**PROJECT NORDICWATT**

**DESIGN DOCUMENT**

COMP.SE.110 Software Desing – Group Assignment

Antti Hakkarainen  
Markus Hissa  
Heikki Hohtari  
Janne Taskinen

Table of Contents

[1 Introduction 3](#_Toc146375906)

[2 Requirements 4](#_Toc146375907)

[2.1 Functional requirements 4](#_Toc146375908)

[2.2 Non-Functional requirements 4](#_Toc146375909)

[2.3 Chart Types 5](#_Toc146375910)

[2.3.1 Line diagram 5](#_Toc146375911)

[2.3.2 Area Chart 5](#_Toc146375912)

[2.3.3 Scatter Dot Chart 6](#_Toc146375913)

[2.3.4 correlation heatmap (ei ole ainakaan protossa vielä toteutettuna) 6](#_Toc146375914)

[2.3.5 Other charts (extra) 6](#_Toc146375915)

[2.4 Available datatypes per chart 6](#_Toc146375916)

[2.5 Example UI 8](#_Toc146375917)

[3 API’s 9](#_Toc146375918)

[3.1 Fingrid 9](#_Toc146375919)

[3.2 Finnish Meteorogical Institute 9](#_Toc146375920)

[3.3 porssisahko.net (extra) 9](#_Toc146375921)

[4 Architecture and design 10](#_Toc146375922)

[4.1 Activity diagrams of high-level use cases 10](#_Toc146375923)

[4.1.1 Program start-up 10](#_Toc146375924)

[4.1.2 Validation of user inputs 11](#_Toc146375925)

[4.1.3 Diagram creation process 13](#_Toc146375926)

[4.2 Diagram of components within the system 14](#_Toc146375927)

[4.3 Design patterns and decisions 14](#_Toc146375928)

[5 Attachments 16](#_Toc146375929)

[6 References 17](#_Toc146375930)

# Introduction

Our application serves as a comprehensive weather and energy tracking platform for desktop environments. The software aggregates information from a minimum of two distinct APIs, displaying the data in a user-friendly format suitable for analysis.

Specifically, users have the capability to review historical as well as forecasted weather data for regions in Finland. The weather metrics provided include temperature, wind speed, and rainfall. Additionally, the application monitors real-time and forecasted energy production and consumption rates within Finland.

As a future enhancement, if time permits, we also plan to integrate Finnish electricity stock market prices into the application.

The concept for this application originated from ChatGPT and was chosen from a variety of options as the most compelling and feasible for production. Alternative possibilities were either less intriguing or necessitated the use of paid APIs or non-public data.

We intend to develop this desktop application as a Java Maven project, utilizing the JavaFX library for its graphical interface. Additional libraries will be specified as they are identified and incorporated into the project.

# Requirements

## Functional requirements

Users workflow of the program follows structure:

1. User types chooses the document style (for example line diagram)
2. User chooses values x and y parameter datatypes (for instance temperature and energy consumption)
3. User selects desired date range
4. User presses Create Diagram to create new diagram

|  |  |  |
| --- | --- | --- |
| ID | Description | Criticality  5 = lowest, 1 = highest |
| F1 | User can save required API keys (Fingrids) | 1 |
| F2 | Program fetches specified data from Fingrid | 1 |
| F3 | Program fetches specified data from Finnish Meteorological institute | 1 |
| F4 | User can choose between line, area and scatter dot charts | 1 |
| F5 | User can change the range of the data | 2 |
| F6 | User can select predefined datatypes for x and y axis | 2 |
| F7 | User can generate chart based on search criteria | 1 |
| F8 | User can see detailed information of the datatype selected (unit, description and source) | 4 |
| F8 | User can hover over chart to see individual datapoints and their values | 5 |
| F9 | User can see the datasets average Y axis value | 5 |
| F10 | User can choose to view average horizontal line on the chart on Y axis | 5 |
| F11 | User can choose to view two horizontal lines on upper and lower quartile on Y axis | 5 |
| F12 | User can generate either chart in new tab or the active tab | 4 |
| F13 | User can close a tab | 3 |
| F14 | User can export data visualized as image or pdf | 5 |
| F15 | User can save search terms of their query for later use | 1 |
| F16 | User can apply saved search terms (check F15) | 1 |
| F17 | User can apply time as either specific timerange or “Last n days/weeks/months” | 3 |
| F18 | Program fetches specified data from porssisahko.net | 4 |
| F19 | User must be able to adjust the parameters of the visualization | 1 |
| F20 | User can create multiple diagrams into same view | 2 |
| F21 | User can save and load chart generation settings | 1 |

