Strukton Project Design Report

University of Twente

J.H. Bannink s2861887 W.B. Koning s2149869 M. Angelov s2749831
B.W. Veen s2253445 J.B.I. Stolk s3133346 O. Adams s2932377

November 2025

Contents

1 Context

In the penultimate module of the TCS bachelor, students are expected to do a "Design Project". During this project, students are tested on their ability to design and develop a complete application from front to back, with a real client, which in our case was for Strukton.

Strukton is a service provider dealing in maintenance and repairs of rail and civil infrastructure in multiple countries. For our project, we worked with Strukton Systems in the Netherlands to create a system that helps them with their maintenance and repairs of train tracks in Sweden, specifically for maintenance on railroad switches. Of all the causes of track downtime, the switch failure was one of the most common ones. Whenever these failures occur, Strukton sends in a maintenance crew which fixes the problem and writes a report. This report contains the issue, the cause of the issue, and information on the maintenance performed in a logbook. To further minimize track downtime, Strukton attempts to analyze measurements they take and fix switches before they break.

Additionally, when we refer to a switch's power curve as nominal in the rest of this report, we do not mean that it is perfect and performs like a switch out of the box from the factory. This is very difficult because, as these switches are used and aging, they deteriorate in ways that cannot be prevented or repaired. Strukton accepts this and only tries to get the switches to work as well as they possibly can given their current state and age.

2 Stakeholders

This project involves a variety of stakeholders, each with their own interests in our product. Some stakeholders are directly involved with our development, providing input and information, while others are primarily affected by our final result. In the following subsections, we describe each of the major stakeholders and how they are impacted by the project.

2.1 Client

Strukton is an infrastructure company that specializes in rail and civil systems. They are the client of our project and work with us to develop the genAI system.

We have had contact with three employees, of which two data analysts and one switch engineer. They have given us valuable insights into the inner workings of the switches, as well as the potential reasons for the maintenance done on them. Furthermore, the analysts have provided us with data that can be used to test and develop the product.

2.2 Maintenance Crews

The input of this project relies on the maintenance crews. These employees write the logs that we use to analyze what has been done to the switch, and the changes they made to the switch influence the measurements as well.

It is important to note that these crews are not employees of Strukton. This means that Strukton does not have any influence on how these crews perform maintenance and how they write maintenance logs. That also leads us to the reason for this project; since Strukton has no control over how the maintenance logs are written, they cannot make the crews log their maintenance in a way that allows Strukton to easily automate their analysis. This is what motivated this project.

2.3 Secondary Stakeholders

Since passengers will not directly use our product, they are secondary stakeholders. They are still affected by the reliability of the railroad system, and thus indirectly by our project. By helping Strukton perform faster and more effective maintenance, our system indirectly benefits passengers through increased train punctuality and decreased cancelations.

3 Problem Statement

To prevent switch failure that could cause delays or train cancelations, Strukton measures the power required to activate switches. These measurements are then used to evaluate the proper functioning of a switch and predict required maintenance. When a failure occurs or maintenance needs to be performed, a maintenance crew will work on the issue and report their work in a logbook. Some of the information that is written consists of recurring values that can easily be interpreted and used by data analysts. However, the most valuable part of the logs is the free text description with specifics of what the crew has done. These logs contain a lot of useful but unstructured information about the operations performed on the switch and its state. Without the use of artificial intelligence, interpreting this information and linking it to observable information in sensor data is an exhausting task. This is why Strukton has tasked us with developing an application that summarizes the work of the maintenance crew and reflects on the sensor measurements before and after the maintenance instance to analyze effectiveness.

4 Proposal

4.1 Initial Proposal

Initially, we planned to develop a generative AI application that uses the railway switch measurements and the maintenance logs from the engineers. It would use this information to create a summary, in English, of what the maintenance crew has done and shows how this relates to the power curve measurements before and after maintenance. To generate this summary, it would take a work order and translate the maintenance crew's description into English, and use other columns that we deem appropriate, together with measurements surrounding the maintenance. This summary would then be shown in a new column in the log file.

We also decided that it would be convenient to have a simple GUI as a user interface, as we consider it to be more suitable and fitting for this application than a CLI tool. A GUI would simplify the workflow of using the application and make it easier to show images and such, if necessary.

We expected that the end product would be able to summarize the maintenance crew work logs and interpret sensor data so that it could make a judgment about the effectiveness of the maintenance. This would be formatted as a summary by itself. The plan was to use generative AI. This ended up being more difficult than expected because we did not have any data to train and test a deep learning model on. In addition, the work logs themselves were full of abbreviations and terms that not even the data analysts would know sometimes.

