Czech Technical University

Faculty of Mechanical Engineering

Department of Instrumentation and Control

FACULTY OF MECHANICAL ENGINEERING CTU IN PRAGUE

Industrial IoT Web Application for Motor Characteristic Monitoring

Master Thesis

Sami Jradi

Supervisor: Ing. Vladimír Hlaváč, Ph.D.

Co-supervisor: MSc. Manuel Aguado Puertas (Siemens DI FA APC3)

Declaration of Authorship

I, Sami Jradi, declare that this thesis titled, 'Industrial IoT Web Application for Motor Characteristic Monitoring' and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a master's degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed:

Date:

MASTER‘S THESIS ASSIGNMENT

I. Personal and study details

Student's name:	**Jradi Sami**	Personal ID number:	**467453**
Faculty / Institute:	**Faculty of Mechanical Engineering**		
Department / Institute:	**Department of Instrumentation and Control Engineering**		
Study program:	**Mechanical Engineering**		
Branch of study:	**Instrumentation and Control Engineering**		

II. Master’s thesis details

Master’s thesis title in English:

Industrial IoT Web Application for Motor Characteristic Monitoring

Master’s thesis title in Czech:

Průmyslová webová aplikace IoT pro monitorování charakteristik motorů

Guidelines:

1. Build the structure of the backend using MVC method (java + spring boot)
2. Create a connection to an SQL database and perform CRUD operations
3. Containerizing the application in Docker (create the image file)
4. Test the application with real hardware

Bibliography / sources:

[1] Craig Walls: Spring Boot in Action. Manning Publications, 2016, ISBN 1617292540.
[2] Robert C. Martin: Clean Code, A Handbook Of Agile Software Craftmanship. Pearson Education (US), 2008. ISBN 0132350882
[3] Chris Fehily: SQL (Database Programming). 2015, Questing Vole Press. ISBN 1937842312

Name and workplace of master’s thesis supervisor:

Ing. Vladimír Hlaváč, Ph.D., U12110.3

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

MSc. Manuel Aguado Puertas, Siemens DE, Application Center, Division of Digital Industries of the Factory Automation (DI FA APC)

Date of master’s thesis assignment: **30.04.2021** Deadline for master's thesis submission: **10.06.2021**

Assignment valid until: ______________

Ing. Vladimír Hlaváč, Ph.D.		prof. Ing. Michael Valášek, DrSc.
Supervisor’s signature	Head of department’s signature	Dean’s signature

III. Assignment receipt

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

.	
Date of assignment receipt	Student’s signature

Czech Technical University

Abstract

Faculty of Mechanical Engineering
Department of Instrumentation and Control Engineering

Master of Science

Industrial IoT Web Application for Motor Characteristic Monitoring

By Sami Jradi

This thesis is based on the technology of web development to build the back-end of an application for motor characteristic monitoring in the field of industrial IoT. The purpose of the web application is to receive data such as velocity, acceleration, position and torque from an industrial servo motor connected to a Siemens SIMOTION D PLC using MQTT protocol, and then analyze this data in the form of charts. The resulting chart would be known as the load curve which will offer the chance to perform predictive maintenance on the hardware functioning inside the factory. The backend of the application will be written in the Java programming language using Spring Boot application framework. This backend will then be connected to an SQL database such as PostgreSQL where all the essential data of the application will be stored. An API documentation will be generated using Swagger containing all required endpoints for the frontend developers to implement the user interface. After building both the backend and frontend of the application, an image file can be created using Docker which will run on industrial devices used in factory automation.

České vysoké učení technické v Praze

Abstrakt

Fakulta strojní
Ústav přístrojové a řídící techniky

Master of Science

Průmyslová webová aplikace IoT pro monitorování charakteristik motorů

By Sami Jradi

Základem této práce je vývoj podpůrné serverové aplikace (back-end), monitorující charakteristiky motorů v oblasti průmyslového IoT (internetu věcí). Účelem této podpůrné aplikace je přijímat změřená data, jako je rychlost, zrychlení, poloha a točivý moment z průmyslového servomotoru připojeného k PLC Siemens SIMOTION D. Data jsou přijímána pomocí protokolu MQTT a poté zpracovávána do formy grafů. Výsledný graf, známý jako křivka zatížení, nabízí možnost předvídat a v předstihu provádět údržbu hardwaru během jeho užívání v průmyslovém provozu. Podpůrná serverová aplikace (back-end) byla vytvořena v programovacím jazyce Java s využitím aplikačního rámce Spring Boot a následně propojena s SQL databází (použit PostgreSQL), kam se ukládají změřená a vypočtená data. Dokumentace API (Application Programming Interface, popis jak volat tuto serverovou podporu) byla vygenerována pomocí programu Swagger (framework pro návrh, tvorbu a dokumentaci API) a obsahuje popis veškerých možných přístupových bodů a volání, které mohou tvůrci nejen webových aplikací (na straně klienta, front-end) využívající vytvořenou serverovou podporu (back-end) implementovat. Po sestavení podpůrné aplikace na straně serveru i aplikace na straně klienta lze pomocí virtualizačního prostředí Docker vytvořit image (obraz paměti), který lze spouštět na průmyslových zařízeních používaných v tovární automatizaci.

Acknowledgements

First of all, I would like to thank my supervisor Ing. Vladimír Hlaváč, Ph.D. for his valuable help, knowledge, and guidance. At the same time, I would like to thank Siemens Digital Industries for providing me with the tools and opportunity for completing this project, as well as giving me the chance to be part of the team at the Application Center in Prague (APC3).

In addition, a special mention goes to my co-supervisor and team leader at Siemens Digital Industries, MSc. Manuel Aguado Puertas. This project was possible thanks to his ability in guiding the efforts of development towards successful delivery as well as his project management skills.

I would like to thank my family and friends that have supported me during my time in Prague, who have also contributed greatly to producing this work.

Contents

Declaration of Authorship 2

Abstract 4

Abstrakt 5

Acknowledgements 6

List of Abbreviations 10

1. Introduction 11

1.1 Load Curve 11

1.1.1 Industrial Edge Platform 12

1.1.2 SIMOTION D PLC 13

1.1.3 OPC Communication 14

1.1.4 MQTT 15

1.1.5 Load Profile 16

2. Research and Requirements 21

2.1 Architectural Overview 21

2.1.1 The Back-end 21

2.1.2 API Controller 23

2.1.3 The Front-end 24

2.1.4 Database 25

3. Implementation 28

3.1 Back-End Implementation 28

3.1.1 Java 28

3.1.2 Spring Boot 29

3.1.3 Spring Data JPA 31

3.1.4 OPC XML-DA Library 34

3.1.5 MQTT Library 36

3.2 Front-End Implementation 38

3.2.1 Angular 38

3.2.2 Communication with the Back-end 38

3.3 Database 40

3.3.1 PostgreSQL 40
3.3.2 PostgreSQL JDBC Driver 40
4. Evaluation 43
4.1 Configuration Screen 43
4.2 Overview Screen 48
4.3 Entity Relationships 54
5. Testing 56
5.1 Results 58
6. Conclusion 61
References 62

List of Abbreviations

IIoT	Industrial Internet of Things
PLC	Programmable Logic Controller
OPC	Open Platform Communications
MQTT	Message Queuing Telemetry Transport
QoS	Quality of Service
N.m	Newton-meter
rpm	Rotations per minute
API	Application Programming Interface
MVC	Model View Controller
HTTP	Hypertext Transfer Protocol
HTML	Hypertext Markup Language
CSS	Cascading Style Sheets
SQL	Structured Query Language
RDBMS	Relational Database Management System
JPA	Java Persistence API
JVM	Java Virtual Machine
JSON	JavaScript Object Notation
JDBC	Java Database Connectivity
CRUD	Create, Read, Update and Delete

1. Introduction

Industrial Internet of Things (also known as Industrial IoT or IIoT) refers to an industrial framework whereby a large number of devices or machines are connected and synchronized through the use of software tools and third platform technologies in a machine-to-machine and Internet of Things context. The idea behind machine-to-machine communication is to reduce human interventions as much as possible so that the highest level of automation could be achieved. [1]

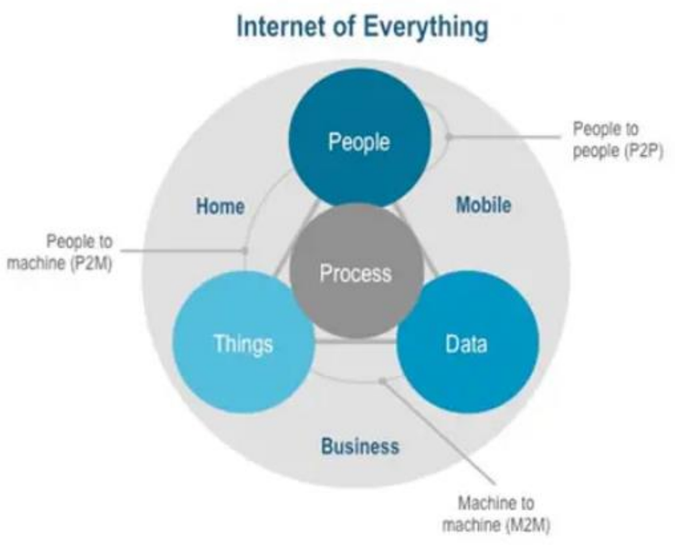


Figure 1 The place of machine-to-machine or M2M on the Internet of Everything [1]

One of the most significant advantages of the Industrial Internet of Things is the elimination of human errors and manual labor, as well as an improvement in overall productivity and cost savings, both in terms of time and money. We must not overlook the potential applications of IIoT in quality control and maintenance. In the future, IIoT is likely to force more unified system protocols and architectures, allowing devices to interact more seamlessly and improving interoperability.

1.1 Load Curve

The Load Curve is a web application designed for monitoring the characteristic of motors which are controlled by a SIMOTION D PLC in a factory. The application will be able to connect to the OPC (Open Platform Communications) server of the PLC and retrieve information about connected motors. After saving the configuration, we will be able to receive data about the motor describing its current velocity, acceleration, position and torque through a communication protocol known as MQTT (Message Queuing Telemetry Transport) that is publishing this data from another web application known as SIMOTION Trace Connector. After processing this data, the application is expected to display four separate charts. The first three charts will represent the velocity, acceleration and torque as functions of time, while the final chart will contain the load profile of the motor, the load point, as well as the torque-speed characteristic that is documented in the catalogue of the motor. The application will be provided to customers as part of the Siemens Industrial Edge Computing Platform.

1.1.1 Industrial Edge Platform

Industrial Edge is the SIEMENS platform to host applications from different vendors on a computing platform close to the shopfloor. Thus, enables the extensions of automation, deployment of demanding stream processing and learning algorithms as well as the hosting from integration code to site automation. [2]

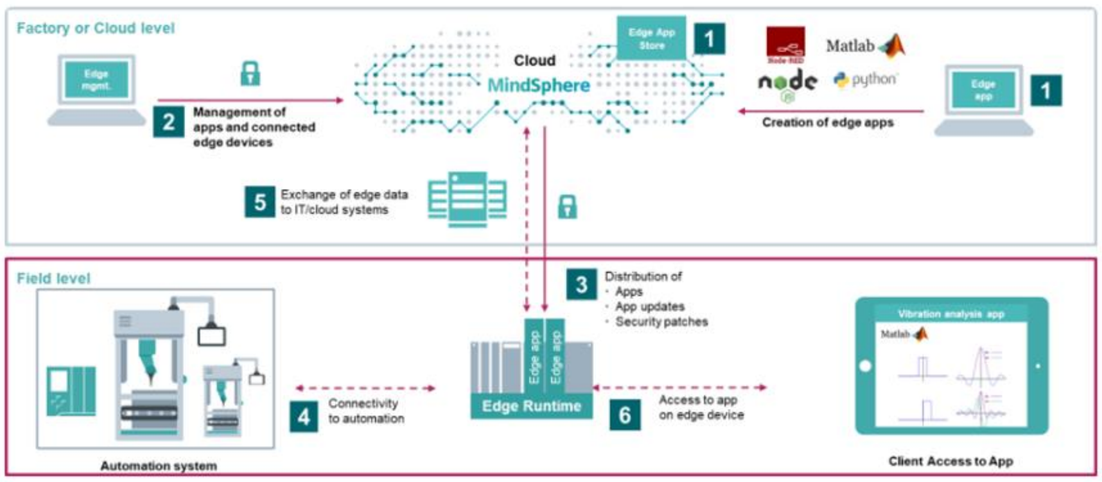


Figure 2 Core Functionalities of Industrial Edge [2]

Industrial Edge addresses the following core functionalities:

a) Write applications in various programming languages, participating from the whole power of the docker community

- Simple creation of applications
- Publishing of applications to different tenants

b) Comprehensive Edge Device Management

   - Simple onboarding of Edge devices operated in industrial networks
   - Secured firmware management with power fail-safe update over the air

c) Integrated Application Management for deployment, configuration, and lifecycle management

d) Publish Web Applications on Site

   - Reverse proxy for controlled publishing from Web Apps on site
   - Integrated user management for on-site access control

This Edge platform combines all the benefits of edge and cloud computing – optimally tailored to the customer's specific requirements. It allows analysis of all the data at the machine or preprocess it quickly and instantly. The optimized data points can then be transferred more quickly to the cloud where, for example, access to more computing power and larger storage capacities is available. Among other things, this permits a precise analysis of data over longer periods of time. Based on the type of application, the customer decides whether and how he would like to use the cloud in addition to Industrial Edge.

1.1.2 SIMOTION D PLC

SIMOTION is a motion control system which offers an optimized all-in-one solution for production machines as well as being able to handle all necessary tasks for the entire machine automation. In this thesis, the hardware used will be SIMOTION D which is a drive-based platform for users who require a drive integrated and a very compact solution for their machines.

SIMOTION D offers all features of a regular Programmable Logic Controller (PLC), in addition to speed and position control, pressure control, temperature control and synchronous operations for electrical and hydraulic axes.

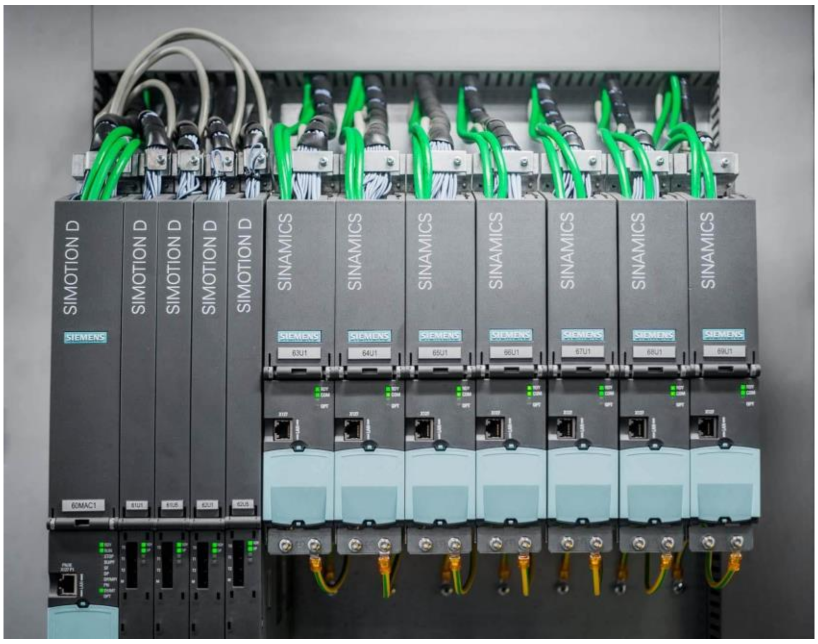


Figure 3 SIMOTION D in a factory setup [3]

SIMOTION D is recommended wherever there is a need to implement complex applications in a small space. As the controller and drive are combined in a single system, SIMOTION D takes up very little space in the control cabinet. It provides an ideal solution for complex multi-axis machines requiring shortest cycle times as well as optimal product quality through a deterministic and reproducible machine behavior. Moreover, multi-axis machines whose modules are each controlled by a SIMOTION controller can mutually exchange process values in isochronous mode (distributed gearing), thus achieving distributed automation solutions.

1.1.3 OPC Communication

Open Platform Communications (OPC) is a series of standards and specifications for industrial telecommunication. These specifications define the interface between Clients and Servers, as well as Servers and Servers, and include features such as access to real-time data, alarm and event tracking, historical data access, and other applications.

The original standard, known as OLE for Process Control, was created in 1996 by an industrial automation task force. The aim of the standard when it was first released was to abstract PLC specific protocols (such as Modbus, Profibus, and others) into a standardized interface that would enable HMI/SCADA systems to communicate with a "middle-man" who would translate generic-OPC read/write requests into device-specific requests and vice versa. As a result, an entire industry of products has emerged, enabling end-users to introduce systems using best-of-breed products that all communicate seamlessly through OPC. [4]

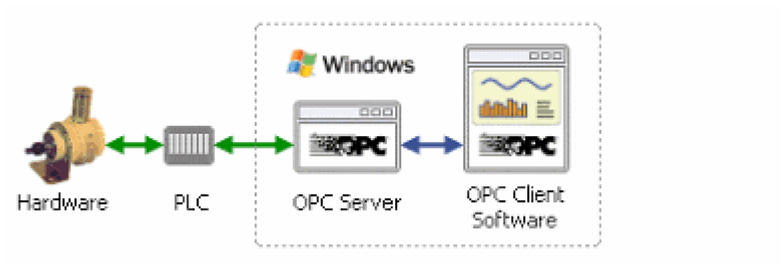


Figure 4 OPC-Hardware relationship [5]

OPC's benefit comes from the fact that it's an open standard, which means lower prices for manufacturers and more choices for consumers. For their devices to connect with any OPC client, hardware manufacturers just need to have a single OPC server. Through including OPC client capabilities in their products, software vendors can immediately make their products compliant with thousands of hardware devices. Users can choose any OPC client software they need, being assured that they will communicate seamlessly with their OPC-enabled hardware.

