



Stock management system for small and medium-sized companies

Matthieu Pierre Brühwiler

Haaga-Helia University of Applied Sciences

Information Technology Bachelor of Business Administration

Bachelor's Thesis

2023

Abstract

Author(s) Matthieu Pierre Brühwiler
Degree Information Technology Bachelor of Business Administration
Report/Thesis Title Stock management system for small and medium-sized companies
Number of pages and Appendix pages 48 + 1
<p>The aim of this thesis is to develop an open-source back-end for a stock management system. The latter goal is to allow small and medium-sized companies to manage their stock with a system that is easily adaptable to their needs. It is developed as a web application and is divided into two distinct parts: a database and a back-end. The database is the component that will store all the system's data. The back-end is an application that implements business logic through methods. These allow actions to be taken on the database in a secure manner.</p> <p>The outcome of this thesis is not intended to interact directly with users. In fact, a front-end will be developed by the author in the framework of another project, in order to allow full use of the system.</p> <p>This thesis report is divided into two parts, the first of which involves theoretical aspects. It begins by exploring the Java programming language. It follows that the added value of certain tools is used in this kind of application. It then explains the Spring Framework and the architecture of REST API. This first part ends by talking about a project management tool, used in various fields including IT. It is understood that this part includes the justification for the use of certain technologies and tools.</p> <p>The second part of this thesis highlights the phases of thesis development by the author. It begins by explaining the general environment of the thesis. Then it explains how the development was thought out and implemented. It ends by demonstrating the result of the project deployed through an Apache Tomcat server.</p>
Key words Back-end, Product stock, Spring Framework, REST API, Database

DEDICATION

*I would like to dedicate this thesis to the memory of my beloved grandfather,
who passed away while I was writing this thesis.*

Table of contents

1	Introduction	1
1.1	Project objectives	1
1.2	Scope & Limitations.....	2
1.3	Application measurables	3
1.4	Future of the project	3
2	Theoretical framework	4
2.1	Java briefly.....	4
2.1.1	History of Java	4
2.1.2	Developing and running in Java	4
2.1.3	Development tools	5
2.1.4	Well-known Java project.....	5
2.2	Apache Maven	5
2.2.1	Key concepts	6
2.2.2	Why Maven.....	7
2.2.3	Alternatives	7
2.3	JPA & Hibernate.....	7
2.3.1	JPA.....	7
2.3.2	Hibernate	9
2.4	Spring Framework.....	9
2.4.1	History	10
2.4.2	Spring modules.....	10
2.4.3	Spring projects.....	11
2.4.4	Spring MVC	11
2.4.5	Spring Data.....	12
2.4.6	Spring and OWASP	12
2.4.7	Spring alternatives	13
2.4.8	Why the choice of Spring	13
2.5	REST API.....	13
2.5.1	REST principles	14
2.5.2	REST & HTTP methods.....	14
2.5.3	Representation formats & HATEOAS.....	14
2.5.4	Security.....	15
2.6	Project Lombok	16
2.6.1	Features & Annotations.....	16
2.6.2	Why uses Lombok	18

2.6.3	Lombok and JPA.....	19
2.7	Trello.....	21
2.7.1	Trello's features	21
2.7.2	Importance of Trello	21
3	Empirical part	22
3.1	Starting point.....	22
3.2	Methodologies.....	22
3.3	Producing the outcome	23
3.3.1	Development phases	24
3.3.2	Deployment.....	32
3.4	Presenting the outcome	34
3.4.1	First steps	34
3.4.2	Routine use.....	36
4	Discussions.....	42
Sources	46
Appendices	49
Appendix 1.	GitHub repository.....	49

1 Introduction

Stock management is crucial for a company. Indeed, stocks are the products that a company owns, which are destined to be sold. In other words, the stocks represent the potential revenue for a company. Poor management of the latter can lead to a lack of stock. In this case, a company may lose the satisfaction of its customers. However, the opposite situation is also possible and implies unexpected additional costs. Whatever the above situation, it means that the company loses money. (IBM s.a.)

A loss of money due to lousy stock management can happen to any company. In order to avoid this problem, some companies are looking for a stock management system. This kind of system can be compared to the heart of a human because it is vital to the smooth running of a business. However, in certain companies, such as small ones, a stock manager may be too expensive for their budget. In the sense of this problem, the author seeks to create an open-source solution that addresses this problem. Through this project, he wishes to aim at small and medium-sized companies. Therefore, it is essential that the project can implement the business processes at lower costs while involving the open-source community for continuous improvement.

The author of the thesis elaborates on this project without a third party, meaning he does everything on his own. Consequently, he considered it as a starting point for a stock management system. Indeed, the result of the thesis can be used without modification. However, the author recommends adapting the system to a company. So that it meets the needs of each company, this adaptation can be made in several ways, either by modifying the database tables or implementing new business processes.

1.1 Project objectives

The goal of this thesis is to develop an open-source back-end for a stock management system. The focus of the thesis on the back-end is explained in Chapter 1.2, Scope and Limitations.

The development of an open-source application implies the use of open-source components. The project uses a MariaDB database to store the system's data. To simplify the database creation and the relationships between elements, the Spring Framework orchestrates the smooth working of the application. Therefore, the latter contains the implementation of the business processes. These processes are exposed securely through a REST API, enabling the system to communicate with others. However, the API is not aimed to be directly used by a human. Indeed, a front-end has to be implemented to make the application more user-friendly.

The back-end features can be classified into three categories, each having at least create, read, update, and delete (CRUD) operations—the first one is related to product management by allowing employees to monitor the stock of products in real time. The second one is used to manage the supplier and customer orders, to avoid a lack of inventory and help to fill customer orders. The last one concerns administrating and securing the application. Because each business process is secured by authentication and authorisation, it requires the management of employees (user accounts) and roles.

Finally, the back-end application should be able to be adapted to each company's needs while remaining secure and easy to be deployed through a web server.

1.2 Scope & Limitations

The thesis scope is based on developing a back-end application of a stock management system. However, this project requires a front-end application to be appropriately used. The aim is to permit small and medium-sized companies with a tight budget to manage their stock of products and supplier and customer orders. In the context of this thesis, the term “back-end” comprises implementing the business processes, which are exposed through a REST API. It also includes the management of the database, which means creating tables and manipulating the records in the database.

