Design Report

Chris van den Hoorn	s2975823
Erwin Loof	s2832461
Kevin Nieuwenhuis	s3012719
Mihai Pop	s2688069
Lu Schoevaars	s2964163
Anna Smit	s2846012

April 17, 2025

Contents

Chapter 1

Introduction

The Ecofactorij [4] is a business park near the intersection of highways A1 and A50 in Apeldoorn, the Netherlands. The companies of the Ecofactorij decided to incorporate a private electricity grid on site. This includes for example batteries and photovoltaic panels. A photovoltaic panel is a chain of photovoltaic cells. A photovoltaic cell converts sunlight into electric energy.

To construct, maintain and analyse the private electricity grid on site, the companies founded the corporation Coöperatief Parkmanagement Ecofactorij U.A. Within this corporation, researchers from among others Saxion University of Applied Sciences and University of Twente are involved to analyse the grid. They intend to be able to assess the efficiency of the grid.

To this end, a digital twin of the electricity grid was constructed by these researchers [5]. A digital twin is a representation of a real-life system, designed to reflect the physical copy by processing data measured by sensors to aid analysing and decision making about its reflection.

From the University of Twente, researcher J.C. López Amézquita from the Energy Management research group is tasked with among others various studies in the efficiency of the grid using the digital twin. He collaborated with students of Saxion University of Applied Sciences to calculate the efficiency of the on-site batteries.

Now, the efficiency of the photovoltaic panels are of interest. The first calculations were on a per cell basis, in the sense that the efficiency of each photovoltaic cell of the panel was determined and used to calculate the panel efficiency. However, this proved not to be reliable as it determined the efficiency of the cell as opposed to the efficiency of the panel.

A better measure is the performance ratio [9]. That is the ratio between the actual yield and the theoretical yield. The actual yield is stored within the digital twin and the theoretical yield can be calculated. This extends to a larger project, so J.C. López Amézquita drafted the authors to work on these calculations.

The project extends to accurately determine the performance ratio of the photovoltaic panels at the Ecofactorij. This enables real-time processing of data

from the digital twin, but also the processing of historical data to calculate the performance ratio. A forecast model was additionally requested, to predict the efficiency for the future.

The product, the process of producing the product and the experience in authorship hereof are discussed in this text. In Chapter 2: Requirement specification, the requirements for the product will be outlined. It will become clear what the product should be able to achieve. In the following Chapter 3: Architectural design, the main components which will achieve the specifications will be described. This specifies the different parts within the product. One of these parts will be explored in Chapter 4: Key Performance Indicators. The entire process of calculating the performance ratio is explained there. Another part will be explored in Chapter 5: Forecasting. There the entire process of forecasting the actual yield is explained. Other components are explored in Chapter 6: Design choices. These are more minor components which receive explanation and justification for their implementation in this chapter. Chapter 7: Testing then makes clear how the components are tested to function correctly. Any other components that are not included in the product, but would in hindsight be valuable additions are described in Chapter 8: Future planning. The product has then been described in entirety. Following these chapters are Chapter 9: Evaluation which reflects on the process of producing the product and Chapter 10: Conclusion which reflects on the product and the production hereof again.

Chapter 2

Requirement specification

In this chapter we explain the planning of our project and what the requirements of our project are.

2.1 Project management

For the project, we used the Agile development strategy SCRUM using sprints of two weeks. SCRUM is a way of developing in short 'sprints' in which you focus on small aspects of a project. Features of the end product are divided into user stories, which are then divided into tasks to be completed. A user story is usually finished in a single sprint. A key feature of Agile development is quick development, flexible standards, small teams, and frequent reflective sessions. To reflect on the progress, weekly meetings with the supervisor were planned. But before we could start on the first sprint, we first identified the stakeholders and requirements.

2.2 Stakeholders

Before specifying the requirements, the project stakeholders were identified. One of the stakeholders is the client, who is the supervisor of the project and at the same time the representative of the client. The other stakeholder is the client, the Ecofactorij, who will be using the product.

2.3 User stories

The requirements were categorized using the MoSCoW method and formulated using a user story. In the following section, the user stories are listed within their corresponding MoSCoW category.

2.3.1 Must

1. As a user, I want to be able to see the final yield of the PV systems in real time in OpenRemote.
2. As a user, I want to be able to see the performance ratio per group of PV systems in real time in OpenRemote.
3. As a user, I want to be able to see the reference yield of the PV systems in real time in OpenRemote.
4. As a user, I want to be able to see the influence of environmental factors (irradiance, radiation) on the performance.
5. As a user, I want to be able to view the statistics (PV details) using OpenRemote.
6. As a user, I want to look up how to use the system in the manual.
7. As the project owner, I want to be able to look up in the manual how to start up the system and train the learning model.
8. As a user, I want to be able to view historical data of the PV systems from at least 1 year back.
9. As a project owner, I want a separate docker container for each component of the solution.
10. As a project owner, I want the solution to use weather data from Open-Meteo.

2.3.2 Should

11. As a user, I want to be able to see the prediction (forecasting) of the KPIs of the PV system for the upcoming 24 hours, based on Open-Meteo and historical data.
12. As a user, I want to be able to view the geographical location of the PV systems using OpenRemote.
13. As a project owner, I want the data to be stored in a MySQL database.

2.3.3 Could

14. As a project owner, I want the learning model to continue training on new incoming data.

2.4 Validation

After creating the user stories, we validated them by showing them to our supervisor who said they were correct and that nothing was missing.

2.5 Project phases

The project is split up into four sprints and for each sprint, we decided what user stories should be completed.

2.5.1 Sprint 1 (24 February - 7 March)

In sprint 1 we focused on implementing the core features and tests of the product. This includes setting up the docker containers, implementing the core of the back end, as well as the connection to the digital twin and Open-Meteo. The user stories related to this are 1, 2, 3, 5, 9, 10, 12, and 13.

2.5.2 Sprint 2 (10 March - 21 March)

In this sprint, we continued on the spillover user stories from sprint 1 and we started on implementing the environmental factors, and making a start on the forecasting. This concerns the user stories from sprint 1 and user stories 4, and 11.

2.5.3 Sprint 3 (24 March - 4 April)

This sprint was dedicated to inserting the historical data, working on the forecasting, and adding improvements to the product, concerning user stories 8 and 11.

2.5.4 Sprint 4 (7 April - 17 April)

The final sprint was dedicated to finishing up the final product, fixing bugs, and writing documentation such as creating the manual as written in user stories 6 and 7. In the end we did not manage to finish user story 14, as we will also state in section 8.1.4.

Chapter 3

Architectural design

This chapter introduces the architectural design and the global design choices that were made.

3.1 Global design prerequisites

Because the project builds further up on the digital twin and because the project is for the Ecofactorij, we had as architectural requirement that OpenRemote was used for the front end. Therefore, we had to design the back-end to work with OpenRemote.

3.2 Global design choices

In order to make our backend compatible with OpenRemote, we made our backend accessible using a REST API. This makes it very flexible and easily to configure and use in OpenRemote. We chose to run the api inside a backend container using Docker, next to the database container. This allows for easy deployment and the ability to run side-to-side with the digital twin. A general overview of our architecture can be seen in Figure 3.1.

3.3 Backend

We decided to create the backend in Python because Python is generally an easy-to-use language and all of the project members are familiar with it. Next to that, it gave us the possibility to create all components of the project, including the API and the prediction model, in the same programming language as the Digital Twin. See Figure 3.2 for a schematic of the backend.

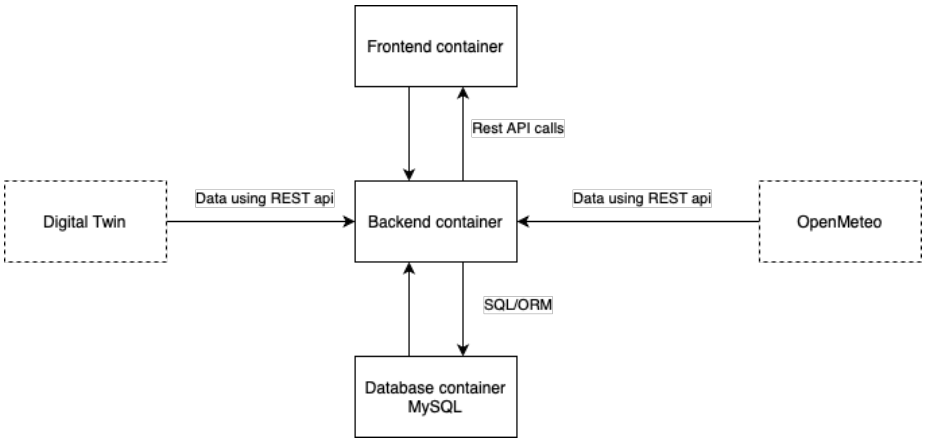

Figure 3.1: General design overview

3.4 Database

We have also created a database that can be used as a local cache for obtaining the PV panels in case the production API is not reachable. This ensures that the basic functionality of our product remains intact. We chose MySQL as the database server, because the database for the digital twin also uses MySQL, so it fits better in the entirety of the project. The database scheme we decided on is displayed in fig. 3.3.

