Views and GUI

The GUI implemented using QML. GUI and GUI implementation are divided into three independent views. Each view has its own features. The views are Electricity view, Weather view and Visualisation view. The views and their features are independent of each other. Goal of partitioning the GUI in this way is to simplify the project structure and ease division of task and responsibilities with multiple developers.

Electricity view

User can see a pie chart of percentages of different power forms and total energy production.

Weather view

The user can request calculations on average temperature at certain location in certain month. The user can request calculations on average maximum and average minimum temperature at certain location in certain month.

Visualisation view

User can plot data fetched from web APIs and combine them. Requestable data sets are listed below. Depending on the source of the data the user can specify different parameters for the fetched data, such as location, begin date and end date.

- Fingrid:
    - Electricity consumption - real time
    - Electricity consumption forecast - hourly
    - Nuclear power production - real time
    - Wind power production - real time
    - Wind power generation forecast - hourly
    - Hydro power production - real time
    - Solar power generation forecast - hourly
    - Electricity production - real time
    - Electricity production forecast - hourly
- FMI
    - Observed monthly average temperature (C°)
    - Observed temperature with 10 min timestep (C°)
    - Observed wind with 10 min timestep (m/s)
    - Observed cloudiness with 10 min timestep (okta)
    - Predicted temperature (C°)
    - Predicted wind (m/s)
    - Hourly average temperature (C°)
    - Hourly maximum temperature (C°)
    - Hourly minimum temperature (C°)
    - Daily average temperature (C°)
    - Daily minimum temperature (C°)
    - Daily maximum temperature (C°)

Components

FMIService and FingridService C++ classes are used via QML-exposed DataService façade. The services are used by the views that need data from them. All components are stateless so the services can be easily used by multiple views.

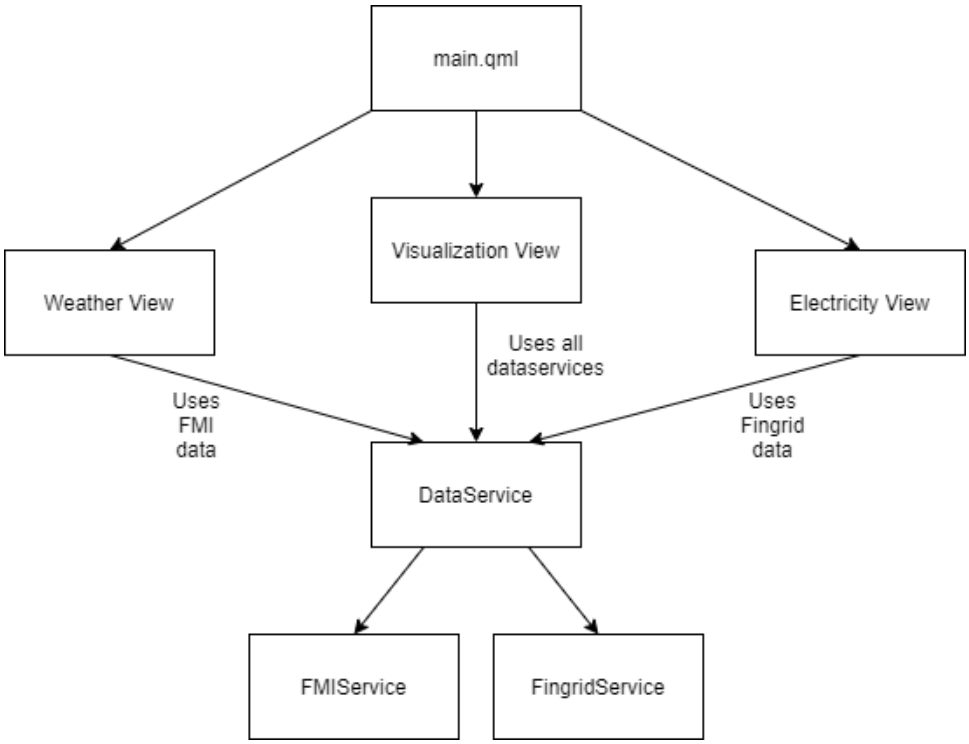

Figure 1. Dependencies between components and views

DataService

DataService is a C++ class which provides all data services in one class. Access to certain API happens with given parameter.

Public methods

- `QVariantList getTimeseries(const QString dataservice, const QString &param, const QDateTime &starttime, const QDateTime &endtime, const QString &place) const;`
    - Gets requested timeseries from FMI or Fingrid api.
    - Parameters
        - Dataservice, requested API data. Options are: fmi or fingrid.
        - param - Requested parameter. Must be in getParameters()
        - starttime - Begin time of the timeseries
        - Endtime - End time of the timeseries
        - Place – position of collected FMI data
    - Return value
        - QVariantList containing QVariantMaps. Each list element has two key-value pairs. "time" contains the time value for timeseries entry as QDateTime. "value" contains the value for timeseries entry as double.
- `QVariantList getParameters(QString dataservice,QString option) const;`

    - Parameters
        - Dataservice, requested API data. Options are: fmi or fingrid.
        - Option, requested data type, options are: production_parameters or production_words
    - Returns a list of parameters requestable from getTimeseries

FMIService

FMIService is a C++ class that provides data from Finnish Meteorological Institute API via simple interface.