4.2 Second Proposal

After meeting with our supervisor and discussing the possibilities of training an AI model, we came to the conclusion that we did not have the appropriate training data. We also did not have the time or knowledge necessary about the railway switches to synthesize these training data. This is why we decided to talk to our client to reconsider the goals of the project and together we shifted the objective towards a new proposal.

The new proposal for the project is an application with a GUI, in which the user can select a maintenance instance. The application will provide the user with a summary of the maintenance logs generated by AI, an option to select and compare the power curves of the switch before and after maintenance, power curve metrics to be used for easier comparisons, and a field in which an analysis report can be written. The objective of the proposed system is to provide the data analysts with all relevant data to evaluate the effectiveness of the maintenance instance. In this way, analysts can create the appropriate data. This data could then later be used to finetune a genAI model which can perform the tasks which were originally proposed for the project.

5 Requirements

5.1 Functional requirements

The functional requirements describe the specific capabilities that the system must provide. Table 1 lists the functional requirements for the application, of which we met all but one.

FR10 was part of our initial proposal, and what we found to be unachievable given our switch knowledge and the nature of the provided data. Instead, we created an algorithm that extracts metrics from all of the selected power curves, such as standard deviation and mean, and displays them all in a table for comparison.

ID	Requirement	Description
FR1	User Authentication	The system shall allow users to log in securely using a unique API key.
FR2	Data importation	The system shall allow users to retrieve work log entries, measurement data and analysis from the database through the GUI.
FR3	Summarization	The system shall use an AI model to generate translated work log summaries.
FR4	Report Writing	The system shall provide a text field in which an analysis report can be written.
FR5	Report Editing	The system shall enable users to modify and update analysis reports.
FR6	Graph Visualization	The system shall display interactive graphs of power curves, allowing users to select and view the curves for specific dates and times.
FR7	Database Storage	The system shall store all user data, summaries, graphs and analysis reports in a database.
FR8	Client–server Communication	The system shall communicate between the client and server via well-defined API endpoints.
FR9	Error Handling	The system shall display error messages and handle invalid input gracefully.
FR10	AI Graph Analysis	The system shall use an AI model to analyze the measurements and predict the cause of the problem and how to fix it.
FR11	Easy Installation Process	The client system will be easily installable using an install wizard.

Table 1: Functional requirements of the system.

5.2 Non-functional requirements

The non-functional requirements define quality attributes and constraints. Table 2 summarizes them, with all requirements being met with the exception of one that is difficult for us to assess.

NFR3 is highly hardware dependent, and on our end summaries took longer to generate as we ran the LLM on laptops with weak GPUs. If Strukton runs our backend on a server with a powerful GPU, the model summarization time should be nearly instant, but we have no way of testing that.

ID	Requirement	Description
NFR1	Database Response Time	Database queries related to logs and reports shall return results in under 2 second for typical workloads.
NFR2	Graph rendering speed	The system shall render and display power curve graphs within 2 seconds after selection
NFR3	Model Summarization Time	The system shall generate a work log summary with genAI in under 5 seconds under normal load.
NFR4	GUI usability	The GUI shall be simplistic and intuitive for efficiency and ease of use.
NFR5	Reliability	The system should handle errors gracefully, letting the user know if anything goes wrong.
NFR6	Scalability	The architecture shall allow for future integration of different or additional models and graph functionalities.
NFR7	Security	All user credentials and data shall be transmitted securely between client and server.
NFR8	Maintainability	The system shall be designed with modular components to facilitate easy updates debugging and feature expansion.
NFR9	Availability	The system shall remain operational with minimal downtime during normal usage.

Table 2: Non-functional requirements of the system.

6 Architecture

For the overall structure of the application, we decided that the best way to set up the system is with a server and a local client for users.

6.1 Frontend

The frontend is a client GUI made with PySide6, the official Python module of the Qt 6.0 framework. In the client, users can search for a maintenance instance using various filters. Then, users can select related measurements and link them to submit an analysis. All data is retrieved from the database and then used to populate tables. Submitting an analysis stores the written text and the linked graphs in the database again. We applied user testing to collect feedback on the ease of use of the application and used the feedback to improve the application.

On the software side, a modular approach was used. Every separate window has its own widget class, which can easily be edited. PySide forces this structure, ensuring a very maintainable and simple code base.

Apart from the PySide code, the frontend only contains a small portion of code which communicates with the server, sending simple GET requests and a POST/PUT request for creating/modifying the analysis. Figure 1 showcases the process of selecting a maintenance instance, viewing relevant power-curves and submitting an analysis to the backend.