Panels	
PK	Id
	meter_name
	Data

Figure 3.3: Database design

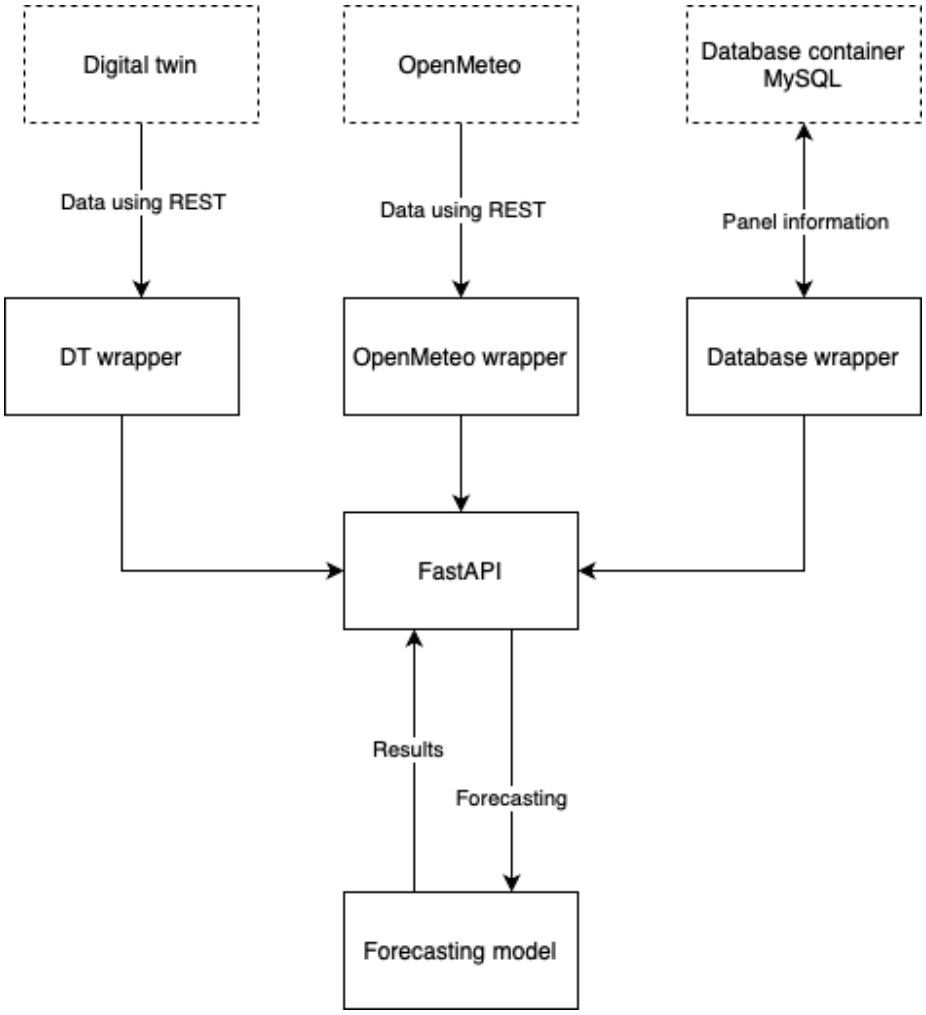

Figure 3.2: Backend design

Chapter 4

Key Performance Indicators

This chapters explains what key performance indicator are used in this project, what their values mean, and how they are calculated.

One of the core parts of the project is to calculate key performance indicators (KPI's). The KPI of interest is the performance ratio. This ratio indicates the proportion of energy between that which actually being generated by a specific PV panel and the theoretical production of the panel.

4.1 Performance Ratio

The performance ratio (PR) is calculated using eq. (4.1) [8]. As the performance ratio is a ratio, it divides the field system yield Y_f over the reference yield Y_{ref}. The performance ratio has no unit.

$$PR = \frac{Y_f}{Y_{ref}} \tag{4.1}$$

This calculates the instantaneous performance ratio for a given time. However, the calculations come with a few challenges that remain to be solved. These will be explored in the following subsections.

4.1.1 Reference yield of 0

First, the reference yield may be 0. That means that theoretically no energy can be produced. This can occur when there is no sunlight to convert to electricity. The performance ratio can then not be calculated, as the denominator equals 0. However, a performance ratio would be nonsensical, since it would describe the amount of energy that was produced out of nothing. Therefore, performance ratios with a reference yield of 0 have been omitted.

4.1.2 Outlier removal

There can be errors in the final system yield and reference yield data. For instance, the final system yield might have measurement errors or the weather model of the reference yield might make inaccurate predictions.

Due to these inconsistencies, the nominator might become several times larger or smaller than the denominator in the calculation of the performance ratio. A particular instance where that might occur is when shutting down the PV system at sunset. The final system yield would be infinitesimally small since the system has been shut of, but the reference yield would still equal just low values, since there is still some irradiance from the sun. These inconsistencies lead to inaccurate performance ratios, where the errors cause ratios of 130% to 150%. However, in the described instance, the performance ratio might get as high as 15000%.

Therefore, these absurdly large performance ratios would impact the average performance ratio significantly. Therefore, these outliers are removed. All performance ratios before the fifth percentile and after the 95th percentile are omitted in the calculations.

4.1.3 Consequences of instantaneousness

The performance ratio at a given time is restricted to the conditions of these times. For instance, it may be cloudy or very hot. These environmental factors impact the performance ratio drastically. To offer a good overview of the performance of the PV system, it is better to look at the average performance ratio over a given time.

However, in the product it was desired to be able to see the performance of the PV system at a given time. This was implemented by using a sliding window algorithm. To calculate the performance ratio at a given time, the average is taken of all performance ratios from a certain amount of time before that given time until 12 hours after that given time. The certain amount of time before the time is decided by the window size. The user can set this to control the amount of time for which the performance ratios get averaged.

As an example, suppose a user wants to calculate the performance ratio of the July 23rd at 12:00 with a window size of 3 days. The window always ends 12 hours after the target time, so at July 24th 00:00. Since the window size is 3 days, the window begin is 3 days before the window end and therefore 2 days and 12 hours before the target time. The window begin is therefore at July 21st 00:00. The entire window is then from July 21st 00:00 until July 24th 00:00, which makes it a size of 3 days.

The window size impacts the accuracy of the representation of the performance of the PV system. The higher the window size is, the better it is represented, as it takes more conditions into account. The supervisor suggested a window size of at least a month for accurate representations such that they can be compared to other PV systems. However, to also be able to indicate how the performance ratio is around the target time, a smaller window would be needed.

Therefore, a window of 3 days was chosen. To that extend, the window has not been fixed and can be easily configured with an input parameter.

Finally, the window is askew around the target time. It always includes 12 hours before the target time and the rest of the time after. This is due to a constraint of the forecasting part of our project. It will predict at most the data for the upcoming 24 hours. With this prediction, the product still needed to be able to compute the performance ratio of the current time. Therefore, the window was chosen to be fixed to end 12 hours ahead, such that the performance ratio of up to 12 hours in the future could be calculated. As a counterexample, suppose that the window was symmetrical and one wants to calculate the performance ratio of the current time with a window of a month, that would imply that the upcoming 15 days should be predicted, which is currently impossible. As a middle ground, the askewed window with 12 hours look ahead was chosen.

4.2 Field system yield

The field system yield Y_f in kWh is calculated using eq. (4.2) [8]. This can be viewed as the actual energy production.

$$Y_f = \frac{E}{P_{STC}} \tag{4.2}$$

E denotes the energy production in kW. The energy production for the calculation is taken from the digital twin. The digital twin includes the data of the energy production in kW of the PV panels at the Ecofactorij for every five minutes. This production is averaged over the hour to be complaint with the reference yield, where data is only available for each hour.

P_{STC} denotes the rated capacity of the PV in kWp. It is the amount of energy in kW that a PV panel will produce under standard test conditions (STC). These conditions are 1 kW of irradiance per square meter panel, a temperature of 25°C within the cells of the PV panel and an air mass of 1,5. The rated capacities of the PV panels at the Ecofactorij are included in the digital twin.

4.3 Reference yield

The reference yield Y_f in kWh/m^2 is calculated using eq. (4.3) [8].

$$Y_f = \frac{H_{POA}}{G_{STC}} \tag{4.3}$$

H_{POA} denotes the plane-of-array irradiance in kWh/m^2. The plane-of-array irradiance is the amount of power received by the PV panel. Intuitively, it can be seen as the amount of sunlight that reaches the PV panel per square meter to be converted into energy.

4.3.1 Plane-of-array irradiance

There are three components towards this plane-of-array irradiance. The panel receives all these radiation components in some way, as seen in fig. 4.1. First, there is direct radiation which is measured by direct normal irradiance (DNI) [11]. This is the radiation from sunbeams that directly originate from the sun. Second, there is the diffused radiation which is measured by diffuse horizontal irradiance (DHI) [11]. This is the radiation from the sun which has been diffused by the atmosphere. Third, there is reflected radiation which is measured by global horizontal irradiance (GHI) [11]. This is the radiation reflected by the surface of the Earth. The summation of these radiation beams (eq. (4.4) [7]) is the plane-of-array irradiance.

$$H_{POA} = G_{direct} + G_{diffused} + G_{reflected} \tag{4.4}$$

Although the components can be summed directly, the measures DNI, DHI and

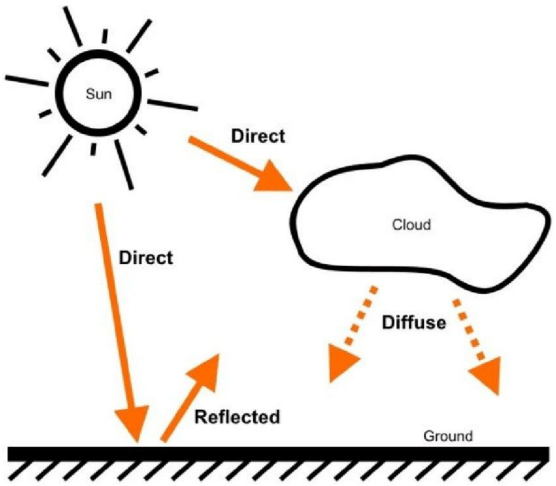

Figure 4.1: Solar radiation components [11]

GHI cannot, since they are under specific angles. There is already open-source software, called pvlib [2], to calculate the plane-of-array irradiance based on the DNI, DHI and GHI when also giving the specific angles and some additional parameters. A discussion about these specific angles will follow. The remaining additional parameter is the albedo. This is how much light reflects on the surface. If more is reflected, then also more is received by the solar panel. Therefore the albedo is also of interest.

4.3.2 Solar angles

Those are the solar angles of the PV panels. The angle of the sun with the panels changes with the movement of the sun and therefore also the amount

of received radiation on the panels. Since the sun is constantly moving, these angles need to be constantly calculated. The angles the open source software needs to calculate the received irradiance are the solar azimuth and zenith and the surface azimuth and zenith.

The solar azimuth and zenith indicate the position of the sun with respect to the PV panel. The solar azimuth, depicted in fig. 4.2, is an angle defined on the projection of the direct sunbeam on the ground [12]. It is the angle between true north and projection of that sunbeam. Intuitively, it can be viewed as how many degrees one needs to rotate around ones z-axis to the right to stare directly into the sun. Usually this is measured from true north, but the digital twin measures the azimuth from the south. However this can be converted with eq. (4.5).

$$azimuth_{north} = azimuth_{south} - 180 \bmod 360 \tag{4.5}$$

The solar zenith, depicted in fig. 4.2, is the angle between the vector pointing to the sun and the vector which is orthogonal to the surface [12]. Intuitively, it can be viewed as how many degrees one needs to look up to stare directly into the sun.