As regards the author, the realisation of this thesis allows him to deepen and demonstrate his knowledge of back-end development. First, with this thesis, he would like to improve his understanding of REST APIs. Indeed, nowadays, applications are generally developed following the microservices architecture. Therefore, the necessity for the use of API grows because all of them need to be able to communicate. (Haute Ecole de Gestion de Genève 2022). Secondly, the author is interested in creating an application that can potentially be used in real-life company cases. The latter's creation should include security because it is a relevant and essential aspect when developing an application. In the course of this thesis, the author wants to build his knowledge of creating applications using the Spring Framework. Thirdly, he also wishes to learn how to prepare an application for deployment and proceed to the deployment. To sum up, the author wants to create an application that can be used in a real case. Through this application, he can show his knowledge of developing and deploying a back-end application with Spring, which includes an API.

As mentioned earlier, the thesis only comprises the development of the back-end. Even though the REST API is not designed to be used directly by humans. The thesis does not include the elaboration of a front-end. Indeed, the system can be considered fully developed with a front-end. However, the front-end development is an overload for the available time.

Therefore, the thesis's author keeps the latter's development in another project. In addition, even if the back-end requires a MariaDB database, it does not include the installation and configuration of the database itself.

1.3 Application measurables

The following factors determine the success of the thesis and the application's quality. First, due to the project context, it must implement the basic features regarding stock management. It understands CRUD operations through a REST API for brands, categories, products, suppliers, customers, and orders. Additionally, the application must provide real-time monitoring of the stocks of products. It is also essential that a minimum of unit tests is written to ensure the proper functioning of the API endpoints. Regarding security, the application must require authentication and authorisation to access the API. It must use parametrised queries when accessing the database to prevent data leakage due to SQL injections. The application's usability should be taken into account. Therefore, each API method must have a comment in the code describing its purpose. Finally, the application must be thought to be adapted to a company's needs by modifying the database modelling and adding endpoints. These last factors describe the minimum requirements to consider the thesis successful.

However, since the author is developing the application alone, the following criteria are optional. First, it is relevant that he creates extensive unit tests by considering all the possible outcomes of each exposed method through the REST API. Second, the application security should also consider other measures, such as Cross-Site Request Forgery (CSRF) and Cross-origin resource sharing (CORS). Third, the application should be conducted to grow with the user company. Hence, it can be interesting to provide an application which can evolve in this direction. Finally, the latter should include documentation for each API endpoint to enhance its understanding. It should also comprise the main instructions to deploy the application.

1.4 Future of the project

Shortly after the publication of the thesis, a front-end application will be developed to complement the back-end application. The latter goal is to make the application more user-friendly and quicker to integrate into a company. It is also possible that new features may be added to the back-end application. Ultimately, due to the open-source of the application, other people than the author can provide new features for the application. Thus, a third party can improve any part of the application as he wishes.

2 Theoretical framework

2.1 Java briefly

The Java programming language is widespread and commonly used. Java is a high-level, strongly typed, and case-sensitive object-oriented programming language. Since its inception, it has been designed to be platform-independent. It means that if an operating system contains the Java Virtual Machine (JVM), it is possible to run all software coded in Java. Nowadays, Java is used for a wide range of applications, whether video games, mobile and web applications, or enterprise software. (Cosmina 2018, chapter 1)

2.1.1 History of Java

In the context of his work at Sun Microsystems, James Gosling had to develop a programming language. Initially called Oak, the purpose of this latter was to be used in large public electronic applications. Only after many years of development and with the help of several contributors the language was renamed. At the same time, this latter has also been refocused on the Internet. From the late nineties to the early two thousandths, Java had its first significant success. This success has led to the development of the language and its libraries. The language story has continued through these years until today, introducing new features, such as genericity, annotations, foreach loops, and many others. (Gosling, Joy, Steele & Bracha 2005, prefaces; Cosmina 2018, chapter 1)

It is also important to note that on 20th April 2009, Oracle purchased the Sun Microsystems company for \$7.4 billion. This purchase understands that Oracle now owns Java. Consequently, Oracle now develops Java. (Lohr April 2009, Oracle 2009)

The Java Development Kits (JDK) provided by Oracle are still free to use in production and redistribute. (Oracle 2021)

2.1.2 Developing and running in Java

As stated before, one of Java's strengths is that any operating system containing the Java Runtime Environment (JRE) can execute Java code. The JRE contains essential components like libraries and class files. It also includes another vital element: the Java Virtual Machine (JVM) responsible for managing the program execution. When you try to execute a Java program, the JVM receives the bytecode from the JRE and converts it into machine code. It is then that the code can be executed by the hardware. (Cosmina 2018, chapter 2)

However, when developing a Java application, the JRE is not enough. In this case, it requires to have the Java Development Kit (JDK). Indeed, the JDK contains a JRE with some additional tools used to compile, execute, debug, and test a Java code. (Cosmina 2018, chapter 2)

2.1.3 Development tools

As stated before, you must have a JDK when you want to write code in Java. Writing some code in a basic text editor like Notepad++ is possible. In this case, you will need to use command lines to compile and execute the code. Another way to write code in Java is to use dedicated tools. It exists as many tools as you want, although some are more convenient and user-friendly. It is also important to note that not all have the same public target and purpose. Among the specialised tools, we can find Visual Studio Code, Eclipse, BlueJ, and IntelliJ IDEA. In agreement with Iuliana Cosmina, IntelliJ IDEA is the most recommended editor. (Cosmina 2018, chapter 2)

From my experience, I started writing Java code with BlueJ, which gave me a good understanding of Java basics. However, BlueJ does not assist with syntax and code generation. This fact means that coding is slow and not practical. For this reason, I changed from BlueJ to IntelliJ IDEA. The latter is dedicated to Java and provides several features that can be extremely helpful for the developer. In contrast with BlueJ, many shortcuts exist to code faster while respecting code conventions. In addition, it also contains several embedded tools like Git or Maven. Installing plugins to personalise the IDEA as you want is also possible. As you might have understood, I used the IntelliJ IDEA during the project development.