1.1.4 MQTT

MQTT is a lightweight publish/subscribe protocol that connects IoT devices with a small footprint and low bandwidth. MQTT is an event-driven protocol that allows messages to be pushed to clients, unlike HTTP's request/response model. This architecture separates clients from one another, allowing for a highly scalable solution without relying on data producers and data consumers. [6]

The MQTT broker and MQTT clients are at the heart of MQTT. The broker is responsible for sending messages between the sender and the receivers. The client publishes a message to the broker, and other clients can subscribe to the broker to receive messages. Each MQTT message includes a topic. A client publishes a message for a specific topic and MQTT clients subscribe to the topics they want to receive. The broker uses the topics and the list of subscribers to send messages to the appropriate clients. Messages that can't be dispatched to MQTT clients who aren't connected, can be buffered by a MQTT broker. When network links are unreliable, this becomes extremely useful. To support the reliable delivery of messages, the protocol supports three different types of quality of service

(QoS) messages: 0-at most once, 1-at least once, and 2-exactly once. QoS is a key feature of the protocol MQTT.

QoS gives the customer the power to choose a level of service that matches their network reliability and application logic.

- At most once - the message is sent only once, and the client and broker take no additional steps to acknowledge delivery (fire and forget).
- At least once - the message is re-tried by the sender multiple times until acknowledgement is received (acknowledged delivery).
- Exactly once - the sender and receiver engage in a two-level handshake to ensure only one copy of the message is received (assured delivery).

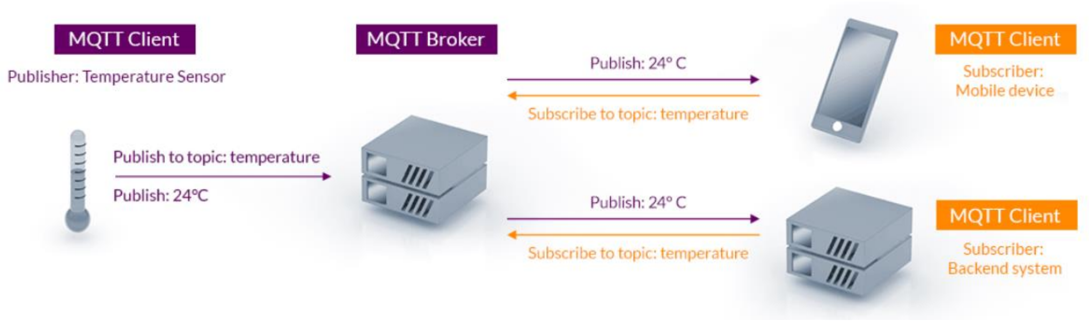

MQTT Client
Publisher: Temperature Sensor
MQTT Broker
Publish to topic: temperature
Publish: 24°C
Publish: 24° C
Subscribe to topic: temperature
MQTT Client
Subscriber: Mobile device
Publish: 24° C
Subscribe to topic: temperature
MQTT Client
Subscriber: Backend system

Figure 5 MQTT Publish / Subscribe Architecture example [6]

1.1.5 Load Profile

The load profile is used for the purpose of motor sizing. Proper sizing is a crucial aspect of motor selection. If a motor is undersized, it will not be able to control the load, leading to overshoot and ringing. If the motor is oversized, it may control the load, but it will also be larger and heavier, as well as more expensive in terms of price and cost of operations. [7]

Every motor will have rated values of voltage, current, speed and power. Normally these are visible on the motor's nameplate and/or given in the documentation. In general, rated values represent the maximum values that the motor should be subjected to in normal conditions. However, the rated torque is often not given on the nameplate but is a very important parameter for appropriately sizing the motor. In both DC and AC induction motors, operating current is proportional to the torque, so exceeding the rated torque is likely to lead to overheating and burnout of the motor windings. Exceeding the rated torque also risks mechanical damage to couplings and the drive shaft. Simply, if the load is constant then sizing the motor consists of choosing a motor whose rated torque is slightly above the torque required by the load. The torque produced by a motor varies with speed and the torque produced by a load also varies with speed. If the motor torque is greater than the load torque, then the load will accelerate. If the load torque is greater than the motor torque, then the load will decelerate.

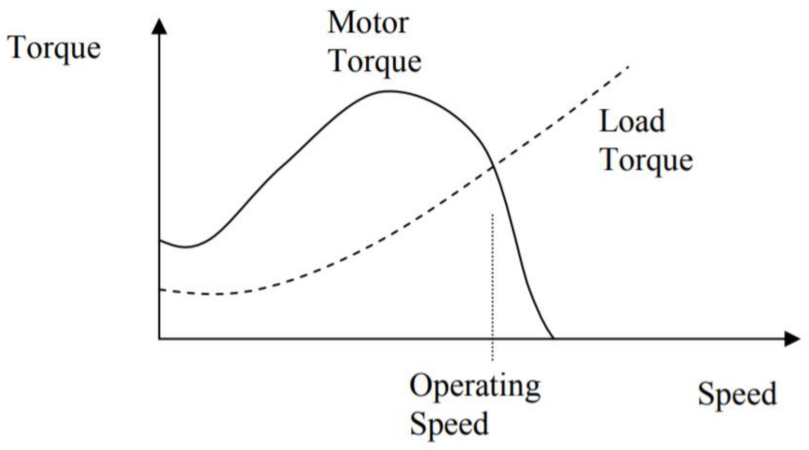


Figure 6 A motor that will start the load and get up to speed correctly

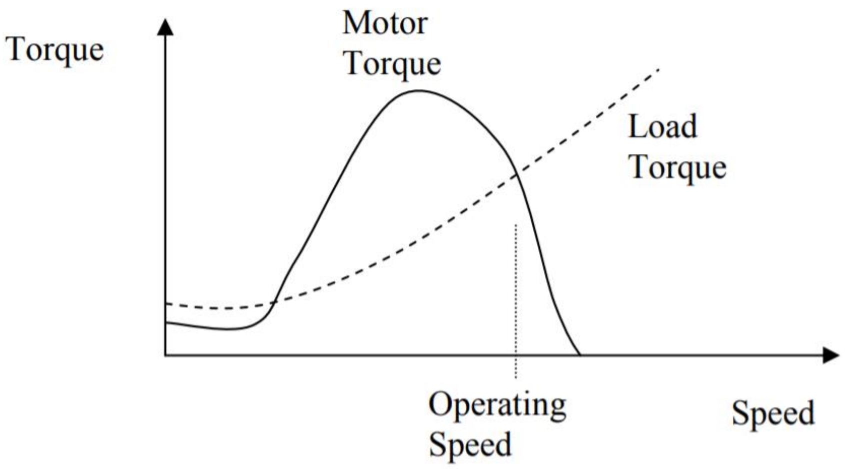


Figure 7 A motor that will never start

Some loads do not present a constant torque even after they have got up to full speed. This presents a variable power to the motor and complicates the sizing problem. In this case we should ensure that:

- Peak load torque < Rated motor torque
- The effective load torque (root mean square load torque) and effective load speed requirements must be less than 100% of the rated motor torque and speed and ideally greater than 75%.
- The motor can start the load and get it up to speed from the initial position.

The point at which the effective torque and effective speed meet is called the load point or operating point. The coordinates of this point can be calculated by the following formulas:

$$M_{eff} = \sqrt{\frac{1}{T} \cdot \sum_{i=1}^{n} M_i^2 \cdot t_i} \qquad (1)$$

$$n_{eff} = \sqrt{\frac{1}{T} \cdot \sum_{i=1}^{n} n_i^2 \cdot t_i} \qquad (2)$$

$T \ldots$ *cycle time of the measurment*

$t \ldots$ *sample time*

$n \ldots$ *number of samples*

If the motor were to be operated at this operating point, the same temperature rise would occur as in the actual load cycle. This point is now entered in its torque-speed diagram for each motor that is considered in the sizing process. If the operating point is below the characteristic curve for S1 operation, the corresponding motor is able to meet the required load cycle from a thermal point of view. All motors that meet this condition are retained for further design steps while all others are eliminated. In figure 8, we can observe that motor 1 is suitable for the load cycle while motor 2 would be thermally overloaded and therefore cannot be used.

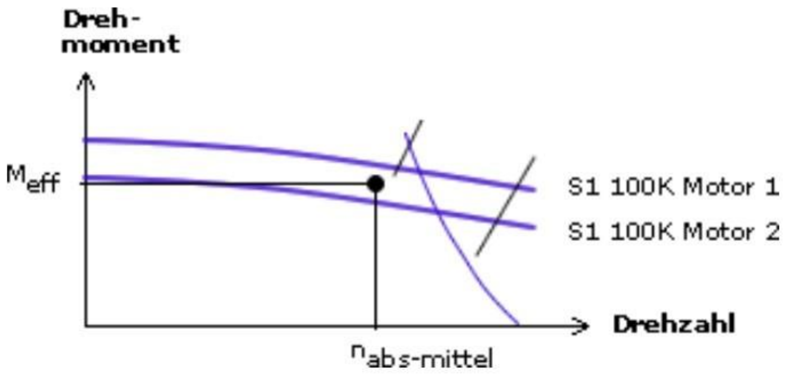


Figure 8 Characteristic of Motor 1 and Motor 2 [8]

To get a better idea of the required motor for a specific application, then the load torque and speed will be compared to the torque characteristic of the motor. This is done by measuring the torque and speed of the motor during operation and then converting these measurements to their absolute values. We call this the load curve, and it represents the possible effective torque vs the possible effective speed.

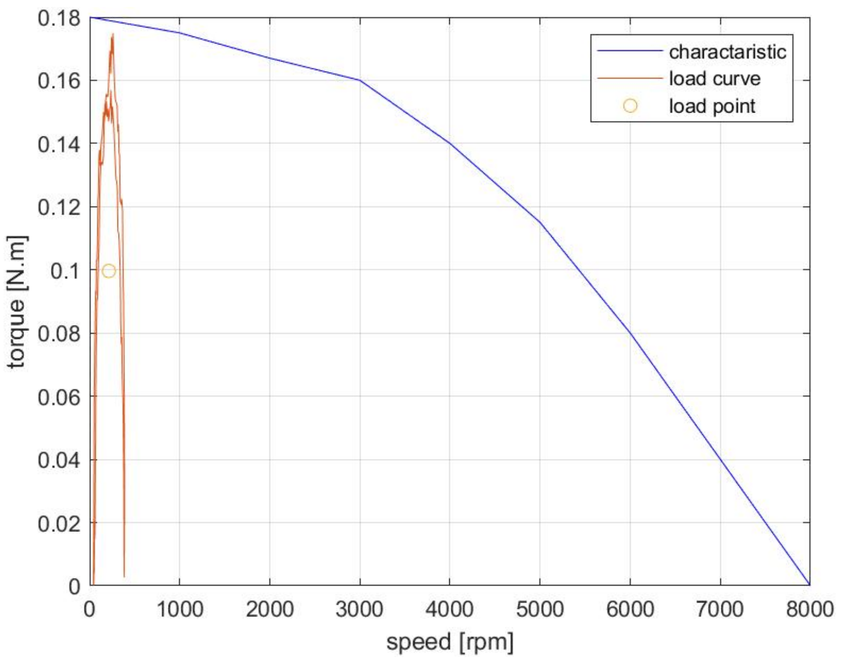


Figure 9 An example of a load curve below the characteristic

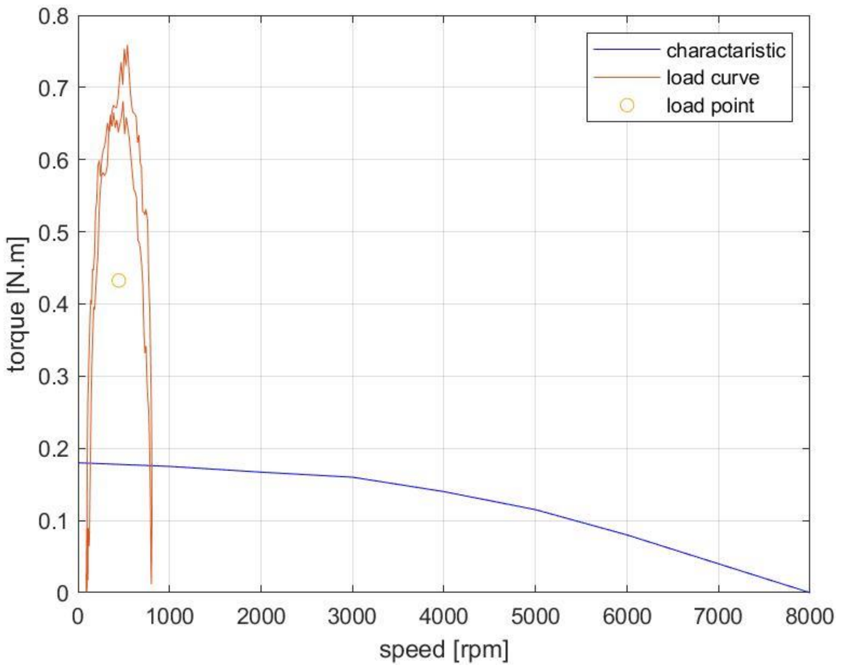


Figure 10 An example of a load curve above the characteristic

In figure 9, we can observe that the load curve or load profile is directly lower than the characteristic, this marks this motor as suitable for operation since the load/operating point falls below the characteristic as well. While in figure 10, the chosen motor should not be used at all and more sizing measurements have to be performed to find a better suited one for the required application.

2. Research and Requirements

2.1 Architectural Overview

This chapter describes an architectural overview of the application. Based on the requirements of the software, the development of the web application has been considered. The architecture consists of three main components: the back-end and front-end systems communicating with an API controller, and the database. The web application will be built as a "3-tier architecture" which comprises from a data tier, business tier and a presentation tier.

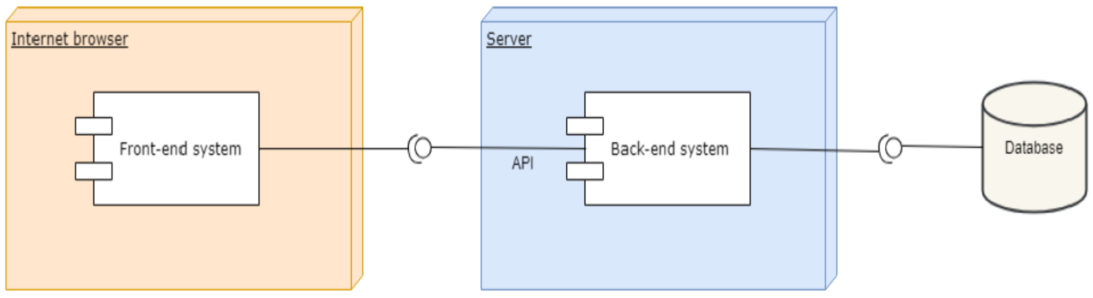


Figure 11 An overview of the proposed architecture

2.1.1 The Back-end

The server side of a website is referred to as the backend. It organizes and stores data, as well as ensuring that everything on the client side of the website functions properly. It's the section of the website you can't see or communicate with. It's the part of the app that doesn't interact with users directly. The parts and characteristics developed by backend designers are indirectly accessed by users through a front-end application. The backend includes activities such as writing APIs, developing libraries, and dealing with machine components without user interfaces or even scientific programming systems.

The back-end uses the Model-View-Controller (MVC) architectural pattern. The MVC pattern in Software Engineering Architecture is defined as an application being separated into three logical components: Model, View and Controller.

- Model:

  All data-related logic will be represented by this part in the architecture. This involves determining the data's structure. To put it another way, this contains the definitions for many of the types used in the application. In many cases, the model here refers to the type of data that we are dealing with. This component also notifies its dependents about data changes.

- View:

  Contains the application's user interface (UI) logic. This section of the framework encapsulates the user interface logic, which involves items like dropdown buttons and web pages that the end user can manipulate.

- Controller:

  Controllers serve as a layer between the Model and View components, processing all business logic generated by user input. It's in charge of handling inputs from the View components, manipulating data with the Model component's models, and eventually interacting with the View components to make the final output to the end user.