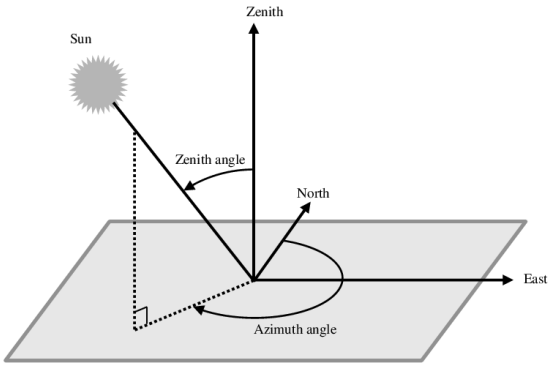

Figure 4.2: Solar azimuth and zenith angles [6]

These angles together determine the location of the sun in the sky. This is also the leverage in how these angles are determined in the project. This location of the sun if fixed by the time and place from which one is looking. For the calculation of the reference yield, the location of the PV panel is known and the time is the one where the reference yield is being calculated for. With these inputs, the solar azimuth and zenith can be calculated. The same open-source software, pvlib [2], can achieve this. Therefore the solar azimuth and zenith can be calculated.

4.3.3 Surface angles

Next to the sun having an specific angle form true north, the same holds for the PV panel itself. The panel has an orientation around its z-axis measured from true north. This is defined as the surface azimuth, depicted in fig. 4.3, which is the angle between the projection of the vector normal to the surface of the PV panel onto the surface of the Earth and true north [1].

The last angle is the tilt angle of the panel. This angle, depicted in fig. 4.3 is the angle between the solar panel surface and the surface of the Earth.

Both these angles are available per PV panel in the digital twin.

4.3.4 Finalizing the Calculation

Now the surface tilt and azimuth, solar azimuth and zenith and albedo are known. The open source software pvlib [2] can now calculate the amount of irradiance from the DNI, DHI and GHI that is being received by the PV panels. Therefore H_{POA} is known. By using this in the equation for the reference yield, see eq. (4.3), the reference yield can be calculated.

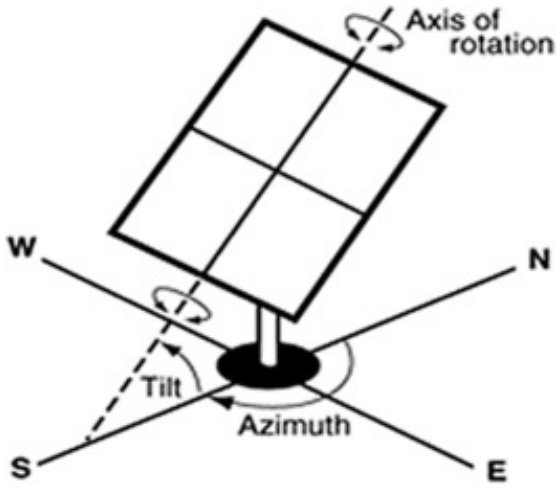

Figure 4.3: Surface azimuth angle and tilt angle [1]

Chapter 5

Forecasting

This chapters explains how the forecast model is trained and how it is used to predict values in the future.

Forecasting is a core part of our project. Our goal was to forecast the field system yield for the upcoming 24 hours. In this chapter we documented all the steps made, including design choices for the architecture, training the model and predicting the field system yield.

5.1 Architecture

As explained in section 4.2, final system yield depends on the energy production and the rated capacity of a PV. Since the rated capacity of a PV is a constant, we have to forecast the energy production. This is dependent on the irradiance and the time of the day, this means we had to forecast this KPI using something that is pattern optimized. After doing some literary research for different solutions for predicting the field system yield, we concluded that we would have to use a neural network with LSTM (Long short-term memory) layers for the most accurate solution. This is due to its ability to handle sequences of data with long-range dependencies, making it ideal for time series analysis [10]. We decided on using 5 layers for our model, as can be seen in fig. 5.1. We reduce the number of neurons per layer as go through each layer, so that in the initial phase, the neural network gets as much context as possible. In the following layers, it identifies and refines patterns, to finally give a prediction in the output layer.

Our model is using several important properties that are influencing the field system yield. We selected these properties based on their correlation with the field system yield. At the start we had the following properties:

1. Direct Normal Irradiance (DNI)
2. Diffuse Solar Radiation (DHI)
3. Shortwave Solar Radiation (GHI)

4. Temperature
5. Cloud cover at 10 meters
6. Wind speed at 10 meters
7. Time sine
8. Time cosine
9. Meter name
10. Field system yield of the day
11. Future Direct Normal Irradiance (DNI)
12. Future Diffuse Solar Radiation (DHI)
13. Future Shortwave Solar Radiation (GHI)
14. Future Temperature
15. Future Cloud cover at 10 meters
16. Future Wind speed at 10 meters

However, after verifying the correlation matrix (a matrix that shows the weight of the dependencies between properties), we discovered that some of the properties do not impact the field system yield significantly. Thus leaving us with the following properties: DNI, Temperature, Time sine, Time cosine, Meter name, Future DNI, Future Temperature.

5.2 Training

To train our model, we use historical data from Open-Meteo and the PV systems from Ecofactorij from 2024. The data that we have from the PV systems was collected every 5 minutes, which enhances the resolution of the predictions due to the amount of measurements per hour that we have. However, Open-Meteo, has only hourly data, so to align the weather data with the data from the PV Systems we interpolated the data to get 5 minute data for the weather.

These properties are using different units with different magnitudes. Meaning that the model would be influenced by the higher magnitude units. To give an example, DNI is can reach values of 800 W/m^2, however, field system yield is a number between 0 and 1. Without normalizing the data, the model would give more attention to the higher values, which in our case is the DNI. Thus we normalized the data to values between -1 and 1, to ensure that every property is using the same scale.

The next step in the process is to create supervised data to train our model. The neural network needs contextual information about the datapoints over time. This means that for each entry in our data, we need to provide data from

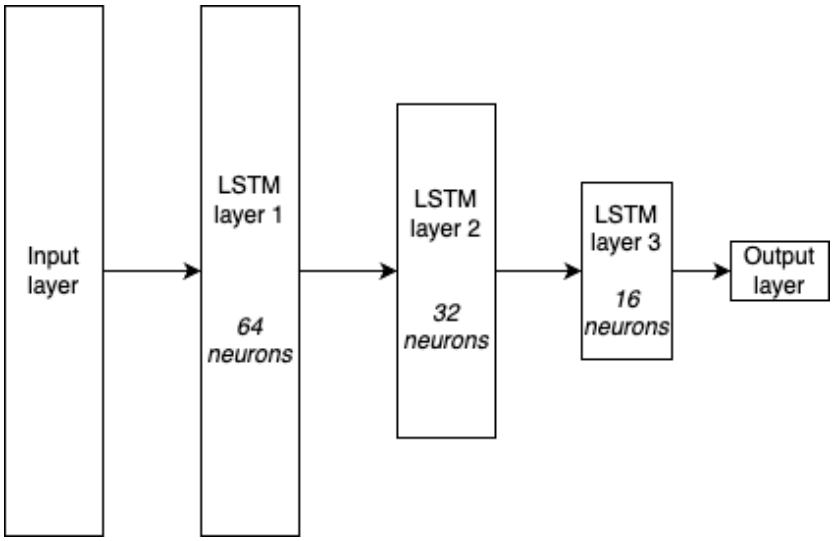

Figure 5.1: Neural network layers

previous timestamps. We do this by shifting the data back and forth by the number of time steps we need to predict in the future. Since we have data every 5 minutes we will have to shift by 288 steps to be able to forecast for the next 24 hours.

After obtaining the supervised data, we split the data into a test and training set, which we use to train the model.

The entire process can also be seen in fig. 5.2

To optimise the training, we added improvements to the process to make the training easier. We added support for early stopping in our training, when the value loss reaches a plateau. Furthermore, we also optimised the learning rate, so when we notice no further improvements after 3 epochs (an epoch is when the model iterates once through the model), it halves the learning rate, so the model learns at a slower rate. In addition, we added checkpoints to the models that we create: every time an epoch passes we save the model verify if the model is the better than previous iterations. We save the best model if that is the case.

The training process can be observed in fig. 5.3

Figure 5.2: Data processing pipeline

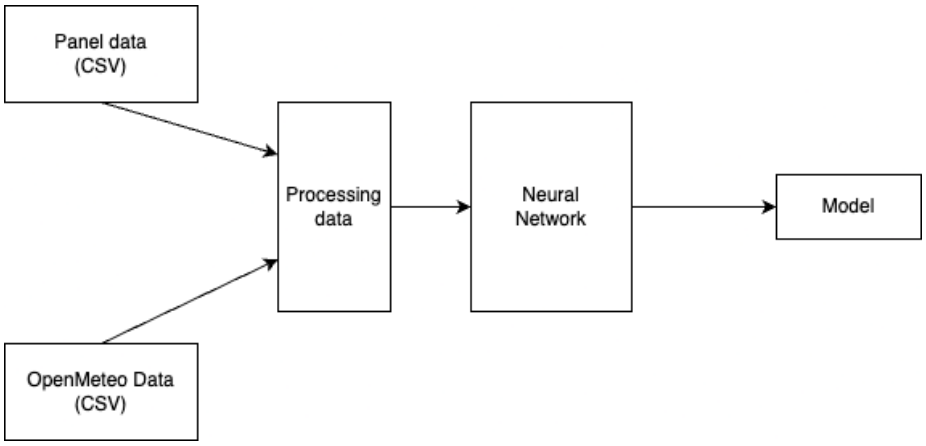

Figure 5.3: Forecasting architecture

5.3 Predicting

To predict, we retrieve the past 24 hours of field system yield from the digital twin of a specified PV system. Additionally, we get the forecast for the next 24 hours from Open-Meteo. We combine this to one big dataframe and we feed it to the model to predict the field system yield for the upcoming 24 hours. The entire process can be observed in fig. 5.4

