YRKESHÖGSKOLAN
NOVIA

Developing an Analysis Add-On

Tool for analysing different parameters from maritime electrical components

Atte Vertanen

Degree Thesis

Thesis for a Bachelor's (UAS) - degree

Degree Programme in Electrical Engineering and Automation

Vaasa 2025

DEGREE THESIS

Author: Atte Vertanen
Degree Programme and place of study: Electrical Engineering and Automation, Vaasa
Specialisation: Information Technology
Supervisors: Jan Berglund, Novia University of Applied Sciences
Pasi Juppo, WE Tech Solutions

Title: Developing an Analysis Add-On

Date: 27.5.2025 Number of pages: 30

Abstract

This bachelor's degree thesis is about developing an analysis tool for WE Tech Solutions as an add-on to their sandbox platform. The sandbox platform contains data from various solutions and is a building block for new add-ons or functions.

The purpose of the work is to be able to analyse various electrical components from their parameter database using the add-on, thus making the tuning of the component parameters easier. The add-on is user-friendly and requires no to minimal coding knowledge to use.

This thesis includes a theoretical part and a practical part. The theoretical part explains the different techniques and programming languages used to develop the add-on. The practical part of the thesis explains the programming and thought behind the logic and functions of the add-on.

This thesis resulted in an analysis add-on that is able to extract data from parameter databases and use them in user-inputted analysis. Providing engineers with useful information about the operational electrical components.

Language: English
Key Words: python, data handling, development

EXAMENSARBETE

Författare: Atte Vertanen
Utbildning och ort: El- och automationsteknik, Vasa
Inriktning: Informationsteknik
Handledare: Jan Berglund, Yrkeshögskolan Novia
Pasi Juppo, WE Tech Solutions

Titel: Utveckling av ett analyseringstillägg

Datum: 27.5.2025 Sidantal: 30

Abstrakt

Detta examensarbete handlar om att utveckla ett analyseringsverktyg för WE Tech Solutions som ett tillägg till deras Sandbox plattform. Sandbox plattformen innehåller data om olika lösningar och fungerar som en byggsten för olika verktyg och funktioner.

Syftet med arbetet var att kunna analysera olika elektriska komponenter utifrån deras parameterdatabaser med hjälp av analyseringsverktyget. Verktyget är användarvänligt och kräver ingen eller minimal kodkunskap för att använda.

Examensarbetet består av en teoretisk och en praktisk del. Den teoretiska delen beskriver de olika teknikerna och programmeringsspråken som använts för att utveckla verktyget. Den praktiska delen beskriver programmeringen av verktyget samt tankegångarna bakom logiken och olika funktionerna i verktyget.

Ett analysverktyg utvecklades för att kunna extrahera data från parameterdatabaserna och använda dem i analyser baserat på användarinmatning. Detta ger ingenjörer värdefull information om de elektriska komponenterna i plattformen.

Språk: engelska
Nyckelord: python, datahantering, programmering

OPINNÄYTETYÖ

Tekijä: Atte Vertanen
Koulutus ja paikkakunta: Sähkö ja automaatioteknikka, Vaasa
Suuntautumisvaihtoehto: Tietoteknikka
Ohjaajat: Jan Berglund, Novia Ammattikorkeakoulu
Pasi Juppo, WE Tech Solutions

Nimike: Analysointityökalun kehittäminen

Päivämäärä: 27.5.2025 Sivumäärä: 30

Tiivistelmä

Tämä opinnäytetyö käsittelee analysointityökalun kehittämistä WE Tech Solutionsille lisäosana heidän sandbox-alustalle. Sandbox-alusta sisältää tietoa eri ratkaisuista ja toimii rakennustyökaluna erilaisille toiminnoille.

Työn tarkoituksena on mahdollistaa erilaisten sähkökomponenttien analysointi niiden parametritietokantojen perusteella analyysityökalun avulla. Työkalu on käyttäjäystävällinen ja sen käyttö ei vaadi lainkaan tai vain vähän ohjelmointiosaamista.

Opinnäytetyö koostuu teoreettisesta ja käytännöllisestä osasta. Teoreettinen osa kuvaa eri tekniikoita ja ohjelmointikieliä, joita käytettiin työkalun kehittämisessä. Käytännöllinen osa kuvaa työkalun ohjelmointia sekä logiikan ja erilaisten toimintojen ajatusprosessia.

Tässä opinnäytetyössä kehitittiin analysointityökalu, jolla pystyy hakemaan tietoa tietokannoista ja käyttämään sitä käyttäjän syöttämiin analyyseihin. Analysointityökalu tarjoaa insinööreille arvokasta tietoa sähkökomponenttien toiminnasta.

Kieli: englanti
Avainsanat: python, tiedonkäsittely, ohjelmointi

Table of Contents

Table of Code Examples

Table of Figures

1 Introduction

This thesis was initiated by WE Tech Solutions with a goal of developing an analysis tool for their installations on a newly developed sandbox platform. The sandbox platform is designed specifically for engineers at WE Tech and allows for seamless integration and testing of new functions without any risk to the production environment.

This thesis delves into what is possible to do with the data from the sandbox platform. With the goal in mind to perform analyses of WE Tech Solutions' installations in maritime vessels. Traditionally, information about different installations has been featured exclusively in their application WE Line as files attached to installations. For instance, if an engineer needs to analyse the voltage frequency between the grid and the vessel, they must first determine the file type, identify the appropriate software for opening it, and then manually examine various values. The analysis tool is going to expedite this by automating the process and boosting productivity.

To simplify, adding different functions to the platform, a dedicated sandbox for WE Line has been developed by another company called Atea Finland Oy Ab. This sandbox enables the configuration of different add-ons that can be set up for different functions in a separate and controlled environment. This ensures that critical data can be examined efficiently without affecting the live application, enhancing both productivity and safety.

This thesis focuses only on developing the add-on and not implementing it on the platform. That part of the project belongs to Atea Finland Oy Ab.

Sensitive data in this thesis has been replaced or hidden in examples and some text in the thesis has been rephrased by ChatGPT to make the text more readable and engaging for readers (OpenAI, 2023).

1.1 Purpose and scope

The purpose of this thesis is to develop a user-friendly add-on for the sandbox platform aimed at simplifying the analysis of various components in installations. The add-on is designed to be easily configurable, requiring no to minimal coding knowledge ensuring accessibility for all engineers.

Useful data from various electrical components such as shaft generators, drives and batteries has been extracted in different forms of files. This data has never been effectively analysed due to the complexity and time-consuming nature of the value extraction process.

Key features of the add-on include the ability to run scheduled analyses, enhancing efficiency and consistency. It will be capable of extracting parameters from different databases, performing analyses, and providing reports directly within the sandbox environment. This functionality will empower engineers to gain valuable insights into the different kinds of installations effortlessly, streamlining their workflow and boosting productivity.

