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 Meterological 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 the project is moving forward. The team has been using Trello to aid in the project and to make notes of the meetings and development process as well as Telegram group chat for communication. Work has been evenly divided between all group members and every member has studied the principles of major functions of the software.

File structure and programming setup

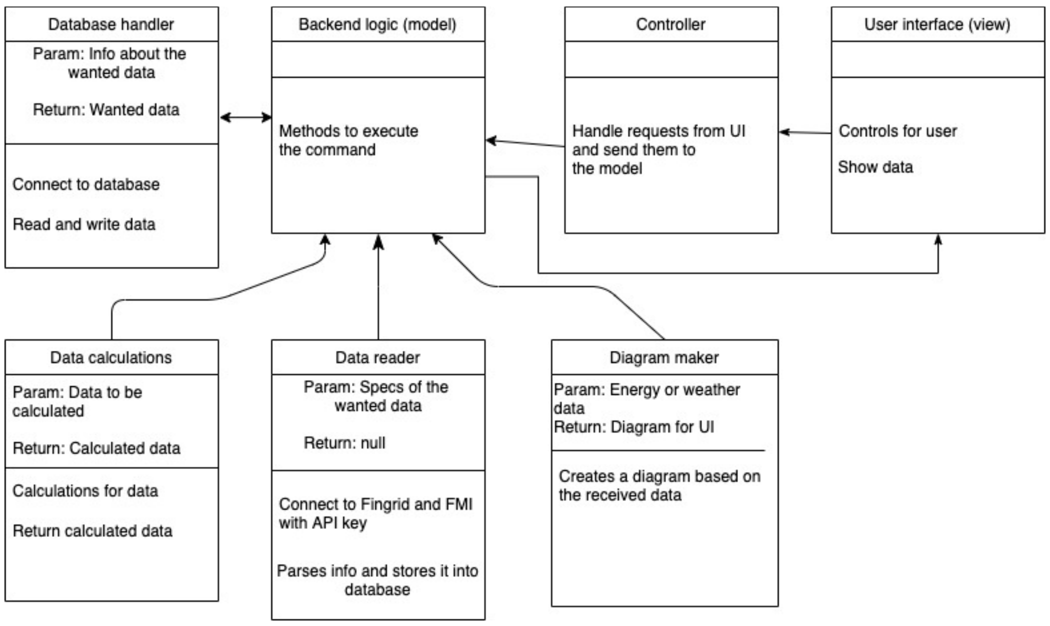
Project's file structure includes following files:

- main: Starts the application and sets up the controller classes
- main.qml: Contains the main user interface
- Data_calculations: Makes needed calculations for data to be used
- Datareader: Gets data from FMI and Fingrid APIs and saves them onto database
- Database_handler: Reads and writes data from/to into database
- 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

We decided to program the app with Qt Creator as it was the one developing environment all group members were most familiar with. We used Qt Creator version 5.12.2. QtCharts 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.


Database handler
Param: Info about the wanted data
Return: Wanted data
Connect to database
Read and write data
Backend logic (model)
Methods to execute the command
Controller
Handle requests from UI and send them to the model
User interface (view)
Controls for user
Show data
Data calculations
Param: Data to be calculated
Return: Calculated data
Calculations for data
Return calculated data
Data reader
Param: Specs of the wanted data
Return: null
Connect to Fingrid and FMI with API key
Parses info and stores it into database
Diagram maker
Param: Energy or weather data
Return: Diagram for UI
Creates a diagram based on the received data

Picture 1: QML-Diagram of project Saagia

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 presentation. At this stage of the project the final number and structure of classes might change when the program takes shape later on, if its seen necessary.

Class responsibilities

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

- User interface:
  - Takes input from the user and presents wanted diagrams and results
  - Communicates user created commands to controller
  - Presents the info and diagrams sent by model
- Controller
  - Receives the user input, processes it and sends it to backend
- Backend logic
  - 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 the user interface
- 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
    - Writes and reads data from to/from database
- Data calculations:
    - Does the math needed for later use
- 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 (Logic) 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. Logic executes command by sending these requests one-by-one to the utility classes and finally sends the results to ui. Database_calculations, datareader, -handler, -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 the logic 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.