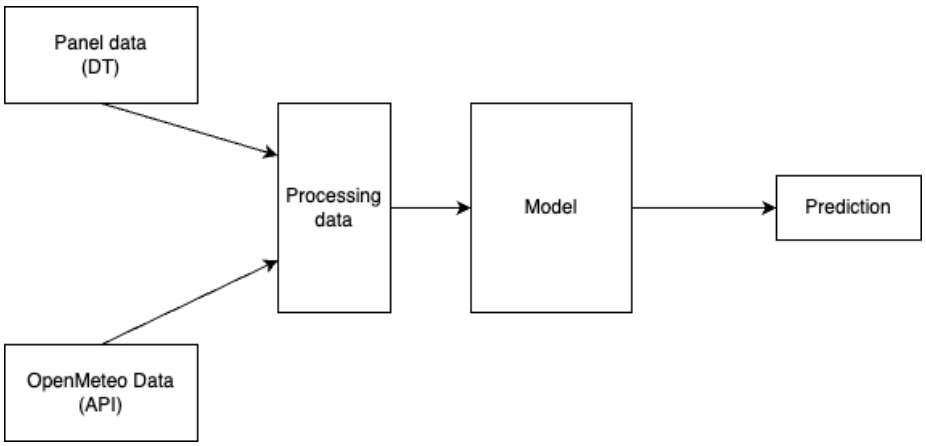

Figure 5.4: Prediction flow

Chapter 6

Design choices

This chapter explains all the smaller design choices that were made for the front end, API, and database.

6.1 Front end

Since OpenRemote already has a fully implemented interface by default, there was not a lot of customization or design left for the appearance of the front end.

6.1.1 Setup

For the setup of OpenRemote, there is the possibility to use their default image or to use their custom template to build the image yourself. We decided to use their default image because the customization of the custom template was not necessary to fulfill the requirements of our project.

6.1.2 Map data

To show the PV panels on a map on the screen, map tiles have to be downloaded to display the map. Since the PV panels are located in the Ecofactorij in Apeldoorn, the map tiles of the Ecofactorij were extracted from the file containing the entire Netherlands, since that file was too big to be uploaded, downloaded, and loaded with ease.

6.1.3 Icon and theme

In the "Appearance" section inside OpenRemote, the logo, favicon and colors of the dashboard can be adjusted. The favicon is the icon that is shown on a tab in the web browser. The logo and favicon were used from the Saxion group who designed the dashboard for the digital twin, since they had already designed a logo for the Ecofactorij which the client liked. For the colors, white was chosen

as the base color, and the green color inside the Ecofactorij logo was used as the accent color, since the Saxion group has already used this color scheme.

6.1.4 Weather data

Each PV panel in OpenRemote contains their associated data, such as their maximum power export and the final yield. For the location of each PV panel, there is a lot of weather data that can be retrieved from Open-Meteo. We decided to show the cloud coverage, temperature, and wind speed since the client said that these weather variables influence the performance of the PV panels. However, the GHI, DHI, and DNI, which are used in calculations, were omitted, because it would add too much complexity for the end-user to show these.

6.1.5 Forecasting and historical data display

Because OpenRemote already has an existing architecture that did not perfectly align with the client's needs, there were some problems with implementing historical and forecasting data points. OpenRemote does not natively allow the modification of timestamps for new data points, so the data points had to be inserted into the database of OpenRemote directly. This is automatically done for both the forecasting when new predictions are made and for the historical data when it is uploaded using the API endpoint available.

6.1.6 Dashboard information

The dashboards from OpenRemote can be used to visualize information. It is up to the client how to use them, but we think that the most useful use case will include comparing the KPIs of different PV systems using line charts.

6.1.7 Creating assets

To show the information for each PV panel in OpenRemote, an asset has to be made. OpenRemote already has a pre-made asset type for PV panels, so most of the data we want to add for a PV panel is already inside there. However, not everything was included so we had to add the variables for the reference yield, final yield, cloud coverage, temperature, and wind speed ourselves. Furthermore, the PV panels were first added to OpenRemote manually, but this seemed to be rather time-consuming and inefficient. Additionally, when the panels are added manually, the database of OpenRemote has to be shared, which is inconvenient and not practical because there's no version control possible for database files in GitLab. Instead, we found out that we could use the OpenRemote API to add assets through HTTP requests. So a Python script was written to automatically insert the information of the PV panels into OpenRemote.

6.2 Backend

For all calls to external sources to be in one place, class wrappers were created. This meant the variables related to the external source, such as the URL and cookies, are stored inside the wrapper. This means they cannot be accessed outside of the wrapper, which improves security. It also ensures that calls to get information from outside the wrapper are limited in what information they need to give. The API calls to all wrappers.

6.2.1 API

The API we created is the core of our product. It is queried by OpenRemote for all the data needed by the dashboard, and it itself queries all required services, such as Open-Meteo for the weather data and the digital twin for all data related to PV systems. As we chose to work in the Python programming language, we decided to make use of FastAPI, as it is easy to work with and is in the programming language the rest of our project is written in as well. It proved to be quick and versatile, and we managed to achieve everything we wanted with it with ease. The API is running in it's own Docker container, and is connected to other Docker containers via a Docker internal network.

6.2.2 Historical data

As OpenRemote does not support inserting historical data directly in its interface, we have created an API endpoint to work around this. When this API endpoint is called a file has to be uploaded, which has to be a Comma Separated File (CSV). This file is then parsed by logic inside our API. In order to add the data from the CSV to OpenRemote, it has to be inserted into the PostgreSQL database used by OpenRemote. In the OpenRemote database, all data points have a timestamp, entity id, a key and a value. Here, the timestamp is the timestamp that the value is for, the entity id is the id of the item it related to, in our case the PV system, the key is what the value is (power, performance ratio...) and finally the value is the (numerical) value. In order to obtain the entity id of the PV system the data has to be added to, all entities are retrieved from the OpenRemote database using SQL queries, after which the output is parsed and the correct entity id is taken. After this, the reference yield, field system yield and performance ratio are calculated for the timestamps in the CSV, based on the uploaded data. Finally, all the calculated and uploaded data is inserted into the OpenRemote database using SQL queries, where conflicting data (same timestamp, entity id and key) is overwritten to ensure up-to-date data.

6.2.3 Weather data

The weather data is queried from Open-Meteo using a wrapper. This makes Open-Meteo easily query-able in the rest of the product. A company must pay

for Open-Meteo, so the option to input an API key is present.

6.2.4 Database

The database stores the information about the PV panels as a backup. As we cannot guarantee the data from the database will remain in the same format, we decided to store a json encoding of it, as well as an artificial primary key, and the name used in the DT as key for that PV system.

Chapter 7

Testing

This chapter explains what we have tested, how we tested, and what the results of those tests were.

7.1 Test plan

For every part of the project, tests were made and executed as far as it was possible. The parts of our project where automatic tests were made for in the backend are the API, the digital twin wrapper, the Open-Meteo wrapper, and the KPI calculations. Python unit tests were made for each of these. These tests can be all be executed by running two commands. For the front end no tests were made since the fully developed application OpenRemote is used, so there should be no tests necessary. Next to that, the forecasting was tested manually.

Open-Meteo wrapper

The Open-Meteo wrapper is tested on correctness of the data request and functionality of the data parsing. For the data request, it is tested that the correct API is queried, since there is a forecast API and a historical API. While the forecast API offers a higher data resolution, it does only include data of up to 90 days in the past. Therefore, the API division needs to be tested to ensure the request is sent to the correct API. Furthermore, the validation of arguments is tested, to ensure for example that there is no request that queries further into the future than the bound of Open-Meteo. Furthermore, the processing and response of the request is tested to check that behaves as expected. Then the processing of the response is tested. This is done by checking the response is in the correct format and that it contains data for the timestamps requested.

With this approach, the entire process of receiving data is tested. The correctness of the data is not tested. This is however tested in the KPI calculations. When they appear to be correct, then the data is sensical.

KPI calculations

The calculation of the final system yield is manually tested. The result is a simple division, so it can be checked manually if it functions correctly.

The calculation of the reference yield is tested. First, the validation of arguments is tested. This ensures no nonsensical reference yields can be calculated, for example with negative direct normal irradiance. Second, the calculation itself is tested. This is not done with pvlib as implemented, but a separate manual literature-supported implementation [7] with less inputs. This leads to a less accurate result, but gives an indication what the value ought to be. In reality, the difference is tested to be at most 0.05. Additionally, the reference yield is also evaluated to give accurate results. Furthermore, the results itself are also manually tested, see section 7.2.2.

The performance ratio is tested using the same methodology. This can be seen in section 7.2.2.

All key performance indications are hereby tested. The reference yield is tested vigorously, since it is used in other calculations where it must be sensical. The final system yield is easy to manually test and the performance ratio needs to adhere to the expectations that literature sets.

7.2 Test results

7.2.1 Backend

To test the logic of different components of the backend, we decided to make unit tests. Every time a component was merged with the main branch of the project, we ensured that the unit tests passed successfully. Additionally, after merging the tests and the code, the tests were still valuable for debugging when updates were made which altered the code.

7.2.2 Key Performance Indicators

To assess the validity of the data produced by the calculations of our key performance indicators, the data can be compared against expectations. These expectations are based on literature or the expertise of the supervisor.

Reference yield

The reference yield should be a multiplication of the direct normal irradiance (DNI) as suggested by the supervisor. Direct normal irradiance is namely the most influential for the reference yield as most irradiance comes directly from the sun.

Therefore we analyzed whether this was the case with our calculated reference yield. The sample chosen for this was the month of April in 2025. For this sample, the direct normal irradiance was queried from Open-Meteo and the reference yield data gathered from the product. Then the factor for the

multiplication needs to be determined to let the reference yield be matched as closely to the direct normal irradiance. This was done by selecting the factor which minimizes the sum of differences between the multiplied reference yield and the direct normal irradiance.