The interaction between the windows in the front end should be as follows; at first, all fields on the main page should be empty. After the API has been configured one can select a maintenance instance. When a maintenance instance has been selected, every field should become filled in according to what has been linked to the analysis report. Some fields can still be empty after this if the analysis report is missing or nothing has been linked to it. If a genAI summary has been generated for a maintenance instance, it will always be visible after that instance has been selected. Furthermore, the graph and curve analysis will be updated each time curves have been added or removed from the graph.

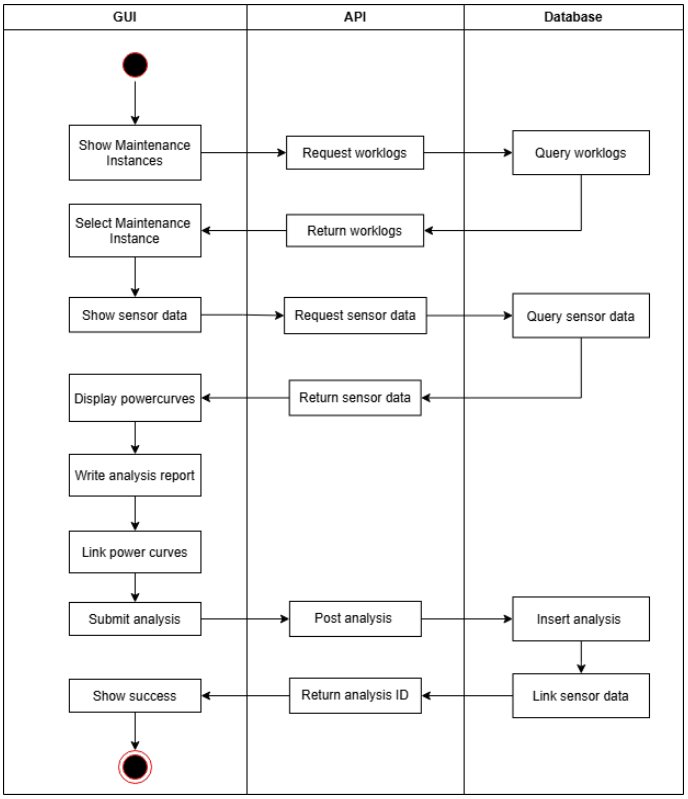

Figure 1: Activity Diagram (Selecting work-log and sensor data, linking graphs and submitting analysis.)

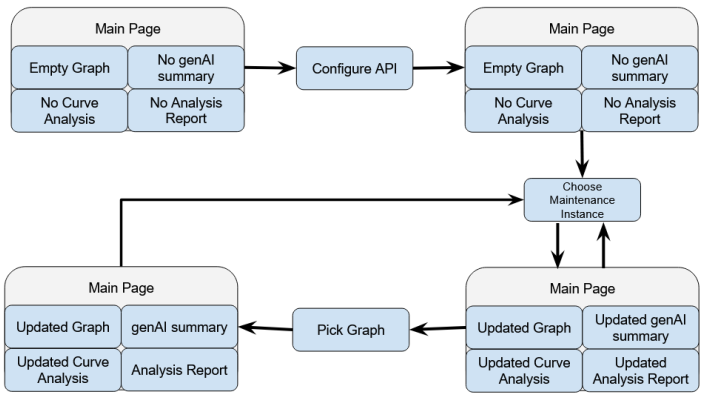

Figure 2: Screen Flow Diagram

6.1.1 API

The frontend also includes an interface for configuring the API credentials. This simple interface allows the user to input the URL of the backend API which has been set up, as well as the corresponding API key. The user can then test the connection, and only after a successful ping confirm the credentials, after which these will be stored locally on the machine. This way, the API credentials only have to be configured once, and can be changed at any time. Additionally, errors related to the GUI or API calls are logged in a locally stored file as well, allowing for easier debugging when an issue occurs in the application.

As the user is interacting with the GUI, the functionalities of the app call functions in the previously instantiated API client. This API client is a Python class that provides prebuilt methods specifically designed to interact with the API. These methods allow for easy access to the data endpoints.

6.1.2 Graph Analysis

Switch power curves contain a lot of useful information for data analysts and switch experts in Strukton, so they were vital information to include in our product. Aside from simply displaying these curves, we wanted to provide some added value for these experts by presenting them with curve metrics such as the peak, the mean of the plateau and standard deviation of different sections of switch operations.

