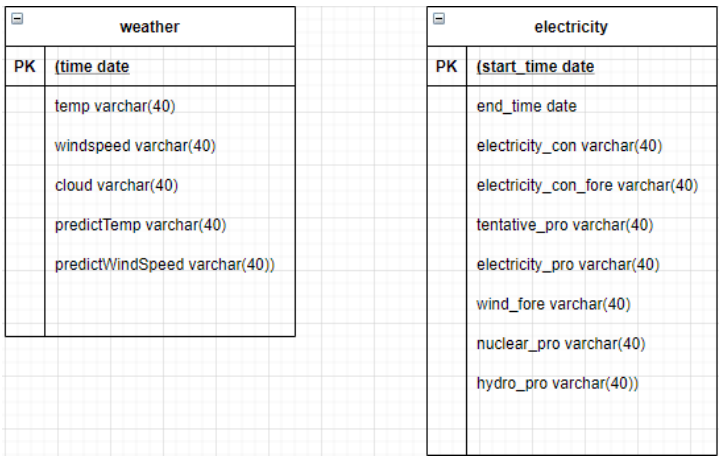

Figure 4: Tables in the database

Boundaries and interfaces

Components as of now:

- Charttool
  - Takes in data as a vector of pair of (int, QDateTime).
  - Has interface for the processed data, currently given to it by the main window.
  - Takes charge of making charts.
- PieChart
  - Takes data as double from MainWindow and data is calculated in Calculator.
  - Return piechart.
- Calculations
  - Takes sql query for data.
  - Returns temp averages for piechart and lcdnumbers.
- Counter
  - Updates progressbar.
- Data
  - Handles starting of getting new data and creates database.
- DataBase
  - Creates sql database and sqlquery object.
- API
  - Is next data from data to get new data.
  - Makes calls over internet and gets xml files.
- XmlToSql
  - Converts from xml to sql.
- Locationdialog
  - Handles of changing place.
- Mainwindow
  - LoadCharts first updates date, time and then fetches the data for weather and power then creates charttool and charts data into graphs using charttools functionalities.
  - UpdateDateTime scales the date of the graph (for now its static 8 days)
  - UpdateAPI calls for datatools fetch updated data so it can be used to create graph.

Self-evaluation

Now that we have fully functional program it is to be said that the original quite lose plans were good and they gave us good working space with development, but we also managed to implement most of the early functionalities and even functions till the end. We were quite busy in our other studies and final week was the time when we worked real hard to create well working program and we are proud of our finished product. We were able to stick to our original design decisions and add buttons, diagrams and widgets and still the old design of the program basically intact.

We had to make some changes and most of them were targeting at making more classes and dividing old ones into many different classes. Only classes that stayed almost fully same were basically charttool and mainwindow. Datatool was divided into 5 different classes that work well together but they still do basically the same thing as the one class earlier, but 5 different classes functionalities are easier to understand which is great.