2.1.4 Well-known Java project

The Java programming language is used in many projects that are now well-known. For example, NASA created an open-source desktop application in Java for their World Wind project. The popular video game Minecraft was also initially developed in Java. Other projects like Netflix, LinkedIn, and Uber use Java. (Suschevich Jun 2020, GitHub 2022)

2.2 Apache Maven

Apache Maven (Maven) is a software project management tool developed by the Apache Software Foundation since two thousand and four. Maven simplifies the build process, provides a uniform build system, presents quality project information, and fosters good development practices. By extending Apache Ant, Maven lets you download dependencies and reuse some Ant scripts. Maven provides several features that are intended to help developers while they are managing their projects. (Apache Maven 2023)

2.2.1 Key concepts

A Java project that uses Maven requires an XML file that contains general information and configuration indication about the project. This latter is a Project Object Model (POM), and Maven relies on it to build the project. In this file, it is possible to manage the global environment of a project, including dependencies, plugins, and lifecycle. (Apache Maven 2023)

A dependency in Maven refers to an external component (a JAR or a Java library) that a project needs. Accessing all Maven's dependencies from the Maven repository is possible. (Apache Maven 2023)

```
<groupId>fi.haagahelia.serverprog</groupId>
<artifactId>todoManager</artifactId>
<version>0.0.1-SNAPSHOT</version>

<name>todoManager</name>
<description>todoManager</description>

<properties>
  <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
  <project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>
  <java.version>1.8</java.version>
</properties>

<dependencies>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-jpa</artifactId>
  </dependency>
  <dependency>
    <groupId>org.postgresql</groupId>
    <artifactId>postgresql</artifactId>
    <version>42.2.1</version>
  </dependency>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-thymeleaf</artifactId>
  </dependency>
  <dependency>
    <groupId>org.thymeleaf.extras</groupId>
    <artifactId>thymeleaf-extras-springsecurity5</artifactId>
  </dependency>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
  </dependency>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-devtools</artifactId>
    <scope>runtime</scope>
    <optional>true</optional>
  </dependency>
</dependencies>
```

Figure 1. Example of a part of a POM file (self-made).

In Apache Maven, Maven's lifecycle is an ordered set containing separate phases used to build, test, and package a project. The lifecycle can be personalised by using a Maven plugin. We can define a plugin as an external software that can add additional features to the Maven build process. (Apache Maven 2023)

2.2.2 Why Maven

In the world of Java development, Maven is a well-known solution for managing a project. With Maven, the project structure and build process are standardised, so it offers a certain simplicity to understand the characteristics of a project. In addition, the build system provided by Maven can be used to compile, test, package, and deploy a project. At this, the tool offers an effortless way to manage dependencies with automatic downloads from the Maven repository. Moreover, Maven is often used with continuous integration tools like Jenkins to automate the building and testing process. (Baeldung 2020)

2.2.3 Alternatives

Although Apache Maven may be the most used project management tool, it exists several other tools. As you might guess, each tool has its features, benefits, and weaknesses. The most known alternative to Maven is Gradle. However, you can also use Apache Buildr or sbt. (Baeldung, 2023)

2.3 JPA & Hibernate

The Java programming language works with classes and relationships. This architecture has similarities with the one used in relational databases, which use tables instead of classes. Object-Relational Mapping (ORM) is a concept that can convert or translate objects of the oriented programming to a relational database and vice versa. To convert a Java class, the ORM relies on the class metadata written by the developer.

2.3.1 JPA

In the Java programming language context, JPA is a standard solution for persistence. Until 2017, JPA stood for Java Persistence API. However, Oracle has entrusted the Java EE project to the Eclipse Foundation since this date. As a result, Java EE was renamed to Jakarta EE. Nowadays, JPA means Jakarta Persistence API. (Späth 2019)

JPA provides a multitude of annotations and interfaces for mapping objects oriented into a relational database. For example, when we want to map a class with JPA, it is ubiquitous to find the following annotations: `@Entity`, `@Id`, `@GeneratedValue`, `@OneToOne`, `@OneToMany`, `@ManyToOne`, `@ManyToMany`.

```

@Entity
@Table(name = "BRU_CUSTOMER")
public class Customer {

    /* ----- FIELDS ----- */

    @Id 3 usages
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "cus_id")
    private Long id;

    @Column(name = "cus_first_name", nullable = false) 5 usages
    private String firstName;

    @Column(name = "cus_last_name", nullable = false) 5 usages
    private String lastName;

    @Column(name = "cus_email", nullable = false, unique = true) 8 usages
    private String email;

    /* ----- RELATIONS ----- */

    @ManyToOne(fetch = FetchType.LAZY) 3 usages
    @JoinColumn(name = "cus_geo_id")
    private Geolocation geolocation;

```

Figure 2. Example of a JPA entity representing a customer (self-made).

The `@Entity` is a class-level annotation that indicates that a class is a persistent entity. This annotation leads to setting a field with the `@Id` annotation to indicate the primary key. It is possible to automatically generate the primary key using the `@GeneratedValue`, which is a field-level annotation. In the case of relationships between classes, four field-level annotations exist (`@OneToOne`, `@OneToMany`, `@ManyToOne`, and `@ManyToMany`) can map the relationships between the database's tables. (Baeldung 2022)

By using JPA, it is possible to create queries and execute them. JPA provides several possibilities, including Java Persistence Query Language (JPQL), criteria API, and native SQL. JPQL is a language like SQL, and it is possible to create a JPQL query using an entity manager's "namedQuery" method.

```

@Query(value = "SELECT o.isSent FROM CustomerOrder o WHERE o.id = ?1")
Boolean isCustomerOrderSentByOrderId(Long id);

```

Figure 3. Example of JPQL query on the customer order table (self-made).

According to Oracle, we can define the Criteria API as a way to write JPA queries. The criteria API enables developers to write type-safe queries. JPQL and Criteria API are likely, and both have similarities in their syntax. (Oracle 2013, 661)

```

EntityManager em = ...;
CriteriaBuilder cb = em.getCriteriaBuilder();
CriteriaQuery<Pet> cq = cb.createQuery(Pet.class);
Root<Pet> pet = cq.from(Pet.class);
cq.select(pet);
TypedQuery<Pet> q = em.createQuery(cq);
List<Pet> allPets = q.getResultList();

```

Figure 4. Example of criteria query (Oracle 2013, 661)

As said before, it is possible to create a native SQL query with JPA by using the “createNativeQuery” of an entity manager.

```

Query query = em.createNativeQuery("SELECT * FROM city WHERE id = 8");
City result = (City) query.getSingleResult();

```

Figure 5. Example of native SQL query (self-made).

However, it must be noted that JPA cannot persist objects by itself. Indeed, an ORM framework such as Hibernate is required to implement JPA.

