CZECH TECHNICAL
UNIVERSITY
IN PRAGUE

F3

Faculty of Electrical Engineering

Department of Computer Science

Master's Thesis

# A system for measured data processing, evaluation of faults and predictive maintenance of a photovoltaic power plant

**Bc. Ondřej Tůma**

Open Informatics, Software Engineering

May 2024

Supervisor: RNDr. Miroslav Kulich, Ph.D.

# MASTER‘S THESIS ASSIGNMENT

## I. Personal and study details

Student's name: **Tůma Ondřej** Personal ID number: **491867**

Faculty / Institute: **Faculty of Electrical Engineering**

Department / Institute: **Department of Computer Science**

Study program: **Open Informatics**

Specialisation: **Software Engineering**

## II. Master’s thesis details

Master’s thesis title in English:

**A system for measured data processing, evaluation of faults and predictive maintenance of a photovoltaic power plant**

Master’s thesis title in Czech:

**Systém pro zpracování naměřených dat, vyhodnocení poruch a prediktivní údržbu fotovoltaické elektrárny**

Guidelines:

For accurate and efficient diagnosis of a photovoltaic power plant (PVP), it is advisable to have a software tool that processes input data in the form of a model and operational data of the plant. This tool should analyze the impact of faults on overall operation and output of the PVP and recommend the most suitable maintenance approach for the PVP, considering the maximization of delivered power, economic efficiency, and impact on the energy system. The task for the student will be to create the core of such a tool in the following steps:
- Familiarize yourself with the open-source electronic circuit simulator ngspice.
- Make a review of current solutions providing diagnosis of PVPs.
- Design and implement an interface for simulating PVP in ngspice.
- Design and implement a web-based user interface for managing the simulation of the PVP and processing results from this simulation. Key functionalities: display of an electronic and physical layout of PVP elements (modules, converters, strings) together with their current state, display of detailed information about modules (thermal images, results of VA and performance characteristics simulation, maintenance recommendation), management of PVPs, users, and module types.
- Design the deployment process, deploy the system on a specified FEL server and verify the tool functionality on the data from a real PVP provided by the superivisor.
- Properly document the implemented software from both programming and user perspectives.

Bibliography / sources:

[1] Nallapaneni Manoj Kumar, K. Sudhakar, M. Samykano, and V. Jayaseelan. On the technologies empowering drones for intelligent monitoring of solar photovoltaic power plants. 2018. International Conference on Robotics and Smart Manufacturing (RoSMa2018).
[2] Giovanni Tanda, and Mauro Migliazzi. Infrared thermography monitoring of solar photovoltaic systems: A comparison between UAV and aircraft remote sensing platforms. Thermal Science and Engineering Progress. 2024, 48 102379.
[3] NGspice Project. NGspice: Open-source Spice Circuit Simulator. 2024-01-01. https://ngspice.sourceforge.io/.

Name and workplace of master's thesis supervisor:

**RNDr. Miroslav Kulich, Ph.D. Intelligent and Mobile Robotics CIIRC**

Name and workplace of second master's thesis supervisor or consultant:

Date of master's thesis assignment: **05.02.2024** Deadline for master's thesis submission: **24.05.2024**

Assignment valid until: **21.09.2025**

| RNDr. Miroslav Kulich, Ph.D.<br>Supervisor's signature | Head of department's signature | prof. Mgr. Petr Páta, Ph.D.<br>Dean's signature |
|---|---|---|

## III. Assignment receipt

The student acknowledges that the master's thesis is an individual work. The student must produce his thesis without the assistance of others, with the exception of provided consultations. Within the master's thesis, the author must state the names of consultants and include a list of references.

| . | |
|---|---|
| Date of assignment receipt | Student's signature |

# Acknowledgement / Declaration

I would like to express my sincere gratitude to my supervisor, RNDr. Miroslav Kulich, Ph.D., for his excellent guidance and support throughout this thesis. I would also like to thank Ing. Karel Košnar, Ph.D., for introducing me to the DiPreFE project, which provided a solid foundation for my work. Additionally, I am grateful to the researchers who introduced me to the photovoltaic industry, offering essential insights and perspectives. A special thanks to my family and my partner for their unwavering support and encouragement during this journey.

I declare that the presented work was developed independently and that I have listed all sources of information used within it in accordance with the methodical instructions for observing the ethical principles in the preparation of university theses.

Prague, date 24. May 2024

.....

# Abstrakt / Abstract

Tato práce si klade za cíl vyvinout softwarový nástroj pro efektivní diagnostiku a údržbu fotovoltaických elektráren s využitím simulátoru obvodů Ngspice společně s daty shromážděnými prostřednictvím termografie za pomoci autonomních dronů. Vytvořený nástroj bude analyzovat dopad poruch na celkový výkon fotovoltaických elektráren a doporučovat optimální strategie údržby za účelem maximalizace výstupního výkonu při zajištění ekonomické efektivity.

**Klíčová slova:** software, web, solární, elektrárna, údržba, termografie, ngspice, simulace, dron

**Překlad titulu:** Systém pro zpracování naměřených dat, vyhodnocení poruch a prediktivní údržbu fotovoltaické elektrárny

This thesis aims to develop a software tool for effective diagnostics and maintenance of photovoltaic power plants using the Ngspice electronic circuit simulator in conjunction with data collected through drone thermography. The developed tool will analyze the impact of faults on overall power plant performance and recommend optimal maintenance strategies for maximizing power output while ensuring economic efficiency.

**Keywords:** software, web, solar, power, plant, maintenance, thermography, ngspice, simulation, drone

# Contents /

# Tables / Figures

# Chapter 1
# Introduction

The demand for renewable power sources has been increasing and the production of renewable energy has doubled over the last 10 years. Solar energy is among the fastest-growing renewable sources, with over 3000% growth[1], while the module installation cost per watt has decreased by over 80% in the same period[2]. With such a significant decrease in the deployment cost, the relative cost of operation and maintenance has increased significantly. After a solar power plant is deployed, the newly installed modules might experience defects that affect the overall power plant performance. The power plant monitoring systems usually cannot pinpoint issues on individual inverters, and the defect discovery process is mostly manual. Inspections and maintenance are necessary at regular intervals. For power plants spanning large areas, this process is expensive due to the human labor involved and the loss of generation capacity.

One of the currently used solutions that attempts to make the maintenance process more efficient is thermography. Certain types of module failures can be recognized with an infrared (IR) camera due to heat accumulation on the broken cells. This results in a distinct feature visible in the IR spectrum. In most cases, the task is currently performed by drones. The drone operators fly the drones equipped with IR cameras over the solar plant, periodically capturing images. The results are then processed either manually or by automatic reporting software.

Commercial software for this purpose, with the support for analysis of the IR images paired with geographic information system (GIS) data, is available. The availability and current state of software for predictive solar power plant maintenance based on IR drone imagery that would also take into account the electrical schema of the power plant and calculate the effect of individual module defects with regard to the rest of the circuit is unknown. One accomplishment of this thesis is the analysis of the availability of such software. As part of the *DiPreFE* [7] research project, the Intelligent and Mobile Robotics Group (IMR) at the Czech Institute of Informatics, Robotics and Cybernetics (CIIRC), Czech Technical University (CTU) in Prague, has proposed a similar method for the inspection of solar power plants using a drone equipped with an IR camera.

The primary goal of this thesis is to build, test, and deploy analytical software to support the proposed solution. The system's main innovation is the integrated simulation capability. The system automatically simulates the circuits based on the failures found in the imagery, with the ability to evaluate the individual module's impact on the rest of the circuit and estimate the power output based on the current state. Based on this, the system is able to estimate the monetary loss over the expected lifespan of the power plant and recommends appropriate maintenance action. This further helps with understanding the impact of the damages and increases the efficiency of the damage as-

[1] `https://ourworldindata.org/renewable-energy`
[2] `https://ourworldindata.org/grapher/solar-pv-prices`

sessment. The built software also includes a standard set of features for navigation and exploration of the captured data, together with the necessary management capabilities.

Apart from providing a fully functional deployment of a software solution for further usage and research by the IMR at CIIRC, the main contributions are the following:

- Evaluation of existing solutions for the thermography analysis of photovoltaic power plants (PvPs).
- Demonstration of the usage of Ngspice simulator for predictive maintenance of solar power plants.
- Evaluation of Ngspice WebAssembly compilation.
- Development of a Node.js application. The application comprises two services and four deployable artifacts with communication over a message queue. The built simulation software solution is vertically scalable.
- Development of a React.js client application communicating over REST API. The application features a web-based user interface, data pdf reporting, advanced table filtering, multitenancy, and role-based access control.
- Implementation of an automated deployment process for a multi-artifact application on a dedicated server using Docker in combination with Ansible.

The built software is expected to be further developed and used for additional research of the drone-based thermography of PvPs. Furthermore, it should serve as the core for an analytical tool offering a wide range of applications in the PvP industry and enabling further research at CIIRC.

# Chapter 2
# Prerequisites analysis

This chapter aims to provide the necessary domain context. This mainly includes the PvP industry terminology and the schematic working of individual components used in the industry. Another vital prerequisite is an understanding of the DiPreFE project, which produces the input data for the built software. Further prerequisites presented in this chapter include a fundamental understanding of the Ngspice circuit simulator used as the core simulation tool in the built software.

## 2.1 Photovoltaic systems

In order to create software that improves the efficiency of processes in a particular domain, the necessary prerequisite is an in-depth understanding of the domain's core principles. The prerequisites to building software capable of simulating a solar power plant are an understanding of the operating principles and the inner workings of individual electrical components of a power plant. Understanding the domain-specific terminology is necessary to properly describe the problem and communicate with the stakeholders.

### 2.1.1 Basic operations

The basic operating principle of a solar power plant is the conversion of absorbed sunlight into electricity. There are a lot of components between the actual component generating the electricity and the final connection to the grid. For the purpose of this thesis, the components that are important for the simulation and related to the circuit inside the power plant are going to be explained.

### 2.1.2 Components

The individual schematic components are described using industry terminology. Their understanding is essential for effective software design and communication with the researchers, who will later be providing the schematics for electronic circuit simulations. The common terminology and component hierarchy are necessary for automated circuit schema generation and database schema design.

#### Photovoltaic cell

The photovoltaic cell is the core building block of a power plant. It is the component that absorbs the sunlight and generates the electricity. It is currently the smallest unit of measure for failure detection, where we can confidently say if a cell is broken or not based on thermography data.

### Substring

A substring is a serial connection of cells inside of a module (described later). Failure inside of a single substring has the potential to highly affect other cells in the same substring. Failure inside a single substring generally does not have to affect other substrings within the same module.

### Module

Solar modules are the smallest practical building blocks of a power plant, meaning that this is the smallest component that is separately handled (bought and replaced as part of maintenance).

From a schematic point of view, it is an assembly consisting of multiple Substrings. From a physical point of view, the module is an enclosure that is able to protect the raw cells from the environment and provide mechanical strength. The schema of a module and its subcomponents can be seen in Figure 2.1.

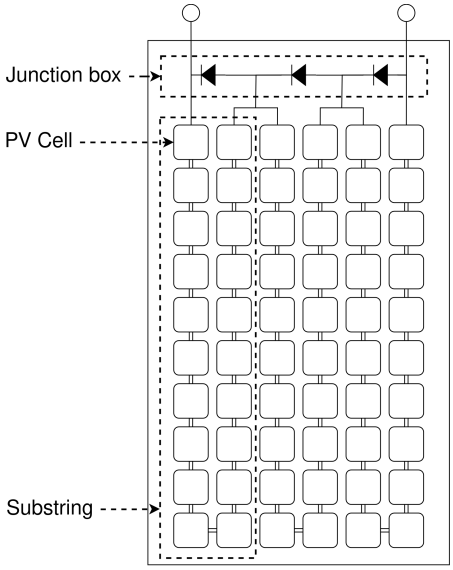



**Figure 2.1.** Module schema

### String

A string is a serial connection of modules. A broken module within a string has the potential to diminish the power output of the whole string, depending on the particular module type.

### Inverter

Inverter is a component that converts the direct current (DC) generated by the photovoltaic cells into alternating current (AC). A single inverter can be connected to multiple strings connected in parallel. A failure within a string might affect the power output of other strings connected to the same inverter, depending on the particular inverter type used. From the schematic point of view, it is the failure boundary, meaning that any failure that happens below an inverter, from a hierarchical point of view, does not spread to the rest of the power plant. We can, therefore, assume that the total

output of the power plant is the sum of individual inverter outputs. For the purpose of the simulation, an inverter is simply a circuit connecting multiple strings. A simplified schema of the connection between inverter and module strings can be seen in Figure 2.2.

#### Junction box

A junction box is a part of a module where the substrings are connected and the place where external connectors are placed. The external connectors are used to connect the photovoltaic (PV) module to the external electrical circuit, enabling the transfer of generated electrical power to an inverter or other load. Usually, it is an enclosure on the back of the PV module. The junction box can usually be seen in the thermography images, appearing as a hotter area at the top or bottom center of the PV module.

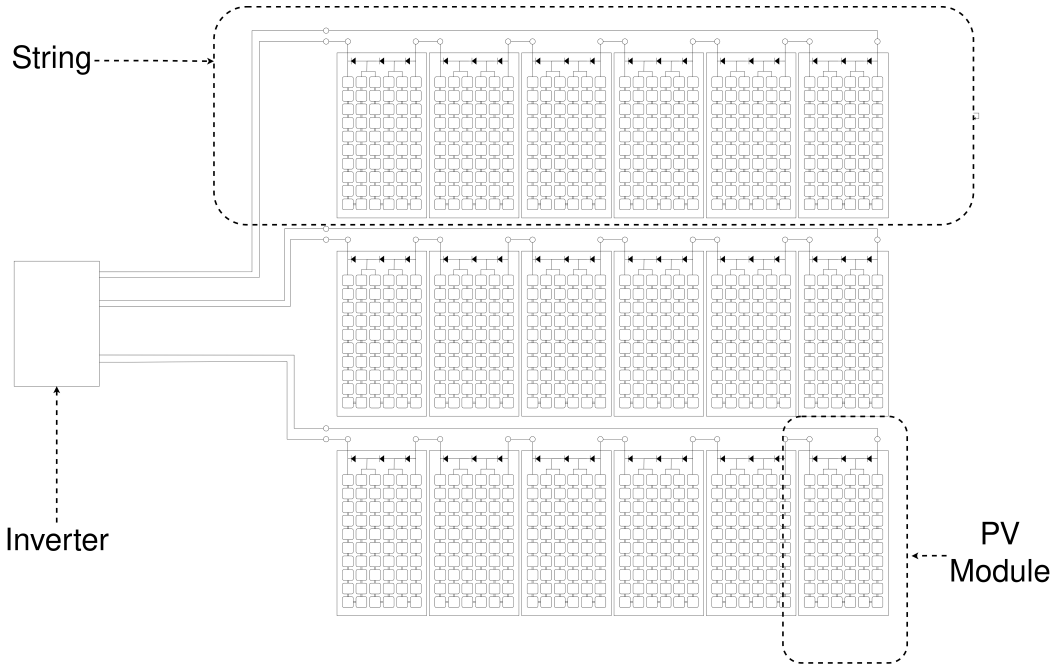
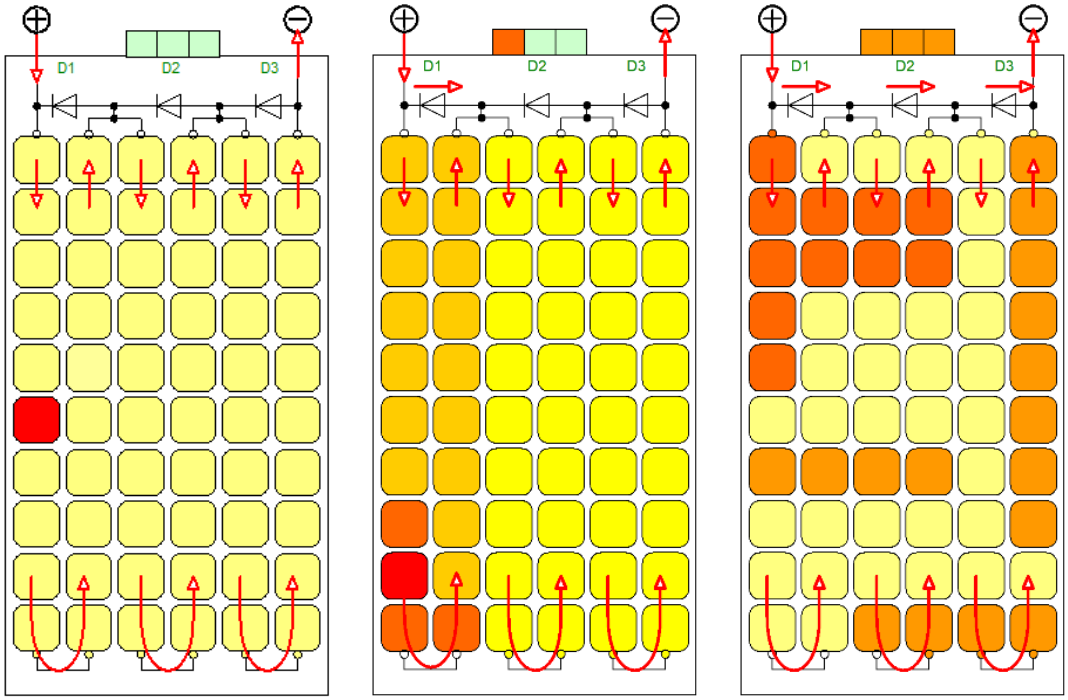



**Figure 2.2.** Inverter schema

### 2.1.3 Failure types

Using thermography, it is possible to distinctly identify temperature differences between individual cells, known as hotspots. This identification ability, paired with a failure-type classification based on the distribution of hotspots, allows for automatic maintenance action recommendations and future failure development prediction based on the existing failure-type knowledge.

Figure 2.3 depicts a few examples of detected failure types, with known future development and performance implications, detected based on the temperature differences between individual cells.




**a)** single hotspot **b)** multi hotspot **c)** chessboard

**Figure 2.3.** Module failure examples [8]

### 2.1.4 Thermography

Digital thermography is the process of capturing image data using a camera capable of capturing in the infrared spectrum. In the context of PV modules, it is possible to detect certain types of anomalies based on the temperature differences between individual cells.

To detect such differences, a basic measurement using a handheld thermal camera can be used. It is possible to assess the results visually. For an effective scaling, a better method is necessary. One of the most popular ones is using a drone-mounted camera. The drones can be scheduled to fly autonomously over the solar plant, periodically capturing images, and pairing them with geospatial data using GPS. The images have to be later analyzed by specialized software.

## 2.2 DiPreFE project

This section takes a detailed look at the *DiPreFE* [7] research project. It mainly elaborates on the project motivation and the input and output components of the project while briefly introducing the inner workings. The output data and database will be necessary for this thesis, as they are used as an input into the newly-built system.

### 2.2.1 Project assignment

The goal of this project is to design, develop, and verify a prototype system for predictive maintenance of a photovoltaic powerplant capable of periodic thermographic and visual

inspection of PV modules making use of UAVs during the operation time of PP. The system is able to predict module failure and evaluate the influence of this failure on the power production of PP. The system for predictive maintenance allows the proposal of optimal actions to maximalize power production and economic effectiveness. Partial goals are [9]:

- Design, development, and verification of specialized methods for accurate navigation and location of unmanned aerial vehicles
- Methods for the collection of thermographic and visual image data, their fusion, and subsequent evaluation to detect potential defects of PV modules with the possible addition of a detailed picture of the expected defective module
- Estimation of the impact of the module defect on the performance, when the module will be dismantled and measured in laboratory conditions
- Creation and maintenance of a catalog of module defects with a specific signature in the visual and thermal area and the measured impact on their performance
- Statistical temporal models of progression of individual types of defects.