1.2 Employer

We Tech Solutions is a company located in the city centre of Vaasa and was founded in 2010. They create and sell different kinds of electrical solutions for different kinds of maritime vessels. Their main objective is to lower emissions and fuel consumption for their customers. We Tech has delivered hybrid solutions to more than 200 vessels globally (WE Tech Solutions, 2024).

1.3 Abbreviations

YAML – Yet Another Markup Language

JSON – JavaScript Object Notation

REST – Representational state transfer

API – Application programming interface

HTTP – Hypertext Transfer Protocol

ODBC – Open Database Connectivity

SQL – Structured Query Language

URL – Web Address

CSV – Comma-separated values

2 Theory

In this chapter, the theoretical part of the thesis will be presented, which will be used in the add-on. The programming languages, methods and reasons for using them in the practical part of the thesis will be explained.

2.1 Sandbox Environment

A sandbox environment is important for every development of software. It allows users to run programs or open files without impacting the main application, system or platform. It is a digital playground where users can play around without impacting the production system. Developers often use sandbox environments for testing new code, while cybersecurity professionals use it to analyse potentially harmful software. Testing software in a sandbox before deploying minimises the risk of production disruptions.

Sandbox environments serve as essential tools for risk mitigation in modern software engineering. Sandboxing provides isolation, which prevents unintended failures in software execution. This isolation ensures that any failure in the test environment remains isolated and does not affect the live system. This encourages testing and fosters innovation, as a developer can trial features or configurations in a safe setting (TechTarget, 2024).

In the context of this thesis, the sandbox provides a realistic yet secure environment for the design and development of the add-on. It enables experimentation, including system crashes or other errors without jeopardizing the stability of the live environment.

2.2 Python

Python is one of the most popular and widely used programming languages in the world. With python, it is possible to create software at any level, from the simplest program to a fully functional application. Its simple syntax and comprehensible code make it a popular general-purpose programming language among developers, engineers and beginner programmers.

The language is as mentioned versatile meaning you can use python for almost anything. For example, you can use python for building websites, automating tasks, creating games,

data analysis and machine learning. It is also easy to read and write python because of how python code almost looks like plain English thus making it easier to understand and write compared to other programming languages (Krishna Kumar & Acharya, 2023).

Recently, python has become more and more popular as a high-level general-purpose programming language. Compared to languages like C, developers can code the same kind of programs with fewer lines of code in python usually, thanks to the different libraries existing and parallel processing (Mueller & Massaron, 2019).

Python is a very portable language meaning it can be run on most other platforms coded in another language and very fast at processing data. Making it an obvious choice for this thesis.

Code example 1. Basic python.

```
#python code
x = 5
y = "John"

print(x)
print(y)

#Output
5
John
```

2.3 Python libraries

Developing more complicated code in python forces you to use python libraries. Python libraries are a collection of pre-written code that you can use in your own programs to avoid writing everything from scratch. Libraries contain functions, classes and tools that make coding easier and faster. It's like using a toolbox, instead of building every tool you need for the project you just grab one out of your toolbox and focus on your own project. Libraries also allow you to do things python can't do on its own, like making charts but libraries like matplotlib enable you to make charts (Geeksforgeeks, 2024). The subchapters of this chapter are python libraries used in this thesis.

2.3.1 Pyodbc

Pyodbc is an open-source Python module that makes accessing ODBC databases simple. This allows python to connect to databases and extract data from them. It works with any database that supports ODBC making it cross-database compatible. Pyodbc also allows the execution of multiple SQL queries in a single transaction. In this thesis, it is used to extract data variables from various parameter databases (Kleehammer, 2024).

2.3.2 PyYAML

PyYAML is a python library that allows YAML code to be parsed into python code. It enables the use of YAML-coded configuration files. It allows python programs to read, write and manipulate YAML data easily. Some key features of PyYAML are loading YAML into Python objects such as dictionaries, lists, etc. All this makes it a powerful and easy-to-use python library for working with YAML files (Simonov, 2024).

2.3.3 Math

The Python Math Library is a built-in library in Python that provides a collection of mathematical functions for performing advanced mathematical operations. It enables the user to do complex math calculations like arithmetic, trigonometry, logarithms, power calculations, factorials and much more (Python, 2024).

2.3.4 Requests

The Requests library is used to communicate with a website. On the website, there's an API that communicates with the application. Requests allow you to handle HTTP requests easily enabling the function to fetch web pages, send data and receive data (Reitz, 2024).

2.3.5 JSON

The JSON library is a built-in library in python. It provides functions for encoding and decoding JSON data. Python objects can be converted into JSON objects and JSON objects into python objects. It also handles file input and output for JSON data (Python, 2025a).

2.3.6 CSV

The CSV library is a built-in library in python, which is used to read and write CSV (Comma-Separated Values) files. It provides functions to manage structured data efficiently (Python, 2025b).

2.3.7 Base64

The base64 library is a built-in library in python. It provides functions for encoding and decoding binary data using Base64 encoding. Base64 is commonly used to encode binary data like files into a text format that can be safely sent over text-based protocols like JSON (Python, 2025c).

2.3.8 OS

The operative system library is a built-in library in python that provides functions for interacting with the operating system. It allows for file and directory management, environment variable access, process control, and system information retrieval (Python, 2025d).

2.4 YAML

YAML is a data serialization language designed to be human-friendly and compatible with most programming languages. YAML has been designed from the start to be useful and friendly to people working with data. There are hundreds of different programming languages but only a handful of them are good at handling data. YAML was specially created to work well for common uses such as configuration files, log files, interprocess messaging, cross-language data sharing, object persistence and debugging of complex data structures.

The first design rule for YAML is that it should be easily readable for humans, the second is that YAML data should be portable between programming languages. The third is that YAML should match the native data structures of dynamic languages (YAML, 2021).

Code example 2. YAML Configuration

```yaml
# A sample yaml file
company: spacelift
domain:
 - devops
 - devsecops
tutorial:
  - yaml:
      name: "YAML Ain't Markup Language"
      type: awesome
      born: 2001
  - json:
      name: JavaScript Object Notation
      type: great
      born: 2001
  - xml:
      name: Extensible Markup Language
      type: good
      born: 1996
author: omkarbirade
published: true
```

2.5 JSON

JavaScript Object Notation (JSON) is a standard text-based format for structured data based on JavaScript object syntax. It is commonly used for transmitting data between web applications such as web and server applications. Almost every website you have ever visited has JSON in it. JSON exists as a string and is useful when transmitting over network data, but it needs to be converted to a native JavaScript object to be used in software. JSON can be stored in a file thus making it useful for configuration files. It is relatively easy to read and manage (Developer Mozilla, 2024).