2.3.2 Hibernate

Hibernate is an ORM framework and a data management tool for the Java programming language. It enables any application to interact with a relational database while using objects. Hibernate provides a wide range of features, including persistence for SQL and NoSQL and implementing the Java Persistence API through the entity manager. (Bauer, King, Gregory 2016, part 1; Keith, Schincariol, Nardone 2018, chapter 1)

A key benefit of Hibernate is that a developer can write less code to access data. Indeed, based on the mapping, Hibernate can automatically generate SQL queries. Another significant benefit of it is the way that it can manage relationships between entities. The latter can manage an inheritance, one-to-one, one-to-many, and many-to-many relationships by considering them when putting them in the database. (Bauer & al. 2016, part 1)

2.4 Spring Framework

The Spring Framework is a well-known Java-based application framework which is open-source. Since its inception, the framework has been developed to simplify the development of applications. Nowadays, it can be used in several kinds of projects, such as microservices, web applications, and enterprise applications. (Spring 2023)

2.4.1 History

The history of the Spring framework began in 2003 when it was first released. This first release was easy to set up and configure and included the design pattern inversion of control (IoC). Because of its simplicity and innovation, the framework quickly gained popularity. Over the years, the framework has been improved and now includes several features such as Aspect-Oriented Programming (AOC), Model-View-Controller (MVC), Data Access, security and many more. Nowadays, the Spring Framework is the most used Java framework in the world of enterprise applications. In addition, it can be used in many kinds of applications, whether web services, micro-services, or serverless applications. The framework has a vast and active community that contributes to developing it. (Walls 2022, chapter 1)

2.4.2 Spring modules

The Spring Framework is organised into different modules. Each module has its features and responsibilities. The first module is the core technologies. It contains several implementations, such as the principle of Inversion of Controller (IoC) and Dependency Injection (DI). The latter combines two design patterns that provide a flexible and performant system that manages an object's dependencies. The core module also offers other features, such as Spring AOT, which enable developers to create an image of their Spring application. (Walls 2022, chapter 1; Cosmina, Harrop, Ho, Schaefer 2017, chapter 1)

The second module is named data access. This module includes the various aspects that Spring has consistent access to handle data. The framework takes them using the Java Transaction API (JTA), the Java Persistence API (JPA), and the Java Database Connectivity (JDBC). In addition, the Spring framework can work with multiple databases like SQL or NoSQL databases and in-memory data storage. (Baeldung 2022, Spring 2019)

As another Spring module, there is one for web development. The latter provides support to enable the developer to create web applications. One of the strengths of the Spring web module is the wide range of features and abstractions provided, which includes the manipulation of HTTP requests and responses. In addition, it has many vital components as Spring model-view-controller (MVC), the support of web sockets, and the ability to build reactive web applications. (Baeldung 2022, Spring 2019)

Integration is another Spring module that manages Enterprise Integration Patterns (EIPs). In other words, the framework can communicate with external architecture using diverse ways like API. (Baeldung 2022, Spring 2019)

The last Spring module is testing. The module provides a way to create and integrate unit tests of a Spring application. In addition, it gives some annotations and classes that facilitate testing Spring components, like controllers or repositories. (Baeldung 2022, Spring 2019)

2.4.3 Spring projects

Although the Spring Framework has many features grace to its modules, one of the most valuable parts of Spring is its projects. Indeed, many projects have been developed on top of the framework. Among this project, **Spring Boot** provides an effortless way to create and configure Spring-based applications. **Spring Cloud** contains several tools and libraries for building and deploying applications in the cloud. **Spring Security** is a tool that can be personalised to secure an application using authentication and authorisation or other features. Finally, **Spring Mobile** is a part of **Spring MVC** used to create mobile web applications. (Baeldung 2022, Spring 2023)

2.4.4 Spring MVC

Model-View-Controller (MVC) is a design pattern that helps implement the application's presentation layer. The latter aims to define which application participant is responsible for what. With MVC, it is determined that the application has three participants. The model participant is accountable for the business data of the application, depending on the user's context. The view participant manages how to format the data to be readable. The controller handles the users' requests by updating the model and redirecting the users based on their actions. (Cosmina & al. 2017, chapter 16)

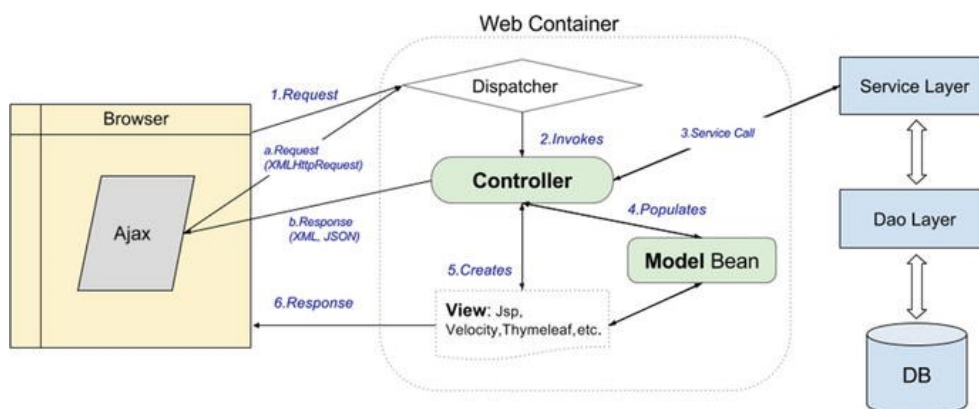


Figure 6. MVC pattern of a web application (Cosmina & al. 2016).

The Spring framework includes Spring Web. This latter includes multiple features, such as Spring MVC, which can implement the MVC design pattern. It can manage HTTP requests and respond to them. The output (view) of the Spring MVC can produce HTML content or a Restful application programming interface. (Cosmina & al. 2017, chapter 16)

2.4.5 Spring Data

The Spring data project is a project that can support various kinds of databases. In other words, it is an easy-to-use API that supports persistence on relational databases, NoSQL databases and data grids. One of the main objectives of Spring Data is to reduce the required code written by the developers to connect to a database. It provides many features, such as automatic query generation, transaction management, pagination, and sorting of the retrieved data. Another advantage of Spring data is that it provides some community modules to expand the number of compatible database types. (Spring 2023)

2.4.6 Spring and OWASP

The Open Web Application Security Project (OWASP) provides a top ten of the most common vulnerabilities found in web applications. According to OWASAP Top 10 2021, it is crucial to consider the following vulnerabilities during the development of one of the latter. The web application should be protected against injections, cryptographic failures, broken access control, logging and monitoring failures, server-side request forgery, identification and authentication failures, and software and data integrity failures. It is also essential that the application does not use vulnerable and outdated components while avoiding misconfiguration situations. The development must include security as a design principle. (OWASP 2021)

The Java Spring Framework provides several features that can be used to avoid the vulnerabilities listed above. As stated before, the Spring Framework is divided into different projects. Each of them provides features and a security layer to prevent attacks. Regarding SQL injections, the Spring data project offers a way to create parameterised queries that avoid string concatenation. In addition, Spring Boot prevents other kinds of injections, which provides a built-in Cross-Site Request Forgery (CSRF) protection mechanism. Many features are supported to avoid broken authentication, such as authentication, authorisation, multifactor authentication, session management, and password hashing.