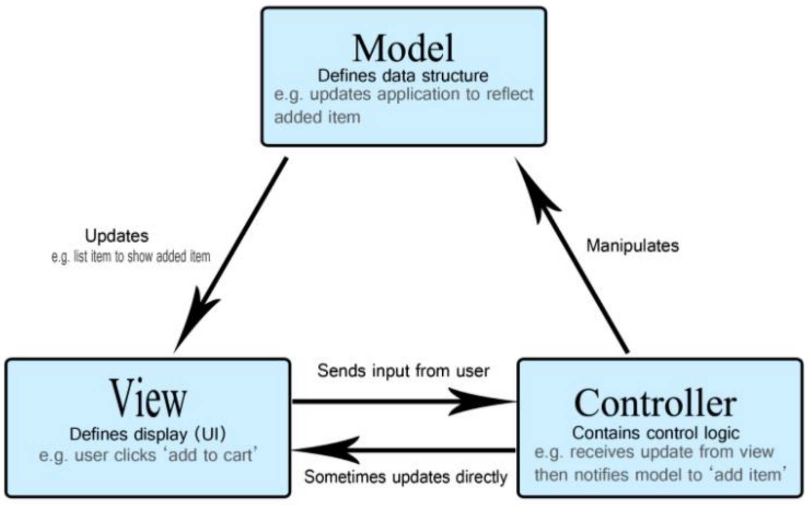

Model
Defines data structure
e.g. updates application to reflect added item
Updates
e.g. list item to show added item
Manipulates
View
Defines display (UI)
e.g. user clicks 'add to cart'
Sends input from user
Sometimes updates directly
Controller
Contains control logic
e.g. receives update from view then notifies model to 'add item'

Figure 12 MVC Architecture [9]

The MVC pattern's benefit is that it allows for a separation of concerns and code. Since the model is independent of the view, any changes to the model have no effect on the overall architecture. It's also simple to test each of these components, and they're reusable. It's possible to make multiple views for a single model. In addition, working in a development team helps you to work on various parts of the web app at the same time without interfering with each other. The key drawback of the MVC architectural pattern lies in its complexity. It requires an understanding of the information flow between the various components.

2.1.2 API Controller

To handle the communication between back-end and front-end, an API is provided. An application programming interface (API) is a collection of concepts and protocols for creating and integrating software applications. It's often referred to as a contract between an information provider and an information user, specifying the content that the client (the call) is required to provide and the content that the producer is required to provide (the response). In other words, if you want to connect with a program or device to access information or perform a function, an API allows you to communicate your request to the system so that it can understand and react. [10]

The API controller is used to execute back-end requests. One method is provided for each available endpoint. This controller builds an HTTP request based on the arguments of the method. HTTP stands for Hypertext Transfer Protocol and is used to structure requests and responses over the internet. HTTP requires data to be transferred from one point to another over the network. The internet contains a large number of resources that are hosted on various servers. Your browser must be able to send a request to the servers and view the resources for you in order to access these resources. HTTP specifies a collection of request methods for specifying the desired action for a given resource. Although they can also be nouns, these request methods are sometimes referred to as HTTP verbs. [11]

- GET: The GET method requests a representation of the specified resource. Requests using GET should only retrieve data.
- POST: The POST method is used to submit an entity to the specified resource, often causing a change in state or side effects on the server.
- PUT: The PUT method replaces all current representations of the target resource with the request payload.
- DELETE: The DELETE method deletes the specified resource.
- PATCH: The PATCH method is used to apply partial modifications to a resource.

2.1.3 The Front-end

The front end of a website is the aspect where the user communicates with directly. It is often referred to as the application's "client side." Anything that users see explicitly is included: text colors and styles, images, graphs and tables, buttons, colors, and the navigation menu. The languages used for Front End development are HTML, CSS, and JavaScript. Front End developers create the structure, design, behavior, and content of anything that appears on the browser screen when websites, web applications, or mobile apps are opened. The front end's main goals are responsiveness and results. [12]

- HTML:

  HTML stands for Hyper Text Markup Language. It is used to design the front end portion of web pages using markup language. HTML is the combination of Hypertext and Markup language. Hypertext defines the link between the web pages. The markup language is used to define the text documentation within tag which defines the structure of web pages.

- CSS:

  Cascading Style Sheets fondly referred to as CSS is a simply designed language intended to simplify the process of making web pages presentable. CSS allows you to apply styles to web pages. More importantly, CSS enables you to do this independent of the HTML that makes up each web page.

- JavaScript:

  JavaScript is a famous scripting language used to create the magic on the sites to make the site interactive for the user. It is used to enhancing the functionality of a website to running cool games and web-based software.

The front-end runs in the users' browser and helps them to communicate with the software in a user-friendly manner. Even if not directly, it uses an MVC-like pattern. Separation of concerns (SoC) is used in the front-end architecture. It is a concept that allows various logic, such as user interface functionality, business logic, and infrastructure logic, to be separated. The MVC architectural pattern is also included in the SoC.

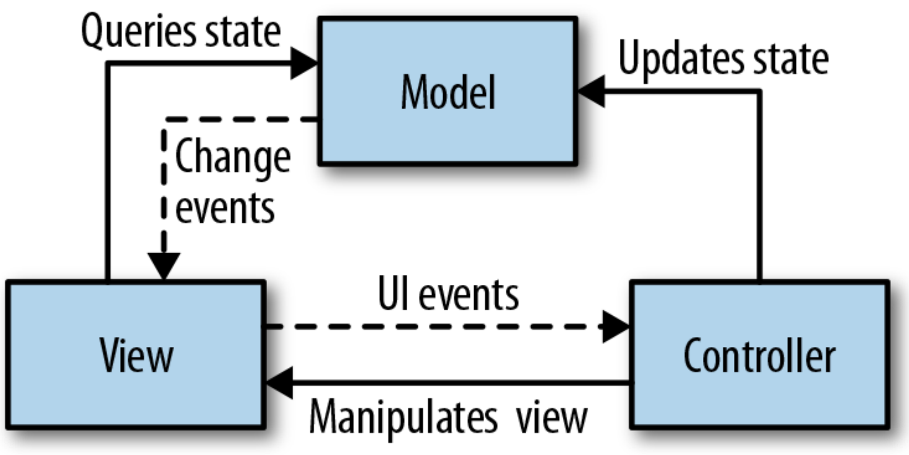


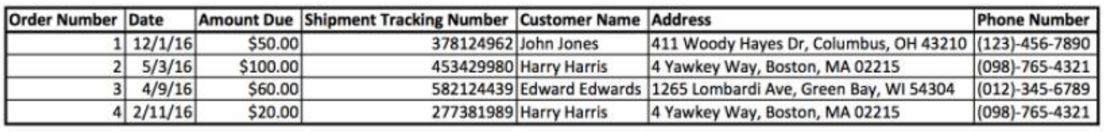
Figure 13 Separation of Concerns Architecture [13]

2.1.4 Database

SQL stands for Structured Query Language. A query language is a kind of programming language that's designed to facilitate retrieving specific information from databases. To put it in other words, SQL is the language of databases. A relational (SQL) database is a database that stores related information across multiple tables and allows you to query information in more than one table at the same time. The computing world quickly realized the value of being able to access several records with a single command that does not require specifying how to get to a specific record.

The relational database management system (RDBMS) that we used today rely on SQL as the engine that allows us to perform all the operations required to create, retrieve, update, and delete data as needed. From an open-source perspective these RDBMSs include MySQL, MariaDB, and PostgreSQL as the most commonly used open-source RDBMS in production today. In this project the used database will be PostgreSQL. PostgreSQL is an object-relational database management system (ORDBMS), instead of a purely RDBMS system like MySQL and MariaDB. This means that PostgreSQL data models can be based on relational database models but can also be object-oriented as well. In practice, that means we see PostgreSQL utilized in more complex and varied data models, while we see MariaDB and MySQL used for more lightweight data models. [14]

It's easier to grasp how a relational database works if we consider an example. Assume a business owner who wants to keep track of sales data. He could create an Excel spreadsheet with all the details he needs in separate columns, such as the order number, date, amount due, shipment tracking number, customer name, address, and phone number.



Order Number	Date	Amount Due	Shipment Tracking Number	Customer Name	Address	Phone Number
1	12/1/16	$50.00	378124962	John Jones	411 Woody Hayes Dr, Columbus, OH 43210	(123)-456-7890
2	5/3/16	$100.00	453429980	Harry Harris	4 Yawkey Way, Boston, MA 02215	(098)-765-4321
3	4/9/16	$60.00	582124439	Edward Edwards	1265 Lombardi Ave, Green Bay, WI 54304	(012)-345-6789
4	2/11/16	$20.00	277381989	Harry Harris	4 Yawkey Way, Boston, MA 02215	(098)-765-4321

Figure 14 Example of an Excel Spreadsheet

This setup would be adequate for monitoring the data he needs at first, but when he begins to receive repeated orders from the same customer, he'll notice that their name, address, and phone number are stored in several rows of the spreadsheet. This redundant data will take up unwanted space and reduce the reliability of the sales tracking system as the company expands and the amount of orders being tracked increases. The company may also have problems with data integrity. There's no guarantee that every field will be filled with the correct data form or that the name and address will be entered consistently.

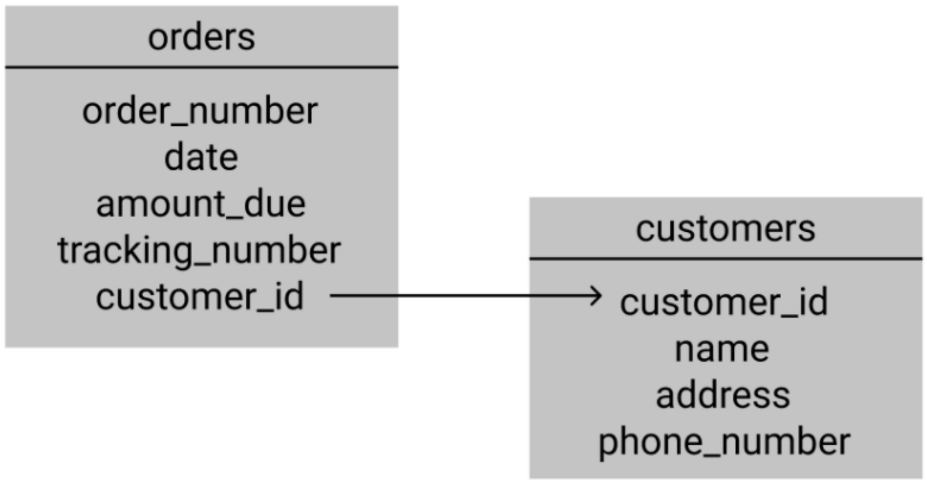


Figure 15 A diagram of how the Relational Database would look like [15]

We can prevent any of these problems by using a relational database, such as the one seen in the diagram above. Two tables may be set up, one for orders and the other for customers. Each customer will have a unique ID number, as well as the name, address, and phone number already being monitored, in the 'customers' table. The 'orders' table will include the order number, date, sum due, and tracking number, as well as a column for the customer ID, rather than a separate field for each piece of customer data. This allows us to retrieve all of the customer information for any given order, but we just have to store it once in our database rather than listing it for each order.

Here's a list of commonly used SQL commands:

- SELECT - extracts data from a database
- UPDATE - updates data in a database
- DELETE - deletes data from a database
- INSERT INTO - inserts new data into a database
- CREATE DATABASE - creates a new database
- ALTER DATABASE - modifies a database
- CREATE TABLE - creates a new table
- ALTER TABLE - modifies a table
- DROP TABLE - deletes a table
- CREATE INDEX - creates an index (search key)
- DROP INDEX - deletes an index

Note: For an expanded view into SQL and its commands, refer to Chris Fehily's "SQL (Database Programming) 2014" from Questing Vole Press. [16]

3. Implementation

This chapter describes the implementation details concerning the architectural components given, as well as the technology choice regarding each component. An overall technology choice is realized with respect to the application scalability and a potential increase of the requirements. The three main architectural components are the back-end, the front-end and the database.

3.1 Back-End Implementation

In this section we will discuss the technologies used to develop the back-end of the web application. These technologies include Java programming language, Spring Boot Framework, Spring Data JPA, OPC XML-DA library by OPC Foundation, and the Eclipse Paho MQTT library. Finally, a small example of the build implementation of the back-end will be demonstrated.

3.1.1 Java

Java is a class-based, object-oriented programming language that is designed to have as few implementation dependencies as possible. It is a general-purpose programming language intended to let application developers write once, run anywhere (WORA), meaning that compiled Java code can run on all platforms that support Java without the need for recompilation. Java applications are typically compiled to bytecode that can run on any Java Virtual Machine (JVM) regardless of the underlying computer architecture. The use of compiled byte-code allows the interpreter (the virtual machine) to be small and efficient (and nearly as fast as the CPU running the compiled code).

Java is extremely portable. The same Java application will run identically on any computer, regardless of hardware features or operating system, as long as it has a Java interpreter. Besides portability, another of Java's key advantages is its set of security features which protect a PC running a Java program from malicious programs (such as viruses). You can safely run a Java applet downloaded from the Internet, because Java's security features prevent these types of applets from accessing a PC's hard drive or network connections. Most Web browsers contain a JVM to run Java applets.

There are three main components of Java – Java Virtual Machine (JVM), Java Development Kit (JDK), and the Java Runtime Environment (JRE). JDK or Java Development Kit is where the developers write their code and run it through the JRE or Java Runtime Environment. Then the code is translated through the JVM which resides inside the JRE along with the java packages or libraries. [17]

In this thesis, Java is chosen for building this application due to its strong support for web development. A Java web application is a collection of dynamic resources (such as Servlets, JavaServer Pages, Java classes and jars) and static resources (such as HTML pages and pictures). A Java web application can be deployed as a WAR (Web ARchive) file. A WAR file is a zip file which contains the complete content of the corresponding web application. A servlet is a Java class which answers a HTTP request within a web container. JavaServer Pages (JSP) are files which contain HTML and Java code. The web container compiles the JSP into a servlet at the first time the JSP is accessed.

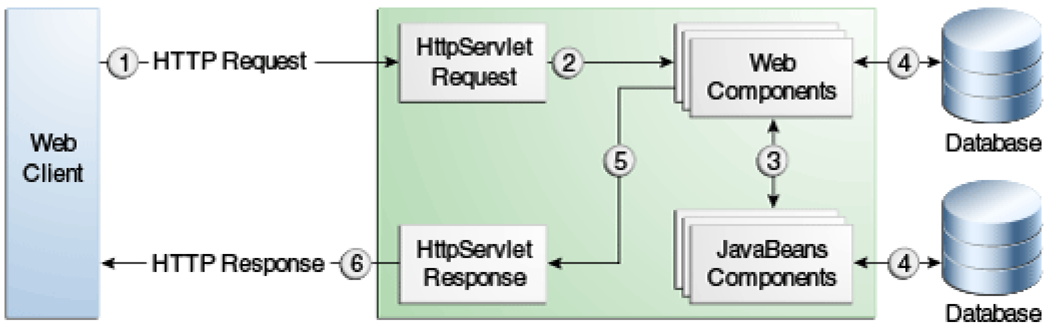


Figure 16 Java Web Application Request Handling [18]

3.1.2 Spring Boot

The Spring Web MVC framework provides Model-View-Controller (MVC) architecture and ready components that can be used to build scalable and flexible web applications. The MVC pattern distinguishes the various aspects of an application (input logic, business logic, and user interface logic) while maintaining a good scalability between them. A DispatcherServlet, which manages all HTTP requests and responses, is at the heart of the Spring framework.

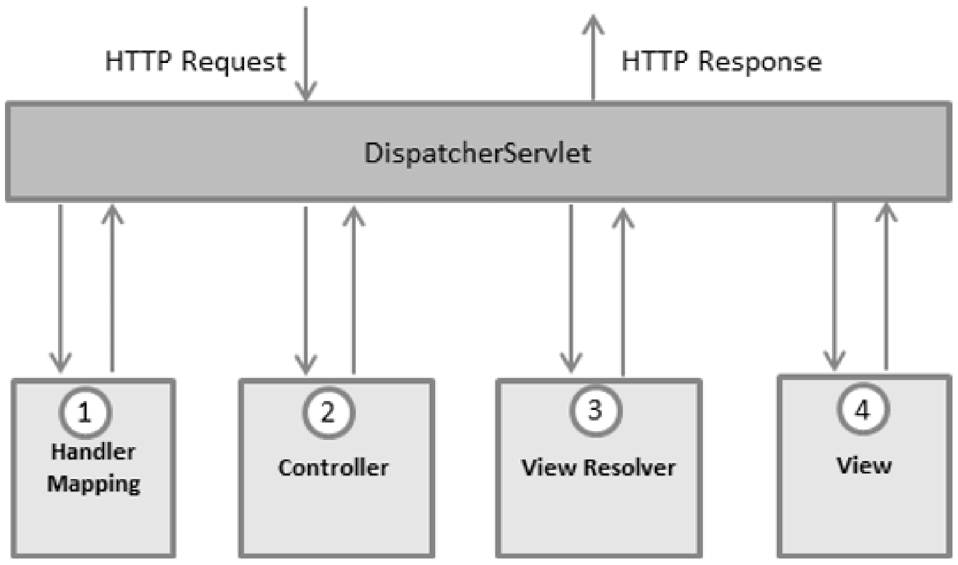
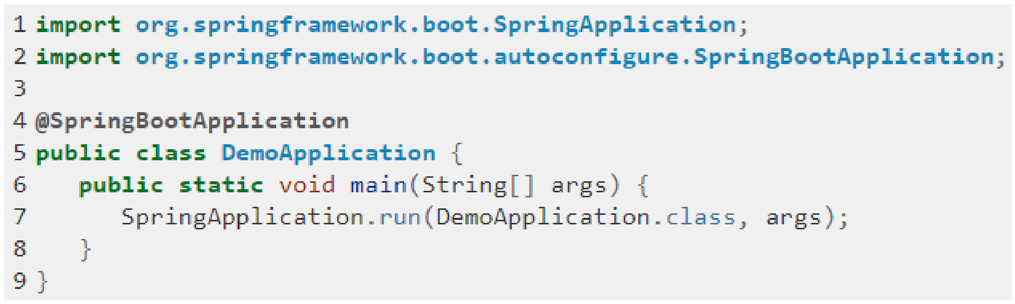


Figure 17 The request processing workflow of the Spring Web MVC DispatcherServlet [19]

After receiving an HTTP request, DispatcherServlet consults the HandlerMapping to call the appropriate Controller. The Controller takes the request and calls the appropriate service methods based on used GET or POST method. The service method will set model data based on defined business logic and returns view name to the DispatcherServlet. The DispatcherServlet will take help from ViewResolver to pick up the defined view for the request. Once view is finalized, The DispatcherServlet passes the model data to the view which is finally rendered on the browser. All the above-mentioned components, such as HandlerMapping, Controller, and ViewResolver are parts of WebApplicationContext.