Code example 3. JSON

```json
{
    "employee": {
        "name":       "sonoo",
        "salary":      56000,
        "married":    true
    }
}
```

2.6 Swagger API

An API (Application Programming Interface) is a set of rules and protocols that allows different software applications to communicate with each other. It defines how requests and responses should be structured, enabling programs to exchange data and functionality efficiently. APIs allow systems to send, receive, and process data without exposing their internal workings. Swagger API is being used in this thesis by We Line to manage data. Swagger offers many tools for developers to design, build, document and use RESTful APIs. It simplifies the use of an API when developing, by creating automatic documentation, ensuring readability and usability. Swagger is widely used by developers making it also compatible with many tools and frameworks (Swagger, 2024).

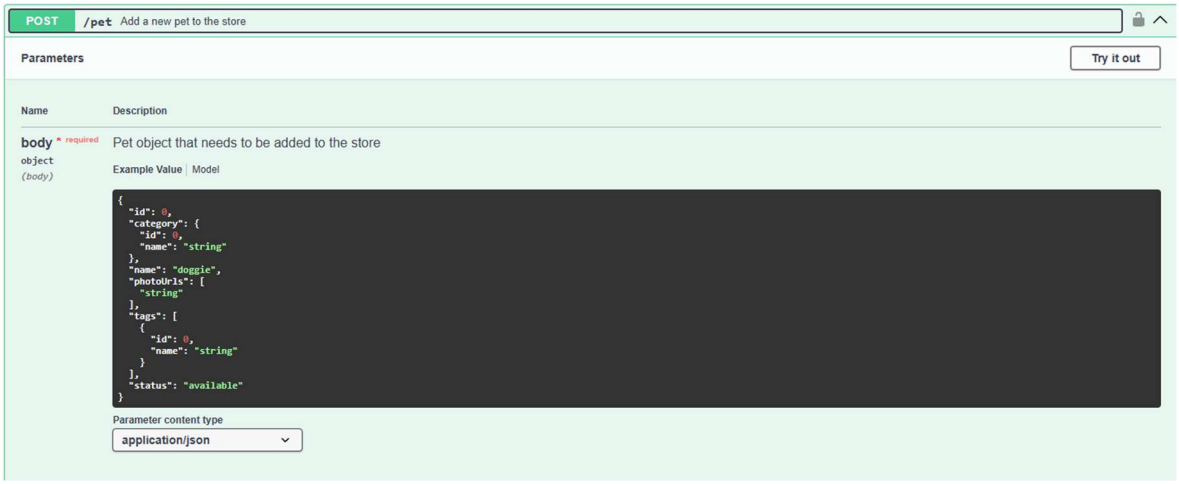

Figure 1. Post in Swagger UI. (Swagger, 2024).

2.7 Comma-separated values

CSV (Comma-separated Values) files are one of the simplest ways to store and exchange data. They are widely used in data processing, spreadsheets, databases, and applications that need structured data in a lightweight format. CSV files are plain text files where each row represents a record, each column is separated by a delimiter and can be opened in text editors, spreadsheets, and programming languages.

Code example 4. CSV

```
ID,Name,Age,Department,Salary
1,Alice,25,Engineering,60000
2,Bob,30,Marketing,50000
3,Charlie,28,HR,55000
```

2.8 SQL

SQL (Structured Query Language) is a standardised programming language for manipulating and managing relational databases. A relational database organizes data into tables, where each table consists of rows and columns. SQL has three main roles such as creating a database and defining its structure, querying the database to obtain data, and controlling database security. It enables users to retrieve, filter, sort, and aggregate data efficiently. There are multiple sublanguages of SQL but the most common is Data Manipulation Language also known as SQL queries. It allows a user to ask a question from the database (Wilton & Colby, 2005).

Code example 5. SQL Query

```
SELECT name, age
FROM employees
WHERE age > 30;
```

3 System architecture

It is essential to establish system architecture before developing an application and in this thesis, it is not any different. The system architecture is simple in this case due to developing only an add-on for a sandbox platform. WE Line is the starting point for system architecture. Before development started, We Tech Solutions gave some key features on the project.

- Add-on configuration should be user-friendly.
- Add-on should be developed in such a way that it is easy to implement in the sandbox.
- Add-on shall run the code and manage data on its own and not interfere with sandbox environment.
- We Line delivers the configuration file, necessary data and files.
- Add-on returns a report of the analysis when it's done.

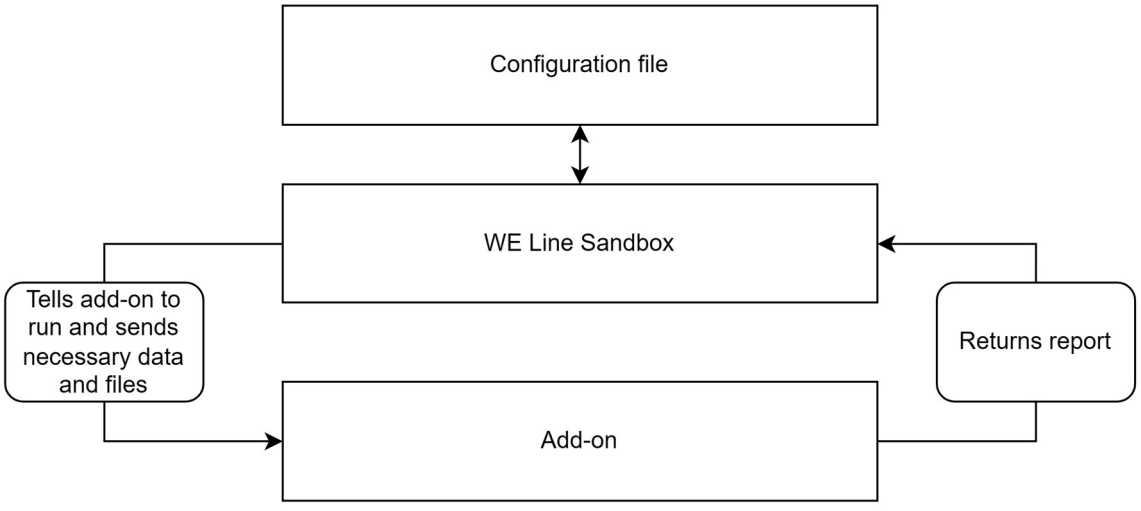

Figure 2. Initial flow map of the project.

The flow map above explains the initial thought about the project.

To begin we start with WE Line, in the sandbox settings, a tab is created to make configuration files. The configuration files are made manually and can be selected manually for each vessel. When selecting a vessel, an overview of its various products and

components is presented (Figure 3). Here a button “Run add-on” is available. When the user clicks it, a list of configuration files is displayed which can be used for the add-on. When the configuration file is selected the user can run the add-on.