Moreover, cryptographic algorithms are also provided to enable developers to secure sensitive data storage. Spring also offers mechanisms to avoid misconfiguration situations. Indeed, these include auto-configuration, content security policies, and support to encourage the use of secure protocols such as HTTPS. It is also important to note that using Maven or Gradle can help find vulnerabilities through dependencies. The latter can scan and detect the imported components to ensure each is unrelated to a known vulnerability. Apropos of logging and monitoring, some popular frameworks, such as Log4J, are also integrated with Spring. (OWASP s.a.)

2.4.7 Spring alternatives

It exists various alternatives to the Spring Framework. We can categorise these alternatives into two categories: Java alternatives and other programming language alternatives. In Java alternatives, it is possible to use Jakarta EE (Java EE), Micronaut, Quarkus, Play Framework, and many others. On the other hand, there are a lot of alternatives. We can take the example of Django, which is used in Python, Ruby on Rails for the Ruby programming language, or Laravel, a framework available in PHP. (Fol 26 October 2021; Kaluža, Kalanj & Vukelić 7 March 2019)

2.4.8 Why the choice of Spring

In the context of my studies at the “Haute Ecole de Gestion de Genève” – University of Applied Sciences and Arts Western Switzerland, I have been led to develop a Rest API with Django. It includes that the project was developed in Python and not in Java. Because my studies have been focused on Java, I wanted to learn to do the same kind of project using Java. At the same time, I had a course that was an introduction to creating web services in Java without Spring. At this point, I started to be motivated to develop web applications. During my first semester at the University of Sciences Haaga-Helia, I followed the “Server programming” course given by Jukka Juslin. This course taught me how to develop web applications with the Spring Framework. As a result of this course, I decided to continue learning about this framework to improve my skills and knowledge. Moreover, Spring is open-source, well-known and often used in the context of applications developed in Java, and not only for web applications. Finally, Spring has extensive documentation and provides several tools and libraries that can be used to facilitate the development and deployment of applications.

2.5 REST API

According to the course “Services Implementation” given by Dr Stefan Behfar (2022) at the “Haute Ecole de Gestion de Genève” – University of Applied Sciences and Arts Western Switzerland, the way to develop applications has changed through the years. A few years ago, the applications were designed in a monolithic manner. As of today, applications are designed in a microservices way. As a result, each system must be able to communicate with others. To enable systems to communicate, the developers must include an application programming interface (API) in their applications. Various API architectures exist, such as REST, GraphQL, or SOAP. (Haute Ecole de Gestion de Genève 2022)

2.5.1 REST principles

REST is for Representational State Transfer, and it is an architectural style that is used to design web-based (HTTP) API. The REST architecture is based on five elementary principles.

The first principle is **addressability**, which means that each resource must be represented by one or several representations. Each representation of a resource has one URL. However, a single resource representation must define only one resource. The second principle is the **uniform interface**. Each interface should be simple, self-descriptive, and easy to understand, and all the resources must use the same interface. The third principle is the fact that the API must be **resource-oriented**. It means that the resources should be represented in a specific format. It can be JSON, XML, or plain text. The fourth principle is **statelessness**, meaning all queries must be independent. In other words, each client can navigate through the API and have the same result. The server does not save interactions with the client. The last principle is **connectedness** (HATEOAS), which means that each resource representation provides links with actions. (Haute Ecole de Gestion de Genève 2022)

2.5.2 REST & HTTP methods

As stated, REST is a client-server architecture based on the Hyper Text Transport Protocol (HTTP). It is why a REST API always uses the four HTTP verbs: GET, POST, PUT, and DELETE. The purpose of using HTTP verbs is to help to identify what will be performed with a request.

The GET verb is used to retrieve the data of a resource through the HTTP request. The client will invoke this verb with the endpoint to target which resource he wants to access. The POST verb is used to create new data in the system. For example, the client provides further information that will be saved in the database. In order to update an existing resource, it is possible to use the PUT verb. The client must send the complete resource representation when he wants to perform a PUT request. Finally, deleting a specific resource by using the verb DELETE is possible. (Haute Ecole de Gestion de Genève 2022)

2.5.3 Representation formats & HATEOAS

As stated before, the response body's format of a REST API should be resource-oriented. Therefore, an API will return the body in XML or JSON format. XML means Extensible Markup Language, which has some similarities with the Hyper Text Markup Language (HTML). The XML format is used to share structured information between different entities.

On the other hand, JavaScript Object Notation has the same use as XML. However, the representation of data in JSON is based on attribute-value pairs. (Haute Ecole de Gestion de Genève 2022)

In 2013, Todd Fredrich stated that as long as costs associated with sharing in XML and JSON are not staggering, it is beneficial to provide both formats. However, if this is not the case, it is better to favour the JSON format. In addition, from Dr Stefan Behfar, returning the XML format through an API developed with Java may require annotating the concerned classes. In the case of having a Java class with annotation related to persistence and for other use, such as Lombok, adding annotation for the XML format could overload the class. (Fredrich 2013; Haute Ecole de Gestion de Genève 2022)

One of the REST constraints is to include a hypermedia text link as a part of the representation. This concept is named HATEOAS for Hypermedia As The Engine Of Application State. This constraint is undoubtedly one of the most significant differences between the REST architecture and the others. The links in the representation enable the client to discover and navigate through the API. The purpose of this constraint is to make the architecture more flexible to change by allowing the API to evolve without breaking the client application. (Fredrich 2013; Haute Ecole de Gestion de Genève 2022)

```

{
  "id": 1,
  "name": "Bonito Flakes - Toku Katsuo",
  "description": "sit amet nunc viverra dapibus nulla suscipit ligula in lacus curabitur at ipsum ac",
  "salePrice": 68.61,
  "stock": 13,
  "_links": {
    "self": {
      "href": "http://localhost:8080/api/products/1"
    },
    "products": {
      "href": "http://localhost:8080/api/products"
    },
    "brand": {
      "href": "http://localhost:8080/api/brands/22"
    },
    "category": {
      "href": "http://localhost:8080/api/categories/10"
    }
  }
}

```

Figure 7. REST representation of a product with HATEOAS (self-made).

2.5.4 Security

The REST architecture does not necessarily require the securing of an API. However, a REST API may be a part of a computerised information system in which data is shared. Furthermore, an API is not necessarily in an isolated network; the requests of an API can be made through the Internet.