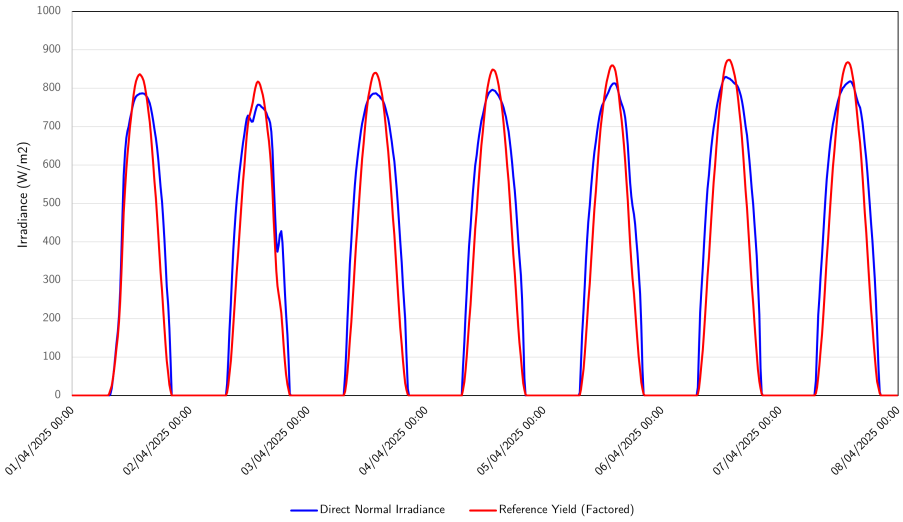

Figure 7.1: Comparison between multiplied reference yield and direct normal irradiance

The distributions can then be compared. A part of the comparison is depicted on figure 7.1. It can be seen that the reference yield roughly follows the direct normal irradiance. It can therefore be concluded that our calculated reference yields appear to be correct. The same result leads from the other data, however, this is not included in fig. 7.1, since it would clutter the figure.

Performance Ratio

The performance ratio of efficient panels should be around 80% [3]. The PV panels at the Ecofactorij are quite high-end and new, so they are expected to have about that performance ratio. To assess the performance ratio, the performance ratio was calculated with all available data in the digital twin on April 10th, 2025. These performance ratio's are depicted in figure 7.2. They seem quite close to 80% which is also the case as their average is 75,98%. Therefore it can be concluded that the performance ratio also appears to be correct. The under-performance of 5% can occur by measurements errors or results of the conditions of the PV panels, however, it is not significant to believe the calculation may be off.

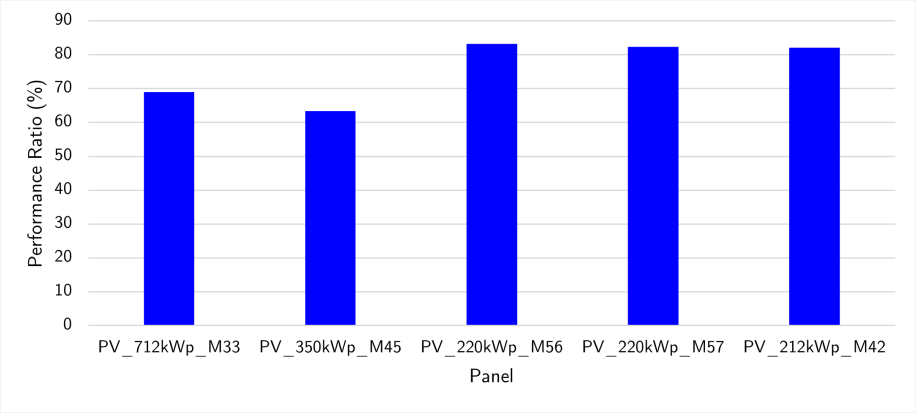

Figure 7.2: Performance ratio per PV panel

7.2.3 Forecasting

We did not create any automatic tests for the forecasting, however we tested the predictions using the digital twin real time data to measure the accuracy. By predicting the field system yield for a day in the past, we could compare it with the values from the digital twin. We plotted the predictions and the real time values in a graph to review the accuracy. For an example, see fig. 7.3 for a comparison of the field system yield, in blue is the prediction, and in orange are the actual values from the digital twin.

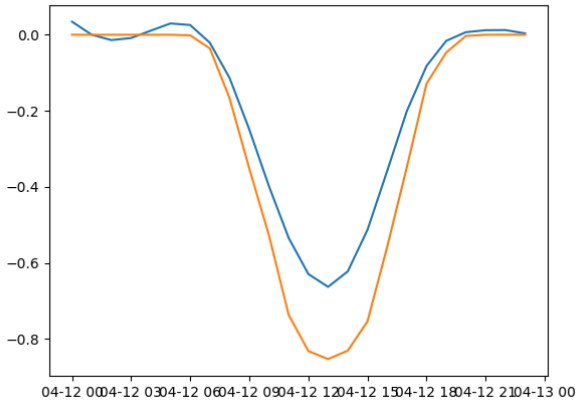

Figure 7.3: Difference between predicted and actual field system yield

Chapter 8

Future planning

In this chapter we mention what improvements or features still could be worked on for this project.

8.1 Forecasting

8.1.1 Making the model more accurate

One of the most evident future improvement is to make the model more accurate. One of the ways to improve the accuracy is by embedding the meters. We are currently using integer encoding, meaning we assign to each meter a numerical value. However, this might lead to the model considering the meters having a numerical relationship, which is not the case since they are distinct. One could try to one-hot encode the meters instead, but we lacked the time to look into this properly.

8.1.2 Extend the forecast to more than 24 hours

Forecasting for the next 24 hours is computational intensive since it requires shifting by $288 * 7 = 2016$ data paints for each 5 minute data entry. Due to its computational insensitivity we decided not to implement this.

8.1.3 Select the correct weather parameters

Investigating the exact relationship of DNI, DHI and GHI could be a topic for future improvements. We only used the DNI, since we considered it to be the most impactful on the field system yield. We based this on evaluating the correlation matrix of the features and our intuition. However, we are not specialists in the area, so further checks would have to happen to find the proper metrics to base the model on, with some scientific evidence.

8.1.4 Retrain model automatically

Adding the possibility to retrain the model automatically could be an improvement of our current work. At the moment, we created a command to train the model but it is not running automatically. A cronjob can be set up to run this command automatically at a specific time interval. We considered implementing this, but it was too computational intensive to run automatically in the background. Our alternative was to save the model locally as a file, resulting in us not needing to train the model every time we made a prediction. Additionally, maybe in the future the real-time data of the digital twin can be used to train the model.

8.2 Performance explanations

A feature that could help the end user better understand the presented data, is by explaining why, for example, the performance ratio is higher or lower than expected. This could be due to the weather conditions or the way the panels are configured. However, we do not have enough knowledge in this area to display these kinds of explanations.

8.3 Database utilisation

At the beginning of the project we decided to use the database to store information about the PV panels, as well as the training data for the forecasting, and the generated results of the forecasting. However, we ended up only storing the information about PV panels. Generating a forecast takes less than a second, so storing it in the database was low priority. However, storing this In general, storing all information we request multiple times from external parties, such as historical weather data, or calculation-heavy results such as the KPI or forecasts. This would mean quicker and less resource-intensive access to the needed data.

Chapter 9

Evaluation

In this chapter we evaluate the entire project regarding the planning, and the division of tasks. Furthermore, we discuss what requirements were fulfilled.

9.1 Planning

We planned the work on the project as described in section 2.5 and we managed to stick to it throughout our module. Some small improvements could have been made such as splitting the forecasting up in smaller tickets. Next to that, some tickets took longer than expected and some additional work needed to be done that was not specified in the tickets before. However, this is to be expected with a development project and considering the experience we have as a team.

9.2 Requirements fulfilled

In the end, we managed to fulfill every requirement except the requirement associated with user story 14 as also described in section 8.1.4. However, since user story 14 was assigned the lowest priority in the project, the fact that it is missing has a minimal impact to the final result.

9.3 Responsibilities

Feature	Person(s) responsible
Digital Twin wrapper	Anna, Erwin, Lu
API	Chris, Erwin, Lu, Kevin
KPIs	Chris
Weather data wrapper	Chris
Database (wrapper)	Lu
Forecasting	Anna, Mihai
Frontend	Erwin, Kevin

Chapter 10

Conclusion

This project offered us the framework to get out of our comfort zone and create something relevant to the outside world too. We learned new skills from managing requirements directly with the client to concepts about neural networks and machine learning. This module offered us the liberty to manage the project ourselves with very loose guidelines, and the deadlines were mostly set by us. We managed to stick to our original plan, something that we think could be because some of us already have some experience working outside of university, which enabled us to create more realistic deadlines and approaches. In addition, we managed to divide the tasks among ourselves quite efficiently, which sped up the process quite a bit.

Like any other project, there were some challenges along the way, one of them being the availability of the digital twin, which caused some delays for our features. Furthermore, since no one had any prior experience with machine learning or neural networks, it was a bit of a challenge to grasp every concept regarding that domain, but we managed to create a product that we are quite proud of. As mentioned in Chapter 8, we had some points that we considered important for enhancing our project, but due to time constraints, we didn't manage to include them in our final product.

Bibliography

[1] Alam Emon, M. S., Ahmad, M. U., and Hasanuzzaman, M. Chapter 2 - solar thermal energy conversion. In *Technologies for Solar Thermal Energy*, M. Hasanuzzaman, Ed. Academic Press, 2022, pp. 25–54.

[2] Anderson, K., Hansen, C., Holmgren, W., Jensen, A., Mikofski, M., and Driesse, A. pvlib python: 2023 project update. *Journal of Open Source Software 8*, 92 (2023).

[3] Ayompe, L., Duffy, A., McCormack, S., and Conlon, M. Measured performance of a 1.72kw rooftop grid connected photovoltaic system in ireland. *Energy Conversion and Management 52*, 2 (2011), 816–825.

[4] Ecofactorij. Ecofactorij. https://ecofactorij.nl/. Accessed: April 16th, 2025.

[5] Ecofactorij. Mooi Project - Ecofactorij. https://ecofactorij.nl/project/mooi-project/. Accessed: April 16th, 2025.

[6] Nou, J., Chauvin, R., Thil, S., and Grieu, S. A new approach to the real-time assessment of the clear-sky direct normal irradiance. *Applied Mathematical Modelling 40* (03 2016).

[7] Perry, M. New product support: Bifacial plane of array (bpoa), solar energy's newest irradiance expression. https://www.campbellsci.es/blog/albedo-resource-assessment, Dec 2018. Accessed: March 31st, 2025.