Spring Boot is an open source, microservice-based Java web framework. It is built on top of the Spring framework, and it comes with many dependencies that can be integrated into the Spring application. So basically, it's an extension of the Spring Framework, but it has some specific features that make the application easier for working within the developer ecosystem. That extension includes pre-configurable web starter kits that help facilitate the responsibilities of an application server that are required for other Spring projects. The Spring Boot auto configuration comes down to three simple annotations: **@SpringBootApplication**, **@EnableAutoConfiguration**, and **@ComponentScan**.

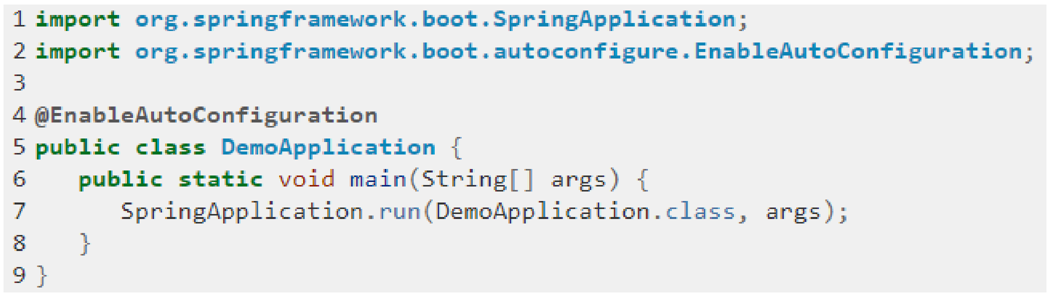
@SpringBootApplication is used in the application's entry point, and the class it resides in must have access to the application's main method. The annotation is needed and will provide each of the other two annotations in the application since the **@SpringBootApplication** includes both inside.



```java
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class DemoApplication {
   public static void main(String[] args) {
      SpringApplication.run(DemoApplication.class, args);
   }
}
```

Figure 18 @SpringBootApplication example

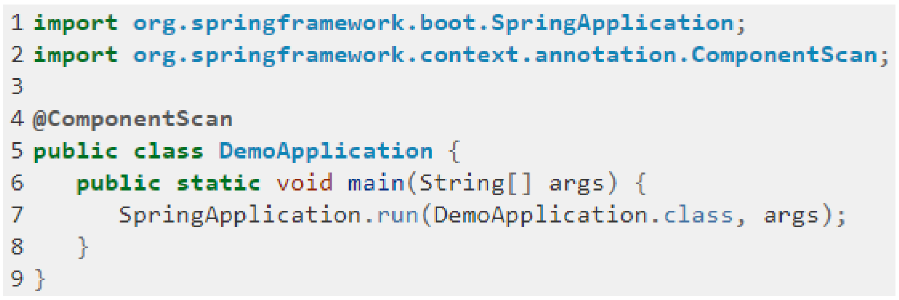
@EnableAutoConfiguration enables Automatic Configuration for each of the representing classes. It automatically configures your Spring application based on the JAR dependencies you added in the project.



```java
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.EnableAutoConfiguration;

@EnableAutoConfiguration
public class DemoApplication {
   public static void main(String[] args) {
      SpringApplication.run(DemoApplication.class, args);
   }
}
```

Figure 19 @EnableAutoConfiguration example

Finally, the **@ComponentScan** will search all of the beans and package declarations at startup.



```java
import org.springframework.boot.SpringApplication;
import org.springframework.context.annotation.ComponentScan;

@ComponentScan
public class DemoApplication {
   public static void main(String[] args) {
      SpringApplication.run(DemoApplication.class, args);
   }
}
```

Figure 20 @ComponentScan example

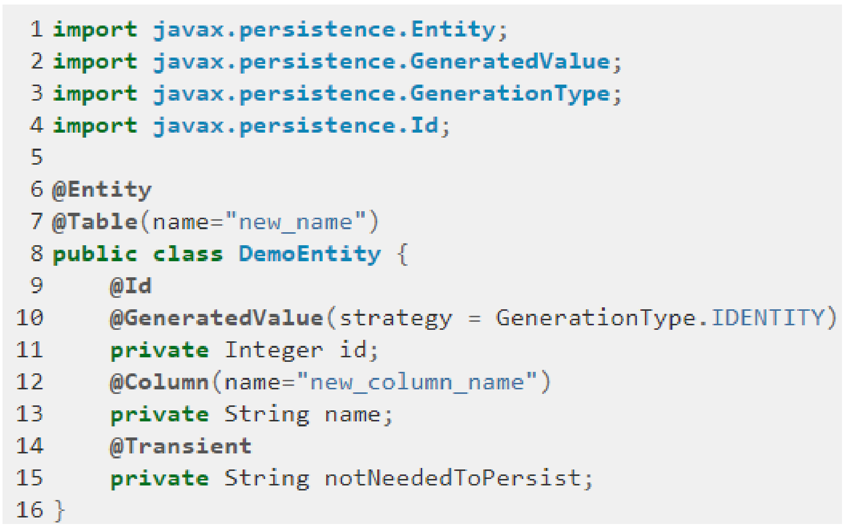
3.1.3 Spring Data JPA

Spring Data JPA provides repository support for the Java Persistence API (JPA). It eases development of applications that need to access JPA data sources. Mapping Java objects to database tables and vice versa is called Object-relational mapping (ORM). The Java

Persistence API (JPA) is one possible approach to ORM. With JPA developers can map, store, update and retrieve data from relational databases to Java objects and vice versa.

A class which should be persisted in a database must be annotated with **@Entity**. Thus, the class will be called an Entity. JPA uses a database table for every entity. Persisted instances of the class will be represented as one row in the table. All entity classes must define a primary key and have a non-arg constructor. JPA allows to auto-generate the primary key in the database via the **@GeneratedValue** annotation. The table name corresponds to the class name. It can be changed with the annotation **@Table(name="NEWTABLENAME")**.

The fields of the Entity are the saved in the database. JPA is able to use either the instance variables (fields) or the corresponding getters and setters to access these fields. It will persist all fields by default, however if some field is not required to be saved then the **@Transient** annotation can be used above the field. Each field will be mapped to a column with the name of the field and can be changed by using the annotation **@Column (name="newColumnName")**.

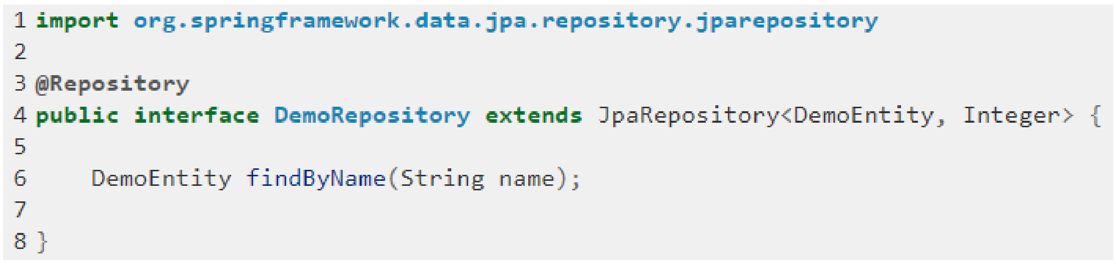


```java
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;

@Entity
@Table(name="new_name")
public class DemoEntity {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Integer id;
    @Column(name="new_column_name")
    private String name;
    @Transient
    private String notNeededToPersist;
}
```

Figure 21 An example of an Entity with JPA

JPA allows to define relationships between classes. These relationships are set using the following annotations: **@OneToOne**, **@OneToMany**, **@ManyToOne**, **@ManyToMany**. A relationship can be bidirectional or unidirectional. In a bidirectional relationship both classes store a reference to each other while in a unidirectional case only one class has a reference to the other class. Within a bidirectional relationship we need to specify the owning side of this relationship in the other class with the attribute "mappedBy", for example: **@ManyToMany(mappedBy="attributeOfTheOwningClass")**.

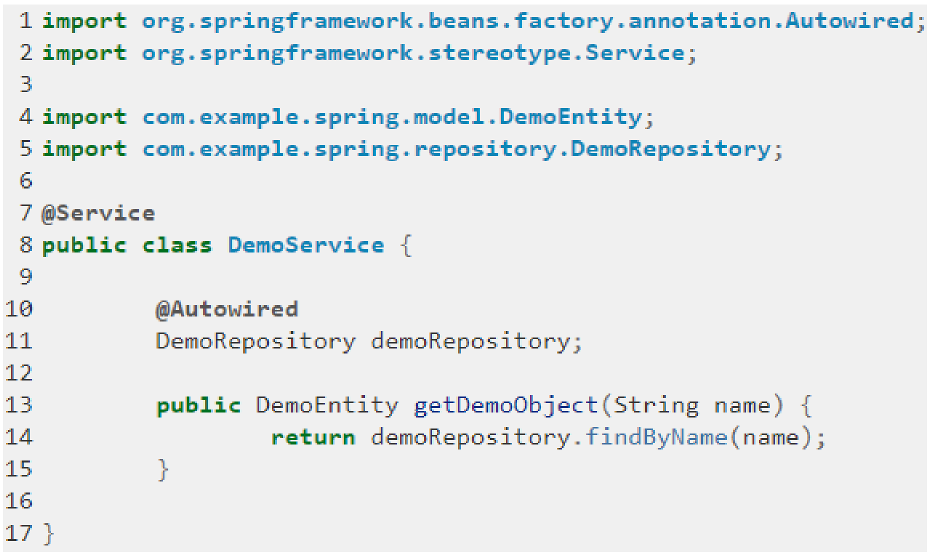
Spring Data JPA provides repository support for the JPA. It simplifies the development of applications that need access to JPA data sources. In addition, Spring Data JPA reduces the amount of boilerplate code required by JPA. That makes the implementation of the persistence layer easier and faster. When Spring Data generates a new Repository implementation, it examines all of the methods specified by the interfaces and attempts to generate queries automatically based on the names of the methods. For generating the repository, we need to add the annotation **@Respository**. [20]



```
import org.springframework.data.jpa.repository.jparepository

@Repository
public interface DemoRepository extends JpaRepository<DemoEntity, Integer> {

    DemoEntity findByName(String name);

}
```

Figure 22 An example of a Repository with Spring Data JPA

After the Spring Data JPA code is ready, next step is to create service class and define methods that we will have to work with database table. We will need to inject the repository class with annotation **@Autowired**.

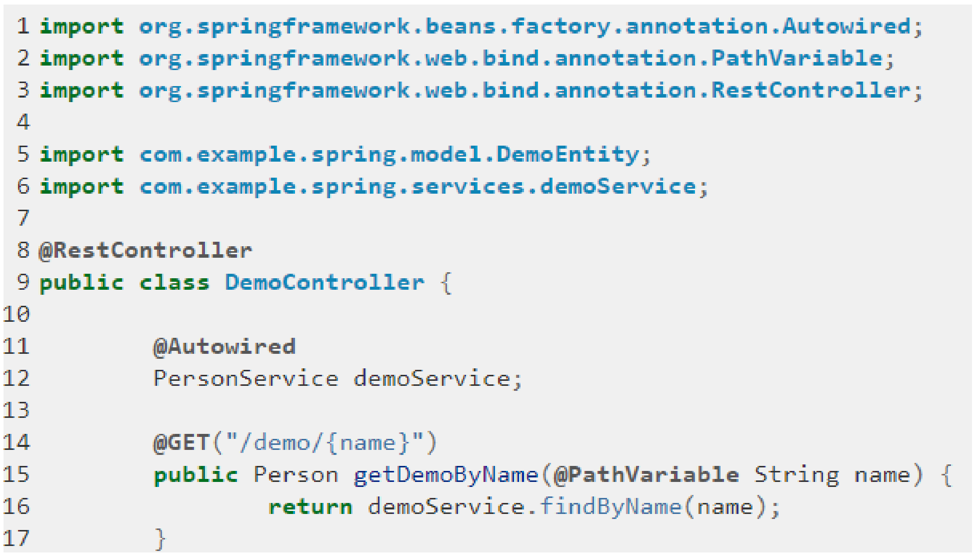


```
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import com.example.spring.model.DemoEntity;
import com.example.spring.repository.DemoRepository;

@Service
public class DemoService {

        @Autowired
        DemoRepository demoRepository;

        public DemoEntity getDemoObject(String name) {
                return demoRepository.findByName(name);
        }

}
```

Figure 23 An example of using the repository in a service

The last step would be to create a controller class to expose the API and allow the front-end of the application to send requests. The controller class in Spring Boot is in charge of handling incoming REST API requests and returning the view to be made as a response.

The **@Controller** or **@RestController** annotations are used to annotate the controller classes in Spring. These classes are labelled as request handlers so that Spring can recognize them as RESTful resources during runtime. **@RestController** is a specialized version of the controller. It includes the **@Controller** and **@ResponseBody** annotations, and as a result, simplifies the controller implementation. While the **@PathVariable** annotation can be used to handle template variables in the request URI mapping and use them as method parameters. [21]

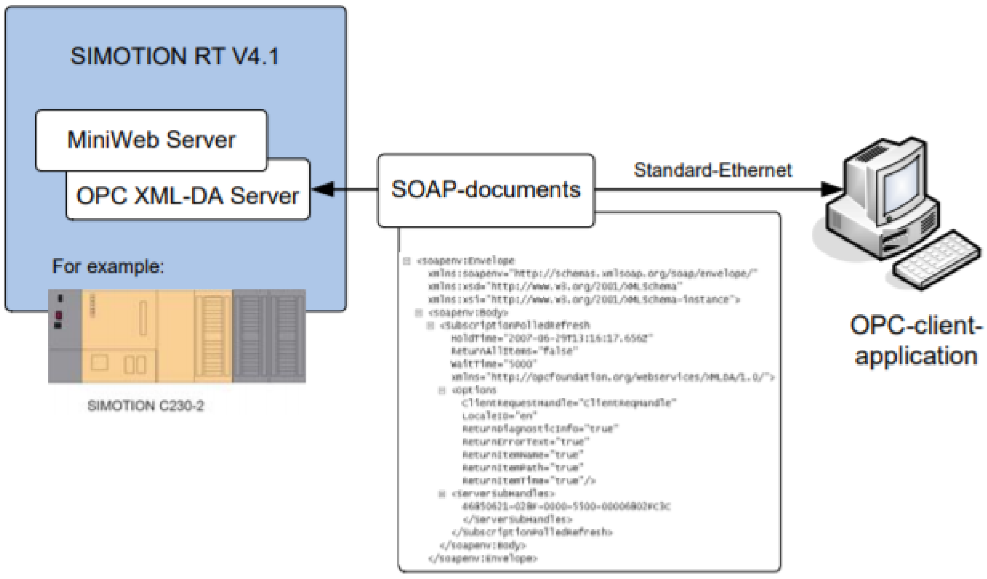


```java
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RestController;

import com.example.spring.model.DemoEntity;
import com.example.spring.services.demoService;

@RestController
public class DemoController {

        @Autowired
        PersonService demoService;

        @GET("/demo/{name}")
        public Person getDemoByName(@PathVariable String name) {
                return demoService.findByName(name);
        }
```

Figure 24 An example of a Spring Boot Controller

3.1.4 OPC XML-DA Library

OPC XML-DA is a java library developed by the OPC Foundation and released in July 2003. From the firmware version 4.1 on, the OPC XML-DA server is integrated in the SIMOTION runtime. By means of standard Ethernet or PROFINET connection, it is possible to realize a data exchange between this OPC server and a client application. This makes it possible to access to certain program variables as well as to system functions of the control without additional engineering tools.



```
<soapenv:Envelope
  xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
 <soapenv:Body>
  <SubscriptionPolledRefresh
    HoldTime="2007-06-29T13:16:17.656Z"
    ReturnAllItems="false"
    WaitTime="5000"
    xmlns="http://opcfoundation.org/webservices/XMLDA/1.0/">
   <Options
     ClientRequestHandle="ClientReqHandle"
     LocaleID="en"
     ReturnDiagnosticInfo="true"
     ReturnErrorText="true"
     ReturnItemName="true"
     ReturnItemPath="true"
     ReturnItemTime="true"/>
   <ServerSubHandles>
     46850621-028F-0000-5100-00006B02FC3C
     </ServerSubHandles>
   </SubscriptionPolledRefresh>
  </soapenv:Body>
 </soapenv:Envelope>
```

Figure 25 Schematic structure of the client-server system [22]

OPC XML-DA uses an open standard that can be used by many operating systems. The real communication between the terminals is realized via the SOAP (Simple Object Access Protocol, a permanently defined XML data structure). SOAP is based upon the http protocol. This already shows that, from the hardware side, a data exchange via OPC XML-DA is based upon an Ethernet connection.