Out of these facts, securing and protecting REST services is essential. According to OWASP, securing one of those services must include the following specifications. Firstly, the services should implement and provide strong authentication and authorisation to restrict access to data. Secondly, the use of HTTPS should be the only one authorised. The latter offers secure data in transit, such as credentials or JSON Web Tokens (JWT). Thirdly, the data entered by the user should be validated to avoid injection attacks such as SQL injections. Fourthly, to prevent attackers from performing attacks on behalf of a user, the REST API should establish a Cross-Site Request Forgery (CSRF) protection. Fifthly, in the case of a server error, the error message must not provide protected information. Sixthly, as far as possible, it should be helpful to implement versioning of the API resources to avoid breaking situations due to unexpected situations. Finally, the system needs to provide rate-limit access to the services. It will protect the server against brute force or denial of service attacks. (OWASP, s.a.)

Implementing other protection measures upstream of the REST services is also possible and essential. These latter will add another protection layer to the services. Among these protection measures, the setup of firewall rules and access lists may be used to restrict access to specific endpoints. Wherever possible, sensitive endpoints, such as management ones, should not be exposed over the web. If this is not possible, it may require the implementation of multifactor authentication. (OWASP, s.a.)

2.6 Project Lombok

The Project Lombok is a Java library developed since 2009. This project aims to reduce the non-valuable and boilerplate code in Java. This enables us to easily read, write, and maintain the Java classes. Furthermore, it provides features that can be used through annotations to simplify the developer's life. Over the years, the Project Lombok has gained popularity and is increasingly recommended when developing in Java. (Baeldung 2022, Lombok s.a.)

2.6.1 Features & Annotations

As stated before, the Lombok provides several annotations. Among these annotations, the most common ones are:

`@Getter` / `@Setter` that are used to auto-generate getters and setters. These two annotations can be used at the class level or directly at the attribute level. Using it at the class level automatically generates getters and setters for the entire class's attributes. However, it is possible to restrict the exposition of getters and setters only for defined fields.

```

import lombok.AccessLevel;
import lombok.Getter;
import lombok.Setter;

public class GetterSetterExample {
    /**
     * Age of the person. Water is wet.
     *
     * @param age New value for this person's age. Sky is blue.
     * @return The current value of this person's age. Circles are round.
     */
    @Getter @Setter private int age = 10;

    /**
     * Name of the person.
     * -- SETTER --
     * Changes the name of this person.
     *
     * @param name The new value.
     */
    @Setter(AccessLevel.PROTECTED) private String name;

    @Override public String toString() {
        return String.format("%s (age: %d)", name, age);
    }
}

```

Figure 8. Getters and Setters on a Java class (Lombok s.a.).

The `@ToString` and `@EqualsAndHashCode` annotations are used to automatically generate “to string”, “equals”, and “hash code” methods. It is also possible to exclude an attribute of one of these methods using the annotation with a dot exclude before the element concerned.

```

@ToString no usages
@EqualsAndHashCode
public class DTOCustomer {

    /* ----- FIELDS -----

    private Long id; no usages

    @EqualsAndHashCode.Exclude no usages
    private String firstName;

    @EqualsAndHashCode.Exclude no usages
    private String lastName;

    private String email; no usages

    /* ----- RELATIONS -----

    @JsonIgnore no usages
    @ToString.Exclude
    @EqualsAndHashCode.Exclude
    private GeolocationDTO geolocationDTO;

}

```

Figure 9. To string, equals, and hash code with Lombok (self-made).

It is also possible to automate the generation of constructors of a Java class. First, the `@NoArgsConstructor` allows the creation of a constructor without arguments. Next, the `@AllArgsConstructor` creates a constructor with all the attributes in the parameter. The last annotation is the `@RequiredArgsConstructor` which can create a constructor with some specific attributes.

```

@Data 14 usages
@AllArgsConstructor
@NoArgsConstructor
public class ProductCuDTO {

    /* -----
private Long id; no usages
private String name; no usages
private String description; no usages
private Double purchasePrice; no usages
private Double salePrice; no usages
private Integer stock; no usages
private Integer minStock; no usages
private Integer batchSize; no usages
private Long brandId; no usages
private Long categoryId; no usages
private Long supplierId; no usages
    }

```

Figure 10. Create constructors with Lombok (self-made).

Project Lombok provides many more annotations, including `@Data`, which replace `@Getter` `@Setter` and `@RequiredArgsConstructor` and reduce the number of annotations. It is also possible to implement the builder design pattern using the `@Builder` annotation or to instantiate a logger (`@Log4j`, `@Slf4j`, `@JBossLog`, etc.).

2.6.2 Why uses Lombok

Nowadays, there is an increasing demand for simple code that is easily read and improved. Therefore, the methods like to-string, getters and setters do not bring value to the code. It is binding to write and maintain these methods. In addition, they are amenable to change depending on the class's attributes, which is also a constraint. (Baeldung 2022)

In order to address this problem, some tools like IntelliJ IDEA include features to generate this boilerplate code, like getters and setters automatically. However, those tools only respond partially to the problem. Indeed, it is easy to create the code, but it does not make a class more readable. The added value of Project Lombok is that those methods are not hard coded in the Java class.

Indeed, this reduces the amount of code in the class and its readability. Furthermore, when we change the attributes of a class, Lombok will automatically handle the update.

2.6.3 Lombok and JPA

I used JPA and Lombok to develop the stock management system. However, I decided not to use both in the same Java classes. The first reason for this choice is related to the annotations. As mentioned above, both projects are based on the use of annotations. Depending on the complexity of a Java class, using the JPA's annotations might sometimes overload it.

```

@Entity
@Table(name = "PRU_PRODUCT", uniqueConstraints = {
    @UniqueConstraint(columnNames = {"pro_name", "pro_sup_id"})
})
public class Product {

    /* ----- FIELDS ----- */

    @Id 6 usages
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "pro_id")
    private Long id;

    @Column(name = "pro_name", nullable = false) 7 usages
    private String name;

    @Column(name = "pro_description", nullable = false) 7 usages
    private String description;

    @Column(name = "pro_purchase_price", nullable = false) 4 usages
    private Double purchasePrice;

    @Column(name = "pro_sale_price", nullable = false) 4 usages
    private Double salePrice;

    @Column(name = "pro_stock") 4 usages
    @Min(0)
    private Integer stock;

    @Column(name = "pro_min_stock", nullable = false) 4 usages
    @Min(0)
    private Integer minStock;

    @Column(name = "pro_batch_size", nullable = false) 4 usages
    @Min(2)
    private Integer batchSize;

    /* ----- RELATIONS ----- */

    @ManyToOne(fetch = FetchType.LAZY, optional = false) 3 usages
    @JoinColumn(name = "pro_bra_id")
    private Brand brand;

```

Figure 11. Product class as a JPA entity (self-made).