### 2.2.2 Introduction

The most important aspect of the project, as far as this thesis is concerned, is the output data format. The data will be in the form of a single SQLite database file, accompanied by a set of detailed thermal images, one for each module. The resulting database schema can be seen in the Figure 2.4.

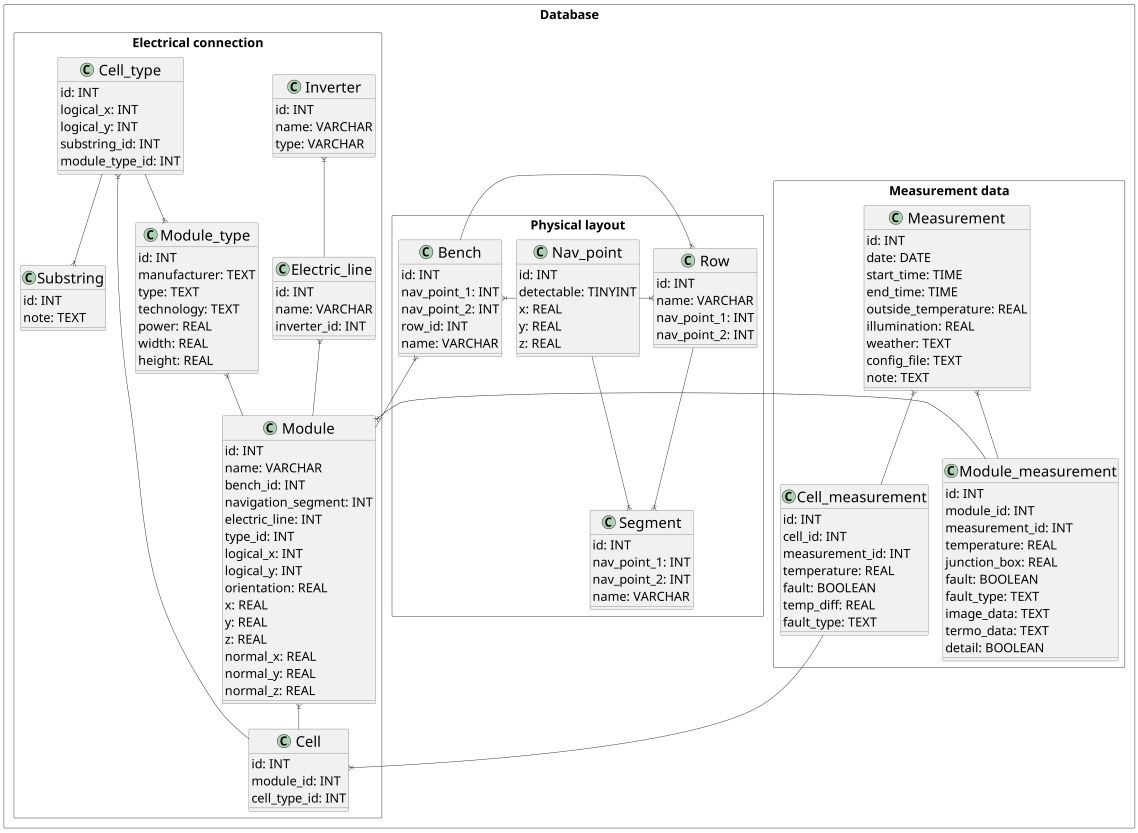



**Figure 2.4.** Database schema

The individual processes that need to happen to get the output database are described in the following sections. Their understanding is not strictly necessary from a software development standpoint, but it plays a crucial role from a product development standpoint. The final software built as part of this thesis is used to accompany the following workflows. Therefore, their understanding is necessary to properly understand the problems the users might be facing.

### 2.2.3 Flight

The flight consists of a series of high-altitude flights to chunk the area into smaller sections and take a set of overview images. These images cover the whole plant and include the GPS coordinate reference points. The pictures are processed using the OpenSfM toolkit[1]. The outcome is a 3D model of the whole power plant. The pictures from the OpenSfM toolkit are then segmented using the U-Net model, which detects the actual modules and benches in the pictures. The high-altitude flight is performed as an initial step when planning the monitoring of a power plant. The outcome of this step can be seen in Figure 2.5.

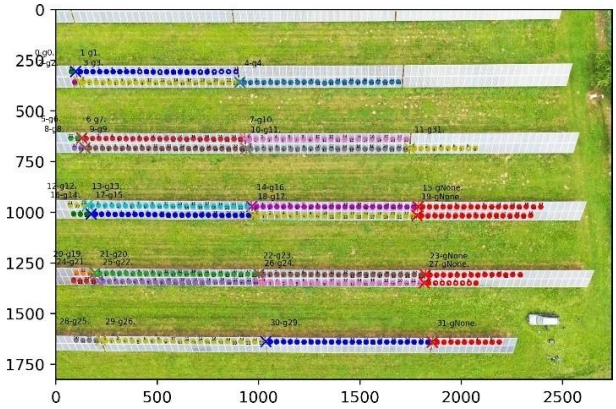



**Figure 2.5.** Processed high altitude image [10].

The high-altitude flight is then followed by a series of overview flights over the detected benches to check individual panel states. The drone flights over every bench, taking a quick and relatively low-resolution picture to evaluate whether the module has an anomaly and needs detailed imaging.

As a last step, the drone flies over the modules that were reported to have an anomaly in the previous step. This is a low-altitude flight to take detailed thermal image data of each module that can later be used to detect the particular anomaly type.

The outcome of this step is the set of individual module detail images paired with the positional and structural data in the database. The available image resolution depends on whether the low-altitude flight was performed for a given module.

### 2.2.4 Image data processing

With the gathered image and positional data, the individual module state evaluation is being performed at this stage. The result is then included in the output data (column

[1] `https://opensfm.org`

*fault_type* in the *Module_measurement* table from Figure 2.4). It consists of analysis of individual IR images, where individual cells have to be recognized, and converted to schematic values, and then the temperature differences have to be evaluated on a per-cell basis (result saved in table *Cell_measurement* from Figure 2.4).

For the purpose of this thesis, the relevant information is that after this step, the data in tables *Module*, *Cell*, *Cell_type* from the *Electrical connection* package, and all the tables from packages *Physical layout* and *Measurement* from Figure 2.4 are populated.

### 2.2.5 Electrical schema

With the individual module records created in the database, it is possible to assign them to the electrical schema in the database structure. This step is performed as an initial step when a new power plant is analyzed or the schema of an existing power plant changes. This process is currently mostly manual and requires a significant effort from the researchers.

The simplest and cheapest variant is if the operators already have an existing document that might be used for such mapping. If it exists at all, it usually has many different forms and is not standardized across power plants, so parsing the document and assigning the modules to the actual electrical schema is a labor-intensive process.

In the case when there was no such documentation before engaging with the research group, it is desired to introduce a solution where the customer could either manually or with automatic assistance, mark the important schematic features in a map view based on the data gathered from the *Flight* and *Image Processing* phase. Right now, this is performed by the researchers manually after consulting with the customer technicians, so such a solution would greatly decrease the researcher's time required to process a single power plant.

In another significant portion of cases, the schema is represented as an Excel sheet. Each cell in the sheet represents either a module or an empty physical space between benches. The assignment to each power line is marked by a number in the target cell or by color-coding of the target cell. This sheet is usually used as a reference for technicians. Currently, the schema can be converted to usable format using ad-hoc scripts (usually Python or bash) or, in most cases, completely disregarded, as processing it takes more effort than coming up with a mapping function based on client specification.

After this step, the individual modules are correctly assigned to the *Electric_line* in the database schema from Figure 2.4.

### 2.2.6 Output data format

As a result of the drone flights, data post-processing and the electrical schema assignment, the power plant data is available as a single SQLite database file, accompanied by a set of images in a bitmap format.

The SQLite format was chosen for easy distribution and data sharing among researchers. For the purpose of the simulation, it is safe to assume that the data already includes the correct module mapping with respect to the connected *Electric_line* and *Inverter*, as described in Section 2.2.5.

## 2.3 Ngspice simulator

Ngspice is an open source simulator for electric and electronic circuits [11], that allows engineers and researchers to model and simulate the behavior of analog and mixed-signal circuits. It is the simulator used as part of this thesis to simulate the circuits within PvPs.

### 2.3.1 Basic concepts

Users define their electronic circuits by specifying the components (such as resistors, capacitors, and transistors) and their interconnections. This description is often written in a netlist format, which is a textual representation of the circuit.

Ngspice supports various simulation types, including DC analysis, AC analysis, transient analysis, and more. Each simulation type focuses on different aspects of the circuit's behavior, such as steady-state conditions, frequency response, and transient response.

#### Netlist

A netlist is a textual representation of an electronic circuit's connectivity. It provides a concise and standardized way to describe the components in a circuit and their interconnections. In a netlist, each component is defined along with its connection points, and the connections between components are specified using a list of nodes and the corresponding interconnecting elements.

Below is an example of a simple circuit with a 1 kΩ resistor connected between nodes 1 and 2.

```
R1 1 2 1k
```

Netlists allow the definition of sub-components called subcircuits. The advantage of such subcircuits is the reusability, where a particular subcircuit can be defined only once, and used at multiple places in a given netlist. The example below is a component that wraps the above-mentioned resistor with parameters for the resistance and terminals.

```
.subckt circuit_1 Res Vpv+ Vpv-
R1 Vpv+ Vpv- Res
.ends
```

The subscircuit definition also supports configuration parameters that are locally scoped. Meaning that each place where the subcircuit appears, can have different value of the input parameters.

```
.subckt circuit_1 Vpv+ Vpv-
R1 Vpv+ Vpv- Res
.param Res 1k
.ends
```

Each netlist can also define a control section, which specifies the simulation steps to perform on a given circuit, together with commands to control the output of the simulation.

Below is an example control section that performs DC analysis from 0 V to 1000 V with a step of 0.1 V, while printing the voltage, current, and electric power for each step of the DC simulation.

```
.control
dc Vpv 0 1000 0.1
print v(0) I(vpv) v(2)*I(vpv)
quit 0
.endc
```

The most important control section commands for the purpose of this thesis are:

- *dc* - Performs a DC analysis
- *print* - Prints the arguments as a column-based tab-delimited ascii output for each simulation step
- *wrdata* - Similar to print, but outputs to a file

The structure of the output from the Ngspice simulation with the above control section can be seen below. It is a simple textual output with no additional formatting support in the Ngspice simulator.

```
Index   v-sweep         i(vpv)          v(2)*i(vpv)
-------------------------------------------------------
0       0.000000e+00    1.139951e+01    2.946705e-25
```

### 2.3.2 Usage

Basic Ngspice usage is as simple as calling the executable with the circuit specification in the *file.cir* netlist.

```
ngspice "file.cir"
```

This loads the circuits in a given file and opens an interactive shell that lets the user perform commands on the given circuit. The user can then proceed by inputting the following into the interactive shell.

```
.dc Vpv 0 1000 0.1
```

*Notice the prefix in the* `.dc`*. The usage of the commands differs slightly when compared to using the* `.control` *section described above*

Another type of usage is using the *-b* flag to run batch operation, which executes the commands in the *.control* section and does not open the interactive shell, allowing for automated execution.

### 2.3.3 Environment configuration

Ngspice supports multiple configuration options. They can change anything from parsing and formatting, to multithreading and error handling.

The simulation behavior and environment configuration can either be performed manually by inputting commands into the interactive shell, in the *.control* section, or via settings in a *.spiceinit* file in the directory where Ngspice is being executed. This file is read automatically on the program startup.

The most important configuration options for the purpose of this thesis are:

- *ngbehavior* - Allows to change the behavior with respect to different netlist formats
- *num_threads* - Number of threads that Ngspice can use (default 2)

The complete documentation can be found on the official Ngspice website[1].

[1] `https://ngspice.sourceforge.io/docs.html`

# Chapter 3
# Analysis of related work

This chapter explores existing commercial software focusing on solar power plant thermography and provides an overview of the state of academic research related to software for such analysis.

By examining commercial software dedicated to solar power plant thermography, we can get a valuable lesson regarding user requirements, feature sets, established standards, compliance requirements, and potential areas for improvement. Similarly, an exploration of academic work in this domain offers a deeper understanding of emerging technologies, theoretical frameworks, and innovative approaches. Only with a comprehensive understanding of the existing landscape is it possible to make informed decisions throughout the software development lifecycle, ensuring that the resulting tool effectively addresses the needs and challenges of solar power plant analysis and visualization.

## 3.1 Methodology

The methodology used to explore the domain should provide insight into the research process, explain how the results were gathered and filtered, and enable the reader to replicate the research. Understanding the used methodology is also crucial for the evaluation of possible biases and systematic mistakes.

### 3.1.1 Academic work

The list of relevant resources was acquired using Google Scholar[1] and Research Gate[2]. The following search queries were used:

- solar plant thermal imagery software
- solar thermal imagery software
- solar thermography software
- solar plant thermography software
- PV thermography software
- photovoltaic thermography software
- drone solar maintenance
- drone solar monitoring

For each search term, results returned on the first page of the respective search engine were considered. Results that were either not focused on the PV thermography or primarily focused on the PV technology itself without any mention of the usage of *thermography* or a *drone* were excluded from the analyzed result set, apart from the examples mentioned below.

[1] `https://scholar.google.cz`
[2] `https://www.researchgate.net/`

### 3.1.2 Related commercial solutions and independent research analysis

The list of resources was acquired primarily using Google search[1]. The following queries were used:

- solar plant thermal imagery software
- solar thermal imagery software
- solar thermography software
- solar plant thermography software
- PV thermography software
- photovoltaic thermography software
- solar plant thermal inspection

Only non-advertisement results were evaluated. Results ranked within the first 25 results were considered for each search term. Results offering or advertising general thermography software (not specialized for photovoltaic measurement, or without automated analytical capabilities) were not considered. Sites with non-explanatory description of the software features, or the description hidden behind a paywall, were also omitted from the research.

Another great resource used for the commercial solutions analysis are the publicly available reviews of such software. The most detailed and comprehensible among the reviewed ones was the one by TheDroneLife [12].

Potential issue with this research method might be the lack of online presence for software providers in this category. Since the sales channel is mostly business-to-business (B2B), the website search engine optimization (SEO) and Google search relevance might not play a crucial role for the business, and therefore this research might not include a well established software.

## 3.2 Commercial usage analysis

Analysis of commercial solutions that offer thermography analysis targeted at the PV industry. The overview is based on publicly available data from the linked websites at the time of publication. The presented data is the result of a subjective understanding of the publicly available materials, and there is no intention to benefit any of the mentioned companies.

[1] https://google.com/

### 3.2.1 Scopito

**Website:** `https://scopito.com/solar-pv-inspection-software`
**Pricing strategy:** Per MW or Enterprise
**Related features:**
- Desktop/Web application - map view
- Mobile application for technicians

**Additional features:**
- Integration with SAP

**Description:**
Scopito is a data management system that supports many industries, with a specialized version for the PV industry. They provide a web-based data analytical platform that supports automatic AI-based fault classification based on the thermal image data, with comprehensive PDF or CSV reporting of the results.

The base pricing is per MW (with the website-available pricing of 20 EUR per MW). The fee per MW is fixed, including all features, regardless of the resource usage or amount of reporting generated.

They do not seem to offer a mobile application for field access to the data.

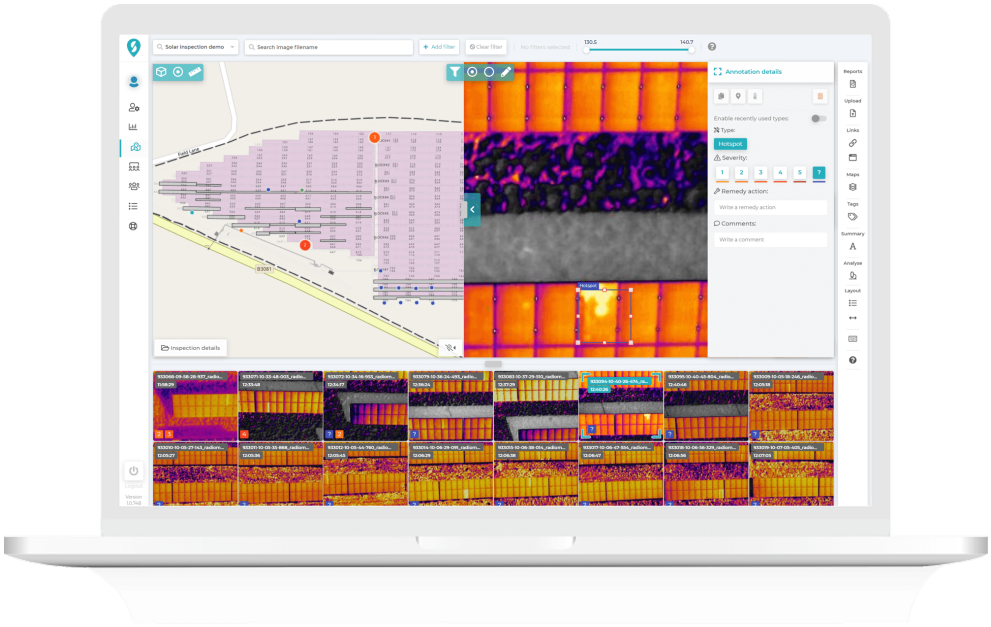


**Figure 3.1.** Screenshot of Scopito platform [1]

### 3.2.2 RaptorMaps

**Website:** `https://raptormaps.com/products/solar-aerial-inspections`
**Pricing strategy:** Not public
**Related features:**
- Desktop/Web application - map view
- Desktop/Web application - thermal anomaly and root cause detection
- Mobile application for technicians

**Additional features:**
- API access

**Description:**
RaptorMaps offers a set of solutions for the digital thermography of solar plants, with anomaly detection, map-based view, and report generation. Their solution offers filtering and aggregations based on the anomaly type. The software can enhance the data by measurements from other sensors. They also offer API connection, so it is possible to analyze the data even further or build advanced tools on top of Raptor Maps. The software supports the management of a portfolio of power plants and aggregate data among them.

In addition to only providing the software, where you can provide the data from your drone, their unique offering is a service where they perform the PV thermography in cooperation with a network of service providers, so the plant owner gets only access to the platform without any physical work on their side.

They offer a mobile application for field access to the data, paired with GPS for technicians to navigate easily.