We calculate these graph metrics and regions in the frontend for display in the GUI. The graph plateau region is identified by taking the greatest positive and negative gradients of a smoothed version of the curve using a Savitzky-Golay filter from the scipy library, which allows us to avoid detecting a small but sharp bump as one of these gradients. Afterwards, the plateau is used as a reference point for finding the other region boundaries, which are hard-coded to be a certain number of indices before or after the plateau. Using this method, we can detect the different sections of switch operation, namely the inrush peak, unlocking, moving and locking sections.

The standard deviation for all but the rise region, along with the mean of the "moving" region, is calculated using numpy. These metrics are then returned along with the indices of each region for graph plotting and to create the curve analysis table.

6.2 Backend

The backend for our system is fully written in Python, and makes use of various libraries to accommodate for the required functionalities of the application.

This backend is developed with the intention of handling all distinct aspects of the system, such as the API, database management, and LLM work-log summaries. These processes are handled by their own docker containers, with two containers for the API (strukton-api, strukton-caddy), two containers for the database management (postgres, db_seeder) and 1 container for the LLM (worker)

6.2.1 Docker

The backend was developed to run on Docker, meaning that all functionalities are separated into individual Docker containers. Docker was chosen because it provides an easy setup for the client to deploy the backend on their dedicated server. The backend consists of 5 services, PostgreSQL, API, translator, caddy and db-seed. The PostgreSQL container runs a PostgreSQL 16 model, to maintain the database. The API container runs a python:3.11-slim image, caddy runs the caddy image, and the translator runs an nvidia/cuda image, so it can handle GPU functionality with the model. The setup of the docker can be found in the .README file.

6.2.2 Database

The database is a key part of the backend as it provides the means to store data. It is a PostgreSQL database designed for scalability with a focus on making future changes easy to integrate.

It consists of various tables, all serving a different purpose. A schema for the full database can be found in Figure 3. The Work_Log table is used to store work order information for maintenance instances. This table corresponds to the schema of Strukton's current database, allowing for easy integration into their system. One of the main considerations when designing this table in particular was how to store certain variables. With the goal of ensuring a scalable and modular design that Strukton could easily expand on in the future, we made separate tables for some attributes where we deemed necessary.

The database also includes two join tables, which are used to represent many to many relations. These relations allow multiple switches to be linked to multiple work logs, providing a convenient access point for tracking which switches were involved in a maintenance instance. Additionally, there is a join table which keeps track of which power-curves were used during the analysis of a maintenance instance. This feature enables the application to display the relevant power-curves whenever a new maintenance instance is selected, and also allows users to reference these power-curves in their written analysis.

In addition, the database stores data from sensors attached to the railway switches. As there are copious amounts of measurements, the raw data are stored in bytes to save space in the database, requiring approximately half as much compared to a hexadecimal string. These measurements are uniquely identified by their measurement id and the id of their corresponding switches. This is done because the measurement id is a unique identifier per individual switch.

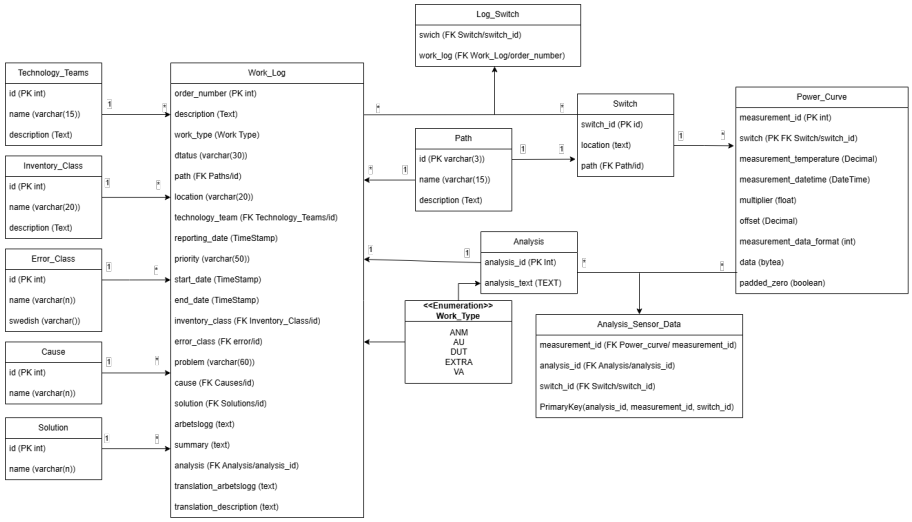

Figure 3: Database Schema

6.2.3 API

For our application, it is very important that the front-end has good communication with the backend. For this reason, an API was created that acts as a bridge between the database and the user interface. This API provides several endpoints, all serving a different purpose in allowing the application to function correctly with the appropriate data and functionalities.