As you can see, it may reduce readability, and code maintenance may be more challenging. In the case of this example, adding Lombok's annotations will unfortunately not compensate for the problem.

To reduce the boilerplate code, the constructors, equals, and hash code, to string, getters, and setters will be replaced by adding dedicated annotations. Implementing the desired logic with Lombok may require a lot of annotations, as shown in the image below.

```

@Entity no usages
@Table(name = "PRU_PRODUCT", uniqueConstraints = {
    @UniqueConstraint(columnNames = {"pro_name", "pro_sup_id"})
})
@Getter
@Setter
@ToString
@NoArgsConstructor
@RequiredArgsConstructor
@AllArgsConstructor
public class ProductLombok {

    /* ----- FIELDS ----- */

    @Id no usages
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "pro_id")
    @NonNull @EqualsAndHashCode.Include @ToString.Include
    private Long id;

    @Column(name = "pro_name", nullable = false) no usages
    @NonNull @EqualsAndHashCode.Include @ToString.Include
    private String name;

    @Column(name = "pro_description", nullable = false) no usages
    @NonNull @EqualsAndHashCode.Exclude @ToString.Include
    private String description;

    @Column(name = "pro_purchase_price", nullable = false) no usages
    @NonNull @EqualsAndHashCode.Exclude @ToString.Include
    private Double purchasePrice;

    @Column(name = "pro_sale_price", nullable = false) no usages
    @NonNull @EqualsAndHashCode.Exclude @ToString.Include
    private Double salePrice;

    @Column(name = "pro_stock") no usages
    @Min(0)
    @EqualsAndHashCode.Exclude @ToString.Include
    private Integer stock;

    @Column(name = "pro_min_stock", nullable = false) no usages
    @Min(0)
    @NonNull @EqualsAndHashCode.Exclude @ToString.Include
    private Integer minStock;

    @Column(name = "pro_batch_size", nullable = false) no usages
    @Min(2)
    @NonNull @EqualsAndHashCode.Exclude @ToString.Include
    private Integer batchSize;
}

```

Figure 12. Product class as a JPA entity, including Lombok's annotations (self-made).

Due to these reasons, I decided to only use the Lombok annotations in the classes used for the presentation layer (returned through the API). Thus, these classes are easy to read and maintain because they only contain attributes and two or three annotations.

In order to resolve this problem stated above, a solution would be for the Lombok project to provide dedicated annotations for the JPA entities, which would merge the JPA's annotations with those from Lombok.

2.7 Trello

Trello is a web project management tool designed to make project and task management easier. Developed since 2010 and launched in September 2011, it quickly found its place in the market. Nowadays, it records almost two million teams that use Trello worldwide. This success is due to its flexibility and simplicity to use. Moreover, the tool can be used with any project and customised as needed. (Trello s.a.)

2.7.1 Trello's features

Because Trello is a web-based project, it is cloud-based. It means the tool can be directly used in the web browser over the Internet and is not needed to install any software. The only requirement is to create an account to access the tool. Registered users can create as many boards as he wants for each project. Then, it is possible to create lists on a board, to manage the distinct stages of a task. To describe a task, Trello allows for making cards and attaching related information such as members, dates, checklists, attachments, etc.

In order to decrease the time spent in project management, an automated workflow can be set up based on rules for almost any action. Finally, Trello also integrates boards in other environments, like Gmail, Salesforce, or others. (Trello s.a.; Johnson 2017)

2.7.2 Importance of Trello

Within the framework of this project, I use Trello to manage the different programming-related tasks. Therefore, I created a board with seven lists representing distinct stages of progress. Then, I added multiple cards, each of which describes a task of the project. As the application was being developed, I moved the cards from list to list based on their progression. As a result, the tasks were organised and sorted, so I could quickly know what I had to do.

In the context of the project's development, Trello was beneficial. As stated before, it enables me to manage the task that I must do. Johnson (2017) says that using Trello for personal and professional projects, if they are not large-scale projects, allows them to be completed on time. Based on my own experience, it is true that the tool is very convenient and can make it possible to manage any project.

3 Empirical part

3.1 Starting point

The project aims to create an open-source stock management system for small and medium-sized companies. Because of the open-source principle of the project, the project must use components in the same direction. The project has not been conducted with a third party, so it is not targeted at a company. Therefore, the outcome can be used without modification in different business contexts. However, the system can be modified or adapted to the needs of a company.

The implementation of the project is made to meet the basic needs of companies in terms of stock management. These include product stewardships (tracking inventory level, purchase, and sales prices, etc.), managing supplier and customer orders (shipping, receiving, sending order details), stock optimisation, which determines which products must be ordered, and analysis of employee sales.

It is also crucial that the project meets the challenges of inventory management for small and medium-sized companies. These companies can be confronted with a lack of real-time visibility, leading to stock loss and additional expenses. In addition, because the project can be adapted to specific needs, it can manage more processes, usually done manually. As a result, it can reduce the time spent on non-valuable operations and the risk of error. Finally, an open-source system brings a cost-effective solution that avoids expensive expenses while remaining scalable.

Regarding limiting factors, I prefer to concentrate on the quality of the features instead of quantity. One of the objectives is to provide a functional back-end that can be easily used. In addition, we must not forget that I am the only developer of this project. Accordingly, depending on the problems encountered, or a lack of time due to an external event, it is possible that some features are not implemented.

Chapter 1.3 already describes the qualitative criteria of the application. These latter allow delimiting the success of the thesis. However, the thesis can be considered successful if all the minimum standards are met.

3.2 Methodologies

In a general manner, I am a person who needs to be organised and to know what to do and when. Therefore, I always use my task manager developed in the Server Programming course at Haaga-Helia. The latter's purpose is to plan the work I must do by the priority level. In this way, I have been able to get organised between this project's thesis and external tasks.

However, this project was insufficient to manage the thesis application development correctly. As a result, I decided to use another tool only for this purpose. I decided to use the Trello web application, which I discussed in Chapter 2.7. The latter offers the possibility to implement the Kanban approach, an Agile framework commonly used in project development focused on reducing the time needed to develop a feature. (OpenClassrooms, 2022) The choice of this organisation is based on avoiding overwork and having a quick overview of the project. Indeed, Trello enabled me to create an store in the cloud on my Kanban board. This board regroups the product backlog and the possible status of the tasks I have to do. These statuses categorise the tasks into four groups: “to do”, “in progress”, “to test”, and “completed”. Through this board and these categories, I could have an accessible overview of the project's progress.

During the whole development of the project, I also had to organise for managing the thesis files. Whether it is the written part of the application, keeping different versions of them in two environments is essential. In order to have redundancy, both were stored locally on my computer and in two different GitHub repositories. The first repository is intended for the application, and the other one is for all the other files, such as the written part or my notes.