As a principle, the communication with OPC XML is a client-server system. An OPC server makes certain data available, and an OPC client is in the position to call data from the server or to change them on the server. To do so, a server has to offer certain determined methods and data interfaces according to OPC specification, which can be accessed to by the client. For example, each server has to offer a read function, which permits the client to read out an existing variable or data set of the server. A further function “write” that realizes a writing, thus a change of a variable made available by the server.

A communication between OPC client and OPC server has to always be initiated by the client side so that the call back mechanism of the server with OPC XML-DA cannot be realized easily. To come to know about a changed value in the OPC server within a certain time interval, an OPC XML-DA client will request this value continuously at least one time within the desired time interval (called polling). If the value to be requested does not change within a series of cycles, it will still be requested each time and transmitted to the client. In case of very short time intervals, this leads to a high network communication and, depending on the system, to a high system load. To reduce this system load, OPC XML-DA offers the function “subscription”.

The following OPC functionalities are provided by the library:

- Read: the current value of the selected variable is read from the OPC server. In addition to this ItemPath and ItemName can be entered to these fields manually.

- Write: Writing of the device's variables being specified in the input fields ItemPath and ItemName.

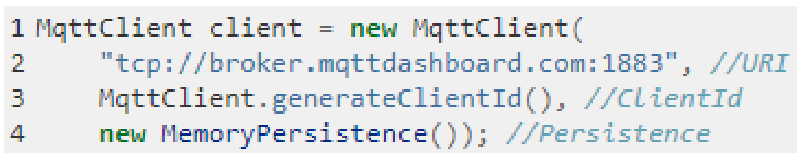
- Connect: By changing the URL a connection to a new device can be established. Doing so, the existing variable tree is renewed.

- Subscribe: Subscription can be initialized. After the input of the required values, the variable being indicated in the field ItemPath and ItemName is registered for a Subscription.

3.1.5 MQTT Library

The Paho Java Client is a Java-based MQTT client library for developing applications that run on the Java Virtual Machine (JVM) or other Java-compatible platforms. Paho is an open-source project that aims to provide stable open-source implementations of open and standard messaging protocols for new, existing, and emerging Machine-to-Machine (M2M) and Internet of Things (IoT) applications. It's reliable, and it's used to link to MQTT brokers by a wide variety of companies from various industries all over the world.

A MqttClient object exposes all MQTT operations. The user must first construct an instance of this MqttClient before linking, posting, or subscribing. In order to link to a MQTT broker, a MqttClient requires a URL and a client identifier. The port to which we want to link must also be specified, and the MQTT client identifier must be unique. [23]



```
MqttClient client = new MqttClient(
    "tcp://broker.mqttdashboard.com:1883", //URI
    MqttClient.generateClientId(), //ClientId
    new MemoryPersistence()); //Persistence
```

Figure 26 An example of MqttClient instance

After the creation of the MqttClient we need to connect to the broker first before we are able to do any MQTT operation. Connecting to a MQTT broker is very easy as well as checking if the client is already connected.

```java
client.connect();
client.isConnected();
```

Figure 27 Connecting to an MqttClient

After the connection to a broker is established, the next step is to publish a MQTT message. With the Paho library this is very straightforward with one line. We can control all relevant MQTT information like the topic, the payload, the Quality of service and if the message should be retained. Paho expects to connect to the MQTT broker first, otherwise we'll receive a MqttException.

```java
client.publish(
    "topic", // topic
    "payload".getBytes(UTF_8), // payload
    2, // QoS
    false); // retained?
```

Figure 28 Publishing a Mqtt message

In order to receive messages a client must subscribe to one or more MQTT topics. If you want to react on incoming MQTT messages, you need to set a callback first. This callback is called on specific events, for example when a message arrives. To subscribe to messages, we can use the method subscribe, which takes a topic and a QoS level as parameter.

```java
client.setCallback(new MqttCallback() {

    //Called when the client lost the connection to the broker
    @Override
    public void connectionLost(Throwable cause) {
    }

    @Override
    public void messageArrived(String topic, MqttMessage message) throws Exception {
        System.out.println(topic + ": " + Arrays.toString(message.getPayload()));
    }

    //Called when a outgoing publish is complete
    @Override
    public void deliveryComplete(IMqttDeliveryToken token) {
    }
});

client.connect();
client.subscribe("#", 1);
```

Figure 29 Subscribing to Mqtt topics

3.2 Front-End Implementation

The front-end of the application will be developed and provided by colleague of mine who is a Front-end Developer at Siemens Digital Industries, thus it is not part of my thesis project. For this reason, this section will be short and brief. It will contain a short description of the framework used which is AngularJS, as well as some explanation about the communication established with the back-end.

3.2.1 Angular

Angular is a JavaScript MVC framework for developing interactive web applications on the client side. There is no need to learn another syntax or language since it is entirely based on HTML and JavaScript. The AngularJS framework transforms static HTML into dynamic HTML. It enhances HTML's functionality by incorporating built-in attributes and elements, as well as allowing users to build custom attributes using simple JavaScript.

The architecture is based on the well-known MVC model (Model-View-Controller). This is a design pattern that can be seen in almost all modern web applications. The business logic layer, the data layer, and the presentation layer are all divided into different parts in this pattern. The division into various parts is done to make it easier to handle each one. To build any application that involved DOM manipulation, a lot of JavaScript had to be written. But with Angular, a lesser amount of code is needed to write for DOM manipulation.

AngularJS can take care of routing which means moving from one view to another. This is the key fundamental of single page applications; wherein you can move to different functionalities in your web application based on user interaction but still stay on the same page. [24]

3.2.2 Communication with the Back-end

Both the backend and the frontend work together to achieve a common purpose. The backend receives HTTP requests from the browser. The HTTP headers or request body of those requests can contain data. The aim may be to request new data or to send data created by the user to the backend. HTTP requests are created and sent from inside the user's browser. Each request receives a response, which includes information from the HTTP headers and the request body. Those responses are returned to the user's browser from the backend. The requests that are executed are as a result of a user clicking a link or as a result of background JS code. However, there are other factors, such as the browser reading the incoming HTML and noticing that it needs to load a resource, such as a JS file, an image, or a CSS file. It then requests each of them with a single new HTTP request. This usually occurs while a website is loading.

The backend normally responds with HTML-formatted responses, other static files (CSS, JS, images, etc.), or JSON-formatted data in the HTTP body. While Simple HTTP requests without a body, Form data, and JSON-formatted data are all sent by the frontend. [25]

3.3 Database

This section will revolve around the database used in the application which is PostgreSQL as well as the database driver library that will connect the database to Java and allow for CRUD operation to be made.

3.3.1 PostgreSQL

PostgreSQL (also known as Postgres) is an open-source object-relational database management system with an emphasis on extensibility and compliance with industry standards. It is available for free as an open-source solution and is licensed under the PostgreSQL License, which is a permissive software license. It can handle a wide range of workloads, from single-machine applications to broad Internet-facing applications with a large number of concurrent users.

PostgreSQL is highly extensible, in addition to being free and open source. When it comes to database configuration and management, PostgreSQL succeeds in two ways. For instance, it complies with SQL requirements to a high degree. This strengthens its compatibility with other programs. Second, PostgreSQL provides users with metadata access. Since its process is catalog-driven, PostgreSQL is extensible. PostgreSQL differs from conventional relational database systems in that it stores much more information in its catalogs, including not only table and column information, but also data types, functions, access methods, and so on. [26]

3.3.2 PostgreSQL JDBC Driver

Java Database Connectivity (JDBC) is a Java application programming interface (API) that describes how a client can communicate with a database. It is a component of the Java Standard Edition framework that offers methods for querying and updating data in a database, with a focus on relational databases. The PostgreSQL JDBC Driver enables Java programs to link to a PostgreSQL database using standard Java code that is database agnostic. It's written in pure Java and uses the PostgreSQL native network protocol to communicate. As a result, the driver is platform independent; once compiled, it can be used on any device. [27]

In this project we will be creating a Spring-Boot application which will interact with PostgreSQL doing CRUD operations. First, we need to configure Spring Boot to use

PostgreSQL as our data source. We can do that simply by adding the PostgreSQL database url, username, and password in the "src/main/resources/application.properties" file.

```
## Spring DATASOURCE (DataSourceAutoConfiguration & DataSourceProperties)
spring.datasource.url=jdbc:postgresql://localhost:5432/postgres_demo
spring.datasource.username= username
spring.datasource.password= password

# The SQL dialect makes Hibernate generate better SQL for the chosen database
spring.jpa.properties.hibernate.dialect = org.hibernate.dialect.PostgreSQLDialect
```

Figure 30 Postgres configuration with Spring Boot

Now we need to create an entity class to map with the corresponding table in the database, as follows:

```java
import javax.persistence.*;

@Entity
@Table(name = "demo")
public class Demo {

    @Id
    @GeneratedValue(strategy = GenerationType.SEQUENCE)
    @Column(name = "id")
    private Integer id;

    @Column(name = "name")
    private String name;

    public Integer getId() {
        return id;
    }

    public void setId(Integer id) {
        this.id = id;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }
}
```

Figure 31 Demo entity to be persisted in Postgres

The next step would be to declare a repository interface as follows:

```java
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;

@Repository
public interface DemoRepository extends JpaRepository<Demo, Integer> {

}
```

Figure 32 Creating a Repository for our Demo entity

The last part is to create a controller for this demo to expose these CRUD operations and allow HTTP requests to be done:

```java
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.DeleteMapping;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RestController;

import java.util.List;
import java.util.Optional;

@RestController("/demo")
public class DemoController {

    @Autowired
    private DemoRepository demoRepository;

    @GetMapping
    public List<Demo> getAllDemos() {
        return demoRepository.findAll();
    }

    @GetMapping
    public Optional<Demo> getDemoById(Integer id) {
        return demoRepository.findById(id);
    }

    @DeleteMapping
    public void deleteDemoById(Integer id) {
        demoRepository.deleteById(id);
    }
}
```

Figure 33 The controller which exposes these CRUD operations

4. Evaluation

After building the web application, this section will focus on the functionalities of the app and the working principle behind them. The app will consist of four screens and one pop up window for configuration and setup. For the purpose of demonstration, some demo data has been imported into the app.

4.1 Configuration Screen

The configuration screen is used to connect to a PLC, browse the variables stored in its memory and create the setup of a motor axis for later measurements. In order to do that we need to click on the "Connect new axis" button. After clicking on this button, a pop-up window will appear to allow us to create our setup.

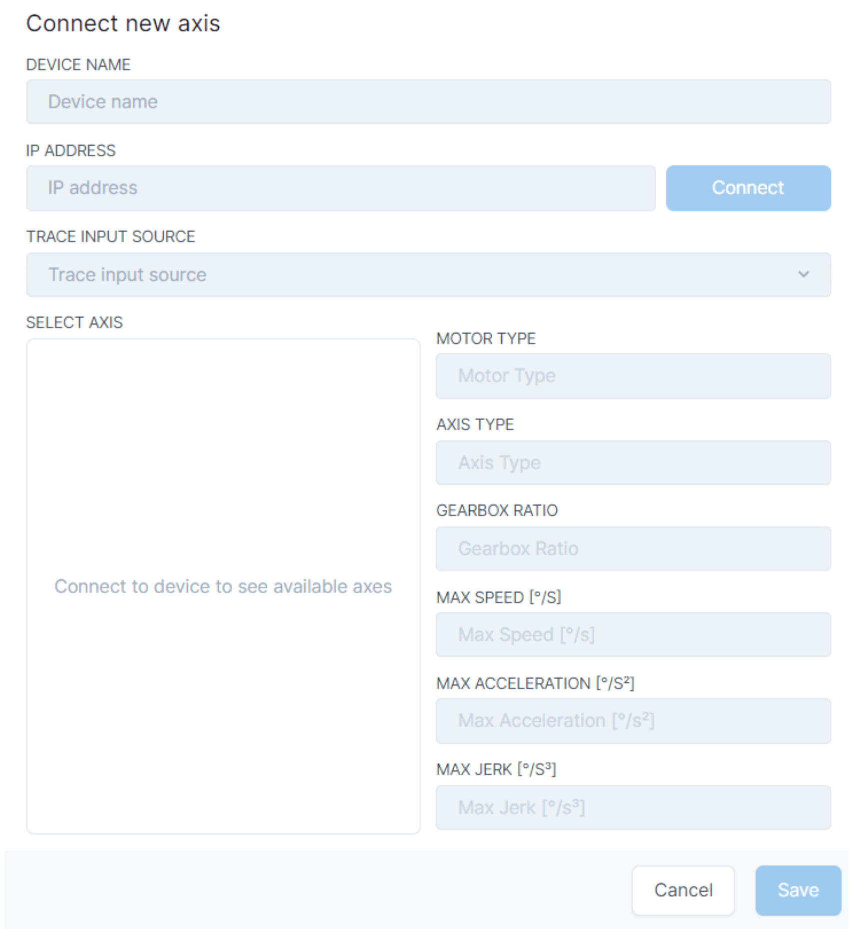


Figure 34 Connect new axis pop-up window

In figure 34 we can see the various fields which need to be filled before a setup came be saved. The first step is to choose a name for our machine and then enter its IP address and hit the connect button. This action will send a GET request to the server and the server will respond by connecting to the machine and retrieving the required data. This data is then sent back to the front end as a response body in the form of JSON. An example of the response should be as follows:

```json
{
    "type": "D445",
    "ipAddress": "192.168.1.7",
    "serialNumber": "ST-KN6031549",
    "firmware": "V 5.3.1.9",
    "status": "CONNECTED",
    "statusTime": "2021-04-12T20:43:33.663+02:00",
    "machineNodes": [
        {
            "node": "Red_Axis",
            "saved": false
        },
        {
            "node": "Blue_Axis",
            "saved": false
        }
    ]
}
```

The response includes a property called “machineNodes”. This property is a list of the available axes inside this machine and each axis contains a Boolean variable showing whether this axis has been already configured and saved or not. If the variable “saved” is false, then we will be able to select the axis from the list in the pop-up window. This action will produce another GET request to the server where the response is a JSON containing all information needed about the axis.

```json
{
    "name": "Red_Axis",
    "motorType": "1FK7 synchronous motor",
    "motorMLFB": "1FK7022-5AK7x",
    "motorCode": "23726",
    "axisType": "LINEAR",
    "gearRatio": 10.0,
    "maxSpeed": 200000.0,
    "maxAcceleration": 10000.0,
    "maxJerk": 500.0,
    "dcLinkVoltage": 328.0,
    "possibleVoltages": "600.0,540.0,650.0,720.0"
}
```

As we can see in the response, we have a property called “dcLinkVoltage” which reads the current DC link voltage at the time of measurement. While below it we have “possibleVoltages” which tells us the possible voltages that this type of motor can have when running in a 3-phase operating state. For obtaining accurate measurements then a voltage has to be selected from the drop-down menu under the name "DC LINK VOLTAGE".

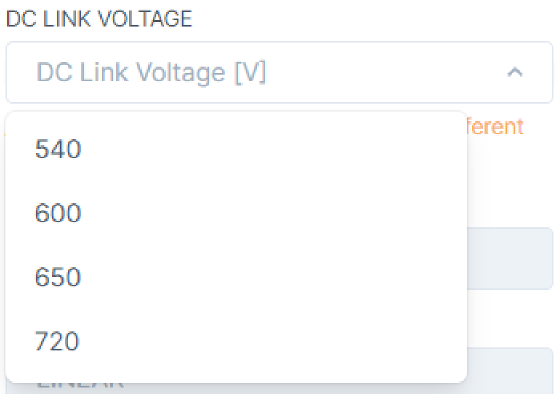
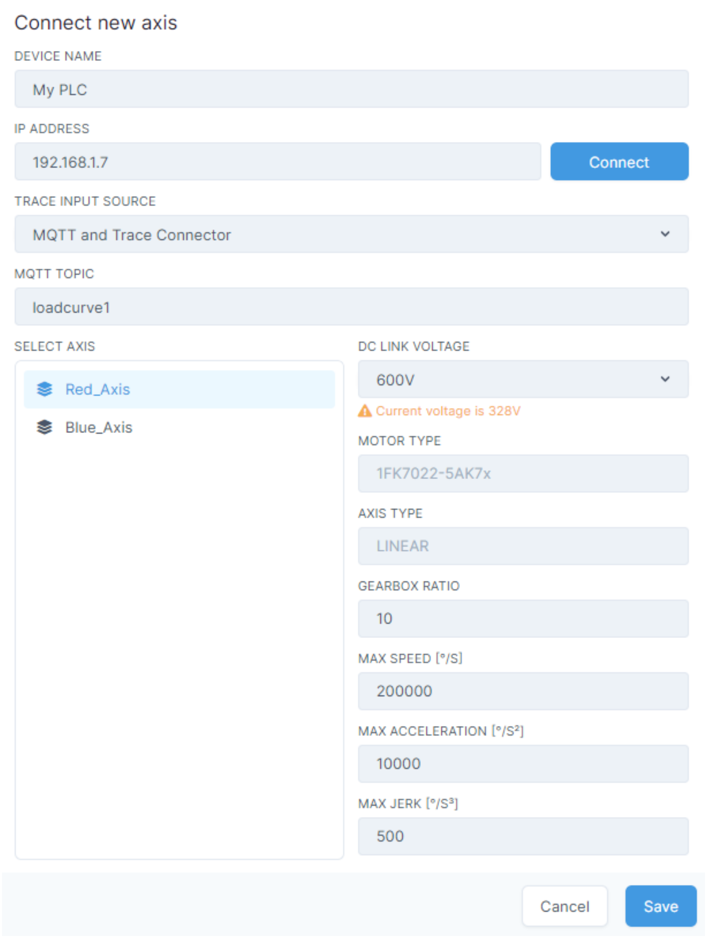


Figure 35 DC link voltage drop-down menu

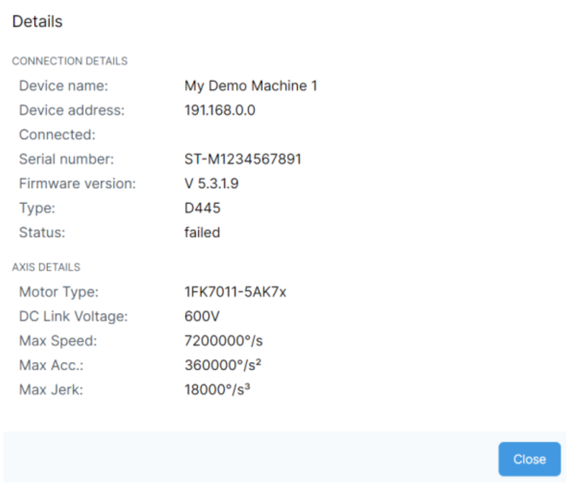
The last step needed before saving the setup is to enter a MQTT topic for which this axis configuration will subscribe to and receive the relevant data from the MQTT server in order to create the charts. By clicking inside the "TRACE INPUT SOURCE FIELD" we are then able to select "MQTT and Trace Connector" and then write our MQTT inside the next field. Trace Connector is another IoT web application by Siemens Digital Industries which performs measurements from a PLC and then publishes the data on an MQTT server. The final setup window should now look as in figure 36.