The design of the API is based on REST principles, ensuring that each resource in the database corresponds to a distinct endpoint. This API is built in Python with the aiohttp library, which allows for the creation of asynchronous endpoints. This detail is very important, as the API should be able to handle multiple requests at the same time for different users. This library was also chosen for its lightweight nature and simplistic design, allowing for easy separation of concerns through the appropriate middle-ware. The API is stateless, resulting in each request being self-contained and without lasting session states being kept between calls.

The API fits very nicely into the overall system architecture; Figure 4 shows how this component interacts with the system. The API makes use of several queries to access the database and returns responses in JSON format. An example of available endpoints is listed in table 3.

As is visible from the table, some endpoints implement full CRUD, specifically the analysis endpoint. This endpoint is used to manage manually written analyses by users. It allows for posting new analyses, but also updating and deleting them. When an analysis is created, it can also be linked to measurement data, such that a later user can see which data were used for a specific analysis.

Although the API is primarily used by functions designed specifically to send a correct request, it is still prone to user errors. For this reason, error handling is very important, and the API should return the correct HTTP responses based on what went wrong with responding to a request. The aiohttp offers a very convenient solution for this, namely middle-ware. Through middle-ware, it is possible to define correct HTTP responses for each distinct error case. By putting this functionality into middle-ware,

the logic will be implemented for all endpoints, resulting in a very clean and modular API. Common errors which are uniformly handled by this middle-ware include invalid JSON errors, database issues, bad parameters, and a catch-all for internal server errors, all including an appropriate HTTP error code.

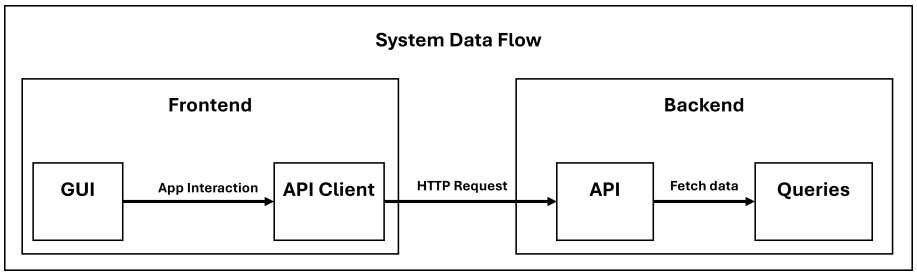

Figure 4: System Dataflow

Endpoint	HTTP Method	Description
/Analysis	POST	Post an analysis for a maintenance instance, allows for linking power-curves to the analysis based on power-curve ID and switch ID.
/Analysis	GET	Retrieve a maintenance log analysis based on analysis ID. Also returns a list of linked power-curve ID's with their respective switch ID.
/Analysis	PUT	Update a maintenance log analysis with new text and possibly new power-curves.
/Analysis	DELETE	Delete a maintenance log analysis. This endpoint is mostly used for testing purposes.

Table 3: Analysis API Endpoints

6.3 Work Log Summary by LLM

It was decided to make use of a local LLM instead of API calls to a third party service, as the client does not want their sensitive data to be collected by these third parties. The application is designed such that the genAI language model runs on the backend. We chose this so that the model does not have to run locally for each user, which would have made the speed vary a lot based on the user's computer hardware. If the client wanted to scale to a better model later, they would have needed to upgrade the GPUs of all local machinery. This is not scalable, and therefore, the choice was made to run the backend on a dedicated server. If the client wants to upgrade to a bigger language model or increase the speed of the model, they just have to acquire a better GPU for the server.

The system is implemented such that the docker service, which is called translator, starts a worker.py file. This file periodically pulls ten work logs from the database, and starts translating and summarizing them, after which it saves these translations and summaries back into the database. The system uses the google gemma-2-2b-it model for generating the work log summaries. This model was chosen because, in addition to being free to use, it also has a low parameter count, which is suitable for the use case and hardware used in this project. Furthermore, we chose to use an instruction tuned model since it was already fine tuned to handle natural language instructions, which is precisely what we need for summarizing work logs.