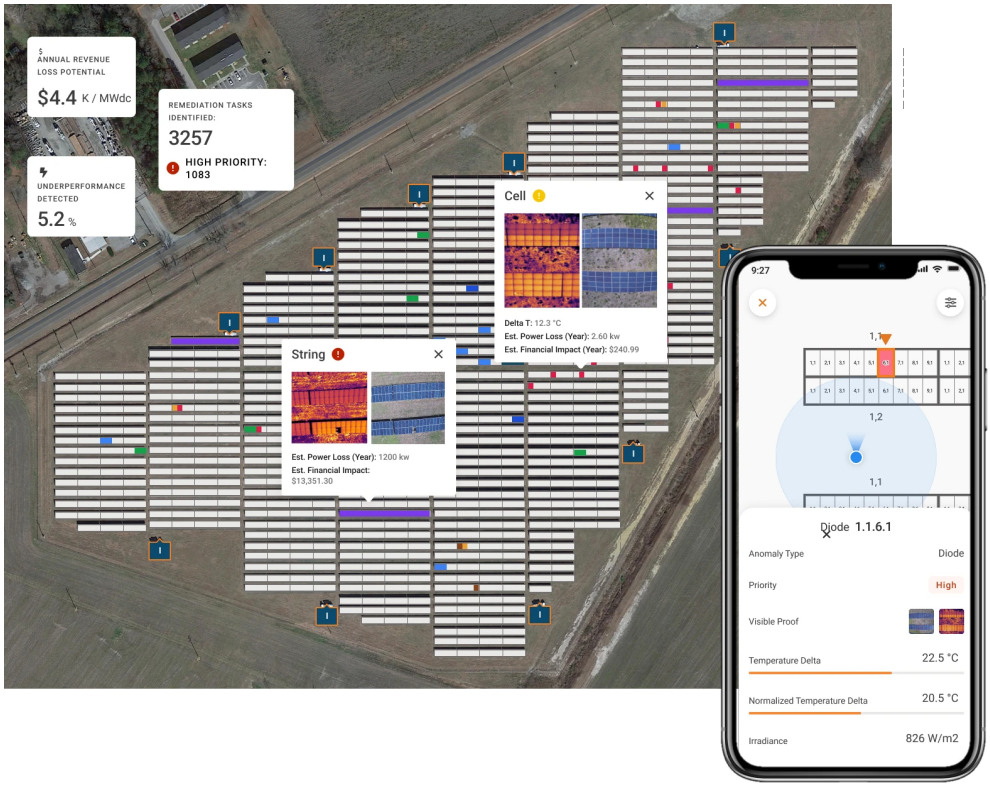



**Figure 3.2.** Screenshot of Raptor Solar platform [2]

### 3.2.3 MapperX

**Website:** `https://mapperx.com`
**Pricing strategy:** Tiered, per module
**Related features:**
- Desktop/Web application - map view
- Desktop/Web application - thermal anomaly and root cause detection
- Mobile application for technicians

**Additional features:**
- Task Manager module - module for managing maintenance tasks
- Process for logging module serial numbers using OCR
- Digital twin simulation

**Description:**
A solution from MapperX offers a platform with core features for digital thermography. They also offer a separate solution that operates the drones, which supports three different DJI drone models. Additionally, their software offers management capabilities for maintenance technicians. They do not offer any sensor data gathering or processing as part of their services.

The pricing is split by tears, where individual tiers offer different sets of features, flight altitudes, image resolution, and anomaly detection capabilities. This structure is similar to Software as a Service (SaaS) companies but uncommon among maintenance software providers.

It is not clear if they have a mobile application. There is a support for tablet devices through a web app or a dedicated application based on the screenshots available.

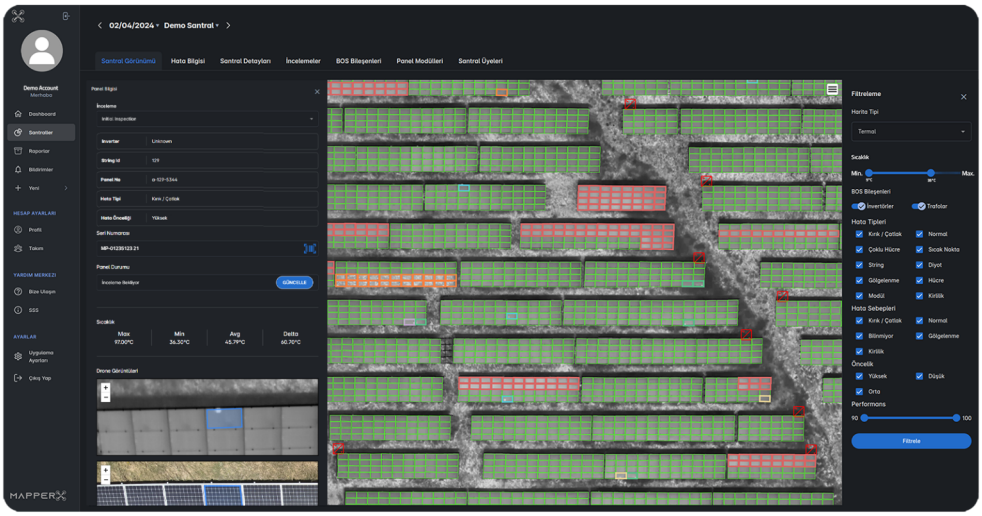



**Figure 3.3.** Screenshot of MapperX platform [3]

### 3.2.4 Sitemark

**Website:** `https://www.sitemark.com`
**Pricing strategy:** Not public
**Related features:**
- Desktop/Web application - map view
- Desktop/Web application - AI-based thermal anomaly and root cause detection
- IEC TS 62446-3 compliant reporting
- Mobile application for technicians

**Additional features:**
- Application for solar plant construction
- Application for data monitoring from sensors

**Description:**
Sitemark focuses mainly on the PV industry while providing a complete set of tools for management, preventive and reactive maintenance, and reporting. Their coverage includes the construction part of the process as well.

The analytical part for the thermographic images was created in collaboration with the Interuniversity Micro-Electronics Center [13]. It supports power loss estimation based on a rule-based classification of thermal signatures on a module level.

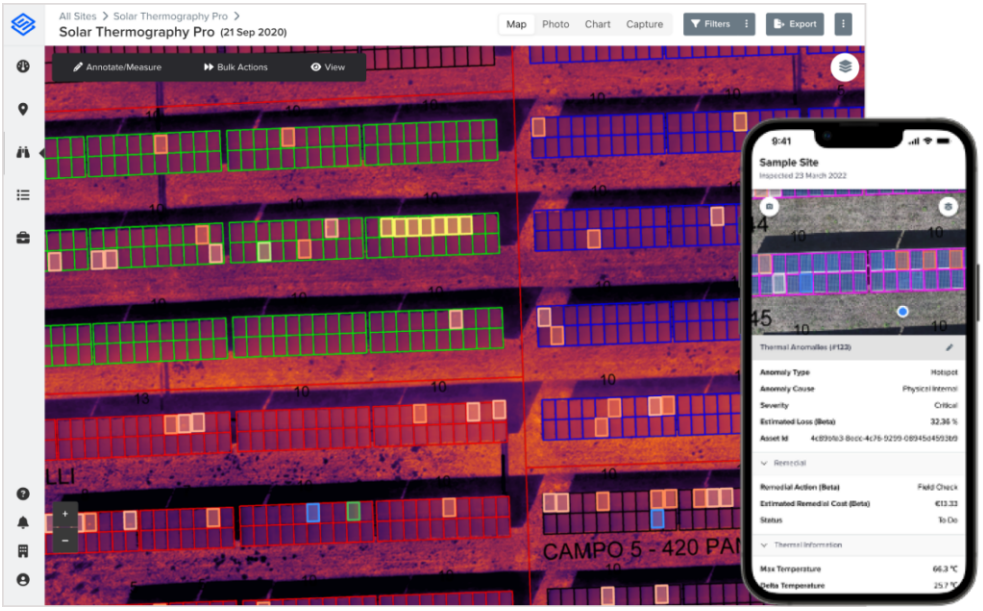



**Figure 3.4.** Screenshot of Sitemark Fuse platform [4]

### 3.2.5 Above Surveying

**Website:** `https://www.abovesurveying.com/inspection/aerial-thermographic-solar-inspection`
**Pricing strategy:** Not public. Base and Pro version available.
**Related features:**
- Desktop/Web application - map view
- Desktop/Web application - Thermal anomaly and root cause detection
- Advanced simulation - String/Inverter level
- Desktop/Web application - Historical inspection data comparison
- IEC TS 62446-3 compliant reporting
- Mobile application with reporting

**Additional features:**
- Built-in task management

**Description:**
Above Surveying offers a solution based on the SolarGain data processing and reporting platform. Beyond the processing of thermal images, the platform offers task management capabilities and automatically generates tasks based on module failure reporting.

In collaboration with C.R.E.S.T[1], they seem to be simulating complex electrical system performance regarding the actual system design. They seem to support simulation at the String and Inverter levels.

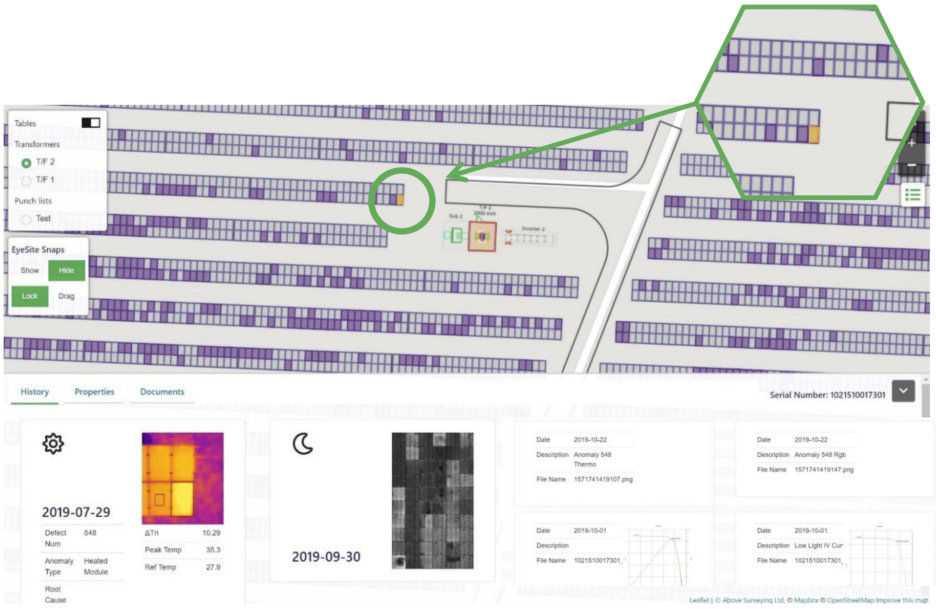



**Figure 3.5.** Screenshot of Above Surveying platform [5]

[1] `https://www.lboro.ac.uk/research/crest`

### 3.2.6 vHive

**Website:** `https://www.vhive.ai/solar-panels`
**Pricing strategy:** Not public
**Related features:**

- Application for drone orchestration
- Desktop/Web application - Fault auto detection
- Mobile application

**Additional features:**

- Digital twig technology

**Description:**
Despite the software not being well documented on the website, it seems that their software offers the core set of features for automated thermography analysis.

Additionally, vHive seems to offer a separate solution for data capture, which supports parallel scanning by multiple drones. It seems that their core offering is the modelling and digital twin simulation software, that focuses primarily on the wind turbine energy generation industry.

### 3.2.7 GeoWGS84

**Website:** `https://www.geowgs84.com/getvision`
**Related features:**

- Web application (Beta)
- Automatic fault detection

**Description:**
The approach by GeoWGS84 seems to be capable of automatic AI-based module failure classification based on thermal image data. They seem to offer automated module failure classification from thermal data and report generation.

They offer a web-based platform currently in beta access. There is no documentation or additional platform description available at the moment.

### 3.2.8 Other

Other resources seem to primarily include service providers, who do not advertise the used software but provide the service and value generated by the data reporting, possibly using one of the above-analyzed software solutions.

Another significant group among the results is general-purpose thermal imaging solutions, possibly able to detect hotspots but with no direct optimization for usage with PV modules or the required segmentation.

### 3.2.9 Summary

Table 3.1 summarizes the overview of the core features of the reviewed applications. Below is explanation of the individual table columns:

- *Application* - Reviewed application name.

- *Sensor data* - Whether the application supports processing and displaying data from equipment sensors.
- *Failure classification type* - The type of failure classification technology. *AI* for technologies based on Artificial Intelligence or *Unknown* when the used technology is not known.
- *Simulation* - Whether the application supports simulating the impact of the failure on the rest of the string or inverter.

| Application | Sensor data | Failure classification type | Simulation |
|---|---|---|---|
| Scopito | No | AI | No |
| Raptor Maps | Yes | Unknown | No |
| MapperX | No | AI | No |
| Sitemark | Yes | AI | No |
| Above Surveying | Unclear | Unknown | Yes |
| vHive | No | AI | No |
| GeoWGS84 | No | AI | No |

**Table 3.1.** Comparison of commercially available software

All the explored solutions offer a map-based overview, with failure classification and direct localization in the map. At least some of the explored software solutions offer advanced search functionality and exports to PDF and CSV formats. Only one solution calculates the effect of a single module failure on the rest of the String / Inverter.

The research suggests that the core features required for such a software platform are consistent across multiple vendors. At the same time, there is a possibility of introducing additional features to provide additional value.

## 3.3 Independent research

Another category found during the research is *Independent research*, which currently spans independent research centers and open-source projects.

### 3.3.1 Interuniversity Micro-Electronics Center

The non-profit organization based in Belgium seems to be doing independent research in the field. They also seem to be cooperating with companies in the commercial sector, based on the mention on the Sitemark website [13]. No other mention of this particular project or cooperation was found during the research phase.

### 3.3.2 Thermography framework

**Website:** `https://github.com/cdeldon/thermography`
**Description:**
A GitHub repository with Python software for AI-based PV module defect detection from thermal images. The license file is missing from the GitHub repository, so the licensing needs to be clarified.

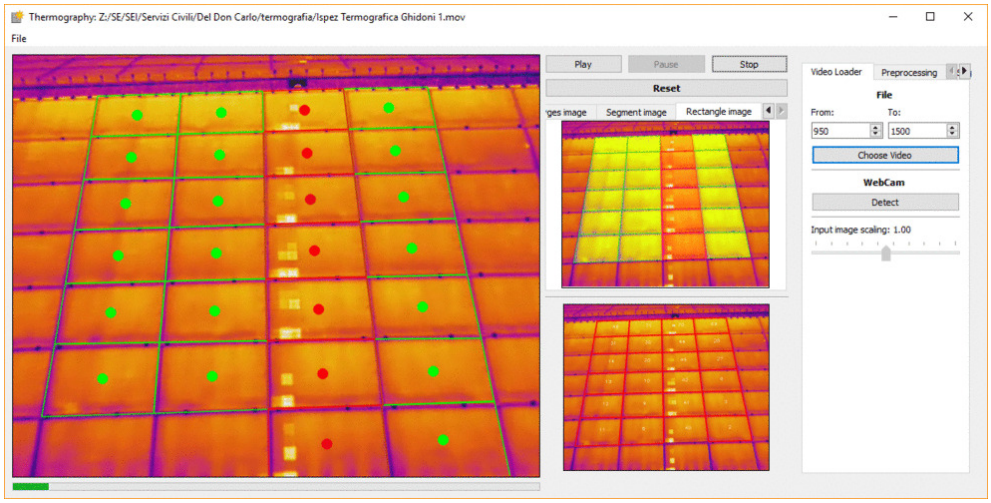



**Figure 3.6.** Screenshot of Thermography desktop application [6]

It focuses only on the detection part from video footage, so it is not a competitive implementation in the context of this thesis. However, it provides a great example of the individual subproblem of defect detection, so it is kept here to enable the reader to understand the subproblem better.

## 3.4 Related academic work analysis

A conference paper from 2014 [14] briefly evaluates the general idea of using UAVs for PvP monitoring, and concludes that the inspection method is reliable, cost effective and time-saving.

An article from 2018 [15] presents the development of a real-time monitoring system that processes data gathered from the power plant using sensors on individual inverters or modules. Such software does provide detailed view into the operation, but requires usage of particular equipment, and upfront investment.

An article from 2018 [16] elaborates on the cost-effectiveness of using drones for inspections. The conclusion is that drone inspections are always more expensive for all power plant sizes while being more accurate than the ones performed by humans.

Publication from 2021 [17] elaborates on the usability of automated AI analysis to predict future PV module failures based on thermal imagery. The publication akcnowledges the added value of having thermal images with GIS data available, evaluates the feasibility of using AI to evaluate the module failures, and concludes, that drone thermal imaging is a possible solution for converting solar power plants into their respective Digital Twins.

Article from 2022 [18] proposes an innovative method for filtering the IR images, resulting in improved defect detection capabilities when measured at lower irradiance levels (cloudy days, winter). As part of the paper, the algorithm is implemented in Matlab.

Article from 2023 [19] focuses mainly on the detection of PV module features and hotspots. The publication mentions a custom developed software, that is able to present the geospatial data together with the thermal images. The software seems to focus on individual module failure analysis.

Article from 2023 [20] proposes an even more scalable monitoring model using aircraft to take image data. Where aircraft becomes more cost-effective for plants with more than $50MW$ capacity. This further supports the idea of creating an application that allows processing such data, as the source of the data is not important for later processing.

Based on the above reviewed publications, I conclude that the publications can be categorized into the following categories based on the core area of focus:

- Module fault type recognition from thermal data
- Thermography data processing
- Drone inspection with emphasis on drone research
- Power plant monitoring system

None of which seem to be doing exactly what this thesis aims to do. That is, based on this analysis, to develop a software similar to the commercial solutions available, that simulates the electrical properties based on the assumed state from thermography results.

# Chapter 4
# Software requirements

In software development, the software requirements are a crucial part of each project. They are used as a specification for the final software. They should clearly and undisputably describe the expectations for the to-be-built software. The software developer should be able to follow the written software requirements to build the software that the other party imagined.

For contractual work, the software requirements are often used as the basis of the underlying contract, holding the developing party responsible for not fulfilling them. Without clearly defined requirements, it becomes challenging to measure the success of the project or to verify that the final product fulfills its intended purpose.

## 4.1 Requirements gathering

Composing a list of requirements for a to-be-build software is a challenging task. It should ideally take into account all expected use cases from all stakeholders. The requirements in the final specification should not be contradictory to one another. This, however, does not have to be the case with real-world stakeholder requirements, where each of the stakeholders might have completely different expectations. This phase, therefore, includes discussions with all the stakeholders and an understanding of their goals.

The requirements for the software developed as part of this thesis were primarily based on the three representative groups described below.

### 4.1.1 Requirements based on the DiPreFE research project

The newly built software is intended to process the output of the previous stages of the DiPreFE research project, as discussed in Section 2.2. Therefore, it is necessary to understand the target users among the researchers and their use cases.

Among the requirements based on the research project are primarily:

- Input data format
- SQLite database handling
- Internal user actions
- Data export specification

The application has to be able to process data from an existing application that is part of the DiPreFE research project and should not put any additional constraints on the input format.

One example of a requirement based on this constraint group, which is highly uncommon in other applications and would typically not be listed as a functional requirement, is the possibility of violating foreign keys in the SQLite database. The common expectation in basically all other software would be that if the database specifies foreign keys, their violation is not a valid state. This might, however, happen, and the application should handle this gracefully and not reject the input database file.

`Foreign key` *in a relational database system is a concept that restricts the possible values of one column to only allow the values based on the specified table and column.*

### 4.1.2 Requirements based on project context

After this thesis is complete, the software being built as part of this thesis is going to be run and maintained by already established research groups at CIIRC. Since they are going to be the ones primarily taking ownership of running this software, their understanding of the software project and the deployment processes is paramount for future maintenance and new developer onboarding.

The deployment and infrastructure part is going to be handled by the infrastructure team that manages the whole faculty. To minimize the support necessary for the maintenance and deployment processes in the future, it is desirable to follow this team's standard deployment practices and avoid introducing novel approaches into this group's workflow, which might require additional training and effort.

The set of requirements based on this category can be mostly split into the following categories:

- Deployment platform limitations
- Maintenance requirements
- Deployment automation

To gather this type of requirements, it was necessary to understand the deployment capabilities of the organization, as well as the approval process for any additional software components and third-party tools. It was also necessary to understand the capabilities of the future maintainers and their opinions on different deployment platforms and orchestration tools.

### 4.1.3 Requirements based on customer requirements

This section primarily includes the requirements that specify what actions the users can perform and how the actions should be performed. These requirements were gathered in coordination with stakeholders and future users. The results of the commercial software analysis performed in Section 3.2 were used to identify the necessary standard features available in the industry.

An important aspect of this part was that the stakeholders are neither the end users of the application nor have any previous formal experience in the User Experience (UX) field. It was, therefore, necessary to ensure that the stakeholder requirements made sense from a user perspective and did not diminish the user experience.

The outcomes of this section primarily include the following requirements categories:

- List of screens
- Data available on each screen
- Actions available on each screen
- Data filtering options

## 4.2 Software requirement specification

One of the popular ways to represent the software requirements is in the form of Functional and Non-functional requirements.

Functional requirements define the features of the product and define what the product should do. They are necessary for the application to provide the desired functionality.