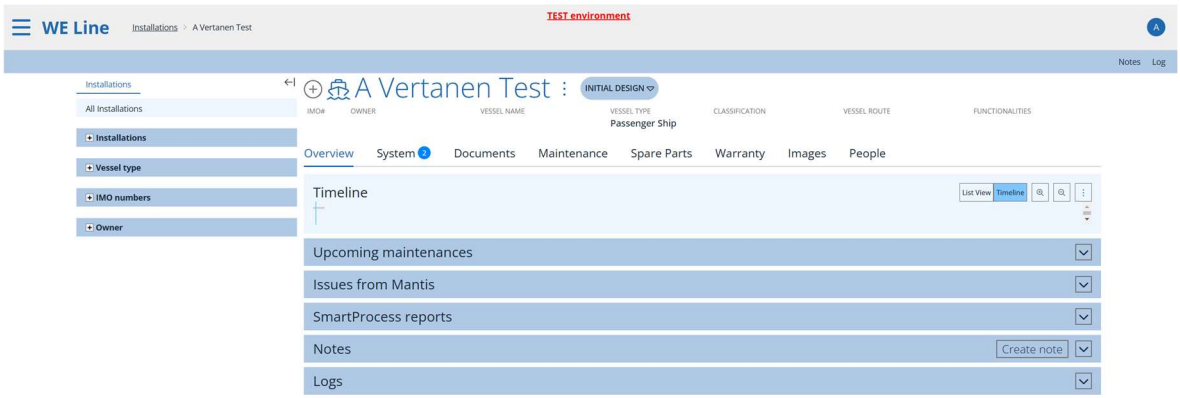

Figure 3. WE Line installation dashboard.

The add-on reads through the configuration file (Figure 4) and fetches data from the WE Line API. It then iterates through all products and components of the vessel to find the correct one. When it finds the right one, it should download the parameter file and parse it. Now, it searches for the different parameters you have given the add-on and goes through the analysing checks. When the analysing is done and everything has been calculated, the add-on returns a JSON format report to WE Line. WE Line displays every report by installation.

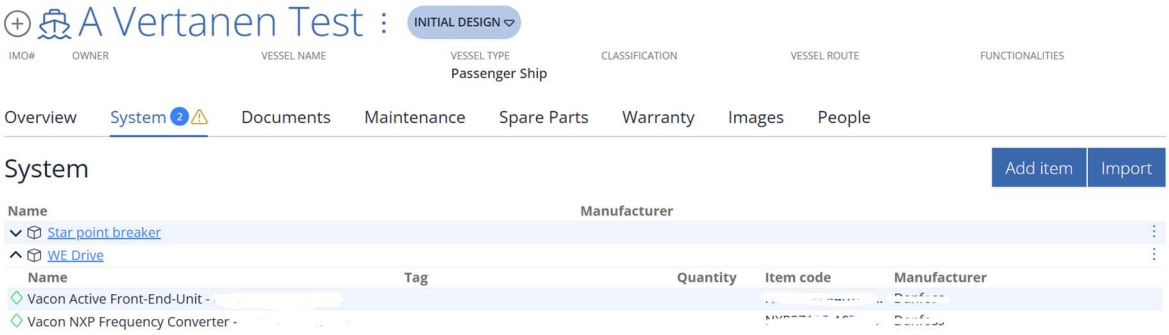

Figure 4. Products and components page.

The parameters are found in databases attached as parameter attachments to products and components (Figure 5). A service manual contains information about what parameters are included in the parameter file.

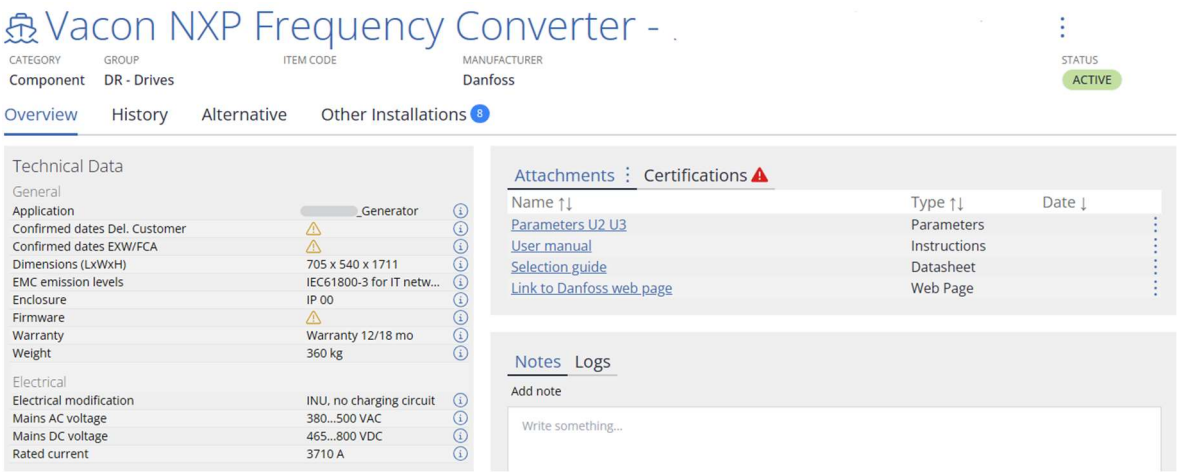

Figure 5. Attachments attached to components.

Here is an example of a service manual containing the chapter parameter list (Figure 6). From this parameter list the user gets an overview of the different parameters and what values they contain. From this list, the user chooses the parameters they want to use later in the analysis.

5. PARAMETER LIST

6.1 Basic parameters

Code	Parameter	Min	Max	Unit	Default	ID	Note
P2.1.1	Grid Nominal Voltage	500V: 380V 690V: 525V	500V: 500V 690V: 690V	V	400	110	Set here the nominal voltage of the grid.
P2.1.2	Grid Nominal Frequency	48	63	Hz	50	1532	Set here the nominal frequency of the grid.
P2.1.3	System Rated Current	0,0	Ih	A	Ih	113	Capacity of supply, used if oversized AFE.
P2.1.4	System Rated Power	0	32000	kW	0	116	
P2.1.5	Parallel AFE	0	3		0	1501	**0** = Single AFE **1** = Parallel AFE **1** = Master **1** = Follower

Table 4-2. Basic parameters

Figure 6. Parameters listed in the user manual.

4 Configuration

Several different well-known formats have evolved over the years to make writing a configuration file easy. It is thanks to these configuration files that your programs have a memory that keeps your settings for each application.

Configuration files can be and often are very simple in structure. If the software only needs data for a single object with a name and description, it could be, for example, written like this.

```
NAME='Tux'
SPECIES='Penguin'
```

Figure 7. Configuration example

Without programming experience, you can see how the code stores data. Now, there are a couple of known formats for config files, but the one used in this thesis is YAML (Opensource, 2021).