For translation, we use the Helsinki-NLP, opus-mt-sv-en machine translation model. We chose this because it already had integration with spaCy, which we were using for Gemma, and it has a relatively small translator model that can run on our local machines for testing. The model is also specifically designed for translating Swedish into English. Since the logs are fully written in Swedish, correctly translating is a vital step for our model. Although gemma-2-2b-it can understand Swedish a bit, the

work logs are quite domain specific, and after a few manual tests it was concluded that the response from gemma is better if the work log is translated using a specialized NLP. Most columns in the work log were translated manually. These columns can only have a set of certain values. These values were manually translated using multiple translation tools to find the complete translation and then written as a set of values. This happened for all instances except the larger free-text fields Beskrivning and Arbetslogg, those were translated by the Helsinki-NLP.

7 Process

Creating a genAI application is a complex and iterative process. At the same time, we had difficulty communicating with our client during the first few weeks due to vacation. During our process of developing the application for our client, we changed the scope of the project slightly which also impacted the design of the project. The following subsections will show what the original design for the application was and how we changed it into our final product. There will also be an in-depth description of how the GUI was designed.

7.1 Initial Design

After the first meeting with the client, we started by becoming familiar with possibly relevant tools that we expected to need for this project. After this, we came up with a data pipeline, to show how we wanted to give shape to our application, and help us with developing. As shown Figure 5, our preliminary design heavily relies on the A.I. model being able to reason using comprehensive understanding of train switches and how the measurements are done, in order to give useful and comprehensible output.

After experimenting with several local pre-trained models, we found that it would be quite difficult to fine-tune a model that would give insightful output. This raised the question whether the usefulness of the output could exceed the usefulness of compiling the input used for the model into a confined and consistent summary without the use of generative AI. After discussing this problem with our supervisor, we decided to discuss this problem with the client.

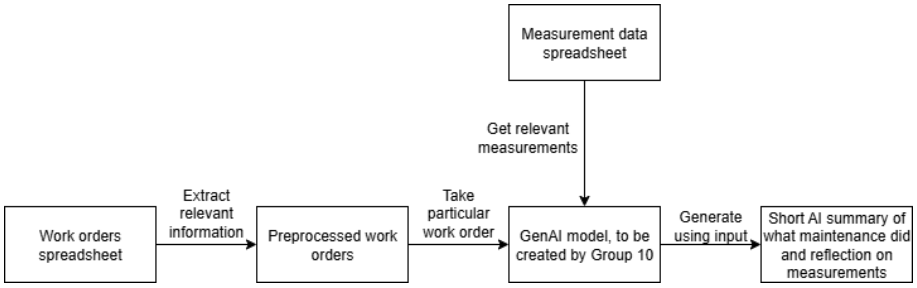

Figure 5: The preliminary design's data pipeline

7.2 Second Design

For the second design, we decided to take the main focus away from the generative AI, which would add a column to the work order spreadsheet, and instead create a GUI which allows for some more functionality.

The new design allowed us to better distribute the tasks, as there was a more concrete goal for the application. We now knew that we had to create a server with a database, a GUI, and a smaller generative AI model that would translate and summarize the work logs. This improved distribution of tasks helped us make substantially more progress in the same time we worked on the initial design.

One of the main design choices we have had to make was choosing a library to create a desktop application. For the first design, we chose NiceGUI. This was mainly because it was easier to set up and the GUI was less important initially. However, for the second design, the GUI became a much more important part of the project, so we had to reconsider whether NiceGUI was the right choice. After some research, the choice was between NiceGUI and PySide6. Where NiceGUI is a more web app oriented library, PySide is dedicated to creating native applications, being the official Python module for access to the Qt 6.0+ framework. We ended up choosing PySide6, as it provides more functionality and allows for creating a more professional looking interface.

7.3 Graph Analysis Design

In a meeting with our client, they mentioned that it's usually best to compare the power curves visually, by looking into the areas that bumps formed in or that previous bumps got flattened down. Due to this, we initially had the idea to use a Visual Language Model (VLM) which would make this comparison for us by being shown the curves along with a prompt asking it to explain what it sees in specific sections of the curve. However, this approach had two problems. Since our team did not have any computers with powerful hardware, the only model we could realistically run and test was paligemma-2-3b. This model only accepted 224x224 input images making them of rather low quality and thus the comparisons were lacking any substantial information. Furthermore, even while being one of the smallest VLM models it still took over 2 minutes for us to receive a comparison between two curves, and while this could have been better with a more powerful GPU, we had no way to test exactly how long it would take. Another issue in our initial approach arose after the client asked to have more than 2 curves displayed on the graph at the same time. Doing so would make it even more difficult to use a VLM for the comparison, as it would have a hard time comparing a large number curves properly and fully. Seeing all these limitations, we decided it was time to explore a different way to compare curves.