Connect new axis

DEVICE NAME	My PLC
IP ADDRESS	192.168.1.7
TRACE INPUT SOURCE	MQTT and Trace Connector
MQTT TOPIC	loadcurve1
SELECT AXIS	Red_Axis, Blue_Axis
DC LINK VOLTAGE	600V
	Current voltage is 328V
MOTOR TYPE	1FK7022-5AK7x
AXIS TYPE	LINEAR
GEARBOX RATIO	10
MAX SPEED [°/S]	200000
MAX ACCELERATION [°/S²]	10000
MAX JERK [°/S³]	500

Figure 36 Resulting setup window with all fields entered

The configuration table consists of six columns: axis name, machine name, IP address of the machine, type of machine, time of creation of configuration and current status. In addition, at the end of every row we have three buttons. The first button opens a pop-up window which shows the details of the saved axis while the last two buttons are for editing or deleting the axis respectively.


Details

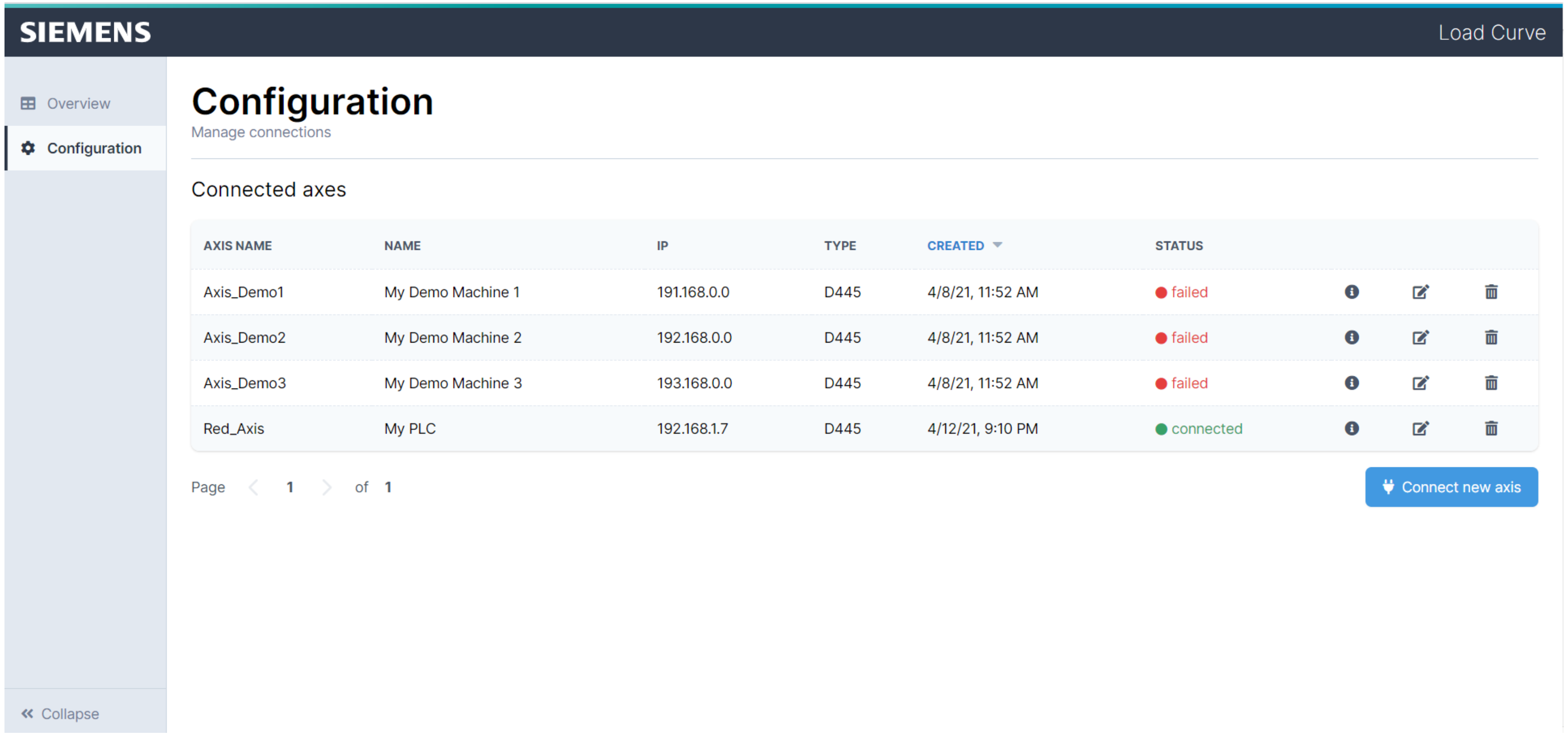
CONNECTION DETAILS

Device name:	My Demo Machine 1
Device address:	191.168.0.0
Connected:	
Serial number:	ST-M1234567891
Firmware version:	V 5.3.1.9
Type:	D445
Status:	failed

AXIS DETAILS

Motor Type:	1FK7011-5AK7x
DC Link Voltage:	600V
Max Speed:	7200000°/s
Max Acc.:	360000°/s²
Max Jerk:	18000°/s³

Figure 37 Axis details pop-up window



Configuration

Manage connections

Connected axes

AXIS NAME	NAME	IP	TYPE	CREATED	STATUS
Axis_Demo1	My Demo Machine 1	191.168.0.0	D445	4/8/21, 11:52 AM	● failed
Axis_Demo2	My Demo Machine 2	192.168.0.0	D445	4/8/21, 11:52 AM	● failed
Axis_Demo3	My Demo Machine 3	193.168.0.0	D445	4/8/21, 11:52 AM	● failed
Red_Axis	My PLC	192.168.1.7	D445	4/12/21, 9:10 PM	● connected

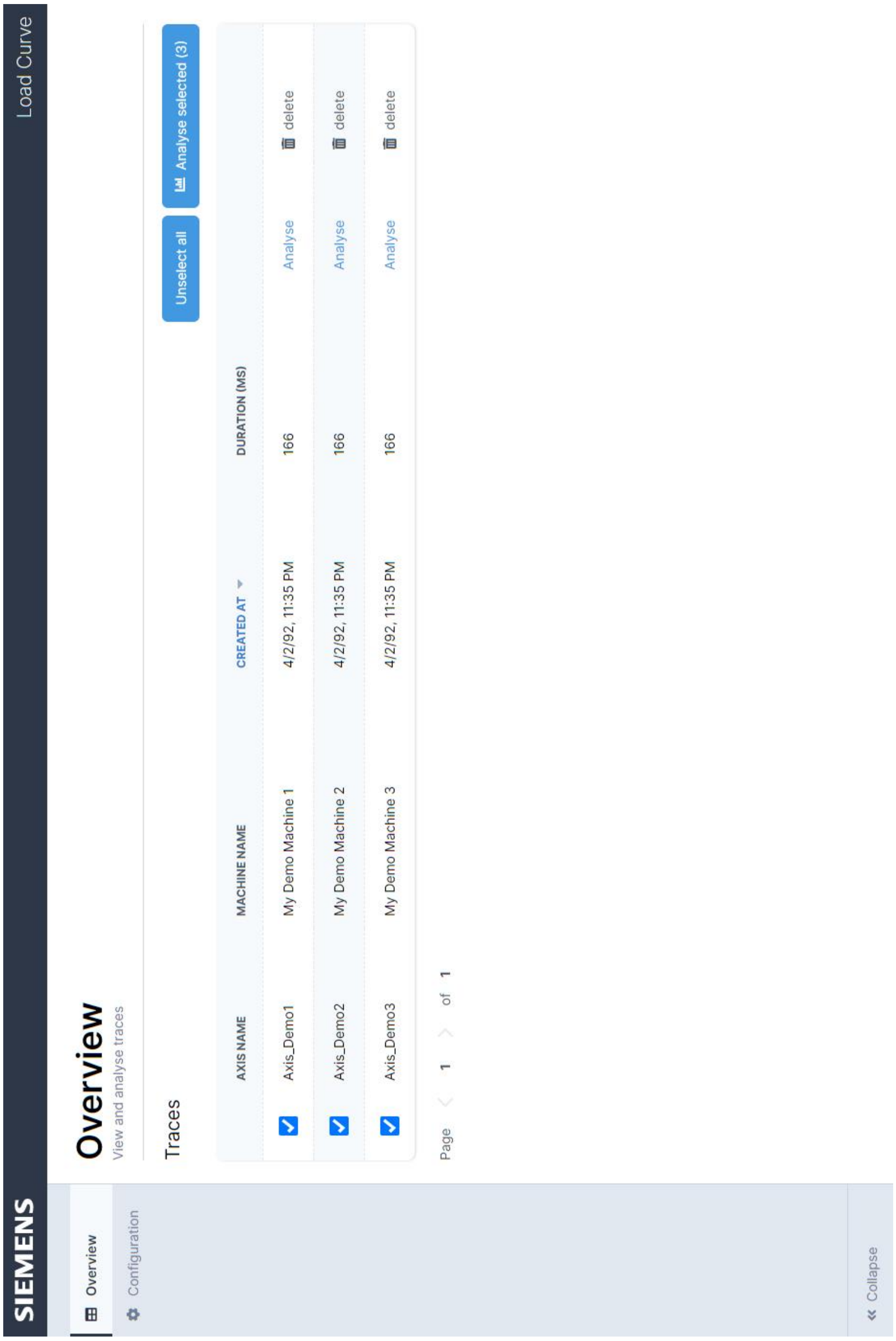
Page 1 of 1

Figure 38 A full screenshot of the configuration window

4.2 Overview Screen

In the overview screen, we will display a list of all measurements of each axis connected. These measurements are known as “Traces” and each “Trace” object will contain information about the axis as well as a list of values of the measurements done. When loading the Overview page, the server will send a response of a list of all available “Trace” object. For this section a demo axis has been created with mock data and it’s would be as follows:

```
{
    "id": 92,
    "axisName": "Axis_Demo1",
    "axisType": "LINEAR",
    "motorMLFB": "1FK7011-5AK7x",
    "gearRatio": 200.0,
    "maxSpeed": 7200000.0,
    "maxAcceleration": 360000.0,
    "maxJerk": 18000.0,
    "dcLinkVoltage": 600.0,
    "machineName": "My Demo Machine 1",
    "measurements": {
        "id": 87,
        "measurementName": "Load Curve Edge App",
        "traceStartType": "IMMEDIATE",
        "duration": 166,
        "ipAddress": "192.168.1.8",
        "tracejobId": 101,
        "cycletime": 1,
        "timestamp": 702250513000,
        "machineSerialNumber": "ST-M123456789",
        "machineFirmwareVersion": "V 5.3.1.9"
    },
    "loadCurve": {
        "id": 93,
        "datetime": 702250513000,
        "cycleTime": 1
    },
    "timestamp": 702250513000,
    "duration": 166
}
```



Overview

View and analyse traces

Traces

	AXIS NAME	MACHINE NAME	CREATED AT	DURATION (MS)		
✓	Axis_Demo1	My Demo Machine 1	4/2/92, 11:35 PM	166	Analyse	delete
✓	Axis_Demo2	My Demo Machine 2	4/2/92, 11:35 PM	166	Analyse	delete
✓	Axis_Demo3	My Demo Machine 3	4/2/92, 11:35 PM	166	Analyse	delete

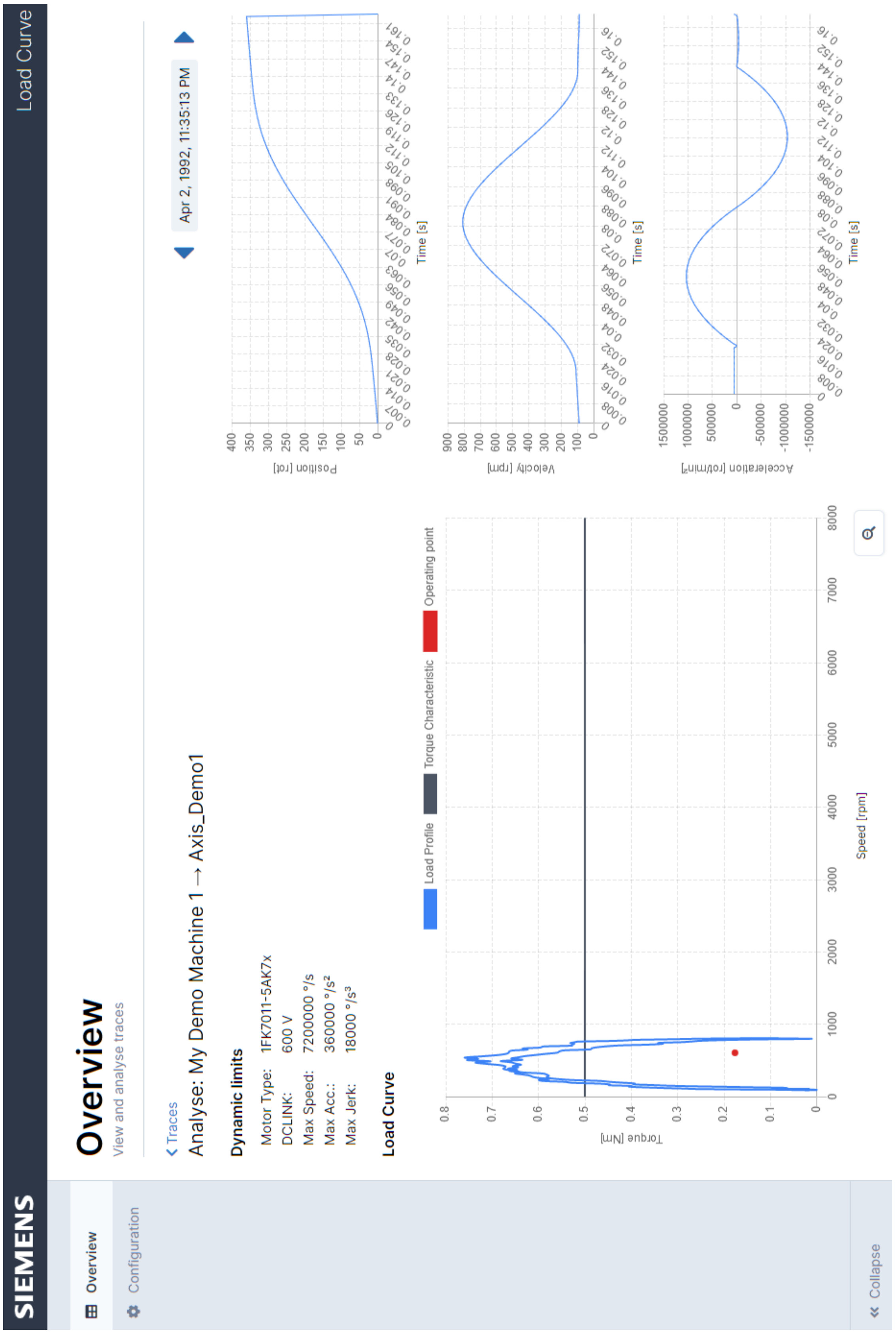
Page 1 of 1

Figure 39 A full screenshot of the Overview window

This response contains an object “Measurements” which includes all information about the measurements received over MQTT. However, in this response the list of values of the measurements is being suppressed to avoid the congestion of too many data being sent on each request. To access this data an “Analyse” button will be available next to each measurement which sends two responses. One response will include the values for the position, speed and acceleration charts and the second response will include the values for the torque characteristic, load curve and operating point. The responses are as follows:

```
{
    "velocity": {
        "axisX": [...],
        "axisY": [...]
    },
    "acceleration": {
        "axisX": [...],
        "axisY": [...]
    },
    "position": {
        "axisX": [...],
        "axisY": [...]
    }
}
```

```
{
    "torqueCharacteristic": {
        "axisX": [...],
        "axisY": [...]
    },
    "loadProfile": {
        "axisX": [...],
        "axisY": [...]
    },
    "effectiveSpeed": 606.8984425107654,
    "effectiveTorque": 0.17676690952747628
}
```


SIEMENS

Load Curve

Overview

Configuration

Collapse

Overview

View and analyse traces

< Traces

Analyse: My Demo Machine 1 → Axis_Demo1

Dynamic limits

Motor Type:	1FK7011-5AK7x
DCLINK:	600 V
Max Speed:	7200000 °/s
Max Acc.:	360000 °/s²
Max Jerk:	18000 °/s³

Load Curve

Apr 2, 1992, 11:35:13 PM

Figure 40 A full screenshot of the overview Analyse window

In addition to the “Analyse” button, we have “Analyse Selected”. This button enables us to view all load profiles of the selected axes in one page for the purpose of comparison. The response will be the same as in the Analyse window, except that it will be repeated three times, once for each axis.