[8] Reinders, A., Verlinden, P. J., Van Sark, W., and Freundlich, A. *Photovoltaic solar energy*. John Wiley & Sons, 2 2017.

[9] Satpathy, R., and Pamuru, V. *Solar PV Power*. Academic Press, 2021.

[10] Srivatsavaya, P. LSTM — Implementation, Advantages and Diadvantages, Oct. 2023.

[11] VR, A. Solar irradiance calculation - everything you need to know. https://arka360.com/ros/solar-irradiance-calculation/, Dec 2024. Accessed: March 31st, 2025.

[12] ZUNSOLAR. Solar panels alignment: Azimuth and zenith orientation. https://www.zunsolar.com/blog/solar-panels-alignment-azimuth-and-zenith-orientation/, Aug 2021. Accessed: April 1st, 2025.

Appendix A

Manual

A.1 Introduction

This is the manual to the design project for calculating the KPI's of the Eco-factorij.

A.2 Prerequisites

For the project to work, some conditions need to be fulfilled.

1. Python installed
2. Internet connection
3. Open-Meteo reachable
4. API with PV system data reachable

A.3 Setup

Start by installing Docker[1] and Python[2] if they are not already installed.

A.3.1 Back end

1. Install Docker.
2. Clone the back-end repository.
3. Copy the `template.env` and rename it to `.env`.
4. Modify the `.env`:

[1] https://docs.docker.com/get-started/get-docker
[2] https://www.python.org/downloads

(a) Set the `DT_USER` and `DT_PASSWORD` with the username and password for the connection to the digital twin.

(b) For commercial usage, set the `Open-Meteo_APIKEY` value to the api key you obtained from Open-Meteo.

(c) Remove variables that are not set to ensure to ensure that no empty variables are set.

5. First, build the docker container with `make build`, or with the following
   - `docker build main -t design-project-group-8-main:latest`
   - `docker build database -t design-project-group-8-database:latest`
6. Start the docker with `make compose` or with `docker compose -p backend up -d`.

A.3.2 Front end

1. Clone the front-end repository.
2. Run `docker-compose pull` to update the necessary containers.
3. Run `docker-compose -p openremote up -d` to start the docker container for the front end.
4. To open OpenRemote, use the URL `https://localhost`.
5. The default admin username is `admin` and the default admin password is `secret`.
6. Change the admin credentials before importing any sensitive data, as explained in section A.4.4.
7. Make sure the back end is running before running the script in the following step.
8. To import the PV panels to OpenRemote, we first have to set the environmental variables by setting the `.env`. First, Copy the `template.env` and rename it to `.env`.
9. Next, we set the other variables. An example of this can be see in figure A.1.

   (a) For the variable `OPENREMOTE_LOCATION`, fill in the URL of the OpenRemote dashboard, by default this is `https://localhost`.

   (b) In the `SERVICE_ACCOUNT_NAME` and `SERVICE_ACCOUNT_PASSWORD`, the user name and password of a service user have to be filled in. To see how to create a service user, check out section A.4.4.

   (c) For the `API_URL`, the URL of the backend has to be filled in, by default this is `http://localhost:8080`.

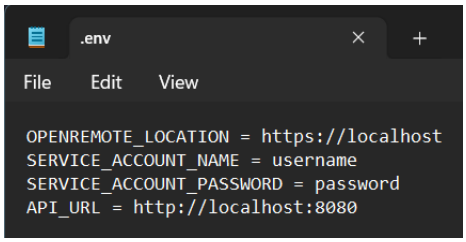

Figure A.1: .env file

10. Install the requirements in the requirements.txt file with the command `pip install -r requirements.txt`.
11. Run the `create_assets.py` script with the command `python scripts/create_assets.py`.

A.4 OpenRemote Usage

A.4.1 Map

The home screen of OpenRemote shows a map of the Ecofactorij with the PV panels as icons on the map, as shown in figure A.2.

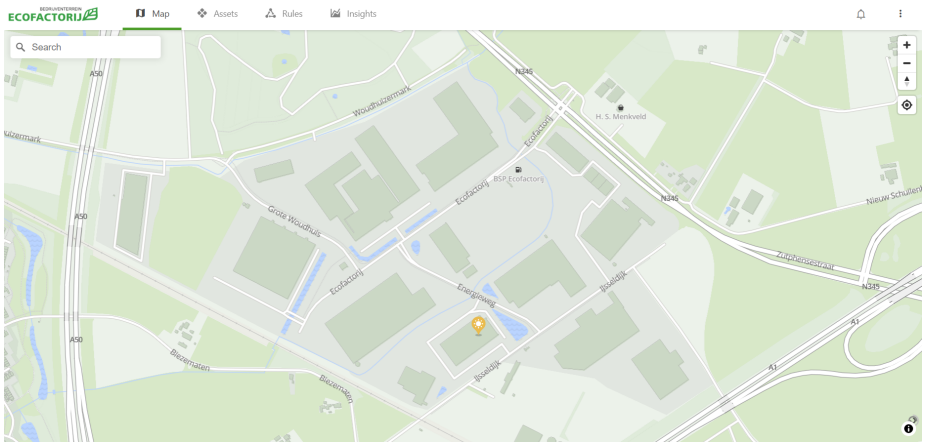

Figure A.2: Map screen

To view the detail of a PV panel, the user can click on the sun icon and the information of the panel will be shown as in figure A.3. To get more information of an asset you can click on the bottom-right `VIEW` button, which will direct users to the assets page.

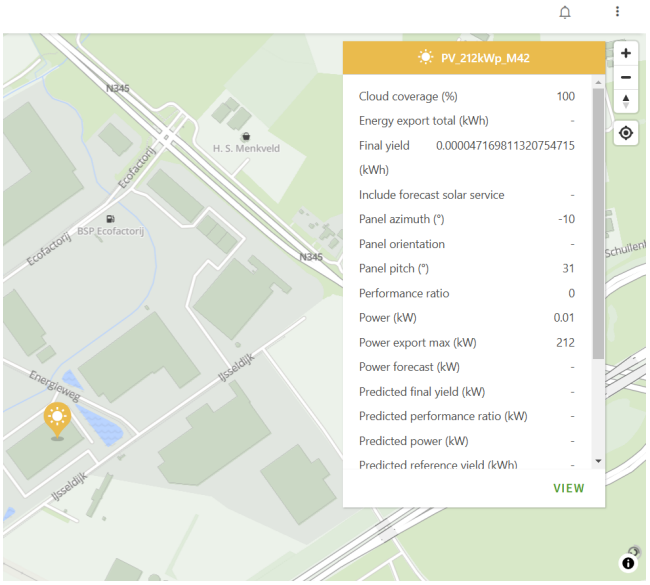

Figure A.3: PV detail pop up

A.4.2 Assets

With the assets page, the user can see a more detailed overview of each asset, and view a line chart of the historical or predicted data of the PV panels. The assets page can be seen in figure A.4.

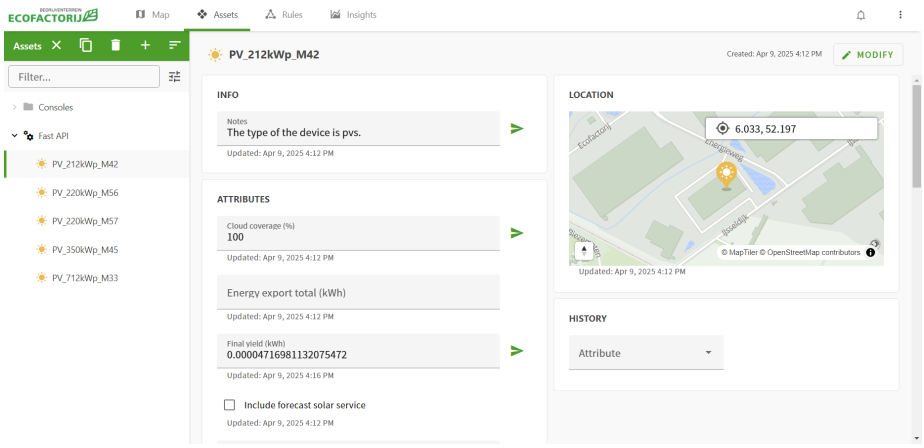

Figure A.4: Assets page

To view a line chart of the historical or predicted data of an asset, select an attribute in the bottom-right history panel. In this panel, the user can also select the time frame of the line chart and the end date. So if the user wants to see data from last week you have to select the time frame `Week` and set the end date `Ending` to today. An example of a line chart can be seen in figure A.5.

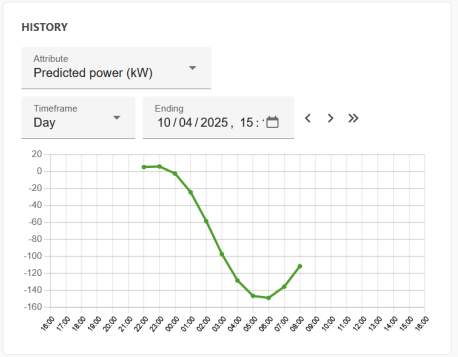

Figure A.5: History panel

A.4.3 Realms

Each Realm in OpenRemote has their own users, assets, rules, and UI styling. The only realm in our project is the `master` realm and it is also the only one needed.

A.4.4 Users

To manage the users of a realm, click on the three dots on the upper right corner of the screen and click on `Users` as seen in figure A.6.

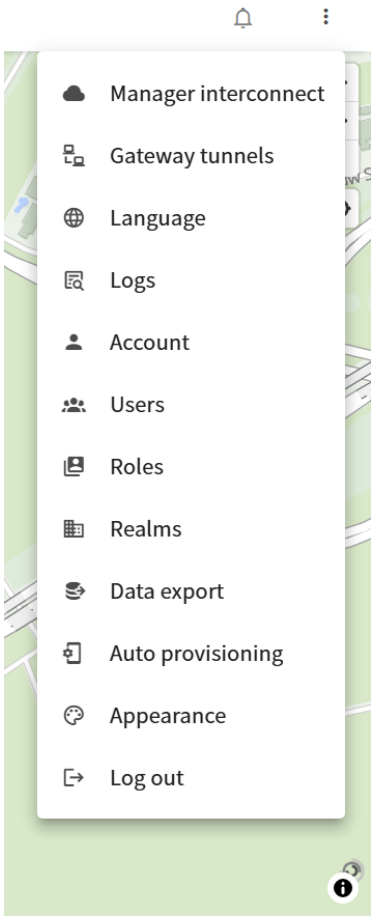

Figure A.6: More options menu

The user will be directed to the screen shown in figure A.7. OpenRemote has an admin user who can manage the other users by default. Other users can be added, modified, and deleted on the users page. There are regular users and service users. The regular users are for logging into the OpenRemote dashboard and the service users are used to run the `create_assets.py` in the setup.