The configuration structure is essential to this thesis because it is what the users will interact with, so it must be user-friendly. YAML was chosen because its primary benefit is readability. It is designed to be easily read and written by humans, using indentation rather than brackets or commas to structure data. This makes the complex configuration easier to understand at first glance. “YAML’s syntax reduces visual noise and allows users to focus on the data structure rather than a delimiter” (YAML, 2021).

The first section of the configuration file is the application you want to analyse. Application values can be found in each product's technical attributes. For example, if the user wants to analyse a grid converter, the user simply enters “Grid Converter”.

Code example 6. The first section of the configuration file.

```
#Config file for add-on

#Enter application you want to analyze
#Example:
#Application:
#  - GridConverter
Application:
  - GridConverter
```

The second section of the configuration is what technical attributes the user wants to use and store for analysis. The technical attributes are found in products and components in the WE Line platform and stored as strings. The user simply needs to copy the string from the technical attribute section and paste it into the configuration.

Code example 7. The second section of the configuration file.

```
#Add-on searches for these values and stores them. WE Drive Tech.
Atr.
#Example:
#Technical Attribute Values:
# "- IN: F/Hz"
Technical Attribute Values:
  - "IN: Un/V"
  - "IN: In/A"
  - "IN: F/Hz"
```

The third section of the configuration file is where the user enters the parameters they want to fetch. In the service manual of a product, they will find the variable ID associated with the parameter they want to fetch from the database. Users also need to specify the type of parameter file. In this version of the add-on, it can analyse two types of databases. One with the file extension ".par" which is a type of Microsoft Access database and the other one with ".csv", which is short for comma-separated values.

Code example 8. The third section of the configuration file

```
#Enter the name of the parameter you want to analyse from the
database
#Example:
#Parameters:
# - 110 # VarID for Grid Nominal Voltage, found in service text in
******
Parameters:
  par:
    - 110  # VarID for Grid Nominal Voltage
    - 1532  # VarID for Grid Nominal Frequency
  csv:
    - 9048  # VarID for Grid Nominal Voltage
```

The last section of the configuration file defines the different analysis checks users want to perform. These can be named whatever the user wants to make it clearer in the report what the check was, as there could be a lot of checks performed in the same installation. The user is then able to do arbitrary calculations using pythons evaluation function. Every parameter and technical attribute that was defined in the earlier parts of the configuration can be used in the analysis.

Code example 9. The final section of the configuration file.

```
#Enter multiple checks, uses pythons eval()
#Example:
#Checks:
# GridConverter  #Application string found in *******
#  - check1: Parameters[110] > 230
Checks:
  #Value comparison operators: <, >, <=, >=, ==, !=
  #Logical (Boolean) operators: and, or, not
  #Membership test operators: in, not in
  #Identity operators: is, is not
  #Mathematical functions:
https://docs.python.org/3/library/math.html
    - Voltage differ check: Parameters[110] <= IN__Un_V
    - Difference between Freq check: math.fabs((Parameters[1532]/100)
- IN__F_Hz)
    - Difference between freq rounded to 2 decimal check:
math.fabs(Parameters[110] - Parameters[9048])
```

5 Add-on

This chapter and its subchapters outline the various functions of the add-on and provide code examples of it. It ties into the theoretical section of this thesis, demonstrating how the concepts discussed in the theory chapters relate to the implementation. Furthermore, it explains the logic behind the data handling and how the data is analysed and utilized from the WE Line platform.

Before beginning development of the add-on, it was necessary to make a flow map based on the earlier chapters, to gain some clarity in the logic and structure of the coding. As previously mentioned, this program should run independently from the sandbox and should only be added as an add-on to the sandbox platform.

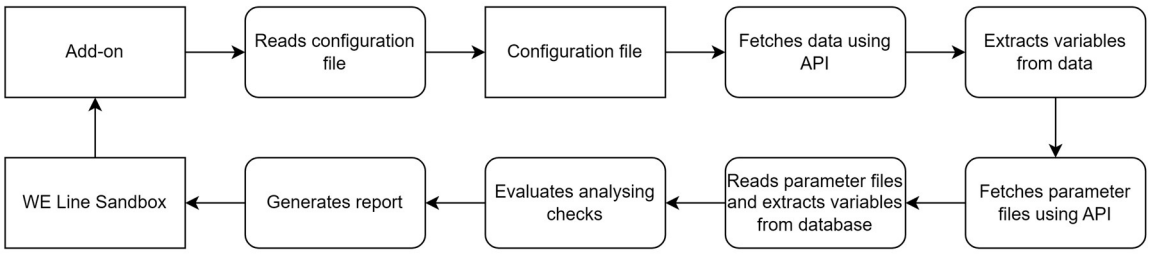

Figure 8. Flow map of add-on

Different functions are developed that follow the same logic and order as the flow map. These functions are developed one at a time and later, tie the add-on together. To execute the add-on, specific variables must be provided, including the configuration file, authorization, and installation ID. Once these variables are supplied, the add-on stores the data and utilizes them as global variables across all functions.

```
def __init__(self, installation_id, authorization, config_file="config.yaml"):
        self.installation_id = installation_id
        self.authorization = authorization
        self.config_file = config_file
```

The next subchapters are going to explain the different functions and they are in order as to how the code is executed. The code examples are only part of the add-on and have been altered to hide sensitive data but give an overview of the developed add-on.

5.1 API call

The WE Line sandbox swagger API returns JSON data that will be used in the add-on. The requirements for calling the API are authorization and installation ID. This function calls the API using a URL and passes through authorization and installation ID. By passing through authorization directly from the sandbox, the user avoids logging in separately to use the add-on. The API call returns different response codes depending on the call. For example, if the API call is successful, the API return code is 200. If the authorisation fails or installation ID fails, the API returns code 401, which means that the call was unsuccessful. These error messages are stored in an error array that is passed on to the report, so the end user knows what went wrong if anything went wrong.

Code example 10. API Call

```
def GetHardwareData(self):
        errors = []
        try:
            url = f"API_URL"
            headers = {"accept": "application/json", "Authorization":
self.authorization}
            response = requests.get(url, headers=headers)

            if response.status_code == 200:
                return response.json(), errors
            else:
                error_message = f"Failed to retrieve data. Status code:
{response.status_code}"
                errors.append(error_message)
                return None, errors
        except Exception as e:
            error_message = f"Error connecting to the API: {e}"
            return None, errors
```

The data that has been returned to the add-on is easily organizable. The JSON data has a specific structure that can be manipulated to select only the data needed for the add-on. For example, the technical attributes defined earlier in the configuration file are the values we want to extract from the data and nothing else. Alternatively, the data could be saved directly in an array but to minimise the runtime of the program it is being filtered out.

Figure 9. JSON data structure from Swagger API documentation.