Non-functional requirements define how the product should work. Often also called non-behavioral, they specify things such as security and scalability requirements.

This section provides an insight into the key requirements used for the development of this software. In a contractually binding specification of commercial projects, the set of requirements would be much more detailed. Given the nature of software development as part of a thesis, as compared to commercial projects, it was not necessary to specify all the aspects of the software before the development phase began and it was possible to develop non-key features without the necessity to specify them in a binding way.

### 4.2.1 General definitions

This section specifies the definitions used in the description of individual requirements.

#### User type

User types used in the formulation of functional requirements.

- *Internal user* - User performing actions and having access to the application on behalf of the software supplier.
- *External user* - Any personnel related to the customer. The access to the application for this user has to be approved by the customer.
- *User* - *Internal user* or *External user*

### 4.2.2 Functional requirements

Below are the specified functional requirements, categorized by the requirement category and detail level.

#### Access requirements

**FR 4.1.** *User* has to log-in in order to access any application data.
**FR 4.2.** Available login method will include email and password.
**FR 4.3.** *External user* access can be restricted to certain power plants.
**FR 4.4.** *Internal user* can access all power plants.
**FR 4.5.** *Internal user* can manage access settings for any *External user*. This mainly includes adding and removing accessible power plants.
**FR 4.6.** *User* has to be manually verified by *Internal user* after registration, before login is allowed.

### Management requirements

**FR 4.7.** *Internal user* is able to create a new power plant.
**FR 4.8.** *Internal user* is able to upload a SQLite file as specified in Section 2.2.6 for a power plant.
**FR 4.9.** *Internal user* is able to upload bitmap image for each module.

### Bulk data access

**FR 4.10.** *User* can see a list of damaged modules.
**FR 4.11.** *User* can filter the list of broken modules by row and bench.
**FR 4.12.** *User* can download a zip file with PDF reports for individual broken modules. The PDF document will include:

- recommended action
- the loss due to the module breakage
- chart with actual power output
- chart with healthy power output

**FR 4.13.** *User* can download a zip archive with PDF reports for inverters with broken modules. The PDF document will include the sum of the losses due to the module breakage and the list of individual broken modules.
**FR 4.14.** *User* can see a map view of the whole power plant. *User* should be able to navigate in the map using click-and-drag functionality.

### Module detail

**FR 4.15.** *User* can access a detail page of each module. The detail page can be accessed from the listing specified in FR 4.10.
**FR 4.16.** The module detail page shows a power output chart based on the current module state in an isolated circuit.
**FR 4.17.** The module detail page shows a power output chart based on the healthy module state in an isolated circuit.
**FR 4.18.** The module detail page shows the colorized thermal image for given module.

### Inverter detail

**FR 4.19.** *User* can see a detail page of each inverter.
**FR 4.20.** The inverter detail page shows a listing of modules connected to the particular inverter.
**FR 4.21.** The inverter detail page shows a power output chart based on the current state.
**FR 4.22.** The inverter detail page shows a power output chart based on the state if none of the modules were broken.

### Module configuration

**FR 4.23.** *Internal user* is able to create new and edit existing module schema definitions.
**FR 4.24.** *Internal user* is able to delete existing module schema definition if it is not assigned to any power plant.
**FR 4.25.** Module types are shared across power plants, so that they do not have to be redefined for every power plant.

#### Power plant configuration

**FR 4.26.** *External user* is able to set basic properties for a power plant. These include
- Cost of single module replacement
- Electricity resale value

**FR 4.27.** *External user* is is able to set the address information for a given power plant.
**FR 4.28.** *External user* is able to set the expected lifespan of the power plant. In the form of setting the *start of life* and *end of life* dates.
**FR 4.29.** Each power plant will be assigned to one module type. *Internal user* is able to change the assignment. The list of available module types has to be searchable.

#### Simulation

**FR 4.30.** *Internal user* is able to run the simulation after the upload of the relevant SQLite file specified in FR 4.8.
**FR 4.31.** *Internal user* is able to manually rerun the simulation after changing settings specified in FR 4.29.
**FR 4.32.** When a simulation is running, *Internal user* should be able to see the progress of the simulation. The interface should automatically refresh the displayed progress.

### 4.2.3 Non-functional requirements

**NR 4.1.** The application can work with power plants with up to $100k$ modules.
**NR 4.2.** Data processing should not affect user interface responsiveness.
**NR 4.3.** Simulation result data should be stored, or transferrable to the original SQLite database file, or available for export in the same format.
**NR 4.4.** The input SQLite database format is open for extension. The application should not require any modification to support new columns in the SQLite database file.
**NR 4.5.** The application should handle data inconsistencies and foreign key violations in the SQLite database gracefully. Such data inconsistency in part of the data should not block the simulation of the rest of the power plant.
**NR 4.6.** Simulation for a given power plant based on the input data should not take more than 24 hours to complete.
**NR 4.7.** The application should be deployable to a publicly accessible virtual machine or a set of virtual machines, given a list of IP addresses and SSH keys.

# Chapter 5
# Implementation

## 5.1 Architecture

The goal of the application architecture is to define a structured solution to develop an application that meets all the functional and non-functional requirements defined in Chapter 4 while optimizing the quality attributes such as performance, security, manageability, and extensibility.

For this particular application, the requirements with the most significant impact on the architecture are the scalability requirements to meet the time constraints for the simulation and PDF generation process, the storage requirements for the SQLite files, and platform limitations, as specified in NR 4.6, FR 4.12, NR 4.3 and NR 4.7, respectively.

The application is split into three components: a backend application, a frontend application, and the Ngspice simulation service. The backend is the central part of the application logic and is responsible for handling user requests, communicating with the central database, scheduling asynchronous tasks, and retrieving their results. The external services used include a message queue (RabbitMQ[1]), a database (PostgreSQL[2]) and a storage server. The decisions behind choosing a particular technology are described in the following paragraphs.

Figure 5.1 shows a schematic overview of the final system. Further decomposition of individual components is explained in the later sections.

The backend server's primary purpose is to serve user requests and send back responses. It is not directly responsible for data processing and does not directly handle computationally expensive tasks. Its responsibility is also to authenticate and authorize the user's actions. It uses PostgreSQL as the primary data store. The PostgreSQL database is used because a managed version is available at the faculty.

One of the most resource-consuming parts of the application is the Ngspice simulation process. In order to achieve the required maximum execution time, as specified in NR 4.6, it is necessary to scale the simulation part independently of the rest of the application. While there are opportunities for caching and deduplicating the necessary simulations, the architecture should be able to handle the worst possible scenario, that is, with no deduplication. Therefore, the simulation process is extracted into a separate service. This service will receive messages, process them by simulating the requested simulation schema, and return the simulation results.

A message queue has been chosen for communication between the backend and Ngspice service due to the ease of scalability of asynchronous message processing. This, however,

[1] `https://www.rabbitmq.com`
[2] `https://www.postgresql.org/`

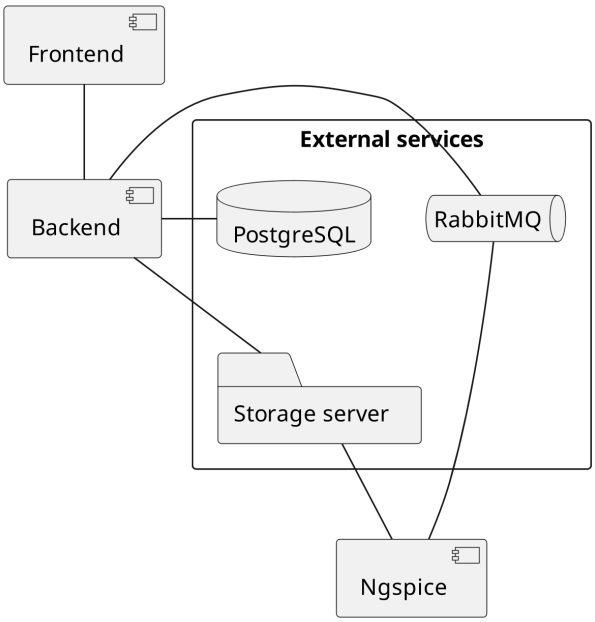



**Figure 5.1.** Architecture schema

imposes additional development overhead for tasks that would be simple to implement in a blocking manner (e.g., a preview of output for schema as described in Section 5.4.8). With a message queue, the application has to implement mechanisms for checking task completion and retry policies, which would not be necessary in the synchronous variant. In this case, the simplified scalability and operational benefits outweigh the increased application complexity. The particular message broker used is RabbitMQ, due to a more developer-friendly approach, when compared to the alternatives like Apache Kafka and ActiveMQ [21].

Further non-standard architectural decisions (for a typical web application) were made due to the use of SQLite storage for the power plant schema. It is desired to keep the data in the original SQLite database, as that decreases the development costs, allows for independent development cycles, and does not require schema updates when the input file schema changes, as specified in NR 4.3. In practice, this means that to work with the SQLite data, the application has to either load the whole database into the local file system or access it on a network-mounted drive. Given that the network and storage solutions were unknown until the deployment phase, it was decided that the only component working with the SQLite database would be the backend.

### 5.1.1 Backend

The main purpose of the Backend part of the application is to expose an API for the Frontend and enable users to perform actions. It is also responsible for authenticating and authorizing the user actions. The request-response processing should not be delayed by any computationally intensive tasks.

One such action that could significantly slow down the server is the generation of PDF reports (as specified in FR 4.12). The PDF document has to be generated for each module after the Ngspice simulation is run. This allows the user to download the PDF documents instantaneously, as generation on the fly is not feasible. This generation also

offers significantly fewer deduplication opportunities than for the Ngspice simulation since the generated PDF needs to be dynamic for each module.

Another action that could impose a significant load on the server is the handling of the events produced by the Ngspice simulation consumer. The Ngspice simulation consumer will emit an event every time a simulation finishes. It is, therefore, desirable to scale the handling of such events accordingly without affecting the processing of user requests.

Due to this, the service was split into three separately deployable and scalable artifacts:

- *Backend server* - server handling user requests
- *PDF Generation consumer* - consumer capable of generating PDF reports
- *Backend NGspice consumer* - consumer handling events produced by Ngspice service

This enables us to scale up or start the individual artifacts on a completely different machine so that the request processing is not slowed down at all. It was kept as a part of the backend service due to the need to access the PostgreSQL and SQLite databases.

### 5.1.2 Ngspice worker

The Ngspice service is not aware of any business logic. Its primary purpose is to run an Ngspice simulation for a given netlist and output the result to the appropriate location. It communicates with the rest of the system through a message queue. This allows us to scale the workers independently on the rest of the application. For a better understanding of the interaction, Figure 5.2 depicts the application interaction with the service.

### 5.1.3 Frontend

The Frontend is a separate client application. It is the only part of the application the user will directly interact with. The main task for this application is handling communication with the backend. The technology used should allow for easy handling of asynchronous tasks and rich user interactions.

### 5.1.4 Implications

As a result of the architecture decision, the simulation can be performed with minimal performance requirements for the primary backend server. The sequential diagram in Figure 5.2 shows the steps necessary to get the simulation result for a single module.

- The Backend initiates the generation by sending a message to appropriate the RabbitMQ queue.
- The Ngspice consumer processes the message by simulating the appropriate Ngspice schema, saving the simulation output to the Network Storage, and sending a message to another RabbitMQ queue upon completion.
- The Backend Ngspice consumer consumes the message sent upon completion and edits appropriate data in the PostgreSQL database.

This sequence is repeated for every module in the power plant. Each highlighted group in the diagram can be scaled separately.

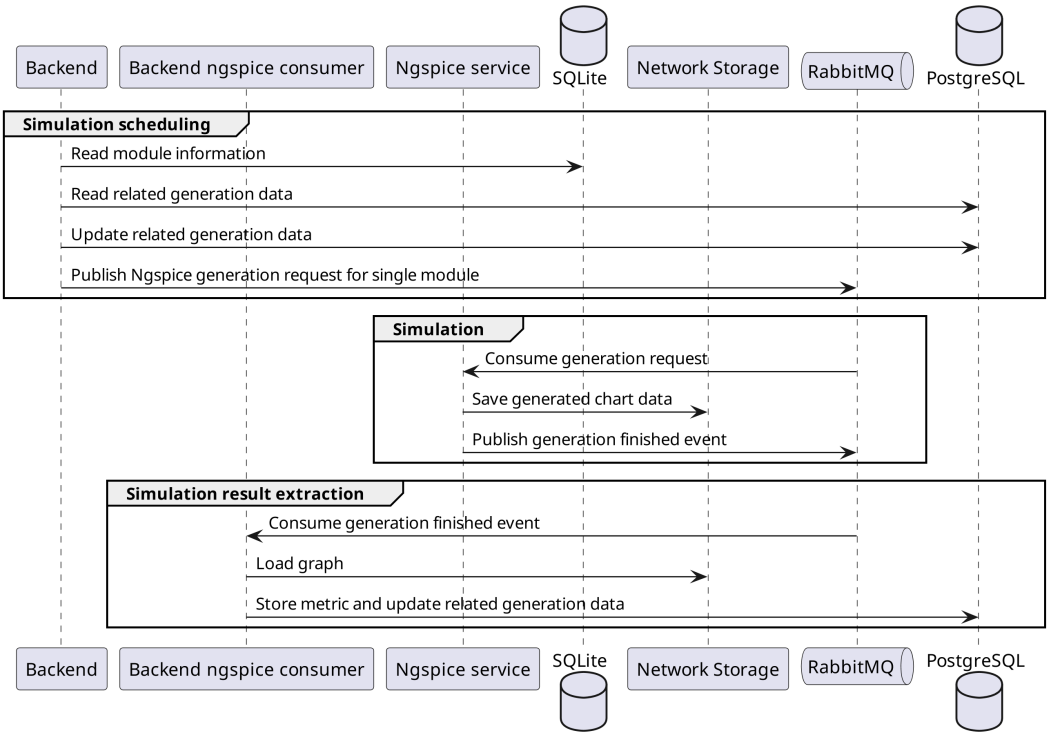



**Figure 5.2.** Simulation process sequential diagram

## 5.2 Implementation details

### 5.2.1 Code structure

Before creating any code, deciding how to organize it was necessary. A single monorepository was used primarily due to the simplified dependency management, organization, and better coordination between developers and simplified tooling development [22]. The structure of the monorepository is the following:

- `deployment` ..... Deployment tools
- `docs` ..... Project documentation
- `nx` ..... Root of the application code
  - `apps` ..... Individual applications
    - `solar-backend` ..... Backend application
    - `solar-frontend` ..... Frontend application
    - `ngspice-consumer` ..... Ngspice consumer
  - `packages` ..... Reusable packages

*Note that this is not a complete directory structure of the repository.*

All services are written in Typescript (the decision for each service is discussed in the respective subsection.). TypeScript is a strongly typed programming language that is a superset of JavaScript. It is the third most popular programming language [23] as of 2021.

The implications of this choice with regard to monorepository management is that it enables the usage of tools specifically designed to work with Typescript-only monorepos-

itories. The tool selected for this task is NX[1]. The tool was selected mainly due to experience from previous projects and relative popularity in the developer community. At the time of writing, the official NX GitHub repository[2] had 22.000 GitHub stars.

### 5.2.2 Backend

The backend service is implemented using Node.js with Typescript. The final decision to use Node.js was due to the overall architecture of the application, where the backend should ideally only process requests and schedule asynchronous tasks. This workflow is well suited for Node.js due to the event-loop and low resource consumption relative to other considered technologies. Due to the architectural decisions, the application should be easily vertically scalable, so the single-threaded nature of Node.js should not be an issue. The place where a potential bottleneck could be seen is with the usage of SQLite database since all the database operations require CPU time from the process performing the queries, which differs significantly from the commonly used databases like PostgreSQL or Mysql, where the application only sends the queries to the database server, and can wait for a response, without blocking the CPU. This might block the event loop and cause delays in the application. This can be easily overcome by vertically scaling the application or separating the functionality into separately deployable artifact, if that ever causes issues, and was considered before choosing the technology.

The particular framework used is Nest.js[3]. It offers a reasonable level of abstraction and a lot of the features known from frameworks in other languages, such as Dependency injection, while simplifying the work by providing existing reference implementation. It provides out-of-the-box support for microservices using numerous communication technologies, including RabbitMQ.

The application is separated into multiple Nest.js Modules. A module is essentially a grouping of controllers, services, and entities. It is an effective way to manage and minimize dependencies between parts of a single application, as the module has to explicitly declare so-called exports that can be imported into other modules. This provides clear visibility into dependencies between individual functionalities of the application and simplifies testing. The module hierarchy is depicted in Figure 5.3.

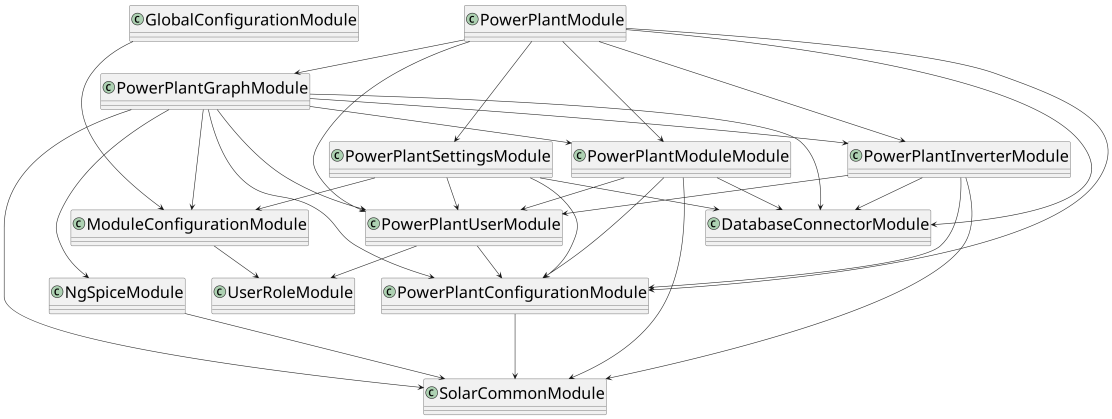



**Figure 5.3.** Module dependency graph

[1] https://nx.dev
[2] https://github.com/nrwl/nx
[3] https://nestjs.com

#### Pagination

All listing endpoints where the data growth is expected are implemented with pagination and filtering by default. Applications without proper pagination and filtering often suffer from a significant degradation in user experience as the amount of data grows.

A generic pagination solution was implemented as a separate Nest.js Module. It allows for a simple definition of a new paginated endpoint, ensures consistency for all such endpoint shapes, and allows the building of generic solutions on the client side. Using the relatively advanced options of type manipulation in Typescript, together with the Typia library[1] allows for type validation without libraries such as class-validator[2] and class-transformer[3]. It enables the developer to write a simple definition and generate complex types, complete with input validation.

### 5.2.3 Ngspice worker

The Ngspice worker service is implemented using the same technologies as the Backend. The code consists only of a thin wrapper that consumes RabbitMQ messages, calls the Ngspice binary in a subprocess, and publishes other RabbitMQ messages. The reasoning behind the decision to choose Typescript and Node.js is the same as for Backend, with the added benefit that this service does not use SQLite database, so there should theoretically be no CPU-bound tasks in the application itself. Its lifecycle is the following:

- *Receive message* - The worker receives a message with an identifier and a Ngspice netlist.
- *Run simulation for given netlist* - Compute the simulation using Ngspice.
- *Save the results* - Upload the graph data to shared storage.
- *Notify* - Emit an event, that the netlist has been simulated and uploaded.

To allow the execution of Ngspice in a subprocess, the instances of this service will be run in a docker container with Ngspice installed, as described in Section 5.3.2.

### 5.2.4 Frontend

For a client application, the choice of technology was limited by the browser support. This means primarily Javascript, or languages that are transpiled into Javascript. The technology chosen for this project is Typescript.