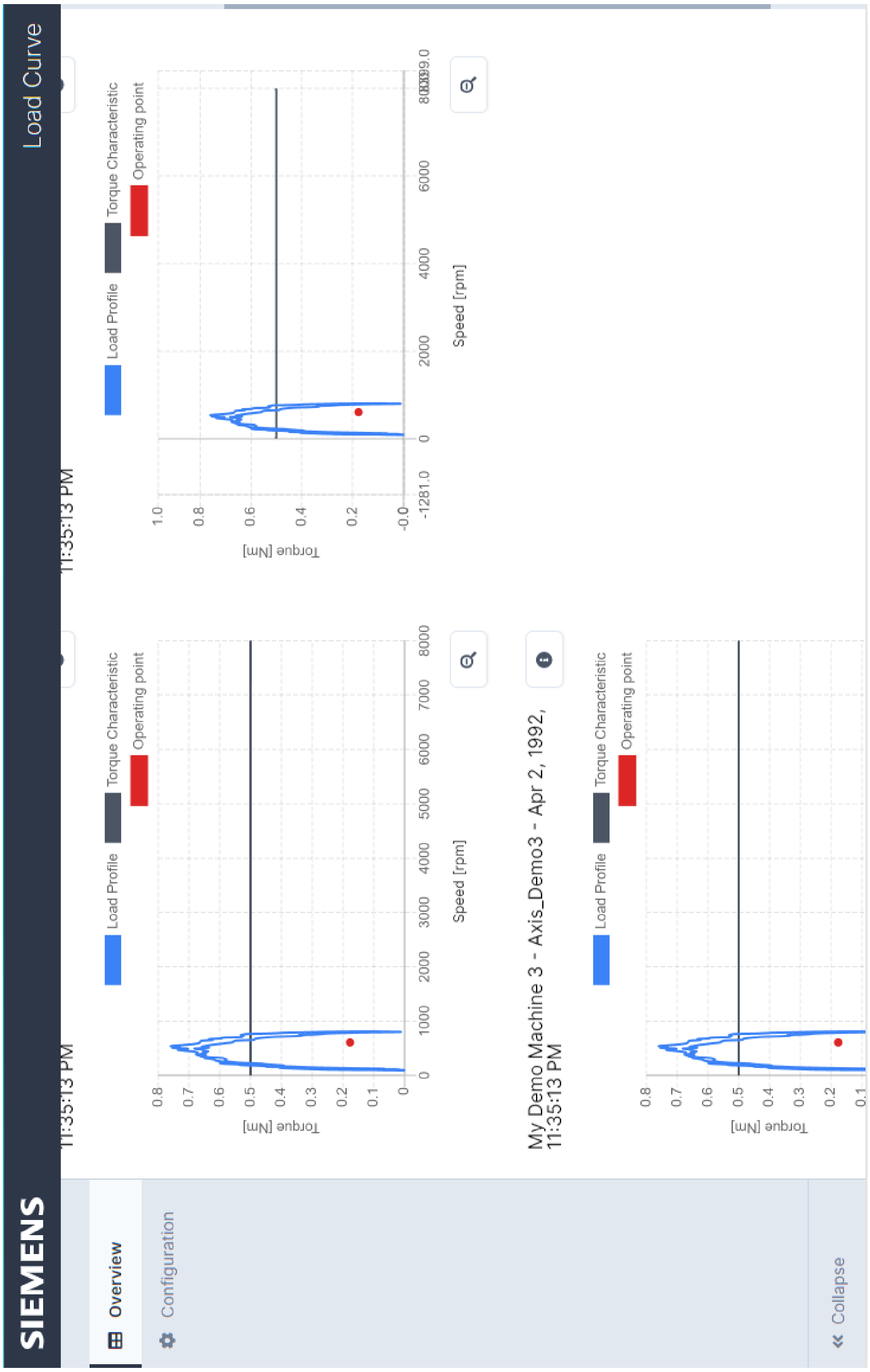
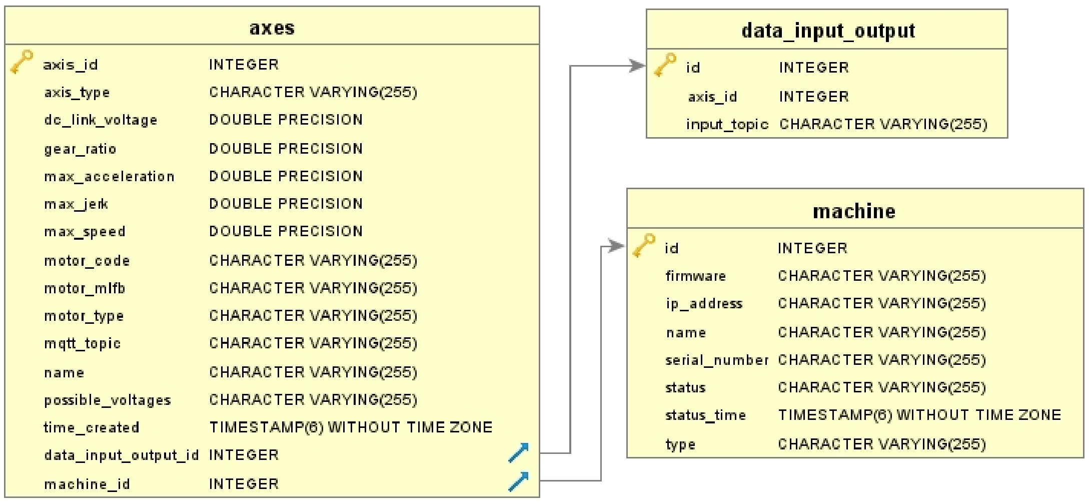


Figure 41 A full screenshot of the Analyse Selected window

4.3 Entity Relationships

When sending requests, the server is accessing the database to make queries and send the data in the form of responses. These responses contain objects that have a relationship with one another. These relationships can be split into two groups.

The first group is responsible for the Configuration window. It includes three entities: Machine, Axes and Data Input/Output. A Machine has a “One To Many” relationship with an Axis, meaning that one Machine can have multiple axes or contains a list of axes. While an Axis has a “One To One” relationship with Data Input/Output. Data I/O is responsible for managing the MQTT connection for each axis thus it has the Axis ID as a foreign key. The resulting Entity-Relationship diagram of this group is as shown in figure 42.



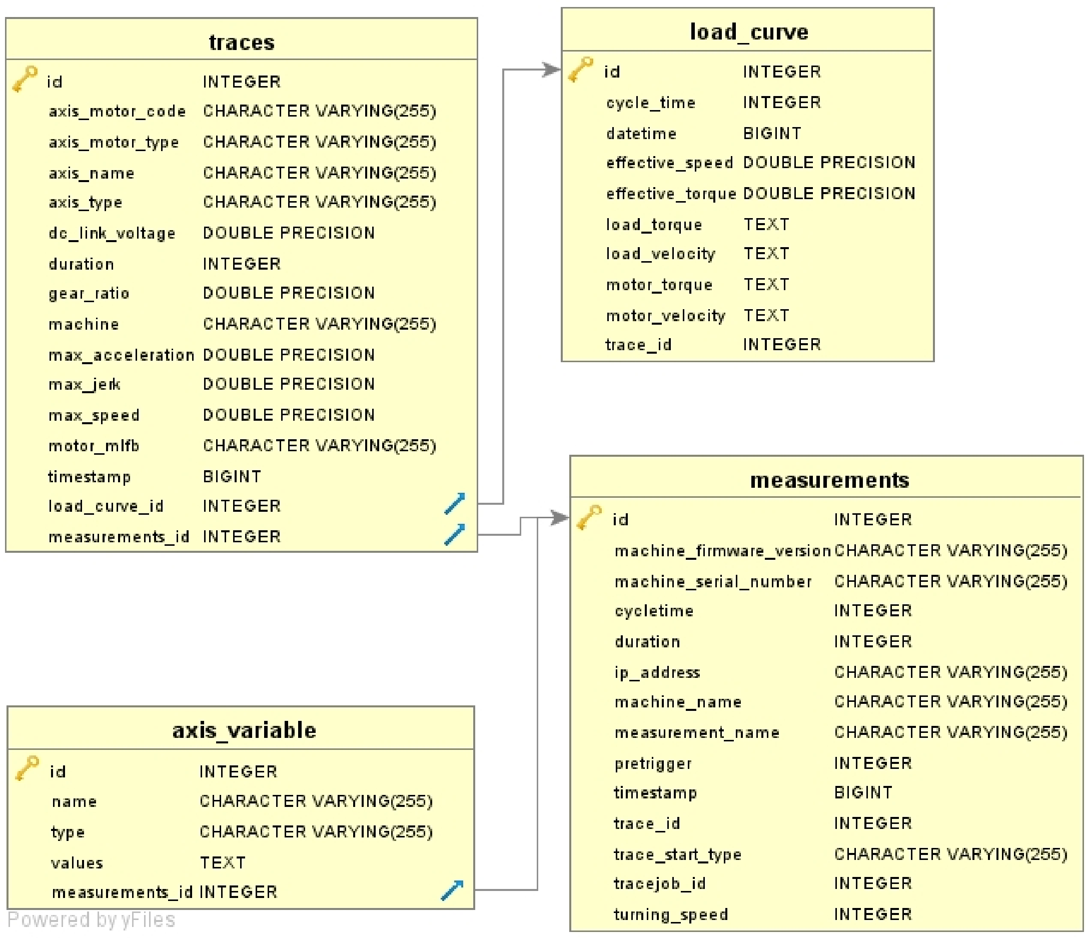
axes	
axis_id	INTEGER
axis_type	CHARACTER VARYING(255)
dc_link_voltage	DOUBLE PRECISION
gear_ratio	DOUBLE PRECISION
max_acceleration	DOUBLE PRECISION
max_jerk	DOUBLE PRECISION
max_speed	DOUBLE PRECISION
motor_code	CHARACTER VARYING(255)
motor_mlfb	CHARACTER VARYING(255)
motor_type	CHARACTER VARYING(255)
mqtt_topic	CHARACTER VARYING(255)
name	CHARACTER VARYING(255)
possible_voltages	CHARACTER VARYING(255)
time_created	TIMESTAMP(6) WITHOUT TIME ZONE
data_input_output_id	INTEGER
machine_id	INTEGER

data_input_output	
id	INTEGER
axis_id	INTEGER
input_topic	CHARACTER VARYING(255)

machine	
id	INTEGER
firmware	CHARACTER VARYING(255)
ip_address	CHARACTER VARYING(255)
name	CHARACTER VARYING(255)
serial_number	CHARACTER VARYING(255)
status	CHARACTER VARYING(255)
status_time	TIMESTAMP(6) WITHOUT TIME ZONE
type	CHARACTER VARYING(255)

Figure 42 E-R diagram of the first group

On the other hand, the second group deals with the data of measurements that we are receiving and want to display in the form of charts. This group includes four entities which are: Measurements, Axis Variables, Traces and Load curve. Measurements has a “One To Many” relationship with Axis Variable, so each measurement contains a set of variables such as velocity, acceleration and position of the axis. While a Trace is an entity that hold all info of its respective axis and has a “One To One” relationship with both Measurements and Load Curve. The Load Curve entity holds information about the torque characteristic of the axis, which is retrieved from catalogue data of the axis’s motor, as well as the load profile and effective torque/speed which are processed on the server side. Figure 43 shows the Entity-Relationship diagram of the second group.



traces

id	INTEGER
axis_motor_code	CHARACTER VARYING(255)
axis_motor_type	CHARACTER VARYING(255)
axis_name	CHARACTER VARYING(255)
axis_type	CHARACTER VARYING(255)
dc_link_voltage	DOUBLE PRECISION
duration	INTEGER
gear_ratio	DOUBLE PRECISION
machine	CHARACTER VARYING(255)
max_acceleration	DOUBLE PRECISION
max_jerk	DOUBLE PRECISION
max_speed	DOUBLE PRECISION
motor_mlfb	CHARACTER VARYING(255)
timestamp	BIGINT
load_curve_id	INTEGER
measurements_id	INTEGER

load_curve

id	INTEGER
cycle_time	INTEGER
datetime	BIGINT
effective_speed	DOUBLE PRECISION
effective_torque	DOUBLE PRECISION
load_torque	TEXT
load_velocity	TEXT
motor_torque	TEXT
motor_velocity	TEXT
trace_id	INTEGER

measurements

id	INTEGER
machine_firmware_version	CHARACTER VARYING(255)
machine_serial_number	CHARACTER VARYING(255)
cycletime	INTEGER
duration	INTEGER
ip_address	CHARACTER VARYING(255)
machine_name	CHARACTER VARYING(255)
measurement_name	CHARACTER VARYING(255)
pretrigger	INTEGER
timestamp	BIGINT
trace_id	INTEGER
trace_start_type	CHARACTER VARYING(255)
tracejob_id	INTEGER
turning_speed	INTEGER

axis_variable

id	INTEGER
name	CHARACTER VARYING(255)
type	CHARACTER VARYING(255)
values	TEXT
measurements_id	INTEGER

Figure 43 E-R diagram of the second group

5. Testing

As we arrive to the final chapter, this application will be tested with a real factory set up that includes a SIMOTION PLC connected to a packaging machine. The packaging machine is provided by Czech manufacturer Viking Mašek and uses servo motors and a SIMOTION PLC provided by Siemens.

