SAAGIA Design Document

Group: Software-Crusaders

Team members: Linnea Viitanen, Sami Naarminen, Niilo Rannikko and Mikko Tuovinen

Introduction

Saagia is a Qt Creator based QML / C++ application for visualizing weather and electricity data from Fingrid and Finnish Meteorological Institute. The app gathers information from the sites and combines them in various ways into visualized form and presents it to the user. App's design follows MVC architecture style, so the view and controller are separated making the app's structure clearer and easily expandable.

Work process

The development team has been actively working on the project right from the get-go. The team has arranged meetings a few days apart from each other in order to continuously divide work to the members, view progress, to plan for the development of the project and to ensure that progress is made. The team has been using Trello to aid in the project and to make notes of the meetings and development process. Work has been divided between all group members and every member has studied the principles of major functions of the software.

File structure and programming setup

Project's file structure includes following files:

C++

main	Starts the application and sets up the MVC classes
data_reader	Contains the functions to read data from Fingrid and FMI
saagia_controller	The main controller class for the user interface
saagia_model	Model class for the application, acts as the main backend logic for the app
saagia_view	View class that communicates the values to the QML side
database_handler	Data class for writing and saving data to the database and storing it in a file
data_calculations	Contains the functions to calculate given data and return calculated data

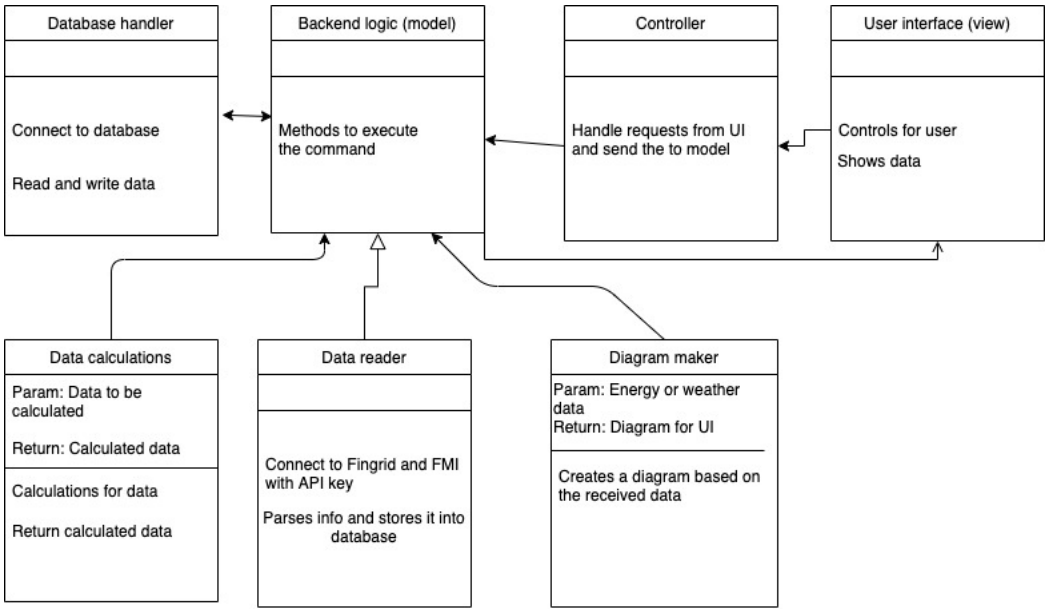
QML

Energy_type_button	Contains the information to form buttons for the energy forms in the UI. In the future, user will be able to press the button and make a specific energy type visible/invisible in the displayed chart
Fetch_data_button	Button to retrieve new datasets, used in UI
Load_data_button	Obsolete button
Save_data_button	Obsolete button
Text_input_bar	Text input bar used to search different cities/areas to show
Date_input	Used in popup. Input, that opens the calendar, from which the user can choose a starting or ending date of the dataset
Time_input	Used in popup. Input, that is used to select a specific starting and ending time of the dataset
Chart_base_test	Contains the information and functions to form a chart from given data
main	Contains the main user interface and position of the elements
New_data_load_popup	Popup window that displays date and time pick for a new dataset to be retrieved
Calendar_model	Contains the calendar that can be opened from the popup
Checkbox_column	Used in popup. Column of energy type checkboxes, from which the user can select which energy type dataset they want to retrieve
Currently_showing_box	Information box in the UI, that tells the user what data is currently being shown in the chart, as well as the date and time range selected
Energy_type_button_row	Row that hosts multiple energy_type_button objects
Output_area	Debug console in the UI for development purposes only
Title_text	Title of the software being displayed as a logo
Window_bar_menu	Obsolete file that will be used later