React.js was chosen as the rendering library, as it was the most commonly used web framework as of 2021 [24]. Login is required to access any part of the application, so SEO or server-side rendering is not necessary. For this reason, only the client-side React.js application was used without any server-side rendering.

[1] https://github.com/samchon/typia
[2] https://github.com/typestack/class-validator
[3] https://github.com/typestack/class-transformer

The application uses react-router[1] for route management, and has the following route hierarchy:

- `/` ..... List of power plants
  - `/plants/:plantUuid` ..... Power plant detail page
    - `/plants/:plantUuid/modules` ..... Power plant - list of modules
    - `/plants/:plantUuid/inverters` ..... Power plant - list of inverters
    - `/plants/:plantUuid/map` ..... Power plant - map
    - `/plants/:plantUuid/settings` ..... Power plant - settings
    - `/plants/:plantUuid/debug` ..... Power plant - internal generation statistics
- `/global` ..... Global PV module settings
  - `/global/substring/:substringUuid` ..... Substring detail page
- `/users` ..... List of users
  - `/users/:userUuid` ..... User detail page

#### Authorization

The application supports role-based route display and role-based element display. In practice, that means that certain routes, buttons, and forms are only visible to internal users, such as the options to delete generated data.

#### API Client

The communication between the Backend and Frontend is done using REST API. Typically, the calls from frontend would be done using the fetch API[2], or a library such as Axios[3]. The calls would be manually written and type-annotated on the client side. This introduces zero coupling between the type systems on both sides but also increases the surface for bugs, as there is no static analysis that would prevent the change of the API shape on the backend side while it is being used. This results in an elevated need for testing after such changes, which drives up the development costs.

As a part of this thesis, the API client is automatically generated based on the generated OpenApi specification. This ensures that the request and response objects are correctly typed, and breaking changes on any side prevent the build of the other application. The OpenApi documentation process is described in detail in Section 5.5.2. The client is not committed as part of the code and is automatically generated during the build process or as part of the developer workflow for local development.

#### Architecture summary

The complete architecture schema, including the individual artifacts can be seen in Figure 5.4.

## 5.3 Ngspice integration

Early on, it became apparent that relying on system-wide installation of Ngspice creates significant friction. Researchers working on the project are using multiple platforms (MacOS, Windows, various variants of Linux). Each of the platforms has component

[1] https://github.com/remix-run/react-router
[2] https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API
[3] https://https://axios-http.com

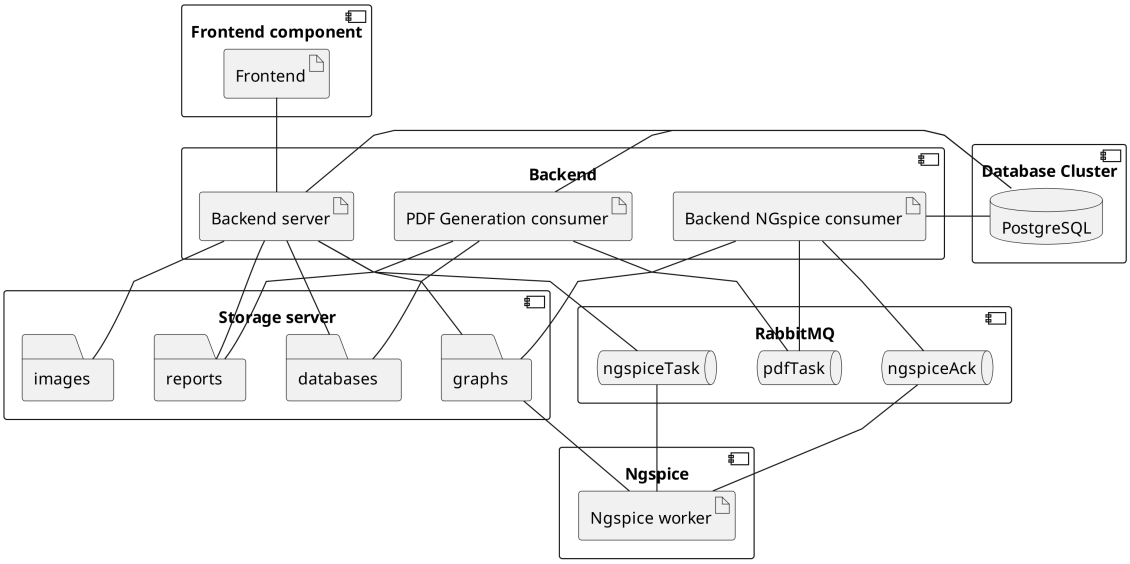



**Figure 5.4.** Architecture schema - complete

libraries installed in different paths, uses different newline delimiters, and other minor nuances. This complicates the sharing of Ngspice schemas and makes the runtime properties of each file dependent on the machine on which they were tested. It is possible that if the engineer tests a schema on his machine, it will not work once it is uploaded to the production server.

It is, therefore, desirable to have a standardized runtime environment for Ngspice that could be shared and used by the whole team. It should behave the same regardless of whether it is the production environment or it is run locally on the developer machines.

In order to be able to use Ngspice in our application, we need to be able to

- Generate Ngspice input
- Provide the necessary component libraries
- Execute the simulation
- Gather and convert the output
- Handle potential errors during the simulation

Two execution approaches were evaluated. They are described in the following sections.

### 5.3.1 WebAssembly

WebAssembly (WASM) is a binary instruction format for virtual machines and a viable compilation target. It is designed to run with near-native performance while being in a memory-isolated sandbox with explicit permission management. This provides undisputed security benefits. Since WebAssembly is supported in most popular web browsers, it is possible to run the same compiled library on the server and on the client.

The benefit of compiling Ngspice this way would be the decrease in the number of external dependencies, as the output would be a single WASM binary that could be executed from any supported environment.

The approach evaluated in this thesis compiles Ngspice to WebAssembly using the Emscripten[1] toolchain. Emscripten is a Compiler toolchain that uses LLVM as its backend. It supports APIs such as pthreads and POSIX.

With such compiled binary, we would then be able to run it with appropriate runtime environment. In the context of this thesis, the assumed runtime environment would be the V8 engine.

Another added benefit is that it would allow Ngspice execution on the frontend, so that users could test new schematics directly. Without this option, if the users wanted to test any Ngspice schema, they would have generate the schema locally and test it on their personal installation of Ngspice. This often result in numerous compatibility issues when they later decide to run the schema against the server installation, because the created netlists would usually depend on static libraries linked through filesystem paths, platform specific paths, or other locally imported file paths.

The chain of events to generate the output would be the following:

- Program generates input file
- Calls the Ngspice WASM wrapper
- Redirects the stdout and parse the output
- Handles potential errors during the simulation

One minor problem with this approach is that since there is no persistent filesystem, all the required component libraries need to be copied to the created web assembly instance. The number of libraries might grow significantly in the future, and it is desired to enable the users to upload their own libraries.

This approach worked well for smaller circuits, but the main drawbacks became visible during later phases of the testing process, with the biggest one being the memory limit of WebAssembly, which is currently at 4GB [25]. The memory limit became an apparent issue for larger circuits (as low as hundreds of modules on a single inverter). Other issues included the difficulties with multithreading. Ngspice uses openMP for its parallelization, which posed significant issues during the compilation to WebAssembly.

Furthermore, runtime was 10 times slower than the native version for single-core runs, which would further increase the project's operational expenses, as more hardware would be needed to provide the same functionality. Larger instances (thousands of modules on a single inverter) could not be run at all.

This approach was abandoned as it would result in decreased reliability for larger instances and the management of component libraries would defeat the purpose of easy unified runtime on the client side, as if the libraries were not part of the distributed binary, we would still need to ensure that client is using only the libraries available on the server.

### 5.3.2 Native binary

Other approach it to utilize the native Ngspice binary. The simplest approach would be to install the binary on the system and rely on that. It solves the memory and

[1] `https://emscripten.org`

multithreading issues from the previous approach, but makes the application deployment dependent on the version of the package installed on the system, as well as its configuration.

To remove the dependency on the system it is running on, a docker image with Ngspice installed is created, which will be used to run the part of the application that requires Ngspice. This makes it independent of the underlying operating system. The overhead of the docker container is minimal.

The flow of the program using native binary would be as follows:

- Program generates input file and stores it on local filesystem
- Executes the Ngspice binary in a subprocess
- Gathers and converts the output
- Handles potential errors during the simulation

The main benefit of this approach is that the final docker image can then be distributed among the researchers for local schema testing, together with providing a standardized runtime environment for the development and production environment.

## 5.4 User experience

User experience is one of the most important aspects of any application, as it directly impacts the user's satisfaction when using the application [26]. The understanding of individual use cases and application screens is necessary to understand the user experience, as each screen directly contributes to the overall usability, functionality, and effectiveness of the application in facilitating the user's goals. This section describes a few selected use cases and screens for a better understanding of the implemented application and evaluation of the overall application capabilities.

### 5.4.1 Power plant overview

After opening the application, the user is presented with a list of power plants. After selecting a power plant, the first screen the user is going to see is the overview of the power plant (Figure 5.5). This screen provides a quick overview of the power plant state while providing the monetary value of the failures based on the currently selected calculation model.

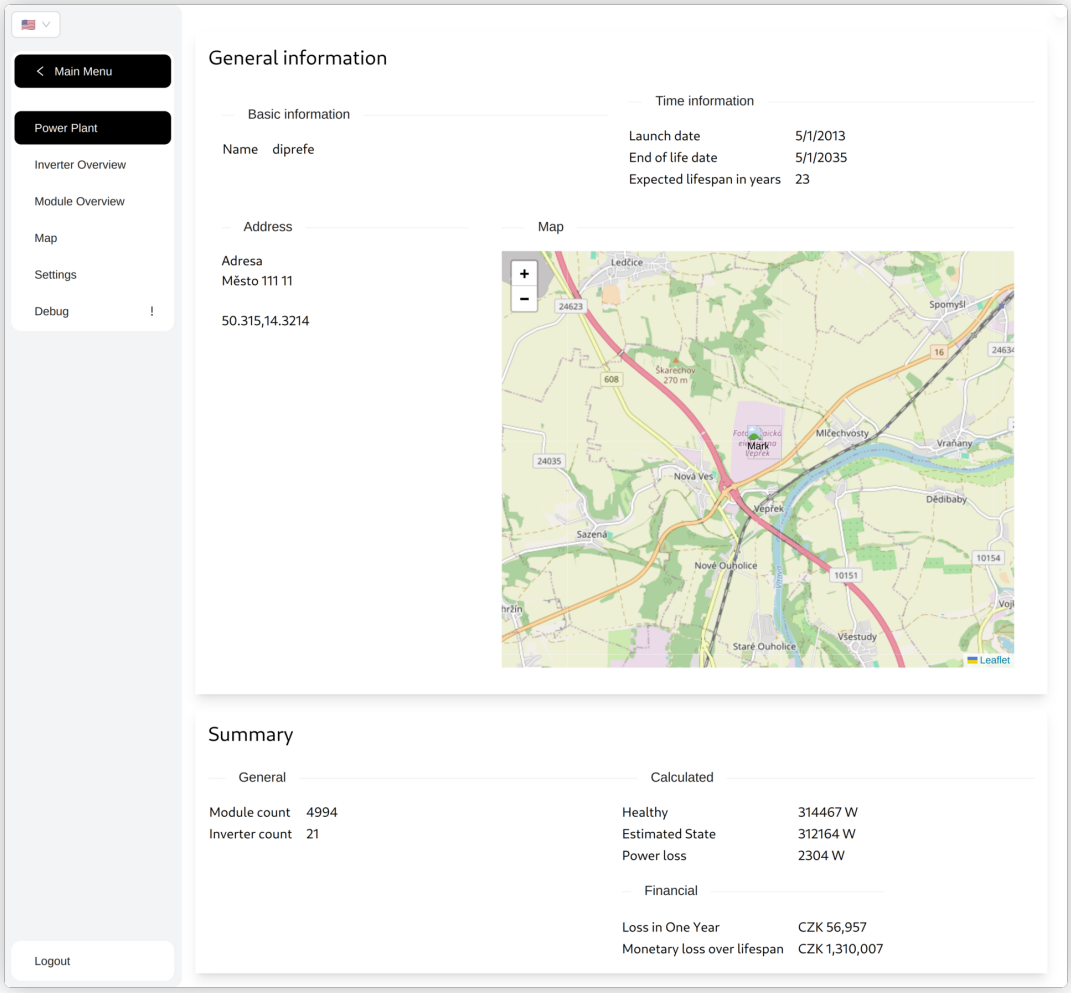



**Figure 5.5.** Power plant overview

### 5.4.2 Map

One of the most important views in the application is the map view (Figure 5.6). It provides the user with a graphical representation of the power plant's physical layout, with a color annotation based on the state of the module. This provides an intuitive way to explore the data and assess the patterns related to physical layout. The user interface is designed to be intuitive and similar to other map applications. Users can zoom in and out and pan across the map.

Users can also interact with specific modules to access detailed information and select them. Once the modules are selected (Figure 5.7), the user can easily navigate to the module detail (Figure 5.12) while keeping the selection state. This allows them to easily cycle between modules to compare their results.

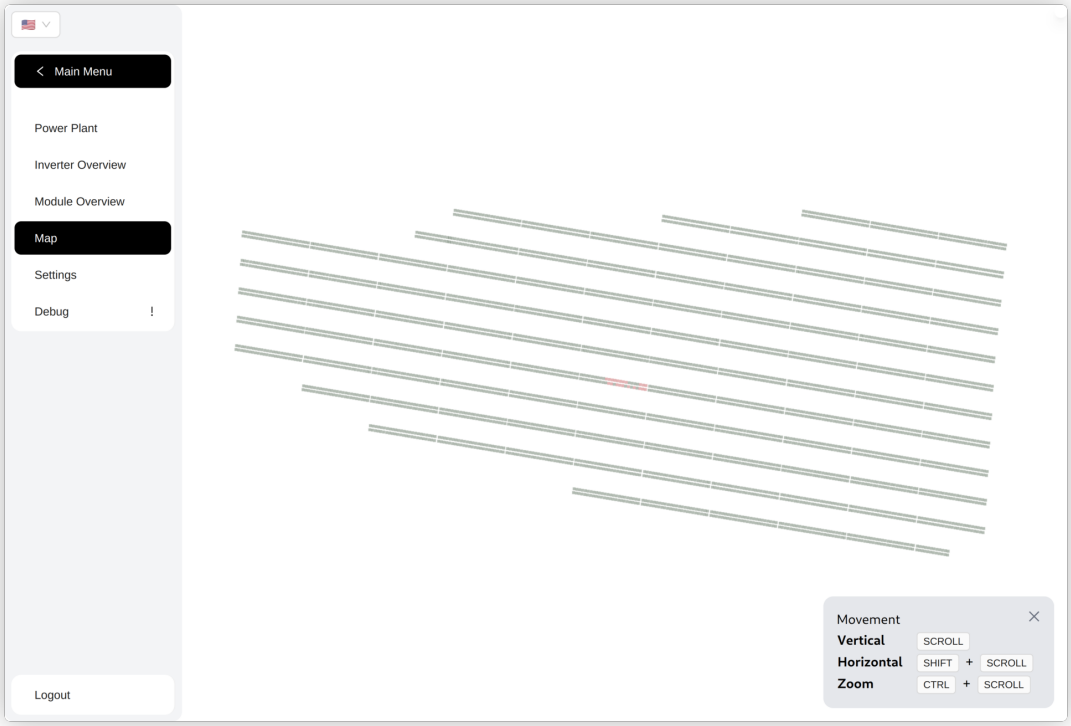



**Figure 5.6.** Map view 1

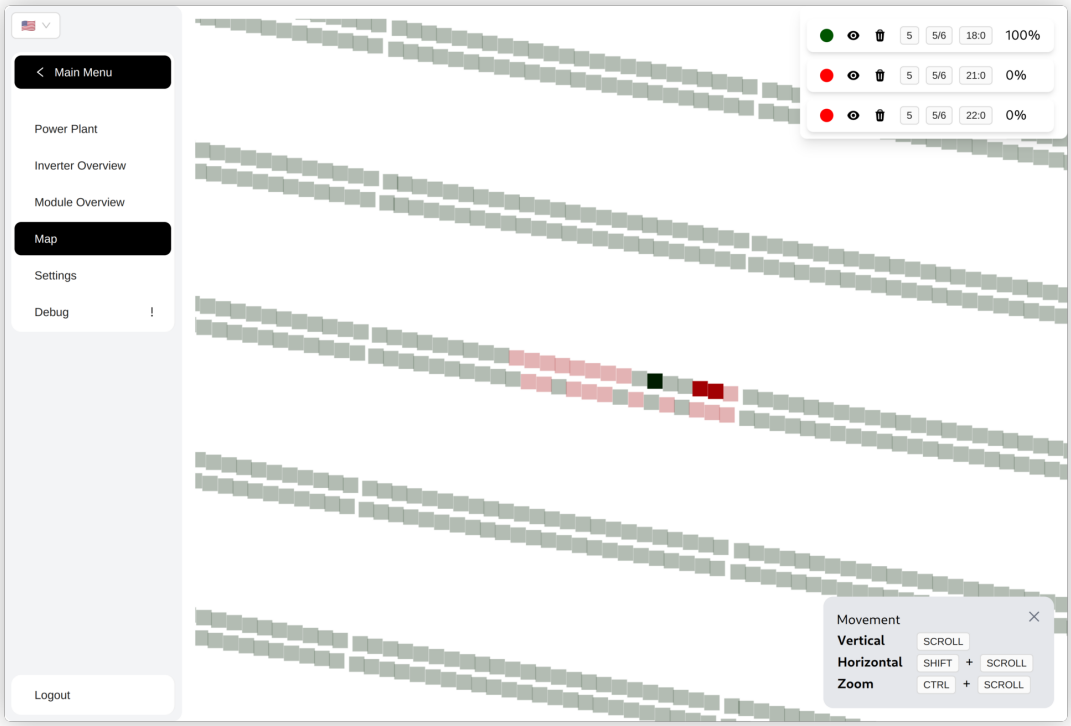



**Figure 5.7.** Map view 2

### 5.4.3 Power plant settings

Each power plant is saved with relevant metadata, such as address (Figure 5.8), that is used primarily to visually assert the user that the correct power plant is selected, since the naming scheme used for the power plant is not necessarily understandable for all users.

Other metadata include the settings related to the power plant lifespan, which is crucial for calculating the economy impact over time. Users can adjust this value to experiment and see the impact on the overall economy of the power plant.

It is also possible to select different module types for the power plant (Figure 5.9) with a single selection. The user can select from many available module types or request to add a new module type from the researchers. This is currently not used by end users since the primary power plant module type does not change, but it is a useful option for experimentation.