Public methods

- `QVariantList getTimeseries(const QString &place, const QString &param, const QDateTime &starttime, const QDateTime &endtime) const;`
    - Gets requested timeseries from FMI api.
    - Parameters
        - Place – position of collected FMI data
        - Param - Requested parameter. Must be in getParameters()
        - Starttime - Begin time of the timeseries
        - Endtime - End time of the timeseries
    - Return value
        - QVariantList containing QVariantMaps. Each list element has two key-value pairs. "time" contains the time value for timeseries entry as QDateTime. "value" contains the value for timeseries entry as double.
- `QVariantList getParameters() const;`
    - Returns a list of parameters requestable from getTimeseries

Private methods

- `QUrl parseURL (const QString &place, const QString &param, const QDateTime &starttime, const QDateTime& endtime) const;`
    - Parses query url for FMI API
    - Parameters
        - Place – position of collected FMI data
        - Param - Requested parameter. Must be in getParameters()
        - Starttime – Begin time of the timeseries
        - Endtime – End time of the timeseries
    - Return value
        - QUrl containing full URL to request data from FMI API
    - Usage
        - This method is meant to be used in getTimeSeries function to parse url using given parameters
- `QString getBaseURL(std::string param) const;`
    - Returns base url for FMI API
    - Parameters
        - Param – Requested parameter. Must be in getParameters()
    - Return value

        - QString containing first part of needed URL to request data from FMI API
    - Usage
        - This method is meant to be used in parseURL function to help to parse URL with given parameter.
- `const QByteArray get(const QUrl &url) const;`
    - Does FMI API request using parsed URL
    - Parameters
        - url– url to FMI API
    - Return value
        - QByteArray containing requested data from FMI API
    - Usage
        - This method is meant to be used in getTimeseries function to get data.
- `QVariantList toTimeSeries(QByteArray &xml) const;`
    - Converts attached data into QVariantList
    - Parameters
        - Xml – requested FMI API data in xml-format.
    - Return value
        - QVariantList containing FMI API data
    - Usage
        - This method is meant to be used in getTimeseries function to convert data into suitable format.

FingridService

FingridService is a C++ class that provides data from Fingrid API via simple interface.

Public methods

- `QVariantList getTimeseries(const QString &param, const QDateTime &begin, const QDateTime &end) const;`
    - Gets requested timeseries from Fingrid api.
    - Parameters
        - param - Requested parameter. Must be in getParameters()
        - begin- Begin time of the timeseries
        - end- End time of the timeseries
    - Return value
        - QVariantList containing QVariantMaps. Each list element has two key-value pairs. "time" contains the time value for timeseries entry as QDateTime. "value" contains the value for timeseries entry as double. Value uses megawatt as unit.
- `QVariantList getParameters() const;`
    - Returns a list of parameters requestable from getTimeseries

Private methods

- `QUrl parseURL(const QString &param, const QDateTime &begin, const QDateTime& end)`
    - Parses query url for Fingrid API

    - Parameters
        - param - Requested parameter. Must be in getParameters()
        - Starttime – Begin time of the timeseries
        - Endtime – End time of the timeseries
    - Return value
        - QUrl containing full URL to request data from Fingrid API
    - Usage
        - This method is meant to be used in getTimeSeries function to parse url using given parameters
- `const QByteArray get(const QUrl &url) const;`
    - Does Fingrid API request using parsed URL
    - Parameters
        - url– url to Fingrid API
    - Return value
        - QByteArray containing requested data from Fingrid API
    - Usage
        - This method is meant to be used in getTimeseries function to get data.
- `QVariantList toTimeSeries(const QJsonArray &json) const;`
    - Converts attached data into QVariantList
    - Parameters
        - json– requested Fingrid API data in json-format.
    - Return value
        - QVariantList containing Fingrid API data
    - Usage
        - This method is meant to be used in getTimeseries function to convert data into suitable format.

Utils.js

Collection of simple multipurpose JavaScript functions used by QML-files.

Functions

- max(arr)
  Returns biggest element in array arr using > operator. If arr is not an array or is empty return undefined instead.
- min(arr)
  Returns smallest element in array arr using < operator. If arr is not an array or is empty return undefined instead.
- yesterday()
  Returns yesterday's date.
- linspace(min, max, n)
  Returns array of n linearly spaced numbers between and including min and max
- unique(arr)
  Returns array containing unique values of arr. Elements are ordered by order of appearance in arr.
- average(arr)
  Returns the average or mean value of the array.

- range(begin, end)
  Return array of integer numbers between begin and end. begin is included, end is excluded

Design pattern

- We are using MV(Model – View) -design pattern
- This design pattern was chosen because it is natural with QML applications. It provides clear separation of concerns.
- View part contains Electricy View, Weather View and Visualisation View
- Model part contains FMIService and FingridService

Self-evaluation

Our design has supported well our implementation and we have mostly stuck with the original plan. We have not made many major changes to the design. Changes to the original design are listed below.

Changes to the design for mid-term submission:

Due to our team size being only two, it was decided not to implement requirements related to saving data non-volatilely and user preferences. Due to this change, *FileService* from earlier version of this document has been removed from this document and will not be implemented. This has also led to some changes in GUI.

Another design change concerns the return types and interfaces of FMIService and Fingrid service. The return type has been changed to be QVariant-based. The original return types were not QML-compatible. Also, instead of every requestable parameter having its own function for getting data, now both FMIService and FingridService have general functions for fetching data which take the requested parameter as argument.

Changes to the design for final submission:

We decided to refactor the FMIService, FingridService, and possible future data fetching services, to be used in QML via façade. Due to this we have only one QML-exposed C++ class DataService. We hope that this change will lead to better maintainability for the software.

Despite of being very similar FMIService and FingridService do not implement a common interface. This is because getTimeseries takes different parameters depending on the service used.