## Non-Functional requirements

|  |  |  |
| --- | --- | --- |
| ID | Description | Criticality  5 = lowest, 1 = highest |
| N1 | Program should keep the data once fetched so we don’t have to fetch it again | 2 |
| N2 | Data and user defined settings should be stored over shutdown | 2 |
| N3 | API fetch should always result in success (provided service is not down). User should not be able to put search terms that cause invalid or erroneous API call. | 1 |
| N4 | Program must implement error handling class | 2 |
| N5 | Fetching of data should not take over 30 seconds | 2 |
| N6 | User can see animation during data fetching and parsing (to see that something is happening) | 4 |
| N7 | The design must be such that further data sources or additional data from existing sources could be easily added. | 1 |
|  |  |  |

## Chart Types

### Line diagram

In a standard line diagram, the X and Y axes represent chosen data types, such as temperature and energy consumption. Optionally, and depending on development time, a second Y-axis parameter can be added to represent a third variable within the same line diagram.

It's important to note the role of the date range in these diagrams:

1. If the date range is set on the X-axis, it serves to indicate the minimum and maximum values for the X-axis variable.
2. However, if a variable other than the date range is placed on the X-axis, the chosen date range will instead define the time interval for which data is collected and displayed.

The date range's function can either limit the X-axis values or serve as a filter for the dataset, and these are two distinct uses.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Data unit | **Time** | **Consumption** | **Price** | **Temperature** | **Wind** |
| **Time** | I | C | C | C | C |
| **Consumption** | C | I | P | P | P |
| **Price** | C | P | I | P | P |
| **Temperature** | C | P | P | I | P |
| **Wind** | C | P | P | P | I |

Table 1. Example of data combinations on line diagram.

|  |  |  |
| --- | --- | --- |
| Sign | **Compatibility** | **Notes** |
| C | Compatible | No problems to display on basic XY-chart |
| P | Partially compatible | Might need different y-axes due to the difference in scale. |
| I | Incompatible | No sense to combine in same chart. |

Table 2. Legend for table 1.

### Area Chart

Area chart is basically the same as line chart, but it will color the area bellow the line.

### Scatter Dot Chart

On scatter dot chart we can visualize individual datapoints on x y axis. Time will not be available on neither axis, but the data is searched by specific date range. The dots on if there multiple dots on top of each other we can have two options to visualize that there are in fact multiple dots at that location:

* Make the dot bigger, if there are multiple dots or
* put a little variance to throw off the dot from its actual point

### correlation heatmap (ei ole ainakaan protossa vielä toteutettuna)

TODO: Tää kappale vaatii aika paljojn tarkennusta

https://duckduckgo.com/?q=correlation+heatmap&t=vivaldi&iar=images&iax=images&ia=images

The cells contain the correlation coefficient between the two variables represented by the corresponding row and column. This coefficient ranges from -1 to 1.

* 1 indicates a perfect positive correlation
* -1 indicates a perfect negative correlation.
* 0 indicates no correlation.

We could use Pearson correlation coefficient for example: <https://en.wikipedia.org/wiki/Pearson_correlation_coefficient>

Example scenarios:

* Cell at the intersection of "Electricity Price" and "Temperature" contains the correlation coefficient indicating how electricity price relates to temperature.
* Cell at the intersection of "Wind Amount" and "Electricity Consumption" contains the correlation coefficient indicating how the amount of wind correlates with electricity consumption.

Colors:

For example Blue to Red Spectrum: Blue can represent negative correlation, white (or a neutral color like gray) can represent no correlation, and red can represent positive correlation.

### Other charts (extra)

While the initial Minimum Viable Product (MVP) will focus on core functionalities, there are several other chart types that could enhance the application's utility and user experience in future releases.