The second, and final, approach relied on calculating curve metrics and displaying them to help the experts at Strukton do the comparisons themselves. As the specific workings of this algorithm are discussed in Section 6.1.2, we won't be going into detail on how it works here. Essentially, this new approach calculated the standard deviation at 3 sections in the curve, the unlocking, movement and locking sections, along with the mean of the movement section. We displayed these metrics, along with other switch data like the date and time of the power reading, as well as the temperature of the switch at that time and the direction in which it was moving. This data is displayed in a table to the right of the graph, and allows for an easy comparison between the different curves.

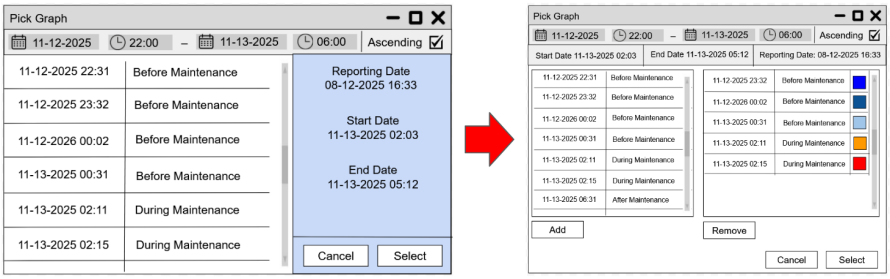

Figure 6: Graph selection design before and after

7.4 GUI Design

The process of creating the GUI had three important steps. Step 1, a design would be made based on the requirements of the project. In this phase, not only functionality was taken into account, but also the important information for making decisions on each screen was to be taken into account. Questions like "Would seeing the start and end date of a maintenance instance be important information when deciding which curves to pick" were brought up and answered during this phase. In step 2, the design would be reviewed with the whole group and afterwards any missing or unclear features would be added or tweaked. Sometimes, a design would be completely scrapped during this phase, and the process would be moved back to step 1. After all tweaks were added, it would be reviewed again. Step 3, when a design was deemed finished, we would start implementing it. Because of this process, a lot of the design stayed the same during this phase. However, small oversights or improvements that became apparent during the implementation phase could cause us to go back to the design and tweak it slightly, similar to step 2. Other times, when the tweak wouldn't change the overall layout, tweaking the design would be skipped. In Figure 7 the final designs of the user interface can be seen.

A singular time during this project, a change in requirements caused us to go back to step 1 after already having implemented the first design. This design of the product allowed for 2 curves to be selected at most, with the curves being selected one at a time. In a meeting with the client, they asked us to allow more than two curves to be on the graph, which meant that the graph selection screen would have to be reworked almost entirely. The layout was changed to cleanly show two tables, one with selectable curves and the other with already selected curves. To combat the messiness that having many

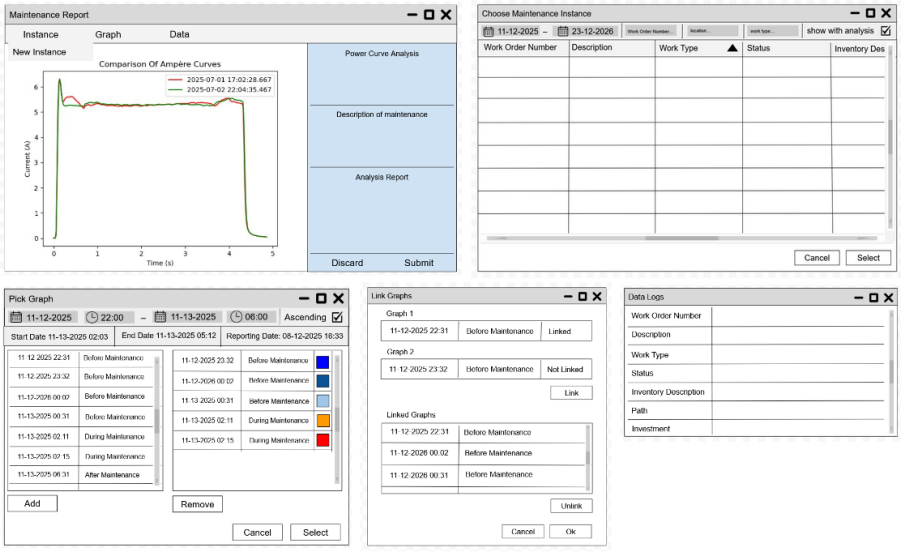

Figure 7: GUI design

curves on a single graph would cause, a feature that the color of the curve would depend on whether a curve happened before or after maintenance was proposed. In Figure 6 the two designs can be seen and the changes between the two can be observed.