5.2 Data handling

After the add-on has received the data and the response code is okay, it proceeds to process the hardware and subitems. It creates a results array where different variables are stored after being processed. The function itself receives the API data and the configuration file. Every hardware could, for example, be a generator, and for that generator, there will be components defined in the API data as subitems. This loops through every hardware and passes the hardware data for extracting technical attributes. For every hardware it also loops through the subitems belonging to that hardware and for every subitem that has an attachment, the attachment is passed to reading the attachment if it's a parameter file and defined in the configuration file.

Code example 11. Processing hardware

```
def ProcessHardwareAndSubItems(self, api_data, config):
        results = []
        for hardware_item in api_data:
            hardware_technical_attributes, hardware_errors =
self.ExtractTechnicalAttributes(hardware_item, config)
            sub_items = hardware_item.get("subItems", [])
            for sub_item in sub_items:
                sub_item_technical_attributes, sub_item_errors =
self.ExtractTechnicalAttributes(sub_item, config)
                links = sub_item.get("links", [])
                param_files = [link["url"] for link in links if
link.get("attachmentType") == "Parameters"]

                if param_files:
                    for param_file_path in param_files:
                        param_values, param_names, read_errors =
self.ReadDB(param_file_path, config["Parameters"])
                        check_results, check_errors = self.EvaluateChecks(config,
param_values, hardware_technical_attributes, sub_item_technical_attributes)
                        results.append({
                            "hardwareName": hardware_item.get("name"),
                            "subItemName": sub_item.get("name"),
                            "paramFile": param_file_path,
                            "technicalAttributes": hardware_technical_attributes,
                            "subItemTechnicalAttributes": sub_item_technical_attributes,
                            "paramValues": param_values,
                            "paramNames": param_names,
                            "checkResults": check_results,
                            "checksMade": bool(check_results),
                            "errors": read_errors + check_errors + hardware_errors +
sub_item_errors
                        })

        return results
```

The function handles the extraction of hardware technical attributes and subitem technical attributes. An error array is created at the beginning to keep track of any errors that may come up during runtime. A variable “relevant_attributes” is created where the defined technical attributes in the configuration file are saved. It saves every matching technical attribute name and value found and later, compares them to the relevant attributes; if they match, the value is saved, and the name of the technical attribute gets sent to the preprocessing function. The preprocessing is there to convert any special characters to underscores. If this is not done, the variables will be unusable in the analysis because python does not support special characters in variable names therefore creating an error. The variable names are originally named in the configuration file and therefore can include special characters.

Code example 12. Extracting technical attributes

```
def ExtractTechnicalAttributes(self, hardware_item, config):
        """Extracts and processes only the relevant technical attributes from the
hardware item JSON based on config."""
        errors = []
        technical_attributes = {}

        # Get the list of relevant technical attributes from the config
        relevant_attributes = config.get("Technical Attribute Values", [])

        for attr in hardware_item.get("technicalAttributeValues", []):
            attr_name = attr["technicalAttribute"]["name"]
            attr_value = attr["value"]

            # Only process attributes that are listed in the config
            if attr_name in relevant_attributes or attr_value in config["Application"]:
                processed_name = self.PreprocessTechnicalAttributeName(attr_name)

                try:
                    attr_value = float(attr_value)
                except (ValueError, TypeError):
                    pass

                technical_attributes[processed_name] = attr_value

        return technical_attributes, errors
```

The code continues to run until every technical attribute value is extracted that was defined in the configuration file. In the sub-item data, you can also find different kinds of attachments. When the code is searching for the sub-item technical attributes, it also gets every attachment, if there are any. If there is an attachment, it gets its URL and checks if the attachment type is “Parameters” If these criteria are met, the code saves the parameter object.

5.3 Database reading

To begin, several parameter files were provided by WE Tech for the study to determine how data could be extracted from them and what type of files they were. The ones with a file extension “.par” were unclear, so the best way to decide what type of database they were, was to open them in a text editor.

```
Standard Jet DB   µn b`   ÂUé©gr@? œ~Ÿ ÿ…š1Åyºí0%ßÌ cÙäÃŸFûŠ¼NMKì7–ôœú
î(æØ ŠˆŸ${6>Âß±¼D C  ±3ÿÑy[Y |*L×|™  ˜ýrUça '†f_•øĐ‰$…gÆ 'DÒîÏeíÿ ÇF¡x
íé-bÔT  4.0
```

From the figure, it can be concluded that the database is a Standard Jet DB. This is a variant of Microsoft Access databases, and therefore, the pyodbc python library is used to execute SQL queries within the database but first, the add-on needs to request the files.

A criterion of the add-on was that it should not have to log in to the file server to access the different parameter files and the WE Line platform should give the file through an API instead. The database reading starts with defining two functions one for decoding the response from the API and one for deleting the temporary file. The API is then called with the JSON attachment object payload, and the API responds with the file content coded in base64 format. Here base64 is chosen because it allows you to include binary data directly in the JSON payload, making it easier to send files through a JSON-based API.

Code example 13. Database API call

```
         def decode_base64_to_file(base64_string, output_file_path):
            with open(output_file_path, "wb") as file:
                file.write(base64.b64decode(base64_string))

        def DeleteTemp(file_path):
            if os.path.exists(file_path):
                os.remove(file_path)
            else:
                print(f"The file {file_path} does not exist")

        url = apiurl
        response = requests.get(url, json=param_file)

        if response.status_code != 200:
            error_message = f"Failed to retrieve data from file retrieval API. Status
code: {response.status_code}"
            errors.append(error_message)
            return {}, {}, errors

        db_data = response.json()
        content = db_data.get("content")
```

The file extension is split from the attachment object to determine what kind of file type the decoder is going to output. The file is saved in a temp folder and then depending on what file extension it was the add-on starts reading with different logic. With a Microsoft jet database ending with a file extension of “.par”. A Microsoft Access driver is used to make SQL queries. The query selects variable IDs, names and values from the parameters table and then extracts them to variables. Finally, when the code has run the add-on deletes the temporary file.

Code example 14. ".par" database

```
if db_path.endswith(".par"):

            decode_base64_to_file(content, file_path)
            try:
                conn = pyodbc.connect(f"Driver={{Microsoft Access Driver (*.mdb,
*.accdb)}};DBQ={file_path};")
                cursor = conn.cursor()

                param_values = {}
                param_names = {}
                query = f"SELECT [VarID], [Var], [Value] FROM [Parameters] WHERE [VarID]
IN ({','.join(['?'] * len(selected_var_ids))})"
                cursor.execute(query, tuple(selected_var_ids))

                for row in cursor.fetchall():
                    var_id, var_name, value = row
                    param_names[int(var_id)] = var_name
                    try:
                        param_values[int(var_id)] = float(value)
                    except ValueError:
                        param_values[int(var_id)] = value

                return param_values, param_names, errors
            except pyodbc.Error as e:
                error_message = f"Database error: {e}"
                errors.append(error_message)
                return {}, {}, errors
            finally:
                cursor.close()
                conn.close()
                DeleteTemp(file_path)
```

The same logic follows for comma-separated values databases. Here, the function starts with reading the first line, and if it starts with items, the code knows that it's the heading for the values separated by commas. With the header, the function knows where to search for variable id, name and value again, saving them and returning the parameters back to the processing function.