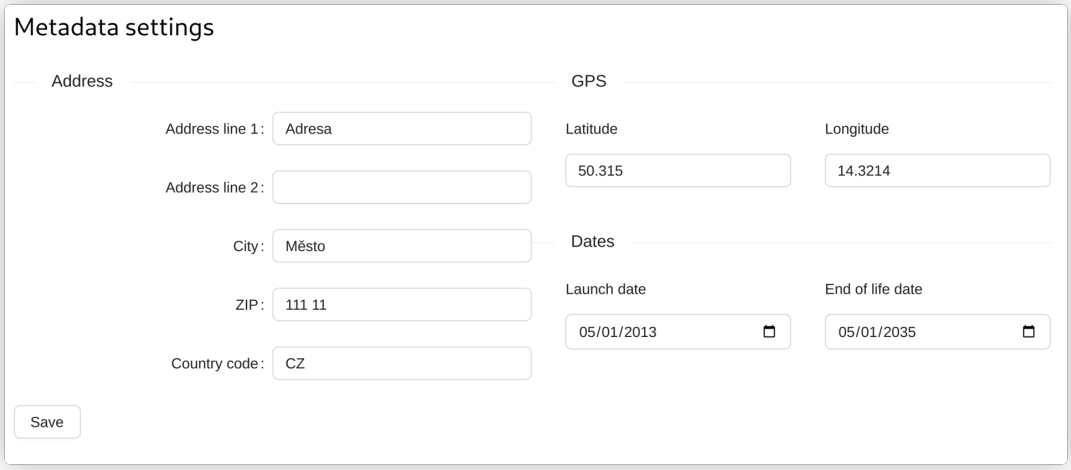



**Figure 5.8.** Power plant metadata settings

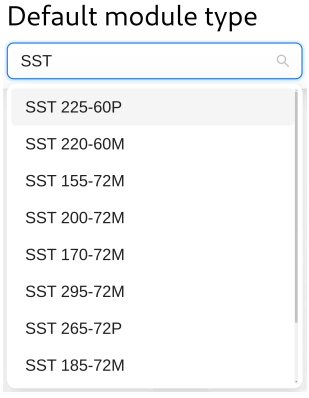



**Figure 5.9.** Module type assignment

### 5.4.4 Power plant economy settings

In order to efficiently calculate the impact of the faults on the overall power plant economy, it is necessary to give users a way to specify the exact economic conditions for their power plant (Figure 5.10). General economy settings are available, while three financial impact calculation strategies are available. One strategy assumes a flat electricity buyout price over the whole lifespan of the power plant. This will probably be used only as an initial example when presenting the data. For the majority of cases, the *Yearly* strategy reflects the economic conditions. It allows the buyout price for each year to be set separately over the lifespan of the power plant. For rare cases, when custom calculation is necessary, it is possible to set a custom function.

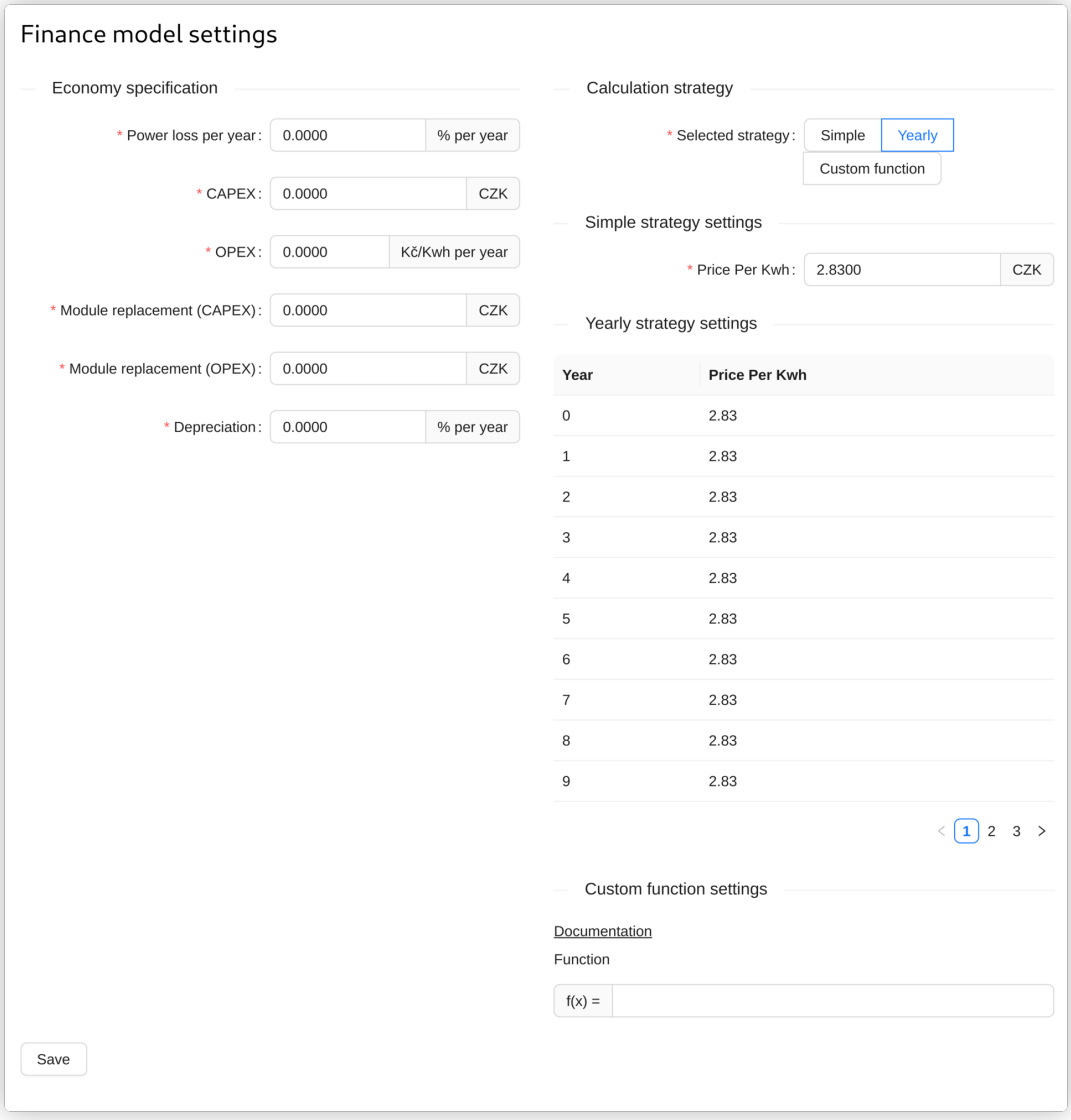



**Figure 5.10.** Power plant economy settings

### 5.4.5 Detail pages

In order to explore individual failures, each module and inverter has an available detail page with detailed graphs and a summary of the data.

For a module (Figure 5.12), the detail page shows the current state based on the thermography data, compared to the state without a failure. For the current state, images from the thermography and their schematic interpretation are displayed.

For an inverter (Figure 5.13), the current state is compared with the state without any failure on all the connected modules. For the current state, the electric schema is displayed, highlighting the individual module states with the appropriate color.

For both states, the result of the Ngspice simulation is shown in the form of a chart. The most important data from the chart is the peak performance value, which is shown in the table at the top of the respective page.

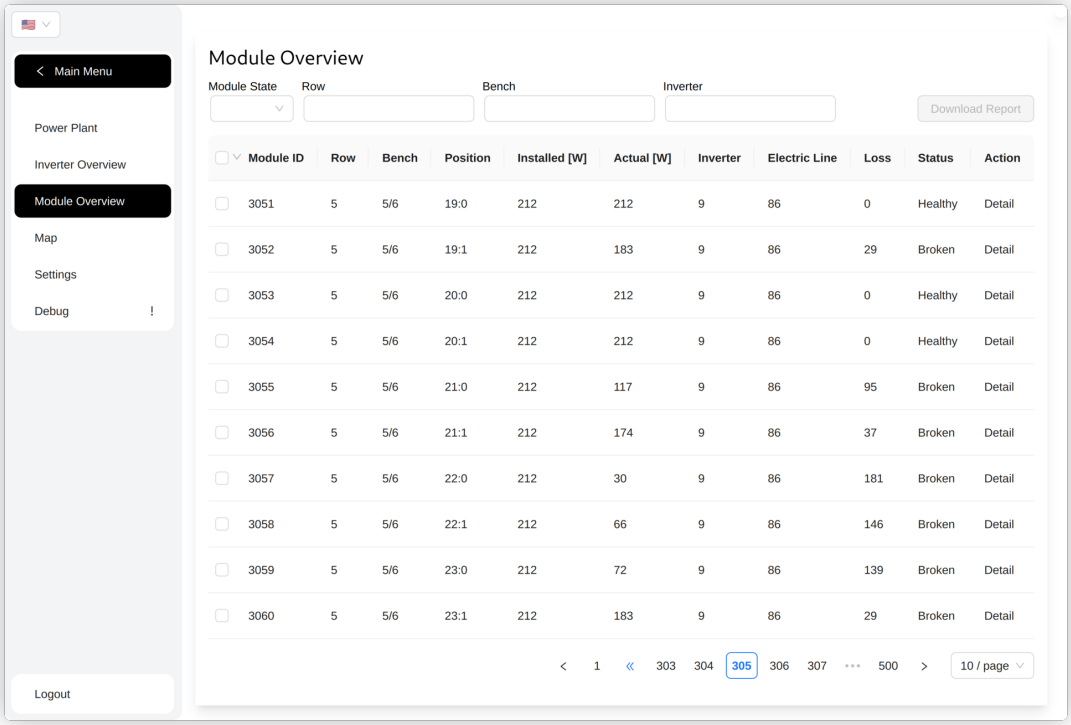



**Figure 5.11.** Module list

### 5.4.6 Listing pages

A method to quickly find the resources the user is looking for is crucial for good user experience. In this application, the relevant searchable resources are the Inverters and Modules. For this purpose, the listing pages are available, as can be seen in Figures 5.14 and 5.11, respectively. The module listing supports filtering based on the status, particular physical location, or the Inverter, while the Inverter listing only supports filtering based on the status. With a single click, the user can navigate to the respective detail page for each resource.

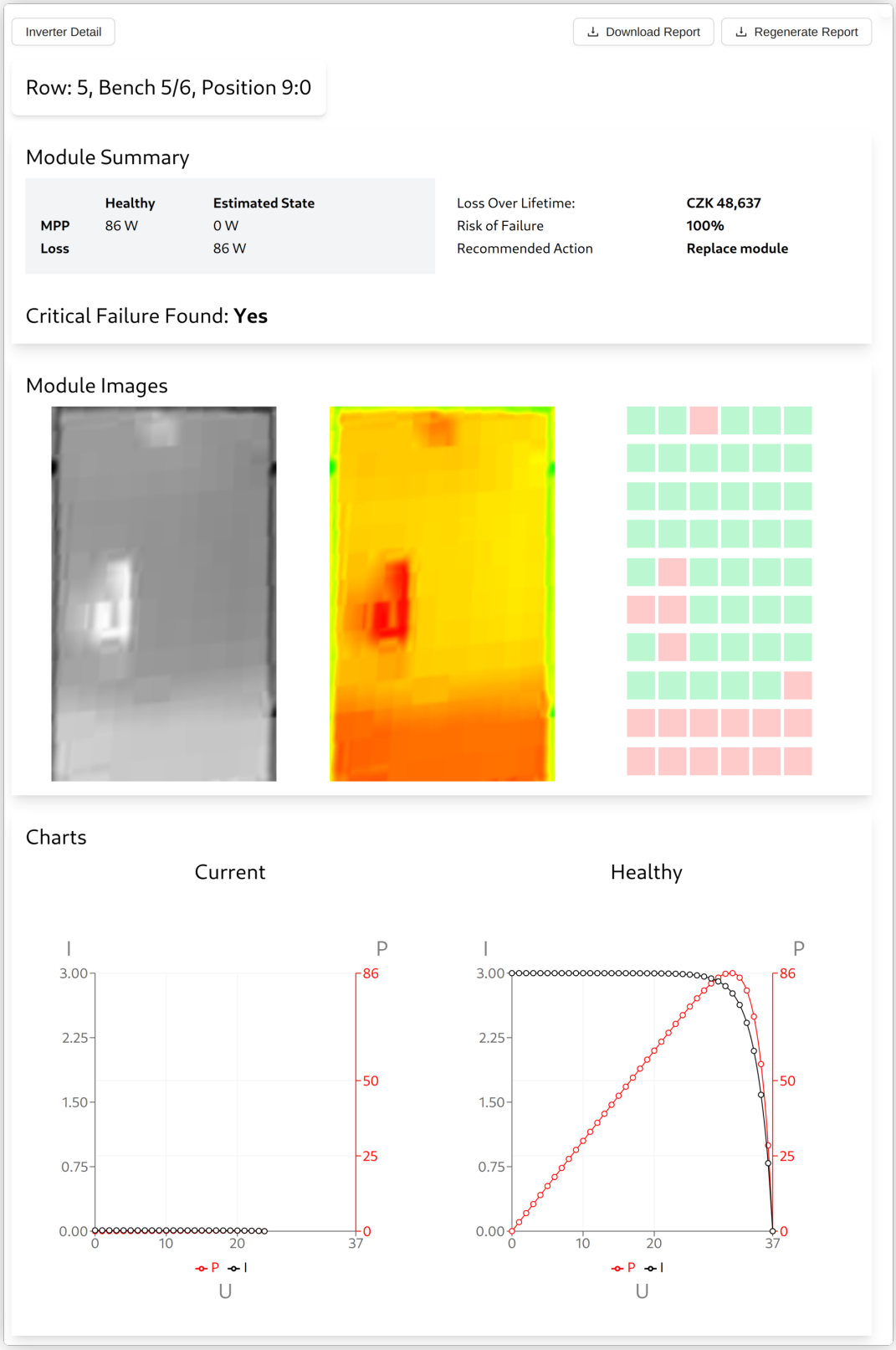



**Figure 5.12.** Module detail


**Figure 5.13.** Inverter detail

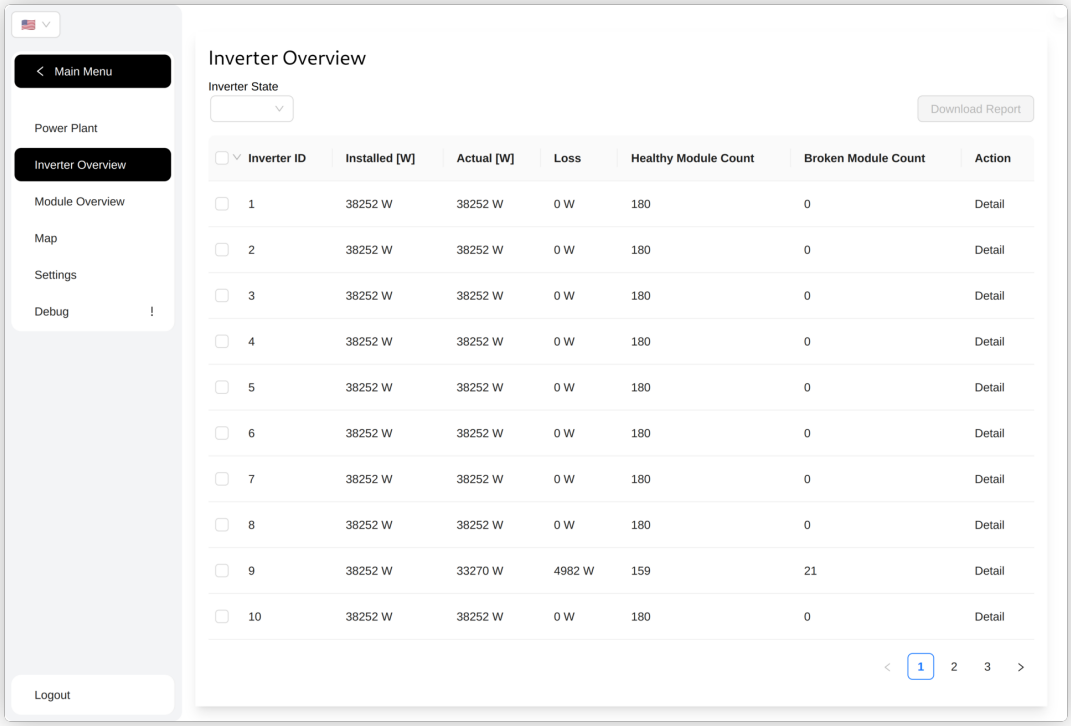



**Figure 5.14.** Inverter list

### 5.4.7 Report generation

The web application is useful for visual exploration and easy navigation, but for a lot of mostly legacy reasons, PDF reporting is still the norm in the PV industry. For the enterprise user, the function of generating and exporting PDF reports is, therefore, as important as the rest of the application.

Application users can download a report for each module or inverter. They can either download a report of a single module or inverter on the respective detail page (Figures 5.12 and 5.13 in the upper right corner of the page) or bulk download the module/inverter reports in bulk on the respective listing pages (described in Section 5.4.6). The structure of the reports for modules and inverters can be seen in Figures 5.15 and 5.16, respectively.

#### Implementation

In order to have a reasonable user experience when downloading the reports, especially for bulk downloads, it is necessary to pre-generate all the data so that the user does not have to wait for the generation, and the server could just return the pre-generatated items. This results in a significantly better user experience, where the majority of time is spent waiting for the transfer of the PDF documents.

Another not obviously complex, yet not an easy-to-solve problem this creates is the actual item selection for the bulk actions. On frontend, the user has access to paginated tables, so he usually sees only about 25 items on a single page. For bulk actions, the user has the option to select all currently filtered items, only the current page, or select

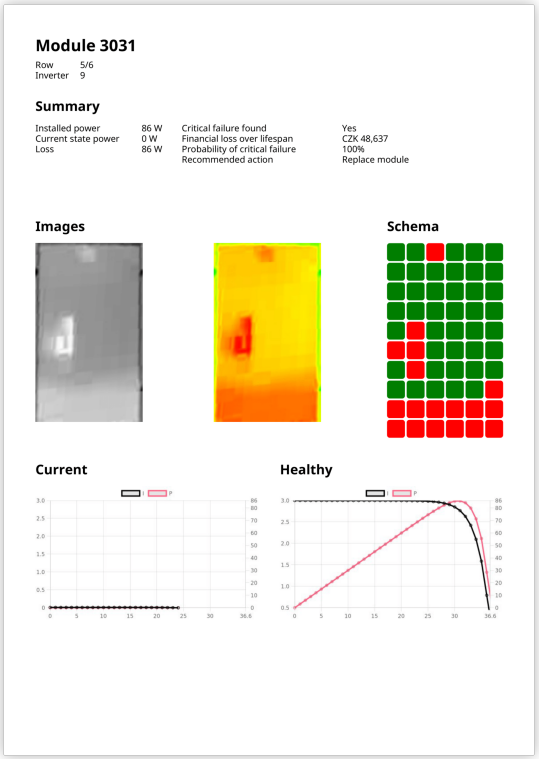



**Figure 5.15.** Module report

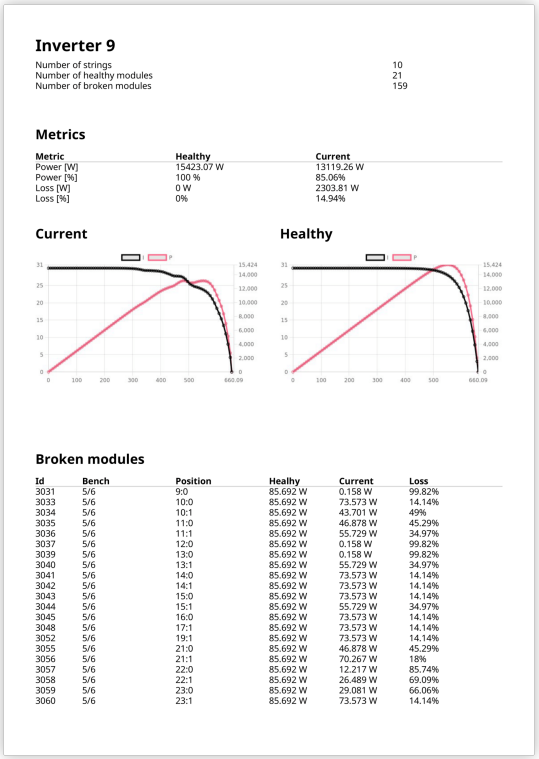



**Figure 5.16.** Inverter report

particular items. In the first case, the list of the affected items can potentially have millions of items. The implemented solution, therefore, is to send the currently selected filters, together with the individually selected items, to the backend, where the data is accessible. The other possible approach commonly used is to only send the identifiers for the bulk actions, but that approach is hard to scale for large datasets.

### 5.4.8 Module type configuration

New module types are being introduced to the market rapidly. This creates a need to support adding new module types directly in the application so that the PV researchers can add PV module types without any developer action. For this purpose, a simple configuration screen was implemented (Figure 5.17). It enables the researchers to modify the electrical properties, as well as the base electrical schema used for the simulation, in case any non-standard configuration is needed.