* radar chart
* Pie chart (how much does wather variable affect price in percentages).
* Price guesstimation
* Caching

## Available datatypes per chart

For line chart and area chart options data options available are as follows:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  | | --- | --- | | A black and white image of a cross  Description automatically generated |  | | **Time** | **Electricity Consumption** | **Electricity Production** | **Hydro Power** | **Nuclear Power** | **Wind Power** | **Temperature** | **Wind** | **Rain** | **Humidity** | **Air Pressure** | **Electricity Stock Price** |
| **Time** |  | yes | yes | yes | yes | yes | yes | yes | yes | yes | yes | yes |
| **Electricity Consumption** |  |  | yes | yes | yes | yes | yes | yes | yes | yes | yes | yes |
| **Electricity Production** |  | yes |  | yes | yes | yes | yes | yes | yes | yes | yes | yes |
| **Hydro Power** |  | yes | yes |  | yes | yes | yes | yes | yes | yes | yes | yes |
| **Nuclear Power** |  | yes | yes | yes |  | yes | yes | yes | yes | yes | yes | yes |
| **Wind Power** |  | yes | yes | yes | yes |  | yes | yes | yes | yes | yes | yes |
| **Temperature** |  | yes | yes | yes | yes | yes |  | yes | yes | yes | yes | yes |
| **Wind** |  | yes | yes | yes | yes | yes | yes |  | yes | yes | yes | yes |
| **Rain** |  | yes | yes | yes | yes | yes | yes | yes |  | yes | yes | yes |
| **Humidity** |  | yes | yes | yes | yes | yes | yes | yes | yes |  | yes | yes |
| **Air Pressure** |  | yes | yes | yes | yes | yes | yes | yes | yes | yes |  | yes |
| **Electricity Stock Price** |  | yes | yes | yes | yes | yes | yes | yes | yes | yes | yes |  |

Chart 1. Line and area chart options.

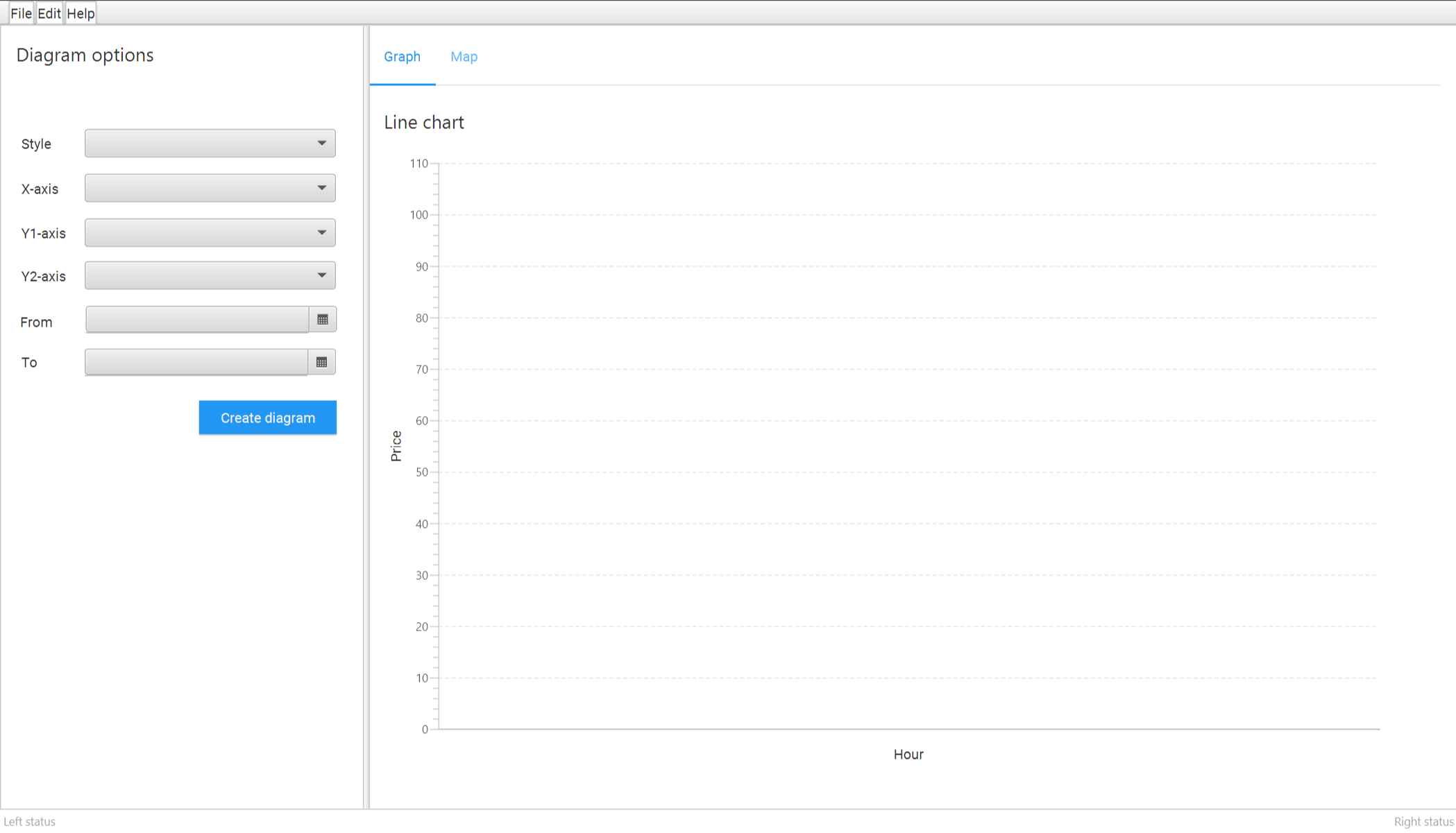
For the scatter dot chart we also have the same options, but time cannot be selected. Therefore in scatter chart date range always is only for data population selection.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  | | --- | --- | | A black and white image of a cross  Description automatically generated |  | | **Electricity Consumption** | **Electricity Production** | **Hydro Power** | **Nuclear Power** | **Wind Power** | **Temperature** | **Wind** | **Rain** | **Humidity** | **Air Pressure** | **Electricity Stock Price** |
| **Electricity Consumption** |  | yes | yes | yes | yes | yes | yes | yes | yes | yes | yes |
| **Electricity Production** | yes |  | yes | yes | yes | yes | yes | yes | yes | yes | yes |
| **Hydro Power** | yes | yes |  | yes | yes | yes | yes | yes | yes | yes | yes |
| **Nuclear Power** | yes | yes | yes |  | yes | yes | yes | yes | yes | yes | yes |
| **Wind Power** | yes | yes | yes | yes |  | yes | yes | yes | yes | yes | yes |
| **Temperature** | yes | yes | yes | yes | yes |  | yes | yes | yes | yes | yes |
| **Wind** | yes | yes | yes | yes | yes | yes |  | yes | yes | yes | yes |
| **Rain** | yes | yes | yes | yes | yes | yes | yes |  | yes | yes | yes |
| **Humidity** | yes | yes | yes | yes | yes | yes | yes | yes |  | yes | yes |
| **Air Pressure** | yes | yes | yes | yes | yes | yes | yes | yes | yes |  | yes |
| **Electricity Stock Price** | yes | yes | yes | yes | yes | yes | yes | yes | yes | yes |  |