Code example 15. "csv" database reading.

```
if db_path.endswith(".csv"):

            decode_base64_to_file(content, file_path)
            try:
                param_values = {}
                param_names = {}
                with open(file_path, "r", newline="") as csvfile:
                    # Read the first line. If it starts with "Items:" then read the
header
                    first_line = csvfile.readline().strip()
                    if first_line.startswith("Items:"):
                        header_line = csvfile.readline().strip()
                    else:
                        header_line = first_line
                    fieldnames = header_line.split(";")

                    # Create DictReader with defined fieldnames and delimiter
                    reader = csv.DictReader(csvfile, fieldnames=fieldnames,
delimiter=";")

                    for row in reader:
                        # Validate that required fields exist
                        if "ID" not in row or "VariableText" not in row or "Value" not
in row:
                            print(f"Skipping malformed row: {row}")
                            continue

                        try:
                            var_id = int(row["ID"])
                        except (ValueError, TypeError) as e:
                            print(f"Error converting ID to int in row {row}: {e}")
                            continue

                        var_name = row["VariableText"]
                        value_str = row["Value"].strip()
                        try:
                            value = float(value_str)
                        except ValueError:
                            value = value_str  # keep as string if conversion fails
                        if var_id in selected_var_ids:
                            param_names[var_id] = var_name
                            param_values[var_id] = value
                return param_values, param_names, errors
            except Exception as e:
                error_message = f"Error reading CSV file: {e}"
                errors.append(error_message)
                return {}, {}, errors
            finally:
                DeleteTemp(file_path)
```

5.4 Evaluating checks

Evaluating the checks or the analysis part of the add-on is a bit more complicated than the other functions. The error handling ensures again that the configuration is valid and that the different sections of it are valid, if they are not, it returns an error message letting the user know what went wrong in the final report. Using Python's evaluation command imposes a security risk, but even accessing the add-on requires authentication. Just to make

sure the evaluation is only done if the commands and variables match things put in the safe context array. If for some reason it does not match, the evaluation will not take place, making sure only the right commands will be executed, and malicious code will not be able to enter.

Code example 16. Evaluating checks

```
def EvaluateChecks(self, config, param_values, technical_attributes,
sub_item_technical_attributes):
        errors = []
        check_results = {}

        if "Application" not in config or not isinstance(config["Application"], list):
            error_message = "Application section is missing or invalid in config"
            errors.append(error_message)
            return check_results, errors

        if "Checks" not in config or not isinstance(config["Checks"], list):
            error_message = "Checks section is missing or invalid in config"
            errors.append(error_message)
            return check_results, errors

        valid_applications = set(config["Application"])

        application_found = any(app in technical_attributes.values() for app in
valid_applications) or any(app in sub_item_technical_attributes.values() for app in
valid_applications)
        if not application_found:
            error_message = "No valid application found in technical attributes"
            errors.append(error_message)
            return check_results, errors

        safe_context = {
            "math": math,
            "Parameters": param_values,
            **technical_attributes,
            **sub_item_technical_attributes
        }

        # Process checks only if an application match was found
        for check in config["Checks"]:
            if not isinstance(check, dict):
                error_message = f"Invalid check: {check}"
                errors.append(error_message)
                continue  # Skip invalid checks

            check_name, expression = list(check.items())[0]

            try:
                check_result = eval(expression, {"__builtins__": None}, safe_context)
                check_results[check_name] = {
                    "expression": expression,
                    "result": check_result,
                    "error": None
                }
            except Exception as e:
                check_results[check_name] = {
                    "expression": expression,
                    "result": None,
                    "error": str(e)
                }

        return check_results, errors
```

5.5 Generating a report

Every function to this point has returned different variables that store information that is used in the final report. The generate report function loops through the variables and lists them in an order as JSON data. The JSON data can then be easily filtered and managed on the front-end side to show the end user their report over the checks. Much of the data are things for the backend such as IDs for installations, hardware and subitems. The report is saved in a folder named Reports where the name of the file is the installation ID and the date when the report was generated.

Code example 17. Generating report

```
 def GenerateReport(self, results, output_file):
        report_data = []

        for result in results:
            report_entry = {
                "installationId": result["installationId"],
                "hardwareName": result["hardwareName"],
                "hardwareId": result["hardwareId"],
                "subItemName": result["subItemName"],
                "subItemId": result["subItemId"],
                "paramFile": result["paramFile"],
                "paramFileUrl": result["paramFileUrl"],
                "technicalAttributes": result["technicalAttributes"],
                "subItemTechnicalAttributes": result["subItemTechnicalAttributes"],
                "paramValues": {},
                "checkResults": {},
                "checksMade": result["checksMade"],
                "errors": result["errors"]
            }

            if result["paramValues"]:
                report_entry["paramValues"] = {
                    str(varId): {
                        "name": result["paramNames"].get(varId, f"Unknown VarID
{varId}"),
                        "value": value
                    } for varId, value in result["paramValues"].items()
                }

            if result["checksMade"]:
                report_entry["checkResults"] = {
                    check_name: {
                        "expression": details["expression"],
                        "result": details["result"],
                        "error": details["error"]
                    } for check_name, details in result["checkResults"].items()
                }

            report_data.append(report_entry)

        report_content = json.dumps(report_data, indent=4)
        print(report_content)

        with open(output_file, "w") as file:
            file.write(report_content)
        print(f"Report saved to {output_file}")
```

5.6 Report Data

The values and names in this example of a final report are altered and are only used as an example. A lot of data gets generated but as earlier said a lot of it will be filtered and only parts are shown to the end user.

Code example 18. Report JSON data


```
{
        "installationId": "1234-Example-Installation",
        "hardwareName": "WE Drive",
        "hardwareId": "1234-Example-Hardware",
        "subItemName": "Frequency Converter",
        "subItemId": "1234-Example-Installation",
        "paramFile": "Parameters",
        "paramFileUrl": "https://www.example.com/freqconverter.par",
        "technicalAttributes": {
            "IN__In_A": 16,
            "IN__F_Hz": 60.0,
            "IN__Un_V": 230.0
        },
        "subItemTechnicalAttributes": {
            "Application": "GridConverter"
        },
        "paramValues": {
            "110": {
                "name": "P_LineVoltage",
                "value": 450.0
            },
            "1532": {
                "name": "P_FreqRef",
                "value": 6000.0
            }
        },
        "checkResults": {
            "Is line voltage bigger than grid voltage": {
                "expression": "Parameters[110] <= IN__Un_V",
                "result": true,
                "error": null
            },
            "Difference between freq": {
                "expression": "math.fabs((Parameters[1532]/100) - IN__F_Hz)",
                "result": 44.0,
                "error": null
            },
            "Difference between freq rounded to 2 decimal check": {
                "expression": "math.fabs(Parameters[110] - Parameters[9048])",
                "result": null,
                "error": "9048"
            }
        },
        "checksMade": true,
        "errors": []
    },
```

6 Result

The goal of this thesis was to develop an analysis add-on for testing the newly developed sandbox platform. In this instance, the goal was met, and the add-on was successfully developed.