8 Testing

8.1 Backend testing

To test the backend, a second set of docker containers was created that duplicates our system. It uses the same database schema to create a database container, the same files to configure the API container, and includes a pytest container to run all test scripts. It is deployed in a separate docker setup so that system functionalities can be tested in a clean and disposable environment, without cross-contaminating or truncating data from tables in the real database. The pytest container runs conftest.py to set up the test configuration and then runs all the 72 tests. A coverage report showing which part of the backend is covered by tests is included in Figure 8.

8.1.1 Database testing

When the test docker is started the database is created using our database schema as shown in Figure 3. The test_fill_db.py includes tests that validate the schema of the database by confirming that the necessary tables exist. Then, using a synthesized data file that is representative of real work log data, we test all functions of fill_DB.py and assert that the correct data are present. The emptying of tables is also tested. As shown in the coverage report Figure 8, a small part of the fill_DB.py is not covered. This is where the data extraction from files and database connection setup are created.

8.1.2 API Testing

To test the API, tests were created that ensure that all endpoints of our REST API as described in section 6.2.3 meet functional and reliability requirements. Before pytests runs the test_api.py the database is seeded with data, then the endpoints are tested using GET, POST, and DELETE requests. Assertions for the desired result tests include requests with a bad payload to check that the correct error status codes are given.

8.2 User Interface Testing

Due to the nature of this product being meant for data analysts, letting regular people test the user interface would not be representative of a realistic use-case scenario. As a large amount of user interface

File ▲	statements	missing	excluded	coverage
src/database/__init__.py	1	0	0	100%
src/database/fill_DB.py	232	28	0	88%
test/__init__.py	0	0	0	100%
test/conftest.py	65	0	0	100%
test/test_api.py	175	0	0	100%
test/test_fill_db.py	156	0	0	100%
Total	**629**	**28**	**0**	**96%**

coverage.py v7.11.0, created at 2025-11-06 20:10 +0000

Figure 8: Coverage report from pytest

testing is about finding inconsistencies, the testee should have a certain level of knowledge on the subject. Therefore, the test was carried out in person with a testee who has knowledge on this subject, a person who gives tasks to the testee, and a person who makes notes. The test showed that certain features were unclear or not intuitive. In the maintenance selection screen, the next button was overshadowing the select button, it was right atop the select button and bigger. To combat this, we moved the button to the center and made it smaller. Furthermore, the tester tried to add a curve to the graph by double clicking on the table. This was later added as a feature. For the most part, the test proved that our design was already decently intuitive, and the features added or changed because of the test will only enhance the clarity of the app.

9 Reflection

The first point that comes to mind when reflecting on our project is the communication with the client and finding a supervisor at the start of the project. The contact persons from the client were on vacation for the first couple weeks and we were unable to find a supervisor, despite asking numerous researchers. Therefore, the start of our project was rather slow and it was quite late that we discovered that training a comprehensive genAI model as described our initial proposal would be too much for a project with the given amount of time, data and knowledge of train switches. After finding this out, we had to meet with the client to discuss the options of what is actually feasible within the given time with our knowledge.

While we could not proceed with the initial design we had in mind due to the lack of data, our product enables Strukton to start creating and collecting this data. By writing an analysis for each maintenance instance and linking graphs to it, structured data is formed that could be used to train the envisioned AI model, perhaps as a future university project.

There was a feature that we did not manage to implement, being able to show a second type of graph, next to the one that we centered our current design around. As our client also asked for this shortly before the deadline of the project, we were sadly not able to implement this and told them that during the meeting. We hope that it would not be too difficult for Strukton to add this themselves, since our code is well documented.

10 Conclusion

In this project for Strukton, our goal was to create an application that translates and summarizes work logs in spreadsheets. It would also have to reflect on the changes in the power curve before and after maintenance. However, during the development of the project, we realized that this was not a realistic goal, so we decided to talk to our client about this. After this meeting with the client, we shifted our aim towards a less AI centered application, but rather an application where the user has all relevant information at his disposal and can write an analysis for a maintenance instance and link related graphs.

Using this application, Strukton can now easily write and save an analysis, and it is possible to use

these analyzes in the future to train AI to fulfill the original goal of the project. The product helps to make sense of many unstructured data and allows analysts to critically reflect on maintenance instances with easy access to all relevant data.

Overall, this project was very educational, in which the group members learned different things based on what they worked on. Many challenges were overcome throughout the project, such as making use of new tools and frameworks, as well as UI and UX design. It gave us valuable experience in working together with a client, and figuring out how to satisfy their requests to the best of our abilities, while also keeping the project in line with what could realistically be achieved.