Chart 2. Scatter dot chart options.

## Example UI

Made with Scene Builder

   
Picture 1. First iteration of user interface prototype

# API’s

In the solution we are using two API’s: Fingrid for energy data, and Finnish Meteorogical Institute for weather data. Also if time permits we can gather electicity stock market price data from porssisahko.net.

## Fingrid

Fingrid provides data related to the electricity production, consumption and multiple other variables that are directly related to electricity. Fingrids publicly listed but is owned by Finnish government and Finnish pension funds. Whole Fingrids dataset is vast, but we found about we are going to use five of them as described in Chart 1 and 2. Even those have some variation. For instance depending on data type it can be searched with hourly intervals, three minute intervals or even as forecast for coming hours (up to four days depending on the variable).

**Fingrids API documentation:** <https://data.fingrid.fi/open-data-api/>

**Fingrids available datasets:** <https://data.fingrid.fi/en/dataset/>

**Collection of multiple Fingrid datasets:** <https://fingrid-public.s3-eu-west-1.amazonaws.com/files/Avoin+data+tietoaineistoluettelo+9.6.2021.xlsx>

## Finnish Meteorogical Institute

Finnish meteorological institute provides weather data on multiple Finnish places. Datasets available are massive, but on our project we selected 5 datatypes to be included in the project (as seen on charts 1 and 2).

Finnish Meterogical Institute API documentation: linkki

## porssisahko.net (extra)

Pörssishäkö.net is a small API service to get hourly electricity stock price rates. Data source it is not clearly stated, but it is strongly implied that at least forecast data comes from Nord Pool (which is an paid API service). This service only produces the stock price (and stock price forecast for the next day). It is a very simple API. On the downside each hourly data has to be looked by separate API calls. Searching for a days data therefore takes 24 API calls in total which makes it somewhat slow service for big datasets.

