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.

Functions

The basic function of the application is to fetch data from given sites and show the fetched data in a visual form. The data contains information about electricity consumption and production, weather conditions, including cloudiness, temperature and wind speeds. Electricity production can be divided into different production types, like nuclear power.

Saagia has a location function implemented which it uses to fetch local weather data of selected location. List of locations is confirmed to work with Finnish Meteorological Institute, and Saagia will fetch legit weather data with those locations.

Data search popup contains radio buttons for the data types user wants to search for, and date menu to choose a timeline. After selection, user presses data fetch button and the data is inserted in the app's database. After the search user can draw the last searched data using the provided button.

Last fetched data can be saved to a file using the "Save data" button. The user is then asked for a text file for the app to save data to. Saved dataset can also be loaded from a text file using the "load data" button.

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 the following files:

C++

main	Starts the application and sets up the MVC classes
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
saagia_controller	The main controller class for the user interface
data_structures	Database class that handles the storing of data properly for different types of data
data_reader	Contains the functions to read data from Fingrid and FMI
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 the calculated data

QML

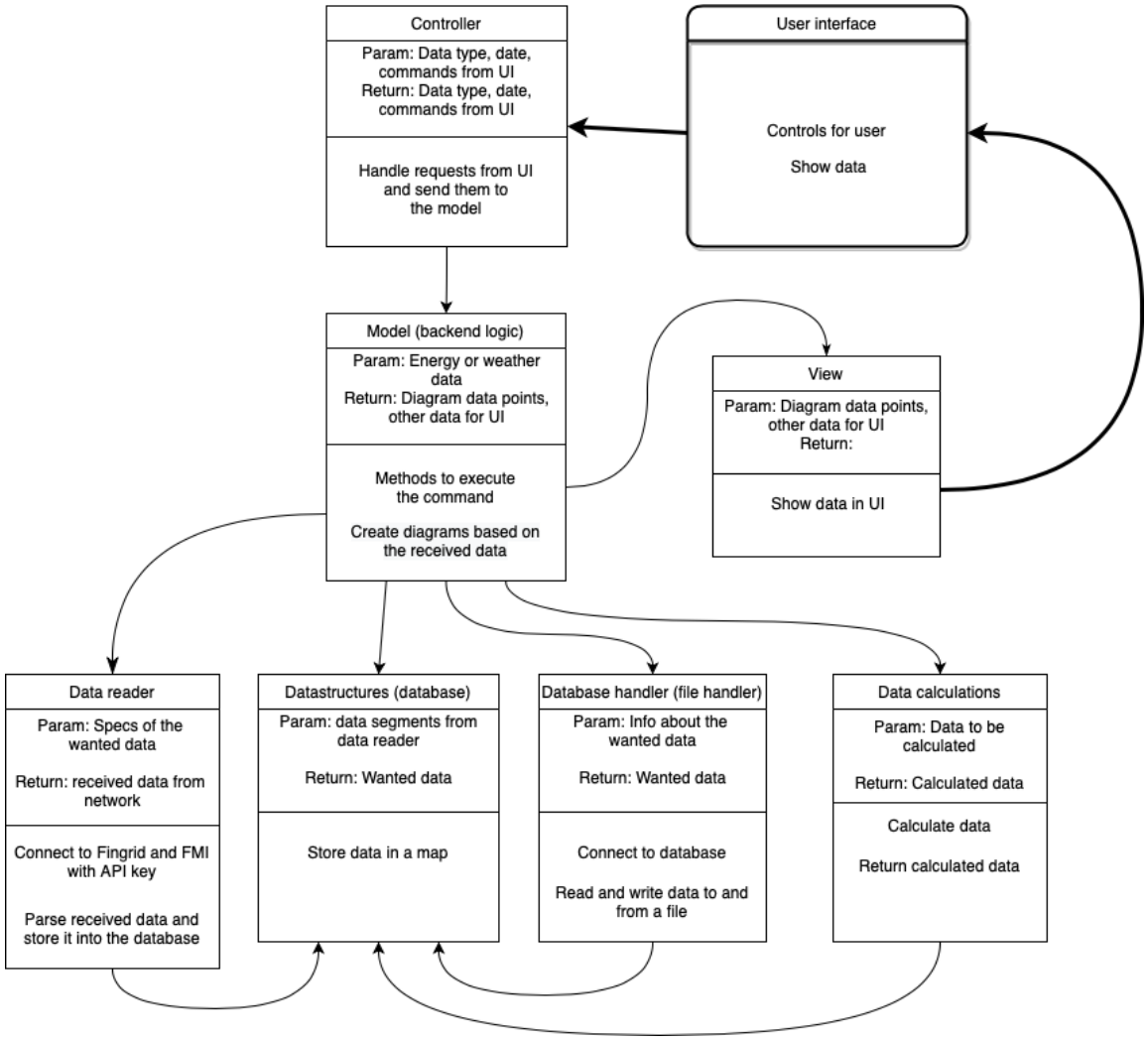
Energy_type_button	Contains the information to form buttons for the energy forms in the UI. In the future, user will be able to press the button and make a specific energy type visible/invisible in the displayed chart
Fetch_data_button	Button to retrieve new datasets, used in UI
Save_data_button	Loads file dialog to save retrieved dataset to file
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
Energy_chart_base	Contains the information and functions to form an energy 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
Radio_buttons	Contains all the radiobuttons for the popup window
Save_file_popup	Popup file for the user to insert data type and dates
Save_file_dialog	File dialog that allows the user to save a file
Title_text	Title of the software being displayed as a logo

Wind_chart_base	Contains the information and functions to form a weather temperature chart from given data
Window_bar_menu	Stores the buttons for saving and loading data structures from file

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.

Program's 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 saagia_model class works through the command with the help of Datastructures, Data reader, Database handler and Data calculations and sends the output to the UI for presenting. In the final stages of development, the structure has stayed almost the same and follows the MVC architecture quite accurately. During the last week we implemented a separate data structure class, Datastructures, to handle the storing of different data types properly.

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 diagram data 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 the 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_structures	- Receives data and stores it in a proper way for each type of data
Data_reader	- Sends request and receives data from Fingrid and the Finnish Meteorological Institute - Parses the data and stores it in the data structure
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	- Performs data calculations for the average temperature of the month and the percentages of energy production

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 and Data_calculations do not interact with each other directly. 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 store the parsed information to Datastructures from where Saagia_model can obtain it and send it to Saagia_view.

In our first design we decided that 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.

The final design also includes Datastructures for handling different types of data and a pointer to it is created in Saagia_model and shared to Data_reader, Database_handler and Data_calculations so that they can access the data when needed.

Known issues

Despite our hardest efforts the program did not turn out perfect or precisely like the group assignment specifications said. The main issue with the project was not doing enough work when we had more time. Our group started the project very well and early on we made great progress, still during the last week before submission the task list grew very large and issues kept coming up that hindered our work.

Some of the functions given in the requirements are partly implemented, but not fully functional in the UI. These include, for example, calculations of the average temperature and different types of energy production calculated.