We decided to program the app with Qt Creator as it was the one all group members were most familiar with. We used Qt Creator version 5.12.2. Qt Charts was also required in order to get the diagrams working properly. At the moment no third-party components are present in the design.

Program architecture

As said in the beginning, the team chose MVC architecture for the application for its simple and easy way of separating view and controls from each other. This helps development in the future and makes the class structure clearer.



Picture 1: UML Diagram of project Saagia

Diagram describes the relations between classes and the roles of each class. Diagram also shows the flow of information in the class structure. The app is built with push-style MVC architecture.

A preliminary version of the program structure is presented in the picture above. Our program is divided into three parts: the view (UI), controller and model (backend logic). In our program, user created command is conveyed through the UI to the controller which then processes the command and sends it to backend logic where the command is executed. In the backend our logic-class works through the command with the help of Data reader, Database handler, Data calculations and Diagram maker and sends the output to the UI for presenting. At this stage of the project the arrangement is subject to change when the program takes shape later on.

Class responsibilities

The purposes and responsibilities of the program's classes are as follows:

UI / QML side	- Takes input from the user and presents wanted diagrams and results - Communicates user created commands to Saagia_controller - Presents the info and diagrams sent by Saagia_model through Saagia_view
Saagia_controller	- Receives the user input from UI / QML side, processes it and sends it to Saagia_model
Saagia_model	- Commands other backend classes - Executes the command given by user with the help of other classes - Orders the data collection, processing, calculations and the making of diagrams and charts - Sends final results and visualizations to UI / QML side through Saagia_view
Saagia_view	- Receives data from Saagia_model and sends it to UI / QML side
Data_reader	- Sends request and receives data from Fingrid and the Finnish Meteorological Institute - Parses the data and stores it in the database
Database_handler	- Searches the database for wanted information - Saves and erases data to/from the database - Reads and writes data to/from a file
Data_calculations	- To be implemented. Performs all the data calculations of the chart.
Diagram_maker	- Constructs diagrams and charts

Reasoning for the design decisions

We chose the MVC style of program architecture since it was prominently presented in the course lectures and seemed appropriate for our program. In the backend we have one class (Saagia_model) that works through the command by commanding the other classes to fetch the data, store it and process it in order to produce wanted output. Saagia_model executes command by sending these requests one-by-one to the utility classes and finally sends the results to Saagia_view which updates the UI / QML side.

Data_reader, Database_handler, Data_calculations and Diagram_maker do not interact with each other. We have divided the backend into these classes as we see them easily separated, independent

functions though we fear, at this point, that Saagia_model class may grow very large when the project advances, so we are prepared to create new classes to share the workload if needed.

Our user interface is designed to be passive and to only show information it is commanded to by the model.

Self-evaluation

The design of the software described in the first phase of the course has stayed the same. The user interface is going to be the same, only with small adjustments. The MVC model that the team decided to implement to the project is now active and is in no need of changing.

New design implemented in the mid-term phase was the popup window for the search options. This provides the user with the tools to give date and time options for the application to search data.

One ongoing change of structure compared to the prototype is that at first parsing the information from the external APIs was done in the view, in QML files. Now, separate parsers have been made into Data_reader for both XML (FMI) and JSON (Fingrid), which send the parsed information to Saagia_model and from there to Saagia_view.

In our first design the class Database_handler is used for reading and writing data into a separate database and files to store to the computer, but in the final program this might be done with a function in Saagia_model and not in a class of its own, depending on how many lines of code it requires.