# Architecture and design

## Activity diagrams of high-level use cases

### Program start-up

## 

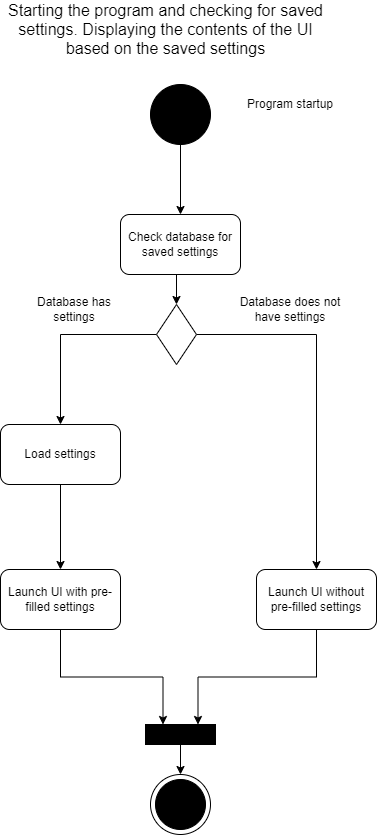


Diagram 1. Activity diagram presenting the process of the programs start up.

Before the programs user-interface is presented to user, the program will check, if the database has certain settings that have been stored. Reason why the check is done before the launch of the user interface is because the settings can have an affect how the user interface is presented. For example, if user has saved chart diagram creation related settings to database, these settings will need to be presented to user.

Note that the diagram does not consider all the possible setting related checks that will be executed. It is demonstrating the high-level idea. Also, as of writing this document it has not been decided whether a “real Database” (such as SQLite) shall be used or whether the settings are stored in a file. Regardless of the decision, the idea presented in the diagram applies.

If the Pörssisähkö API (section 3.3) is implemented, initiation of parallel process of fetching price-related information can be launched after the launch of the user-interface in case where the database lacks price-related information. The reason for this is that Pörssisähkö API is quite slow and in case user wishes to create price-related diagrams, it is good to cache price related information in advance for better user experience. The paraller process mentioned above is not described in the diagram.

### Validation of user inputs

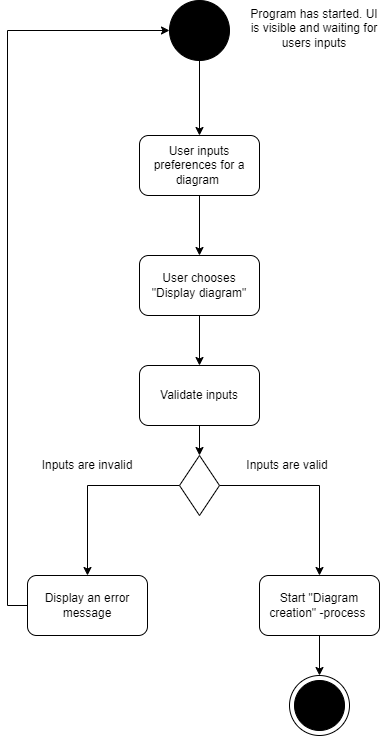


Diagram 2. Activity diagram presenting the process of user inputs validation.

Users’ inputs (such as variables regarding a diagram to be created, start date and end date) should be validated before a diagram will be created. Reason to validate user inputs is to prevent execution of expensive operations, such as API calls in case where user has for example forgotten to something mandatory from some of the fields in the user-interface. While the diagram does not clearly present it, the idea is that user-input related checks are done in user-interface layer (controller) before a service from a model is called. Assuming user’s inputs are valid, meaning that all the mandatory fields have appropriate values, the “Diagram creation” process can be started.