As mentioned earlier, my part of this project at WE Tech Solutions was to develop the add-on, not implement it on the sandbox platform. Atea Finland Oy Ab is responsible for the platform and the deployment of the developed add-on. This does not mean that the add-on cannot be tested locally as it should be possible to do in sandboxes.

The testing has been done with a couple of parameter files given by WE Tech Solutions and done with limited resources. This has resulted in only being able to do a couple of analyses on mostly the same electrical component with the same database.

The result is a working add-on that has been implemented to their platform, which can extract variables from databases and use JSON data from the platform to execute user input analysis of various components. The add-on runs on its own and performs analyses on each component independently. When an installation has been run with a certain configuration the add-on gives a report on the analysis of every component. The resulting report is then used to gain valuable insight into the state of a solution. This benefits the engineers to further optimize settings in the electrical components.

All in all, the goals of this thesis were met, and WE Tech Solutions was satisfied with the result.

7 Further Development

A lot of further development can be done and some need to be done for continuous operation. Many of the further development points can only be done by ATEA as they include actual changes to the WE Line platform.

- The configuration files could be for example made into objects rather than the user needing to write a whole configuration file. The objects could be displayed in the front end in such a way that makes it easier for the user to use the add-on.

- Configuration files should be made by Application Engineers at WE Tech as they have done the analyses before manually.

- A list of commonly used parameters for vessel applications could be implemented to ease the user of the process of searching the parameter variable ID from the service manual every time.

- Bigger database support as it currently only supports two types of databases which are databases of only two of the endless electrical components used in the solutions.

- Timed execution, when a database is added to a component, the add-on would run automatically with a certain configuration. This would be useful to validate if the factory settings in the component are compatible with the solution.

- User guide, and documentation on how to use the add-on.

A couple of these points were brought up by engineers in a demo meeting of the add-on at WE Tech Solutions.

8 Discussion

When this task was brought to me by, We Tech Solutions it felt like it was not possible. The first step of the thesis was to learn python. Learning a new programming language can be hard but if you have the right fundamentals from another language, it should not be too challenging. The first hurdle I encountered was the user input analysis until I read the theory about python evaluation function. The evaluation function made the work easier because I thought that in the beginning, the evaluation was necessary to code from the beginning meaning that the different mathematical operators would have to be coded separately. After realising that the evaluation part of the add-on was built-in in python, the whole picture of the add-on became clearer.

When starting to build the add-on it was easier to take it part by part instead of focusing on the result. At that time, I created the flow map which was easy to follow and build the different functions from. When following the flow map it became clear how the add-on should be structured and how the data should be handled and from there on it was only a matter of coding the functions and testing it. Function by function was created and tested and after the last function was created it was a matter of testing the whole add-on. The lack of examples given by WE Tech Solutions was a bit of a challenge. The add-on needed to be tested locally and only with the given databases or example analysis checks. When the first version was created and working, I consulted Atea Finland about it and if it is possible to implement it in the sandbox platform for furthermore testing. This resulted in re-coding the whole add-on to better suit the sandbox functionalities. When demoing the working version to WE Tech we noticed that some features were missing such as having the possibility to also use sub-item technical attributes in the analysis. A lot of the work went back and forth like this until both parts were satisfied with a version.

This thesis has been extremely educational for me for multiple reasons such as learning a whole new programming language, managing a lot of data, attending and consulting on meetings and simply just working on a bigger project than usual. I have also noticed how important it is to stick to a plan and have version controls of the project. I will continue to further develop this add-on and expand my knowledge at WE Tech Solutions.

9 References

Developer Mozilla. (2024). *mdn web docs.* Retrieved January 07, 2025, from mdn web docs: https://developer.mozilla.org/en-US/docs/Learn_web_development/Core/Scripting/JSON

Geeksforgeeks. (2024, August 1). *Geeksforgeeks*. Retrieved February 12, 2025, from Geeksforgeeks: https://www.geeksforgeeks.org/libraries-in-python/

Kleehammer, M. (2024, October 16). *pypi*. Retrieved January 13, 2025, from pypi: https://pypi.org/project/pyodbc/

Krishna Kumar, M., & Acharya, M. (2023). *Basics of Python Programming.* Bentham Science Publishers.

Mueller, J. P., & Massaron, L. (2019). *Python for Data Science for Dummies.* John Wiley & Sons, Incorporated.

OpenAI. (2023). *ChatGPT*. Retrieved January 2025, from ChatGPT: https://chat.openai.com/chat

Opensource. (2021). *Opensource.* Retrieved January 10, 2025, from https://opensource.com/article/21/6/what-config-files

Python. (2024, August 1). *Python docs*. Retrieved January 13, 2025, from Python docs: https://docs.python.org/3/library/math.html

Python. (2025a, January 1). *Python Docs*. Retrieved February 25, 2025, from Python Docs: https://docs.python.org/3/library/json.html

Python. (2025b, January 1). *Python Docs*. Retrieved February 25, 2025, from Python Docs: https://docs.python.org/3/library/csv.html

Python. (2025c, January 1). *Python Docs*. Retrieved February 25, 2025, from Python Docs: https://docs.python.org/3/library/base64.html

Python. (2025d, January 1). *Python Docs*. Retrieved February 25, 2025, from Python Docs: https://docs.python.org/3/library/os.html

Reitz, K. (2024, May 29). *pypi*. Retrieved January 13, 2025, from pypi: https://pypi.org/project/requests/

Simonov, K. (2024, August 6). *pypi*. Retrieved February 16, 2025, from pypi: https://pypi.org/project/PyYAML/

Swagger. (2024). *Swagger*. Retrieved January 26, 2025, from Swagger: https://swagger.io/docs/specification/v2_0/what-is-swagger/

TechTarget. (2024, January 1). *Definition sandbox.* Retrieved January 4, 2025, from TechTarget: https://www.techtarget.com/searchsecurity/definition/sandbox

WE Tech Solutions. (2024). Retrieved November 7, 2024, from https://wetech.fi/company/

Wilton, P., & Colby, J. (2005). *Beginning SQL.* John Wiley & Sons, Incorporated.

YAML. (2021). *YAML*. Retrieved January 2025, from https://yaml.org/spec/1.2.2/#chapter-1-introduction-to-yaml