As a means to minimize the required support with invalid schemas and misbehaving simulations, a validation step was added before saving the schema. A test simulation must succeed for the edited schema before saving it is allowed in the application. This eliminates most of the manual user errors. This verification step is shown in Figure 5.18.

As an optimization for better user experience, a bulk upload option was added after identifying the root data the researchers use for their representation of the modules. It is possible to upload raw schematic files, and the system automatically parses them and creates or edits module types in bulk, saving hours of work when compared to manual creation in the application using the standard edit flow.

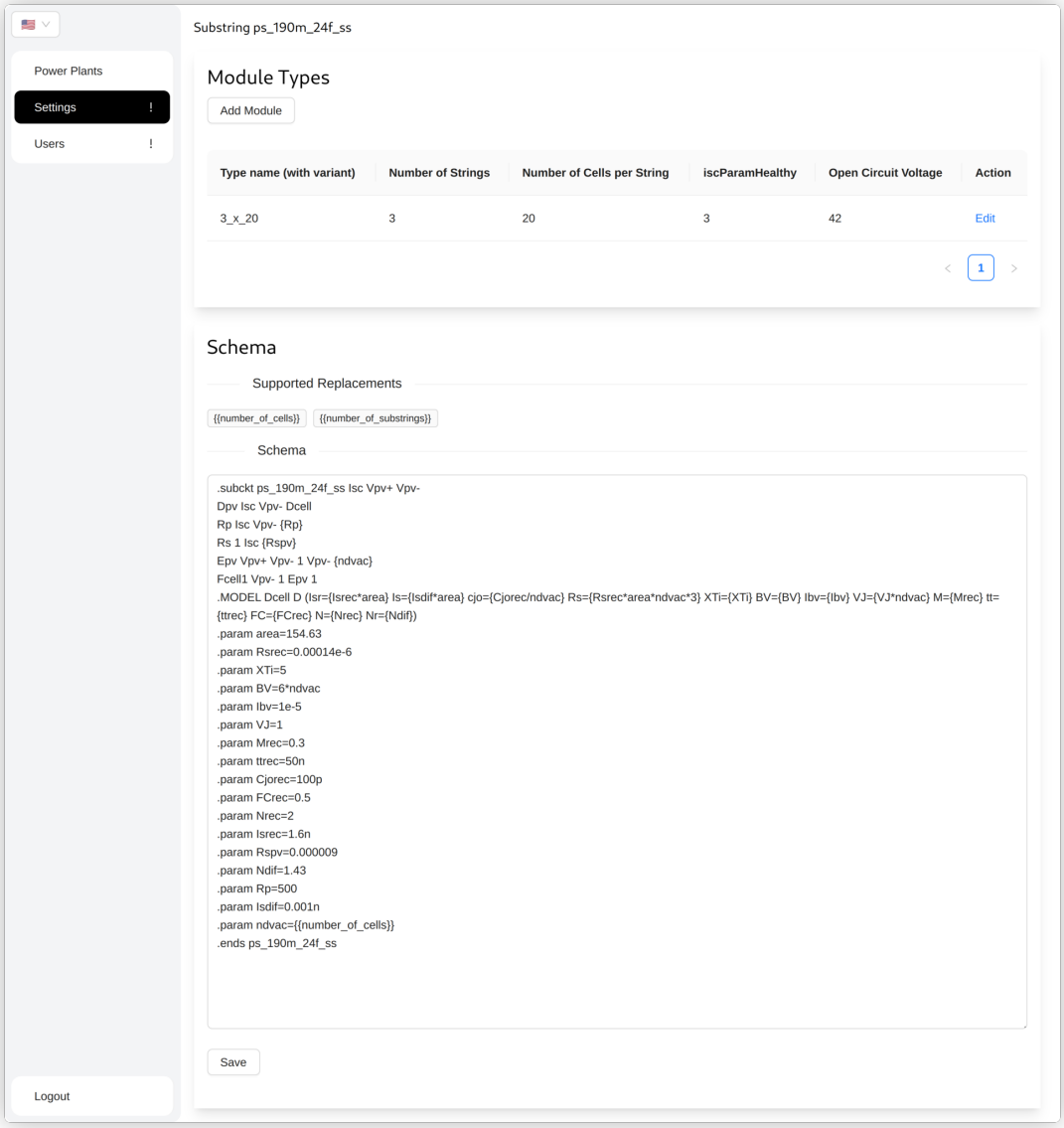



**Figure 5.17.** Module type configuration

### 5.4.9 Simulation management

The simulation of a power plant can take a significant amount of time (up to 24 hours, based on NR 4.6). A screen, available to internal users, showing the current simulation process with additional controls for the simulation was added and can be seen in Figure 5.19. It primarily allows the researchers to reschedule the simulation and regenerate PDF documents when the automatic scheduling is not enough (e.g. after a manual schema update).

### 5.4.10 User management

Internal users can manage the access for all application users. For this purpose, a page that lists all the registered users was added to the application. Each user has a detail page, as can be seen in Figure 5.20, where the necessary management actions

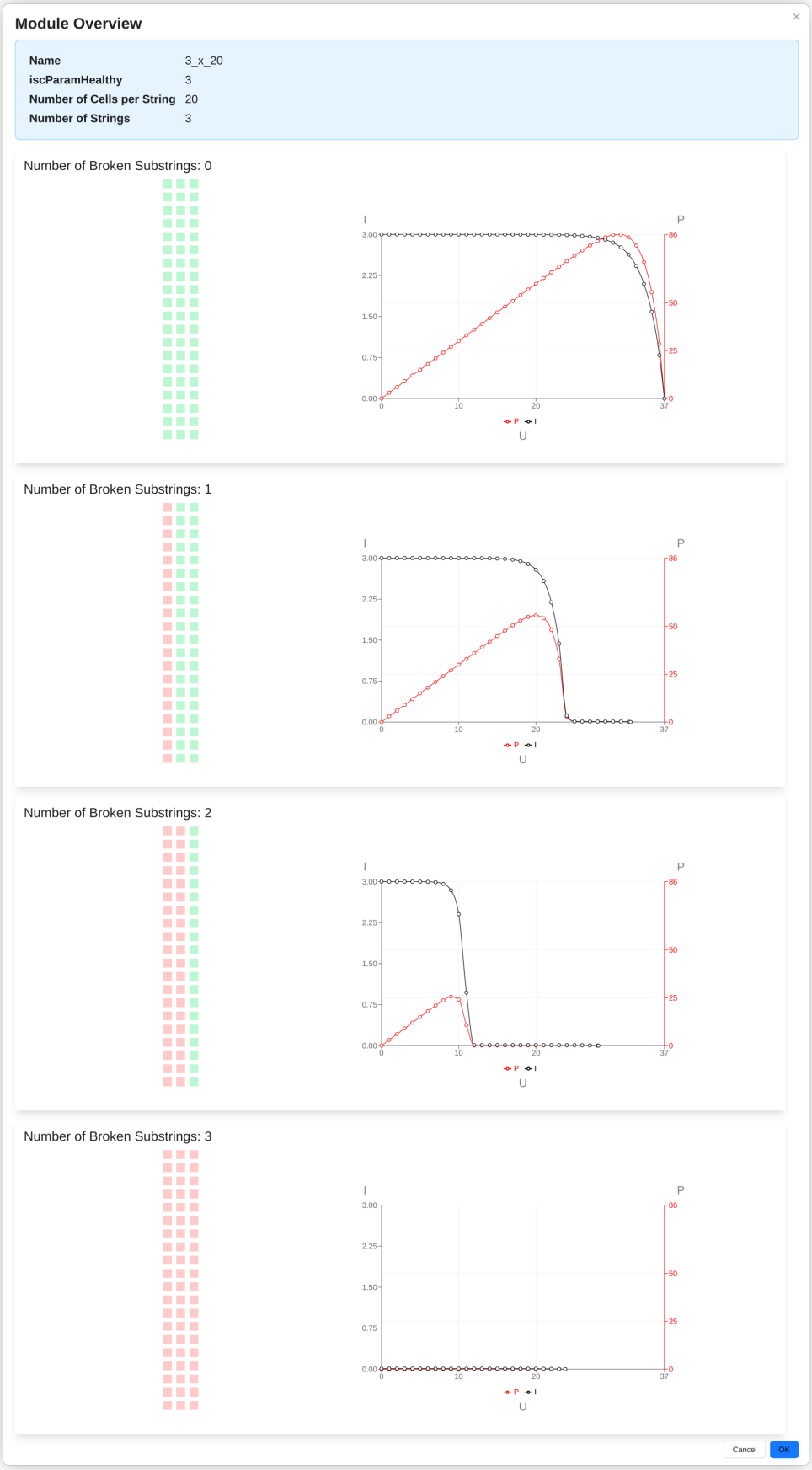



**Figure 5.18.** Module type verification

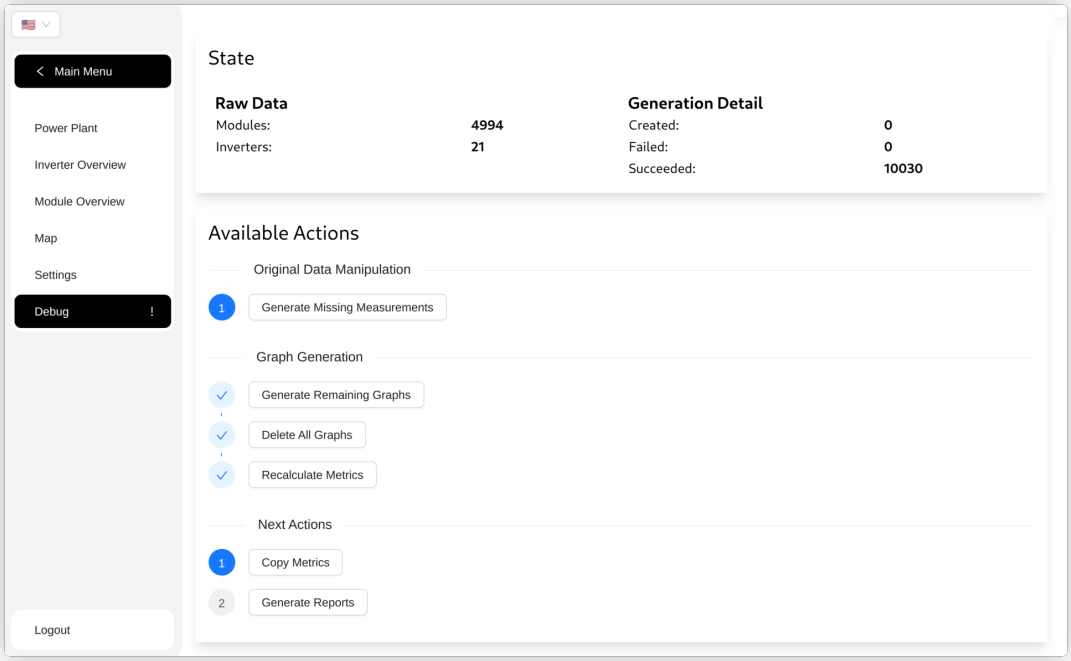



**Figure 5.19.** Simulation management

are available. After registration, each user has to be verified by an already registered internal user to prevent unmanaged access to the platform. A button for the verification process can be seen in the upper right corner of the user detail page. It is also possible to limit the user access to certain power plants by moving the power plants between the *Available Power Plants* and *Power Plants to Assign* sections at the bottom of the user detail page.

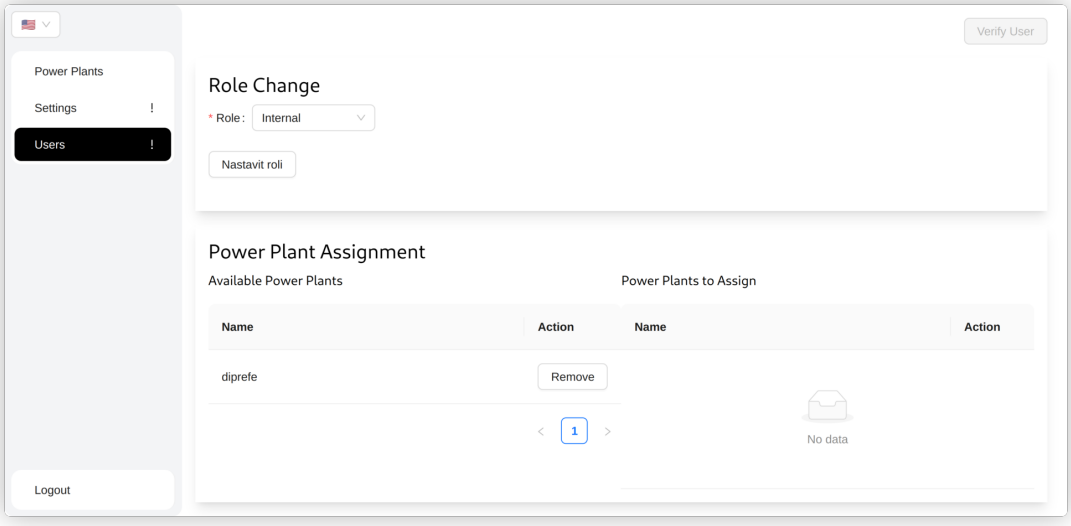



**Figure 5.20.** User detail

## 5.5 Documentation

Documentation is an integral part of software development and should be one of the outcomes of a software project.

Technical documentation plays a crucial role during project handover and future maintenance and development. User documentation should be available and should answer the user's question of how to perform certain actions.

### 5.5.1 Technical documentation

#### Developer documentation

To simplify new developer onboarding and to help with understanding the codebase and related developer processes, a comprehensive developer documentation was composed. This documentation is in the form of Markdown[1] documents with a hierarchical structure. It guides the developer through the process of setting up the repository on a new machine and documents the necessary steps to successfully commit to the codebase, such as linting and automatic code generation setup. This documentation can be found in the *docs* folder as part of the repository to eliminate any additional deployment steps.

#### OpenAPI

Formerly known as Swagger, OpenAPI is a specification for building APIs, focusing on machine-readable interface files. These files, typically in YAML or JSON format, outline the structure, endpoints, parameters, responses, and authentication requirements of an API. This detailed documentation clarifies API functionalities and standardizes its interface, aiding seamless integration and interoperability.

In this project, OpenAPI specification is automatically generated from the Backend code. It provides clear documentation for future project developers and enhances automated type-checking across the whole application.

Based on this specification, an API client is automatically generated for the Frontend application. This ensures that if a breaking change was introduced in any part of the application (e.g., changing the shape of the API, invalid usage of the API on Frontend), the type system would not transpile the code, and the programmer would have to fix the issue, instead of the error being discovered at runtime. The specification can also be used for manual API testing through tools such as Postman[2] and Insomnia[3].

### 5.5.2 User documentation

Especially among enterprise users, it is necessary to have all the processes documented, even though they may seem straightforward to the average user. Any interface change that is not documented might pose a need for additional training on the customer side. Documentation of the common actions performed by end-users was composed to

[1] https://docs.github.com/en/get-started/writing-on-github/getting-started-with-writing-and-formatting-on-github/basic-writing-and-formatting-syntax
[2] https://www.postman.com
[3] https://insomnia.rest

simplify the onboarding and training of new customers. This documentation consists of the steps to accomplish the basic actions such users might need. Such as:

- The registration process
- Map navigation
- Explanation of the individual detail pages

Additional documentation was composed for the internal users, primarily focusing on the management processes. The actions documented for this user group include:

- Adding new power plant
- Uploading a database
- Uploading images
- Managing simulations
- Module type management

With this documentation, the onboarding process for both user types should not pose a problem even after passing the ownership of the software. Future maintainers of the software will be able to simply document new processes as they are developed.

# Chapter 6
# Deployment process

This chapter elaborates on the deployment process, the tools used, and the selection of the target platform.

It is necessary to evaluate the decisions made in this chapter in the organization's context and with the available maintenance capabilities in mind. There are numerous highly sophisticated orchestration tools available that would accomplish all the discussed requirements, but the maintenance of these tools would impose a significant overhead for the organization. The decisions made in this chapter were primarily made with the intention to minimize the total cost of ownership of the deployment process while keeping the required functionality.

## 6.1 Considerations

Among the most important considerations for the deployment platform is compliance with the non-functional requirements introduced in Section 4.2.3. This section elaborates on the most impactful ones in detail.

### 6.1.1 Peak performance requirements

For the expected application access patterns, the peak performance is going to be hit during the calculation of new power plant data after uploading the relevant SQLite database file. If we assume that a large power plant can have over three million individual modules (such as the *Longyangxia Dam Solar Park, China* power plant, with a capacity of more than 850MW)[1], and we want to generate a PDF for each one. Each of which takes approximately $1s$ (a conservative assumption to simplify the calculations). PDF generation alone would have a serial runtime of:

$$T_s \approx 3000000s \approx 833h \approx 34d$$

This is an unreasonable amount of time for anyone to wait if the application had to perform this operation sequentially.

It was decided that the maximum time for the generation phase of such a large power plant is about 12 hours. (The strict requirement introduced in the non-functional requirements NR 4.6 sets this limit to 24 hours. The 12-hour strict limitation is introduced here as a means to improve the worst-case user experience, even though it is not required based on the requirements alone). For a great user experience, we would ideally want to process the whole power plant within 1 hour.

[1] https://www.power-technology.com/features/the-worlds-biggest-solar-power-plants

This gives us the necessary speedup of at least:

$$S = \frac{T_s}{T_p} = \frac{3000000}{43200} \approx 70$$

Where $T_s$ is the serial runtime, and $T_p$ is the parallel runtime in seconds. That means we would need at least 70 parallel processors to complete the task in a reasonable time. *This is a simplified calculation which disregards any potential overhead in a real-world scenario.*

### 6.1.2 Load characteristics

The primary requirement for the application is that it has to be able to serve requests continuously without stopping. This can be handled by a single machine using approximately 0.1 CPU cores with 256Mb of RAM, as the request count is expected to stay in the single-digit numbers even during peak load.

The computationally intensive load is expected to burst when new data is uploaded. This will typically be a few times in the lifetime of a power plant, but it does require a huge amount of computing power for completion in a reasonable timeframe. We can assume that new power plant data will be uploaded once a month.

The assumptions introduced above give us a clear understanding of the computational resources necessary to operate the platform while meeting user expectations.

### 6.1.3 Durability

The majority of application data will be stored in a PostgreSQL database, together with images and simulation results stored on a filesystem, and further data will be temporarily stored in messages in exchanges.

For production usage, all of these data stores need to be properly backed up. In the case of managed solutions, the backup and disaster recovery process is handled by the service provider.

For self-hosted PostgreSQL, the backup process could either be solved by a near real-time replication or periodic snapshots of the database [27], Both of which require a proper maintenance and backup data validation. Such maintenance is not feasible, as it would have to be done by the research team. For the self-managed file system backups, the imposed maintenance is not acceptable as well. For the RabbitMQ, the disaster recovery process is outside of the capabilities of the future maintainers as well.

Therefore, the only feasible solution is the one where all three services are managed by the infrastructure providers, and the backups and disaster recovery responsibilities are delegated.

### 6.1.4 Maintainability

The deployment platform has to be chosen so that faculty full-time employees can manage it without problems or any additional training requirements.

The faculty currently operates a set of computational clusters, together with a set of managed services, such as managed PostgreSQL and S3-like storage solutions.

After discussions with the infrastructure providers, a solution that met all the computation requirements while being maintainable was introduced. Since there is an established process of managing virtual machines (VMs) and physical servers in the cluster, the feasible solution included the usage of a VM or a dedicated server.