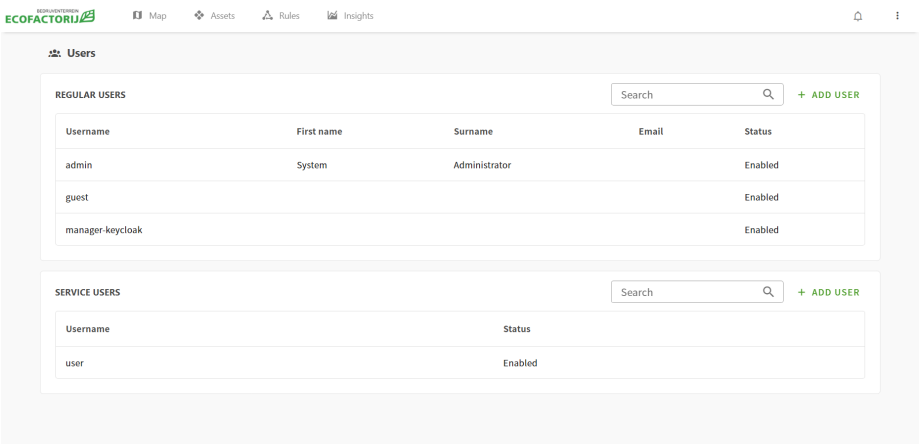

Figure A.7: User page

Changing the credentials of the admin user

To change the default credentials of the admin user, click on the admin user and type a new password in the password field, repeat the password and click `SAVE`.

Creating regular users

To create a regular user, click on `ADD USER` in the top right in the `REGULAR USERS` panels as shown in figure A.7. Then you will see the screen in figure A.8. In this screen, you only have to insert the username and password, and repeat the password. Before clicking create, it is important to set the permissions for the users. Here you can set what a user can see and modify, and for which assets they can. Furthermore, there is a realm role and a manager role. The realm role `Super admin` gives the user access to create and manage realms. The realm role `Restricted user` merely gives a user this role. With the manager role you can give read and write permissions in batch to a user.

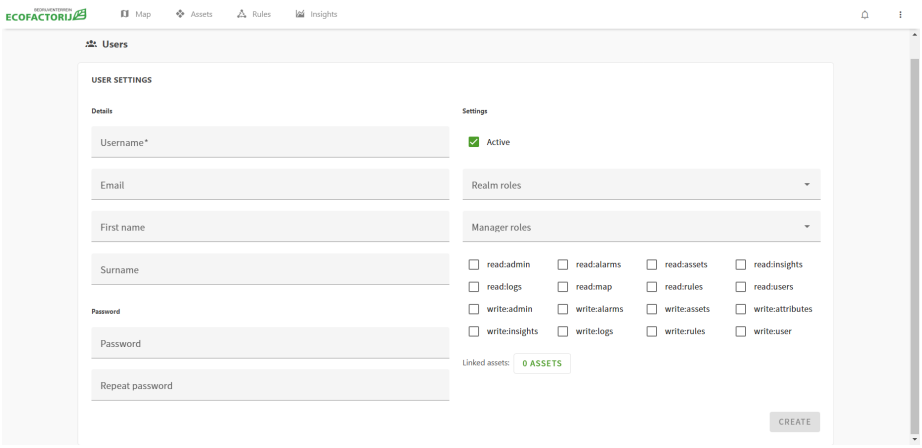

Figure A.8: Create a regular user page

Creating service users

To create a service user, click on `ADD USER` in the `SERVICE USERS` panel. Then you will see the screen in figure A.9. Here, you only need to fill in the username. Furthermore, check the permissions `read:assets` and `write:assets` for the `create_assets.py` script to work. After clicking create, a password will be generated which you have to store in the `.env` file together with the username.

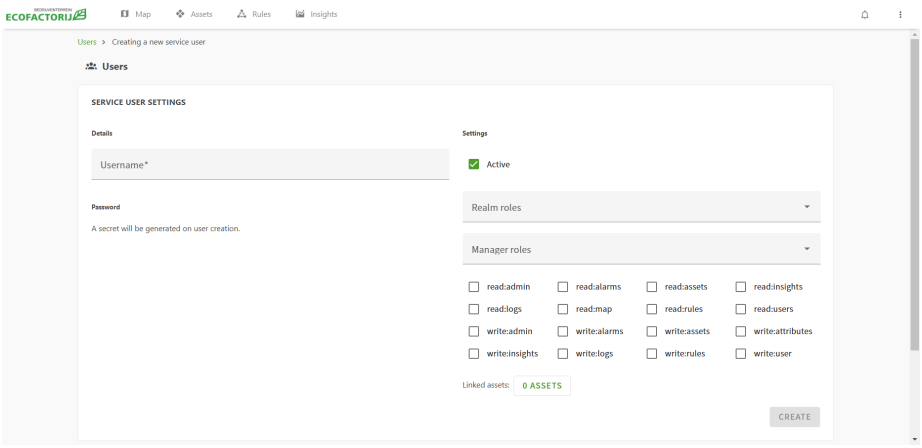

Figure A.9: Create a service user page

A.4.5 Insights

In the insight page, the user can create their own dashboards and add widgets to it. By clicking `+` as shown in figure A.10, the user can add a new dashboard.

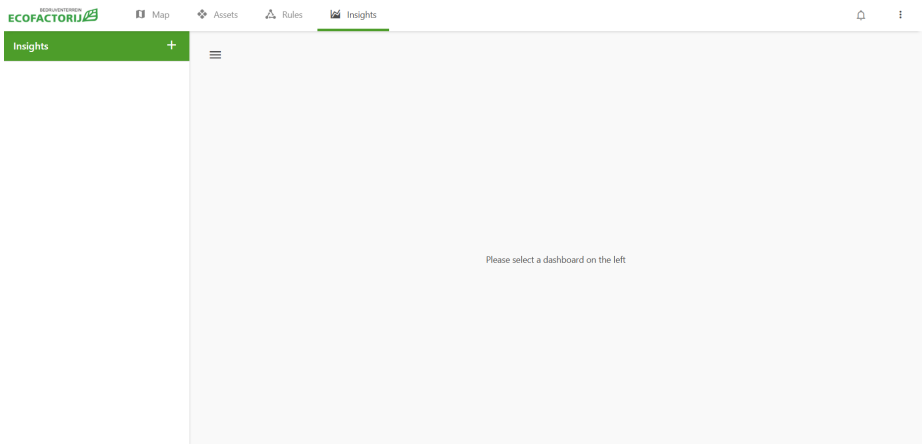

Figure A.10: Insight page

Then to add widgets, users can drag them from the right panel onto the dashboard, as shown in figure A.11.

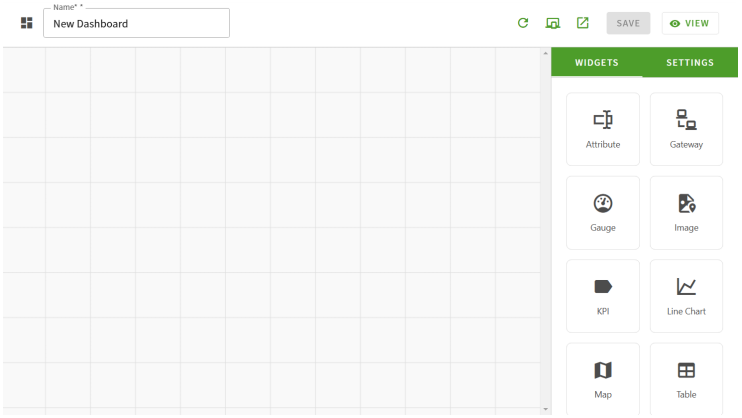

Figure A.11: Dashboard edit screen

To graph an attribute in the dashboard, the `Line Chart` widget should be dragged onto the dashboard. After that, to add an attribute, click on the dragged line chart and click on `+ATTRIBUTE` as shown in figure A.12.

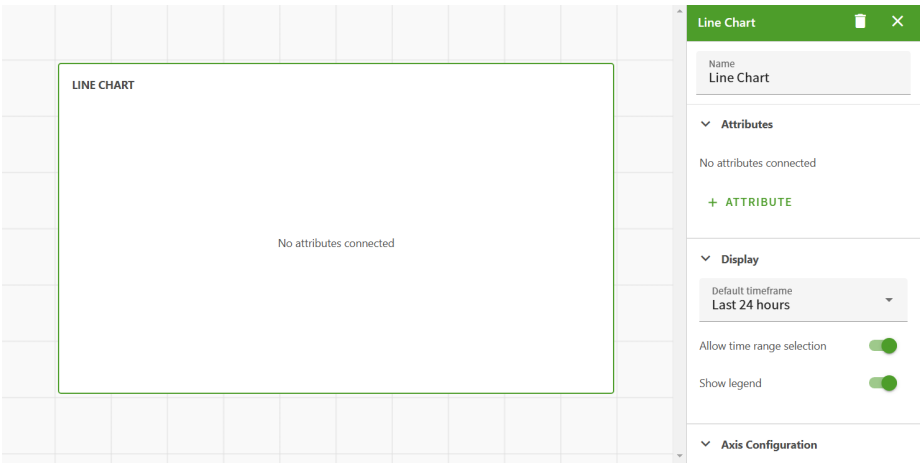

Figure A.12: Line chart edit panel

In the new pop up shown in figure A.13, the attributes of each PV panel can be chosen.