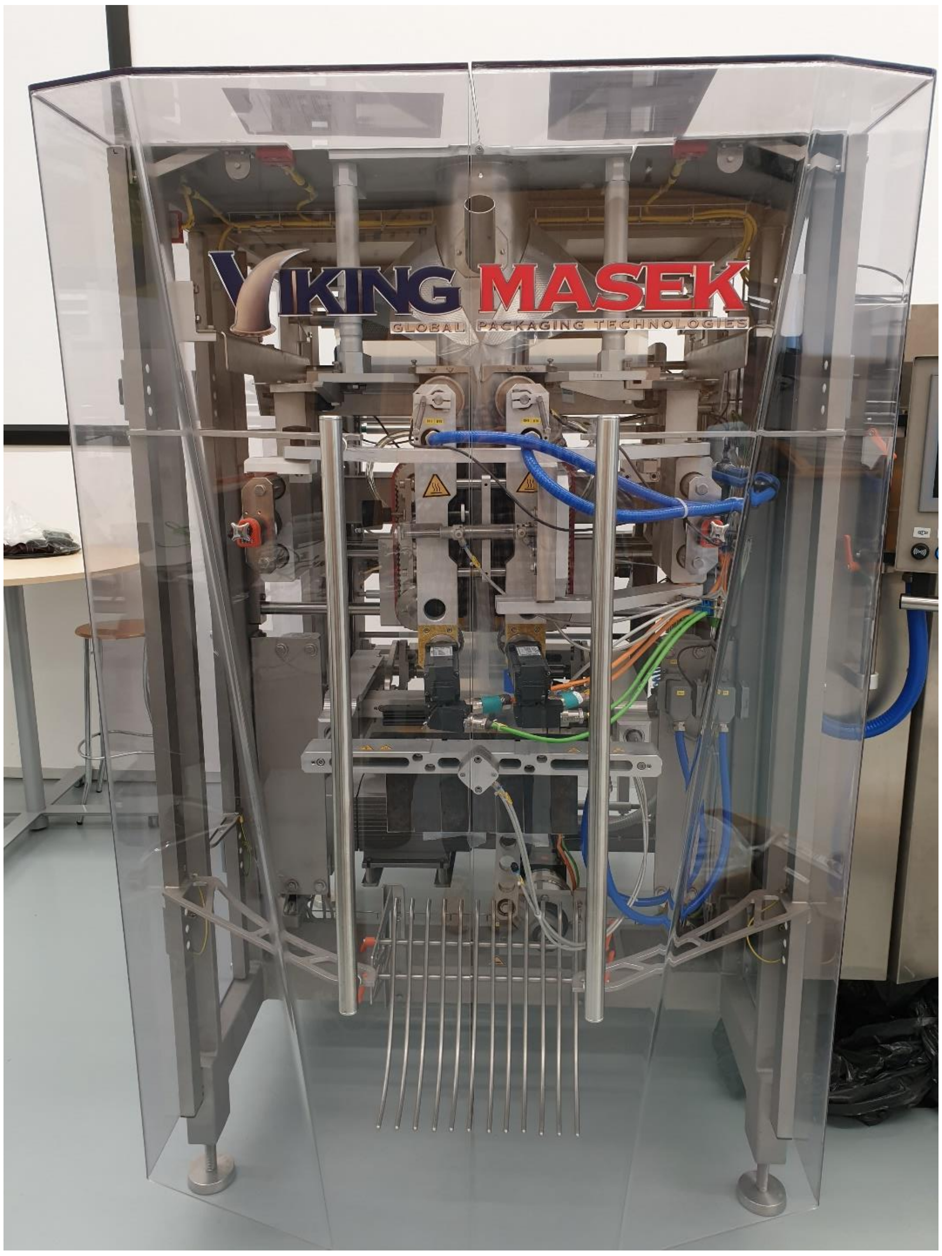


Figure 44 Packaging machine from Viking Mašek

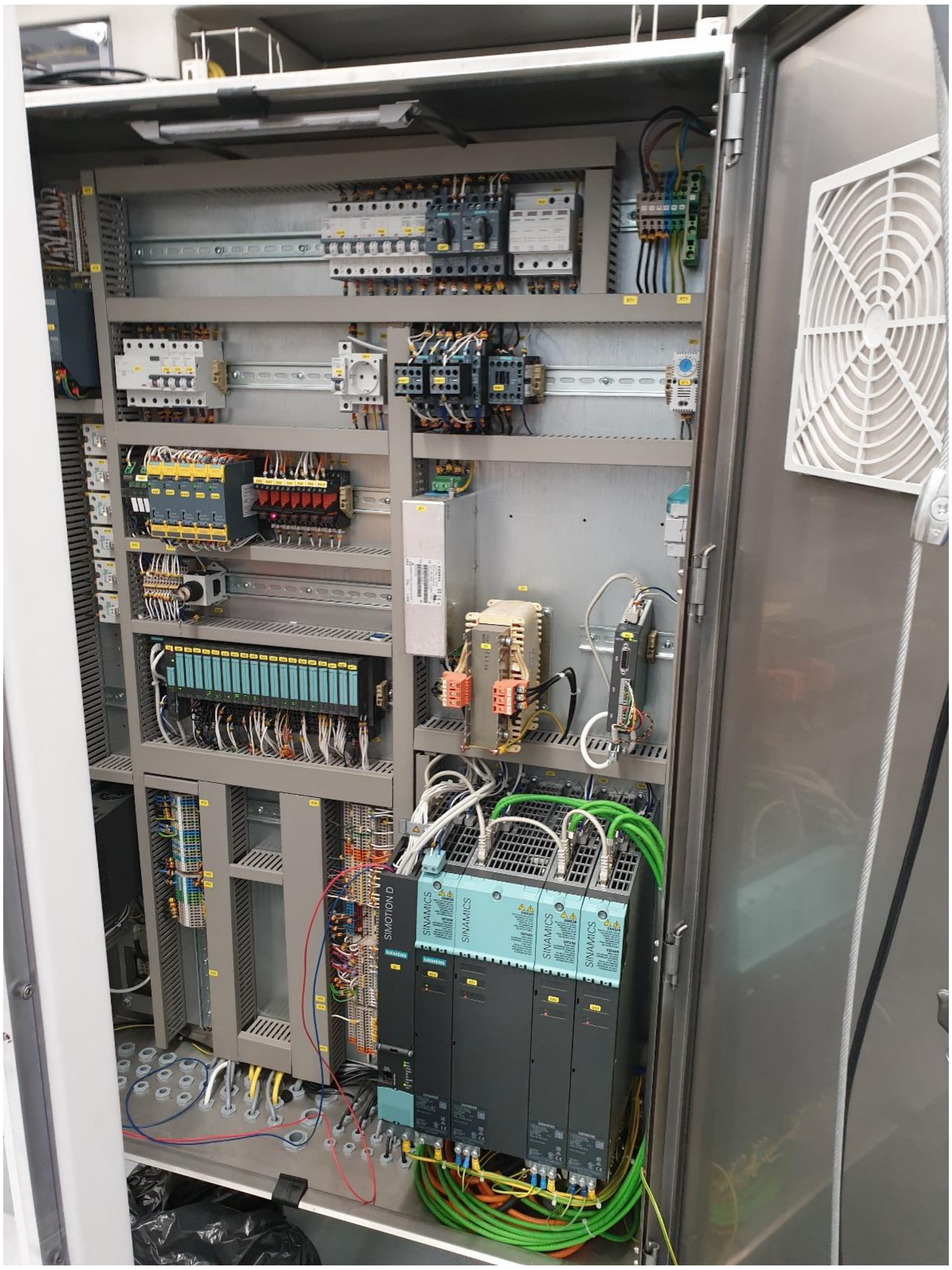
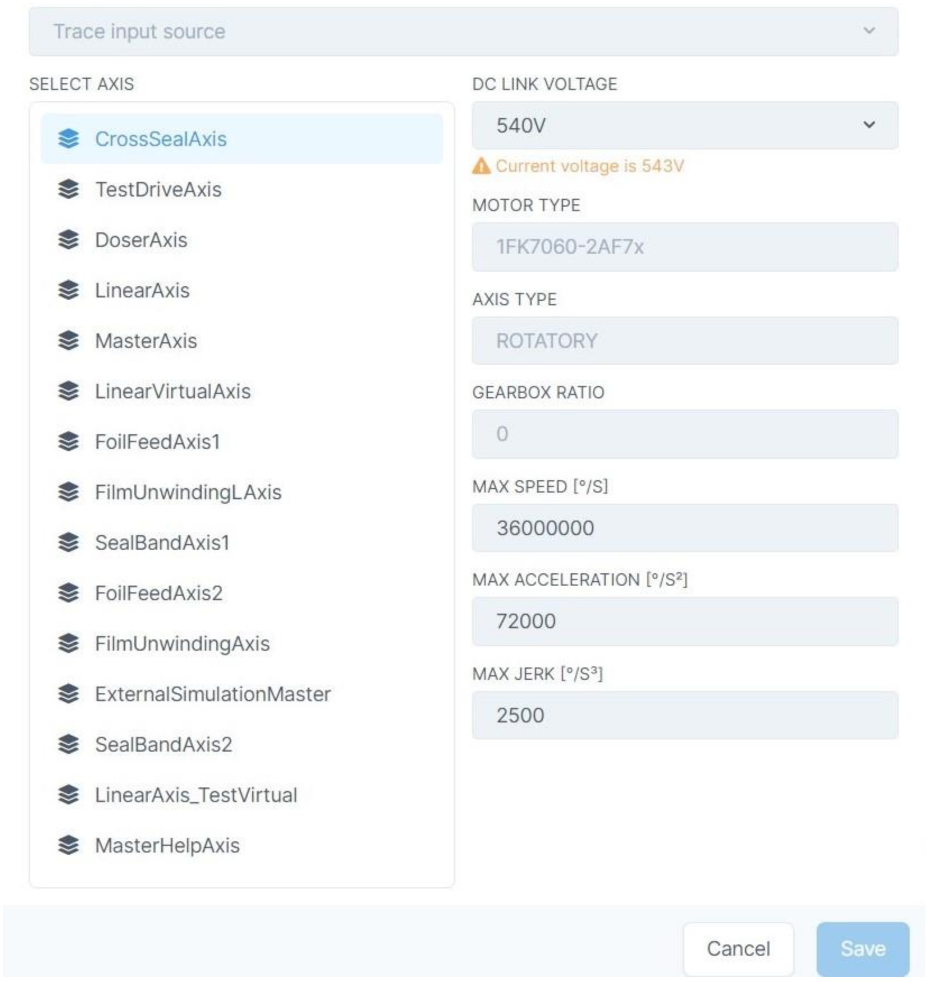


Figure 45 Electrical cabinet of the packaging machine

5.1 Results

In the test, the Cross Seal Axis was selected for the measurement since it would carry a significant amount of torque. The configuration of the axis is shown in figure 46 below.



Trace input source

SELECT AXIS

- CrossSealAxis
- TestDriveAxis
- DoserAxis
- LinearAxis
- MasterAxis
- LinearVirtualAxis
- FoilFeedAxis1
- FilmUnwindingLAxis
- SealBandAxis1
- FoilFeedAxis2
- FilmUnwindingAxis
- ExternalSimulationMaster
- SealBandAxis2
- LinearAxis_TestVirtual
- MasterHelpAxis

DC LINK VOLTAGE: 540V

Current voltage is 543V

MOTOR TYPE: 1FK7060-2AF7x

AXIS TYPE: ROTATORY

GEARBOX RATIO: 0

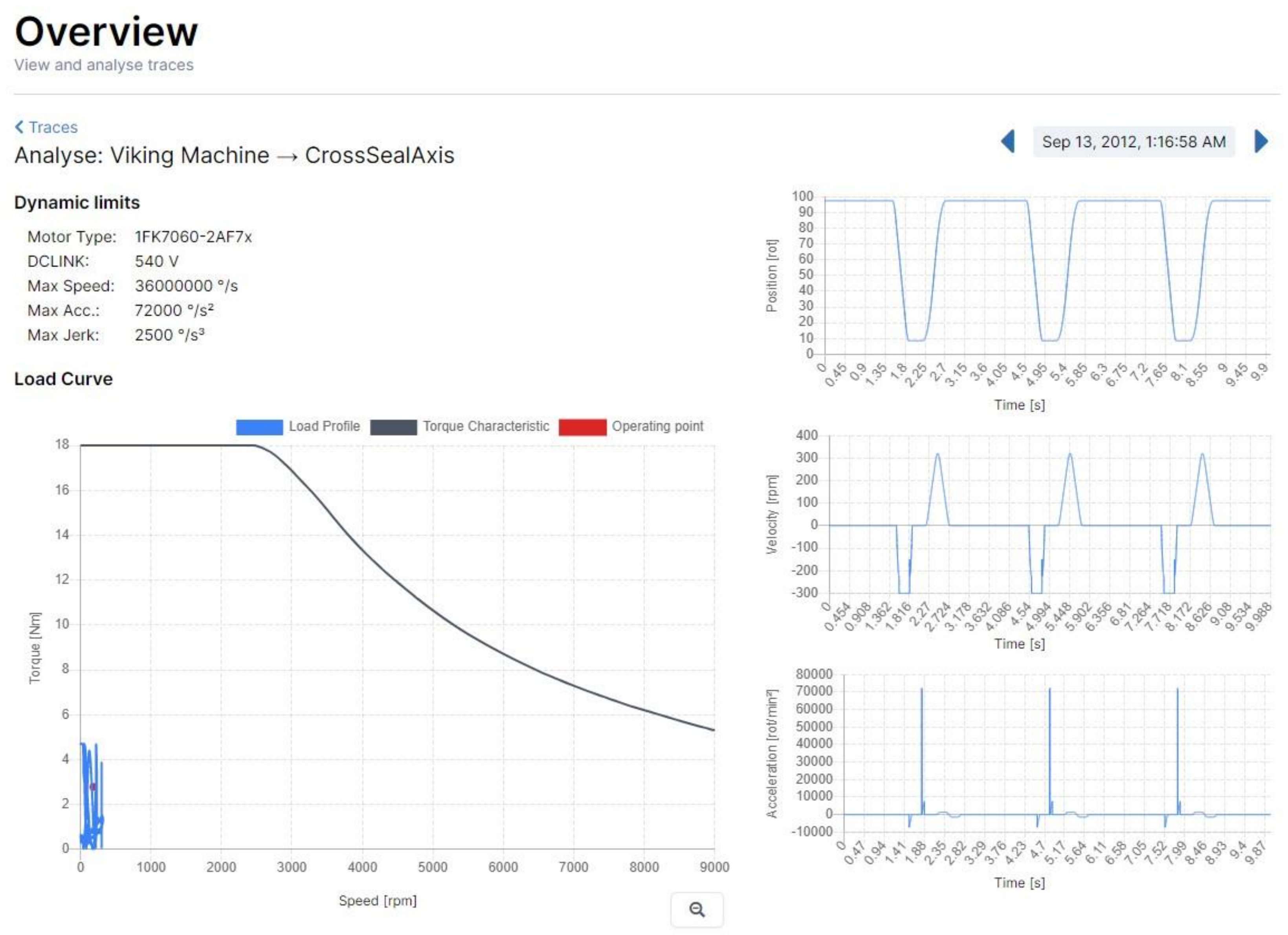
MAX SPEED [°/S]: 36000000

MAX ACCELERATION [°/S²]: 72000

MAX JERK [°/S³]: 2500

Figure 46 Configuration of Cross Seal Axis

After saving the configuration, then we are able to trigger a measurement while the machine is in its normal operating state. The duration of the measurement has been set to 10000 milliseconds to allow the axis to perform many cycles. Four different properties are expected to be measured: speed, acceleration, position, and torque. In addition, the torque-speed characteristic of the motor type will be loaded from catalogue data stored on the server side of the application.



Overview

View and analyse traces

< Traces

Analyse: Viking Machine → CrossSealAxis

Sep 13, 2012, 1:16:58 AM

Dynamic limits

Motor Type:	1FK7060-2AF7x
DCLINK:	540 V
Max Speed:	36000000 °/s
Max Acc.:	72000 °/s²
Max Jerk:	2500 °/s³

Load Curve

Figure 47 Results of the measurement of the Cross Seal Axis

In figure 47, we can see the output of the measurement and proof that the application is able to provide the required functionality which it was intended to when connected to a real-life factory setup. In the figure we observe how the position, speed, accelerations are changing in a cyclic manner according to the movement of the axis in the machine. In addition, the load profile and operating point fall below the torque characteristic of the motor which confirms that a suitable motor was selected with a lot more room left to increase the torque due to the big gap between the two curves.

6. Conclusion

This thesis has examined the possibility of developing an Industrial IoT web application used in the field of factory automation and production machines. The work done for this thesis includes developing the back-end or the server-side of the application where the logic is being done, or in other words, the data is being processed. In addition, the server-side of the application includes the creation and administration of the database as well. Other parts of the project such as the front-end of the application was completely developed by other software developers at Siemens Digital Industries.

The application is able to connect to a PLC using its OPC server and read variables stored inside its memory. By selecting an axis from the list, the application is then able to detect the properties of this axis and allow the user to save the configuration. After finalizing the setup, the user is then able to send a measurement over MQTT to the application and perform visualization of the data in charts. The application has proved to be functional, practical and user friendly when tested with a real-life factory setup. Future work can be done to enhance the usability of the application such as adding compatibility with other types of PLCs other than SIMOTION and allowing the user to trigger measurements directly from the application rather than sending the data from another application using MQTT.

References

[1] "IIoT- the Industrial Internet of Things (IIoT)," [Online]. Available: https://www.i-scoop.eu/internet-of-things-guide/industrial-internet-things-iiot-saving-costs-innovation/industrial-internet-things-iiot/.

[2] "Official Siemens Documentation," [Online]. Available: https://documentation.mindsphere.io/resources/html/manage-my-sinumerik-edge-app-management/en-US/user-docu/industrialedge.html.

[3] "Siemens.com," [Online]. Available: https://new.siemens.com/global/en/products/automation/systems/motion-control/simotion-hardware/simotion-d.html.

[4] "OPC Foundation," [Online]. Available: https://opcfoundation.org/about/what-is-opc/.

[5] "OPC Data Hub," [Online]. Available: https://www.opcdatahub.com/WhatIsOPC.html.

[6] "MQTT.org," [Online]. Available: https://mqtt.org/.

[7] G. Kirckof, Servomotor Sizing and Application, International Society of Automation, 2012.

[8] "Servo Technik," [Online]. Available: http://www.servotechnik.de/fachwissen/auslegung/f_beitr_00_708.htm.

[9] "Medium," [Online]. Available: https://medium.com/@eugeneteu/intro-to-software-engineering-architecture-model-view-controller-c29805284de6.

[10] "Red Hat," [Online]. Available: https://www.redhat.com/en/topics/api/what-is-a-rest-api.

[11] "MDN Web Docs," [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods.

[12] J. Duckett, Web Design with HTML, CSS, JavaScript and jQuery Set, Wiley, 2014.

[13] "O'Rielly," [Online]. Available: https://www.oreilly.com/library/view/programming-javascript-applications/9781491950289/ch05.html.

[14] "Open Logic," [Online]. Available: https://www.openlogic.com/blog/what-sql-database.

[15] "Data Quest," [Online]. Available: https://www.dataquest.io/blog/sql-basics/.

[16] C. Fehily, SQL (Database Programming), Questing Vole Press, 2017.

[17] "Science Direct," [Online]. Available: https://www.sciencedirect.com/topics/computer-science/java-programming-language.

[18] "Oracle Documentation," [Online]. Available: https://docs.oracle.com/javaee/6/tutorial/doc/geysj.html.

[19] "Tutorials Point," [Online]. Available: https://www.tutorialspoint.com/spring/spring_web_mvc_framework.htm.

[20] "Vogella," [Online]. Available: https://www.vogella.com/tutorials/JavaPersistenceAPI/article.html.

[21] "Stack Abuse," [Online]. Available: https://stackabuse.com/controller-and-restcontroller-annotations-in-spring-boot/.

[22] "industry.siemens.com," Data exchange via OPC XML - Description and example for the data exchange, [Online]. Available: https://cache.industry.siemens.com/dl/files/938/27097938/att_78439/v1/faq_opc_xml_da_datenaustausch_v10_en.pdf.

[23] "Eclipse," [Online]. Available: https://www.eclipse.org/paho/index.php?page=clients/java/index.php.

[24] "Java T Point," [Online]. Available: https://www.javatpoint.com/what-is-angularjs.

[25] "Build.vsupalov," [Online]. Available: https://build.vsupalov.com/how-backend-and-frontend-communicate/.

[26] "Fastware," [Online]. Available: https://www.postgresql.fastware.com/what-is-postgresql.

[27] "PostgreSQL Official Documentation," [Online]. Available: https://jdbc.postgresql.org/documentation/head/intro.html.