## 6.2 Infrastructure

As discussed in Section 6.1.4, the feasible solution included a VM or a Dedicated server. After evaluating the available options for both, the most appealing solution consisted of:

- Single VM for the Backend server
- Dedicated server for Ngspice consumers and other Backend artifacts
- Managed version of PostgreSQL
- Managed version of RabbitMQ
- Network attached storage

Due to the operational delays and other external factors, the final solution was changed to the following:

- Dedicated server for Backend server, Ngspice consumers, and other Backend artifacts
- Managed version of PostgreSQL
- Network attached storage

For this deployment, a single dedicated server was provided to host the application. The specification includes 128 CPUs and 256GB of RAM. The resources should be more than sufficient for the intended workflow. As a result of this choice, all the application components are deployed on this single machine.

This could, in theory, mean that a significant load on the simulation could deplete all the available resources for the request processing part, therefore not meeting the non-functional requirement NR 4.2. Due to the significant cost of acquiring new infrastructure and the minor probability of such a thing happening, it was disregarded after thorough discussions with the stakeholders.

Another implication of this decision is in direct conflict with the requirements introduced in Section 6.1.3. In this case, the RabbitMQ instance has to be self-hosted on the dedicated server. Due to the imposed overhead, the backup and disaster recovery process for the RabbitMQ component has been disregarded. Instead, an option to regenerate all the messages that could possibly be lost in the event of a failure, was added to the application. This allows the researchers to reinitialize the RabbitMQ node and resume the operation with a single click of a button in the application.

The deployment diagram of the system can be seen in Figure 6.1. *Please note that the individual components are not split into individual artifacts, since all of the artifacts will be deployed on a single machine.*

### 6.2.1 CPU utilization evaluation

With the deployment platform specified, it is possible to calculate the utilization of resources based on the expected usage. For such calculation, we assume the size of

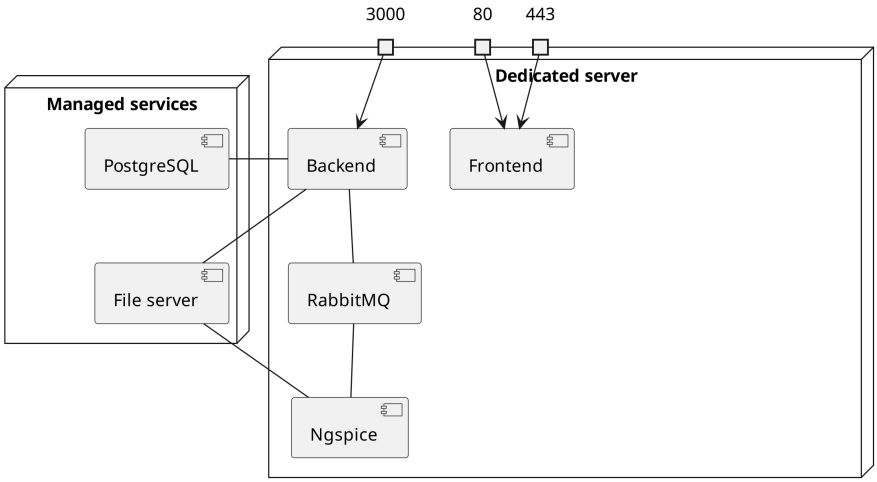



**Figure 6.1.** Deployment diagram with exposed port numbers

a power plant ranges from 5000 modules to 3000000 modules, with the median being 50000 modules, calculated once a month.

This means that the required compute time for the application is going to be approximately

- 267840 compute-seconds per month for serving requests
- 50000 compute-seconds per month for report generation

With the dedicated server provided and a single median power plant calculated each month, the average CPU utilization will be:

$$E \approx \frac{\overbrace{267840 + 50000}^{Useful\ CPU\ time}}{\underbrace{128 * 2678400}_{Total\ CPU\ time}} = \frac{317840}{342835200} \approx 0.09\%$$

If we assume that a single largest possible power plant will be calculated each month, the average CPU utilization will be:

$$E \approx \frac{267840 + 3000000}{128 * 2678400} = \frac{3267840}{342835200} \approx 0.95\%$$

This assumes that there will be only one application running on the server. The used server was deployed with the plan that there would be more applications running on it in the future. Therefore, this number might vary significantly based on future development.

### 6.2.2 Additional services

The deployed system consists of the components deployed as a part of this thesis, as well as all the used resources provided by third parties.

#### Network filesystem

A network filesystem was provided by the infrastructure team and was already mounted to the provided dedicated server. Therefore, any management related to the underlying server or the filesystem itself is delegated to the infrastructure team. The application can access the filesystem as if it were a local filesystem.

#### PostgreSQL

A managed version of PostgreSQL was provided by the faculty infrastructure team. The responsibilities for managing the database cluster are, therefore, delegated and not part of the deployment process. The application can use the database as a service without worrying about the underlying management.

#### RabbitMQ

As discussed in Section 6.2, no managed RabbitMQ instance was provided. The solution is currently to use a single-node dockerized version of RabbitMQ running on the provided dedicated server. Given the usage patterns for this particular application and the fault tolerance for the long-running bulk actions, it is not expected to become an issue in the future.

## 6.3 Continuous integration

GitLab CI/CD[1] was used because the organization relies entirely on Gitlab.

The standard process for continuous integration would be to run a pipeline that would trigger automatic build and testing of the pushed code.

Due to the organizational policies, it is forbidden to run docker-in-docker on the provided runners, so we are unable to build images in the integration pipeline under the current policies.

Other options could be to use self-hosted runners or third-party services. Using third-party services is not feasible, as the faculty approval process was not reasonable to accomplish during the time window provided for this thesis. Setting up self-hosting runners would be a viable option, but from a long-term perspective, it would increase the project's long-term maintenance while not providing many benefits compared to a local pre-commit hook, given the infrequent updates planned in the future, so it was discarded as well.

As a result of the policies in effect, the project ended up not using continuous integration, as the necessary communication with the IT department was not feasible as part of this thesis.

## 6.4 Continuous delivery

The standard process for Continuous Delivery would be to use an already built image from the Continuous Integration phase or to build a new image and switch the image deployed to the latest one as part of the configuration on the deployment platform.

[1] `https://docs.gitlab.com/ee/ci`

Since there is no way to build docker images directly in the pipelines, we opted for a non-standard solution. The deployment process is managed using Ansible, and the steps are following:

- Connect to the target server
- Verify the docker network setup
- Build all of the artifacts one-by-one, tagging them accordingly
- Execute pending migrations
- Stop the old version of services and start the latest version
- Start new application version

Due to the stopping and starting of the services, a short downtime happens. Typically it is less than 3s, according to the performed empirical tests.

Zero downtime updates were considered, but due to increased maintenance, they were disregarded. The application usage will be sparse and the frequency of deployments will not be enough to justify the development time spent on this solution.

Standard deployment platforms, such as docker-swarm/Kubernetes, have zero-downtime deployments solved by default. This is achieved by first scaling up the new version of the service, then redirecting all the traffic and stopping the original service as the last step. These options were disregarded, as they would impose additional knowledge requirements on future maintainers while not providing a justifiable benefit.

## 6.5 Deployment tools

The runtime environment on the target machine is a plain docker daemon. Docker swarm was evaluated, but due to the increased complexity, the future maintainers disregarded it.

### 6.5.1 Ansible

At its core, Ansible[1] is an open-source automation engine primarily designed for configuration management, application deployment, and task automation. Through modular playbooks and declarative language syntax, configuration tasks are executed and should be idempotent, mitigating the risk of configuration drift.

Developed by Red Hat, Ansible operates on agentless architecture, distinguishing itself from traditional configuration management tools. This agentless nature alleviates the need to install client software on managed nodes, thereby simplifying the setup process and reducing overhead.

[1] `https://github.com/ansible/ansible`

For the deployment process of this particular application, Ansible is used to declaratively

- Setup the networking
- Clone a new version of the repository
- Build new images
- Start docker containers

It should be possible to run exactly the same Ansible script against a different machine and end with the same application deployment. *This disregards any additional configuration needed on the infrastructure side.*

## 6.6 Local development

For local development, docker-compose[1] is used. This ensures that the environment is as close to production as possible, apart from the network topology. The applications are run in development mode, which enables features like debugging and hot-reloading. It also lets developers run the same configuration on any machine without the need to install specific software versions, further reducing new developer onboarding overhead.

[1] `https://docs.docker.com/compose`

# Chapter 7
# Conclusions

The primary outcome of this thesis is a fully functional web application for the analysis of data from drone-based thermography of photovoltaic power plants, integrating the Ngspice circuit simulator for advanced assessment of the impacts of various failures within the photovoltaic system. Similar functionality is only available in one of the commercial solutions, so its first-class integration is a unique offering.

The resulting system consists of a React.js client application and multiple backend services built with Node.js. Asynchronous processing is enabled by multiple RabbitMQ consumers. Together with an automated deployment process, the application will serve as a core platform for future development of thermography analysis software at the IMR at CIIRC, CTU in Prague.

The initial step consisted of the introduction to the problem domain and analysis of existing solutions in the form of a competitive analysis. The analysis provided valuable insights into the existing market and allowed for the identification of common features and potential improvements.

A specification for the software was then composed, aided by the insights gained by the competitive analysis. The specification required significant cooperation with domain experts. Which, in turn, required a clear definition and mutual understanding of the business goals and added value of the built software. Based on the specification, a clear definition of software requirements and the expected capabilities of the final application were composed.

The composed software requirements were then used to guide the software design process and ensure that all critical aspects of the application were considered and addressed. The design phase involved creating architectural plans for the software and selecting appropriate technologies for each software component. The final architecture enables a scalable solution to the presented problem.

The implementation phase consisted of writing each software component based on the application architecture plan. This phase consisted primarily of coding the application, integration of the necessary libraries and frameworks, and testing to ensure proper function and performance. User documentation was created to simplify the onboarding of new customers.

With the working software, it was then possible to specify the platform requirements and design the automated deployment process. This primarily included the selection of the platform itself based on the available resources and the writing of the deployment scripts to support the automated deployment process. After the software deployment was stable, it was possible to upload data and execute a simulation on the production server. Access was given to testing users to explore the new software and find potential issues.

The resulting software is, therefore, the product of extensive research and competitive analysis in the target domain of PvPs, cooperation and collaboration with domain specialists, and the combination of the gained knowledge with software engineering expertise to build a product that addresses the specific needs and challenges of the industry.

This thesis, along with the analysis and software developed, serves as a foundation for further advancements in monitoring and predictive maintenance in the PV industry at the IMR, at CIIRC, CTU in Prague.

While further development of the software is needed to provide a fully featured and commercially competitive platform, the developed software serves as a feature-rich, scalable, and extensible core for future development. A few of the most notable opportunities for future development are the following:

- Among the primary considerations to further enhance the usability of the platform, is the development of a mobile user interface. The current version primarily targets desktop users, but the potential for enabling field usage of the application could unlock further use cases of the application.
- Industry standardization and compliance play a critical role for the enterprise usability of the application. One of the fields of focus can be ensuring that all the output data are compliant with industry certifications.
- The enhancement of the electric schema editing functionalities on top of the map layer can pose a significant improvement in the researcher's workflow while saving hours of time for each power plant.

These features were, however, out of the scope of this thesis and further demonstrate the broad range of opportunities that lie in the successful combination of software engineering expertise with related domain-specific knowledge to develop software with high added value for the end users while enabling novel approaches in established industries.

# References


[1] Scopito. *Solar panel inspection software*. https://scopito.com/solar-pv-inspection-software. Accessed: 2024-05-20.

[2] Raptormaps. *Raptor Solar platform screenshots*. https://raptormaps.com/products/solar-aerial-inspections. Accessed: 2024-05-20.

[3] MapperX Global. *MapperX: Solar PV panel inspection software*. https://mapperx.com/en/main-page. Accessed: 2024-05-20.

[4] Sitemark. *Software for solar operation*. https://www.sitemark.com/solutions/solar-operation. Accessed: 2024-05-20.

[5] TheDroneLife. *Above Surveying Solar Reporting*. https://thedronelifenj.com/drone-solar-inspection-software/. Accessed: 2024-05-20.

[6] Carlo Del Don. *Screenshot of Thermography desktop application*. 2017. https://github.com/cdeldon/thermography. Accessed: 2024-05-20.

[7] Starfos. *Implementation of diagnostic and predictive maintenance for efficient control of photovoltaic powerplants using autonomous vehicles*. https://starfos.tacr.cz/en/projekty/TK03020144#project-main. Accessed: 2024-05-20.

[8] Ing. Ladislava Černá, Ph.D. *LDFS Report*.

[9] Department of Electrotechnology, FEE, CTU in Prague. *Implementation of diagnostics and predictive maintenance for efficient control of photovoltaic power plants by autonomous means*. https://technology.fel.cvut.cz/diprefe/. Accessed: 2024-05-20.

[10] Intelligent, and Mobile Robotics Group. *Autonomní vzdušný prostředek pro inspekci fotovoltaických elektráren*. Report number: TK03020144_V1. Czech Institute of Informatics, Robotics and Cybernetics. Available at: https://imr.ciirc.cvut.cz/uploads/Research/DiPreFE_popis_vysledku_V1.pdf, Accessed: 2024-05-20.

[11] NGspice Project. *NGspice: Open-source Spice Circuit Simulator*. https://ngspice.sourceforge.io/. Accessed: 2024-05-20.

[12] TheDroneLife. *The Best Drone Solar Inspection Software for 2024*. https://thedronelifenj.com/drone-solar-inspection-software/. Accessed: 2024-05-20.

[13] Sitemark. *Anomaly Loss Estimation — Sitemark Help Center*. https://support.sitemark.com/en/articles/5664646-anomaly-loss-estimation. Accessed: 2024-05-20.

[14] Mohammadreza Aghaei. *Unmanned Aerial Vehicles in Photovoltaic Systems Monitoring Applications*. In: 2014.

[15] Václav Beránek, Tomáš Olšan, Martin Libra, Vladislav Poulek, Jan Sedláček, Minh-Quan Dang, and Igor I. Tyukhov. New Monitoring System for Photovoltaic Power Plants' Management. *Energies*. 2018, 11 (10), DOI 10.3390/en11102495.

[16] Nallapaneni Manoj Kumar, K. Sudhakar, M. Samykano, and V. Jayaseelan. *On the technologies empowering drones for intelligent monitoring of solar photovoltaic power plants*. 2018.
`https://www.sciencedirect.com/science/article/pii/S1877050918310366`. International Conference on Robotics and Smart Manufacturing (RoSMa2018).

[17] Maximilian Lowin, Domenic Kellner, Tobias Kohl, and Cristina Mihale-Wilson. *From Physical to Virtual: Leveraging Drone Imagery to Automate Photovoltaic System Maintenance*. INFORMATIK 2021. 2021.

[18] Kuo-Chien Liao, Hom-Yu Wu, and Hung-Ta Wen. Using Drones for Thermal Imaging Photography and Building 3D Images to Analyze the Defects of Solar Modules. *Inventions*. 2022, 7 (3), DOI 10.3390/inventions7030067.

[19] David Hernández-López, Esteban Ruíz de Ona, Miguel A. Moreno, and Diego González-Aguilera. SunMap: Towards Unattended Maintenance of Photovoltaic Plants Using Drone Photogrammetry. *Drones*. 2023, DOI 10.3390/drones7020129.

[20] Giovanni Tanda, and Mauro Migliazzi. Infrared thermography monitoring of solar photovoltaic systems: A comparison between UAV and aircraft remote sensing platforms. *Thermal Science and Engineering Progress*. 2024, 48 102379. DOI 10.1016/j.tsep.2023.102379.

[21] Alibaba Cloud. *Message Brokers Revealed: Kafka vs. RabbitMQ vs. ActiveMQ*.
`https://www.alibabacloud.com/tech-news/a/message_queue/gugz0vw2oh-message-brokers-revealed-kafka-vs-rabbitmq-vs-activemq`. Accessed: 2024-05-20.

[22] Gleison Brito, Ricardo Terra, and Marco Túlio Valente. Monorepos: A Multivocal Literature Review. *ArXiv*. 2018, abs/1810.09477

[23] Stack Overflow. *Stack Overflow Developer Survey 2021*.
`https://survey.stackoverflow.co/2021#section-most-loved-dreaded-and-wanted-programming-scripting-and-markup-languages`. Accessed: 2024-05-20.

[24] Stack Overflow. *Stack Overflow Developer Survey 2021*.
`https://survey.stackoverflow.co/2021#section-most-popular-technologies-web-frameworks`. Accessed: 2024-05-20.

[25] MozDevNet. *WebAssembly.memory() constructor - webassembly: MDN*.
`https://developer.mozilla.org/en-US/docs/WebAssembly/JavaScript_interface/Memory/Memory`. Accessed: 2024-05-20.

[26] Marc Hassenzahl. *Experience Design: Technology for All the Right Reasons*. 2010.

[27] Gunnar (Nick) Bluth. *An overview of PostgreSQL's backup, archiving and replication*. FOSDEM VZW. FOSDEM 2017. 2018.
`https://doi.org/10.5446/41887`.

# Appendix A
# Abbreviations

| | |
|---|---|
| API | Application programming interface |
| B2B | Business to business |
| CIIRC | Czech Institute of Informatics, Robotics and Cybernetics |
| CTU | Czech Technical University |
| GIS | Geographic Information System |
| GPS | Global positioning system |
| IMR | Intelligent and Mobile Robotics Group |
| IR | Infrared |
| PP | Power plant |
| PV | Photovoltaic |
| PvP | Photovoltaic power plant |
| REST | Representational State Transfer |
| SEO | Search Engine Optimization |
| UAV | Unmanned aerial vehicle |
| UX | User experience |
| WASM | WebAssembly |

# Appendix B
## Ngspice example schema - module

```
Module 3031
Vpv 2 0
* PowerLine 1
XModul3031 1 0 x20_3x20_3x20_3
Rload 2 1 0.01

* Substring ps_190m_24f_ss
.subckt ps_190m_24f_ss Isc Vpv+ Vpv-
Dpv Isc Vpv- Dcell
Rp Isc Vpv- {Rp}
Rs 1 Isc {Rspv}
Epv Vpv+ Vpv- 1 Vpv- {ndvac}
Fcell1 Vpv- 1 Epv 1
.MODEL Dcell D (Isr={Isrec*area} Is={Isdif*area} cjo={Cjorec/ndvac}
Rs={Rsrec*area*ndvac*3} XTi={XTi} BV={BV} Ibv={Ibv} VJ={VJ*ndvac}
M={Mrec} tt={ttrec} FC={FCrec} N={Nrec} Nr={Ndif})
.param area=154.63
.param Rsrec=0.00014e-6
.param XTi=5
.param BV=6*ndvac
.param Ibv=1e-5
.param VJ=1
.param Mrec=0.3
.param ttrec=50n
.param Cjorec=100p
.param FCrec=0.5
.param Nrec=2
.param Isrec=1.6n
.param Rspv=0.000009
.param Ndif=1.43
.param Rp=500
.param Isdif=0.001n
.param ndvac=20
.ends ps_190m_24f_ss

* Module x20_3x20_3x20_3
.subckt x20_3x20_3x20_3 PV_out+ PV_out-
D1 2 PV_out+ MBRB2545CT
Isc1 2 3 3
XSS1 3 PV_out+ 2 ps_190m_24f_ss
D2 4 2 MBRB2545CT
Isc2 4 5 3
XSS2 5 2 4 ps_190m_24f_ss
```

```
D3 PV_out- 4 MBRB2545CT
Isc3 PV_out- 7 3
XSS3 7 4 PV_out- ps_190m_24f_ss
.model MBRB2545CT D( Trs1=0.002824 Tnom=25 Is=19.4n Rs=0.01 N=1 Eg=1.1
Xti=3 Cjo=2.05n Vj=0.4 M=0.41 mfg=GI type=Schottky)
.ends x20_3x20_3x20_3
.temp 25
.control
dc Vpv 0 46.2 1
print v(0) I(vpv) v(2)*I(vpv)
quit 0
.endc
.end
```