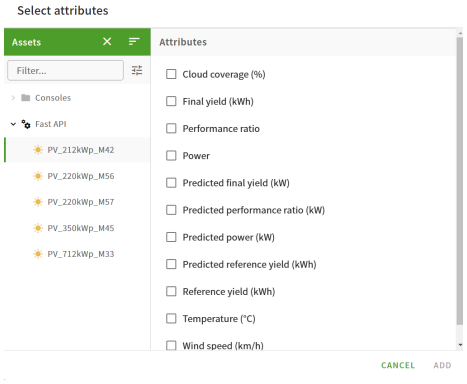

Figure A.13: Attribute selection pop up

After selecting the desired attributes to be chose, click `ADD`. The default time frame shown in the graph can be adjusted in the `Display/Default time frame` section, shown in figure A.12, when the line chart is selected. To view a certain time range in the graph, click on the date picker icon shown in figure A.14, and adjust the start and end date as shown in figure A.15. Finally, save the dashboard by clicking the `SAVE` button.

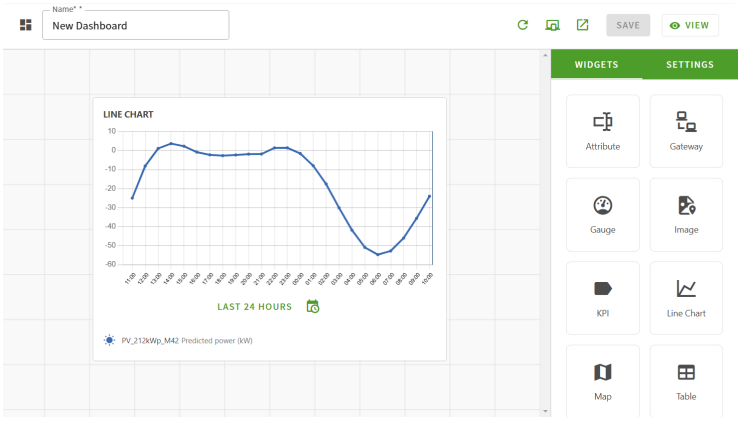

Figure A.14: Line chart example

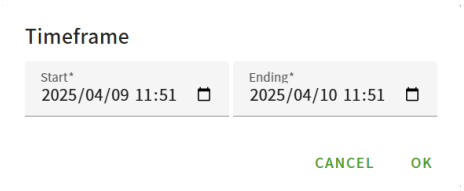

Figure A.15: Date selector

A.4.6 Rules

The rules page lets users view and create rules. Only users with the read:rules or write:rules permission can view or modify rules. To open the rules page, click `Rules` in the top navigation bar. In the rules page as shown in figure A.16, click on the + icon to create a new rule.

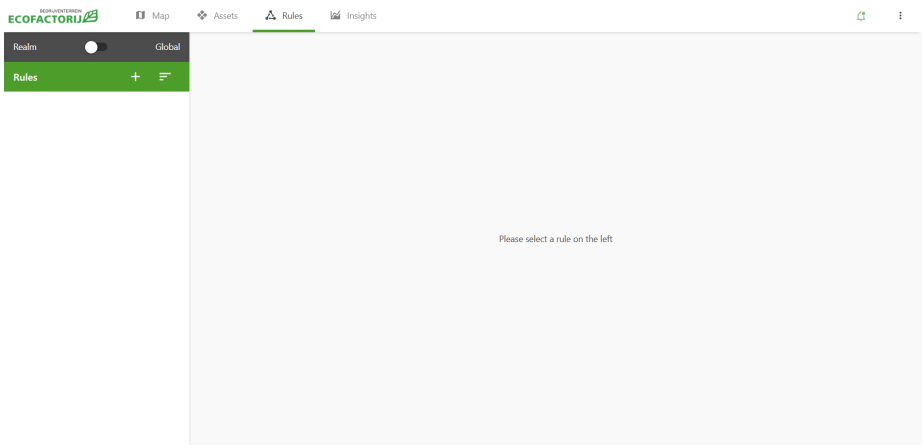

Figure A.16: Rules page

As an example, we will make a rule to create an alarm when the performance ratio is below 70%. To create a rule for alarms, select `When-Then`. In the When-Then edit screen, shown in figure A.17, users can type a name for the rule, and add a "When" and "Then" condition.

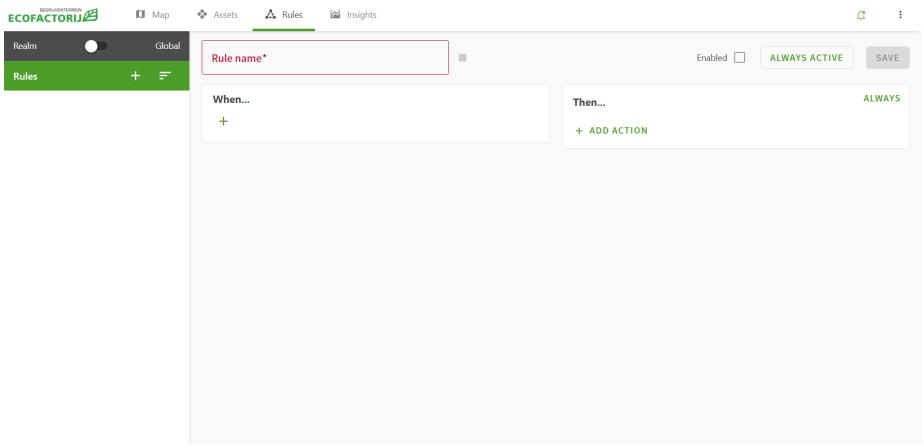

Figure A.17: When-Then edit screen

For the "When", select `PV Solar Asset` from the drop down menu. Select a PV panel as asset and select the performance ratio as attribute. For the operator, choose "Less than" and type 0.7 in the performance ratio field. In the "Then" panel, select `Alarm` as action. An example of the filled in Rule can be seen in figure A.18. Finally, click on save.

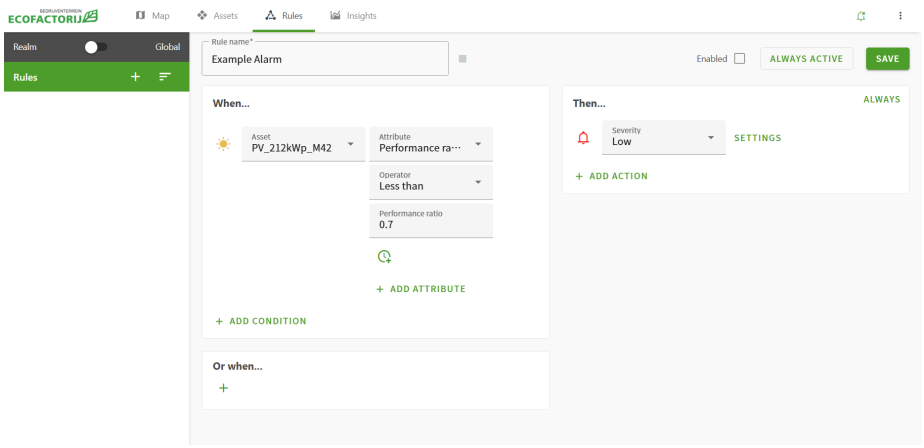

Figure A.18: When-Then rule example

A.5 Forecasting

The forecasting consists of two parts: the model and making the prediction. For training and predicting we use 5 minute data, so 288 data entries per day. To train and predict correctly all 288 data entries should be available, this also applies to the data in the digital twin.

A.5.1 Setting up the environment variables

We have the following variables for our forecasting model:

- `STEPS` - the amount of steps that are used to predict (288 at the moment = 24 hours of prediction)
- `NR_FEATURES` - the amount of variables the model is using
- `FILES_PATH` - general path to the `/files` folder
- `COLUMN_CSV` - the column that should be used from the CSV files to get the kwH for the field system yield
- `DATAFILE_PATH` - path for the data entries
- `SAVEMODEL_PATH` - path where the model should be saved
- `LAT` - latitude of the PV systems
- `LONG` - longitude of the PV systems

We provide default values in the file `main/forecasting/forecasting_model.py`, in the `__init__` constructor, but if the default have to be changed we encourage to specify the changes in the `.env` file.

A.5.2 Adding new data to the model

To add new data to the model, there are a couple requirements. Firstly, the integrity of the data should be correct. This means that there should not be any gaps in time stamps for the data used in the model. We trained with data from 2024, if data needs to be added it needs to be added starting after the last entry point in the `.csv` for every 5 minutes consecutively. Secondly, the data should be added in the form of a `.csv`. The naming should correspond to the following format: `METER_ADDITIONAL_5m.csv`, so for example: `M56_GROLLEMAN_ZON_M56_5m.csv` would suffice. However, `M_56_GROLLEMAN_ZON_M56_5m.csv` would not suffice. The `.csv` needs to be uploaded to the location specified by `DATAFILE_PATH`, that can be found in the `.env` or to the default location.

For every data entry, there should be available weather data. We use Open-Meteo to gather weather data for 2024. We specified the following attributes:

- `Time`
- `Temperature`
- `DNI`
- `DHI`
- `GHI`

Some processing needs to happen on the `.csv` such that only the attributes mentioned above remain in the `.csv`. The resulted file should be saved in `../files/weather_data`.

A.5.3 Training the model

To train the model, we created a `makefile train_model` command, that when running, uses the files specified in the `DATAFILE_PATH` to train the model. Depending on the hardware, the training could take some time. We recommend using a system with dedicated GPU for training the model.

A.6 Importing historical data

As OpenRemote does not natively support adding historical data, we have created an endpoint for this. In order to upload a `.csv` file to OpenRemote, create a `POST` request to the `API_URL` as seen in Section A.3.2, on the path `/historical/<panel_name>`. The file should be added as `form-data` in the body, using the key `file` and as value the file. An example of how this would look in Postman can be seen in Figure A.19

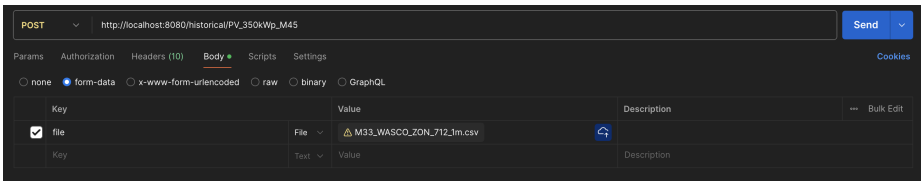

Figure A.19: Example of uploading historical data