### Diagram creation process

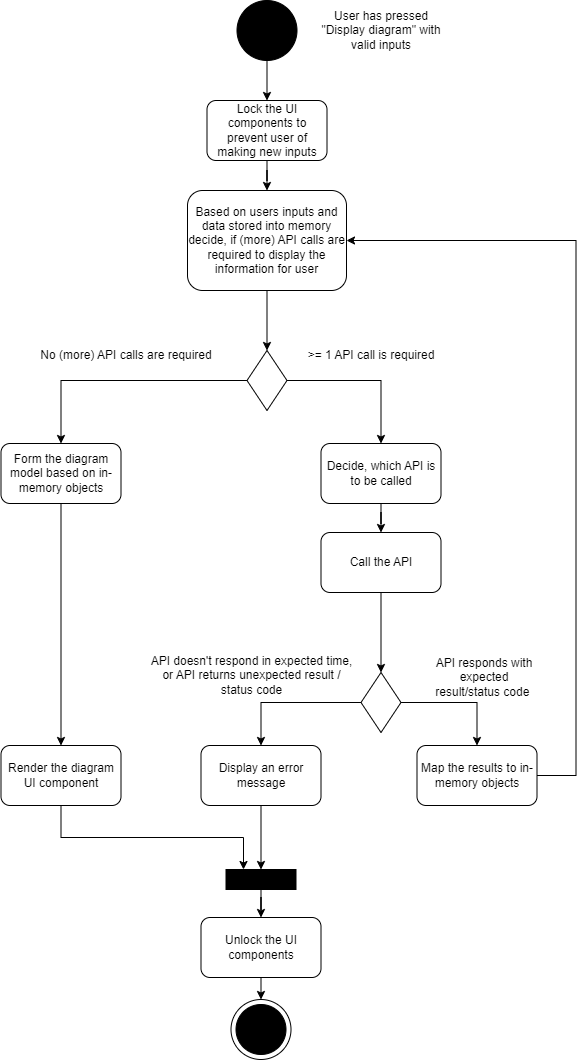


Diagram 3. Activity diagram presenting the process of diagram creation.

Once the users’ inputs have been validated, the diagram creation process can begin. The diagram creation process happens in the background, during which the user-interface should be locked. Locking the user-interface can mean for example a loading spinner to be shown in the user until the diagram creation process is ready. The diagram creation process in a nutshell means collecting the necessary data to serve users request, mapping the data to models of the program, and providing this information to user interface layer (controller), so that the diagram can be visually built and rendered.

Depending on the users request, getting the data requires 0..n API calls to be executed. In some situations, the program can already have the required data in memory. For example, if user has recently created a line chart with variables x and y for date range z to v, creating second diagram will not require an API call if the date range of new diagram is a subset of the date range z to v. In this case the program should have all the necessary information in memory.

Since API call can result an error caused by 3rd party (e.g. the API is down), these errors need to be properly dealt with. The program should not crash but instead display an error message stating “Technical error happened. Please try again later” as an example.

## Diagram of components within the system

@Heikki your diagram here

## Design patterns and decisions

The program will be implemented utilizing MVC (Model View Controller) design approach. Our approach can be seen as “stack-like” where View is the highest element of the stack and model is the lowest element of the stack. In our approach the view never communicates directly with the model, meaning that the controller is responsible for acting as an interface between the model and the view.

User inputs are given from the user interface, which we consider as “view”. For any given input from the view, the controller validates the input and decides, whether the model is required. For example, when user presses “Generate diagram” button, the controller checks whether all the required fields have a proper input. If something mandatory is missing (e.g. end date has not been set), the controller informs the view to display proper error message and therefore it does not need to interact with the model.

The controller does validations such as mandatory field is not missing, or end date has been selected to be after start date. However, the controller does not know (and is not supposed to know) if an API is down, or whether date x has any information that an API can provide or not. This means that data and API related check is done in model and input related check is done in controller.

From technical point of view the view is static .fxml -file in the “resources” folder of the project. The controller will be implemented by creating classes which dynamically change the UI components defined in the .fxml -file depending on users interactions and information received from the model.

The model can be seen as the brain of the program. The model is responsible for acting as an interface toward the controller, executing API calls, maintaining the data models, mapping the API results to internal objects, and interacting with database. Heart of the model is data management service, which offers and interface towards the controller. The data management service is responsible for maintaining the data (internal objects regarding e.g. Weather, Energy) and knowing where new information can be asked if the controller requests information that is not stored in-memory during the time of the request. New information is mainly received from API’s, but the data management service is not the one executing the API calls. Instead of making the API calls, the data management service can call API service. The API service is responsible of knowing the API-specific requirements (such as required HTTP methods, parameters) and executing the API calls based on those requirements. The API service is never called from controller.

# Attachments

Attachment 1. Discussion with ChatGPT to form the idea.

# References