The quality of the code is also an essential element. A code that does not respect conventions has no comments, is too long and complex, and is difficult to maintain. Therefore, during the application development, I tried as much as possible to describe the importance and purpose of each method. Furthermore, I applied as much as possible the good practices of the Java language, like the "camelCase", which I learned during my studies.

3.3 Producing the outcome

The final goal of this project is to create a back-end application. The back-end is made in Spring and should be able to manage a company's stocks and orders. In order to deal with them, the back-end application provides several methods through REST services.

Nevertheless, the purpose of a REST API is to enable communication between two systems. It means the user (the company's employees) will not directly interact with the back-end through the API. Therefore, the user will perform actions using a front-end application.

We can illustrate the previous scenario with the diagram below. The pink box represents the context of this thesis. In fact, developing a front-end is an external component of this thesis.

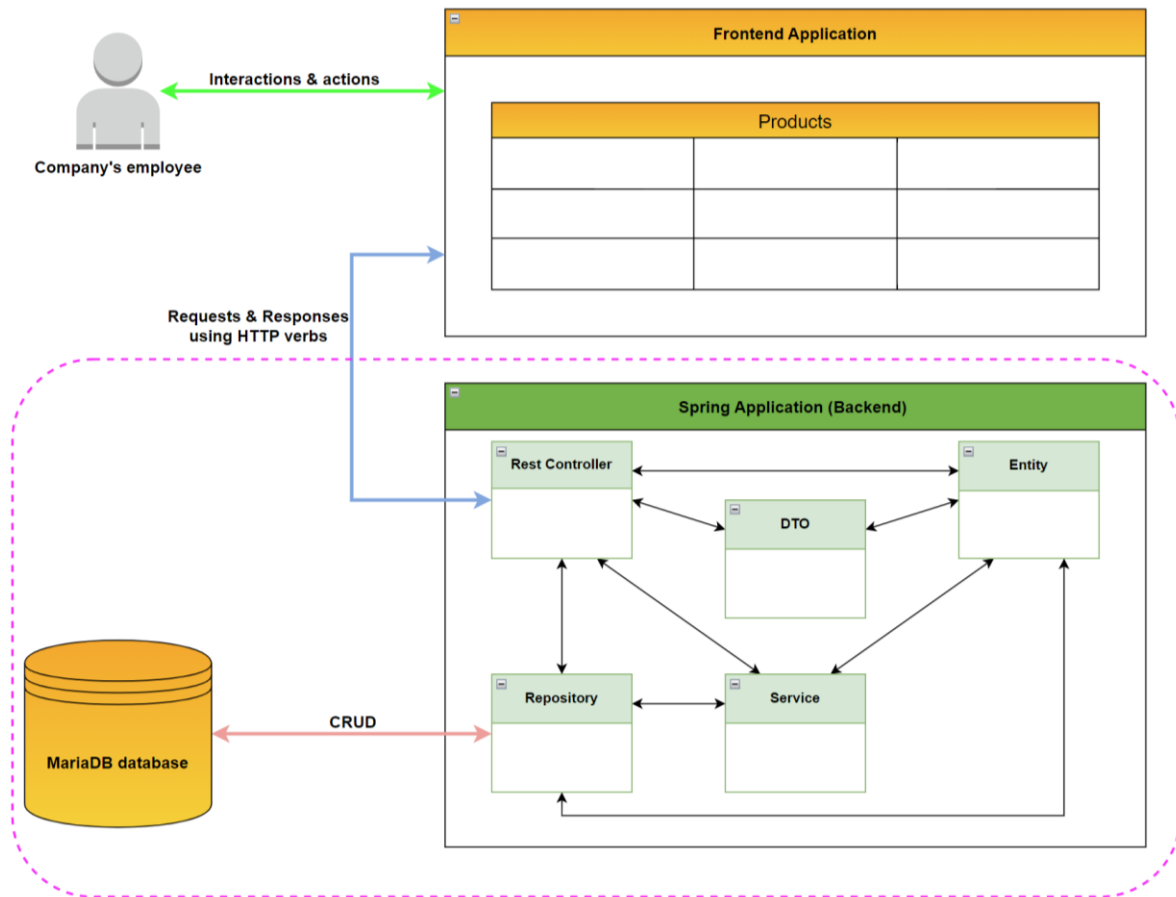


Figure 13. Final project environment and entities interactions. (self-made)

As we can see in the diagram below, when the server receives the request, it redirects it to the corresponding REST controller based on the URL routes. Depending on the request, the controller calls the requested repositories or services. The latter performs CRUD actions on the database and returns the entity to the controller. Before sending the response to the client, the controller converts entities into DTO objects.

3.3.1 Development phases

The development of this project took place in several stages. The first stage took place after establishing the context of the thesis. At this point, I knew I would like to create the back-end of stock management, but I still had to define how to do it. As stated in Chapter 2.4.8, the choice of the Spring framework is based on various criteria. It includes my background and the advantages of the framework, such as its extensive documentation and community. During this first stage, I also had to choose a database for the project. The choice of a relational one was evident in this kind of project because of the relationships between each object. I chose MariaDB because of its open-source performance and compatibility with Spring.

The second phase involves deciding to work in an Agile manner, following the SCRUM methodology. I decided to integrate the method of the project because, nowadays, it is widely used in software development. Due to this fact, I created a product backlog that included the project requirements and the features which I wanted the outcome to have. By initiating the project development, I quickly realised that integrating the SCRUM methodology was overwork since I was alone to work on the project. Finally, I kept the product backlog and broke it into several sub-tasks. These are the ones I used to manage the project's progress via Trello.

The next stage was database modelling, based on the defined product backlog. First, I created a conceptual data model (CDM) that includes what the system contains. Secondly, I transformed the CDM into a logical data model (LDM). This transformation determines how the data will be organised into a relational database. I used the "draw.io" application to draw the two previous models. However, I decided not to create a physical data model (PDM) because Hibernate and JPA can generate the database scheme automatically. The CDM and LDM also helped me to set up the required Java classes. In addition, due to the ultimate version of IntelliJ, I can automatically generate the PDM. I will show the latter about the Java classes and JPA in the stage.

Another key point is the project's adaptability to the needs of companies. It was necessary to take this into account to allow future modification of the database. Ultimately, to make the project more compliant with the General Data Protection Regulation (GDPR), it contains the minimum amount of information about the customers. Only the most essential information is stored. In case of a data deletion request, it is possible to separate the customer from his orders to delete his data. However, it is impossible to delete sales orders to keep track of the company's sales.

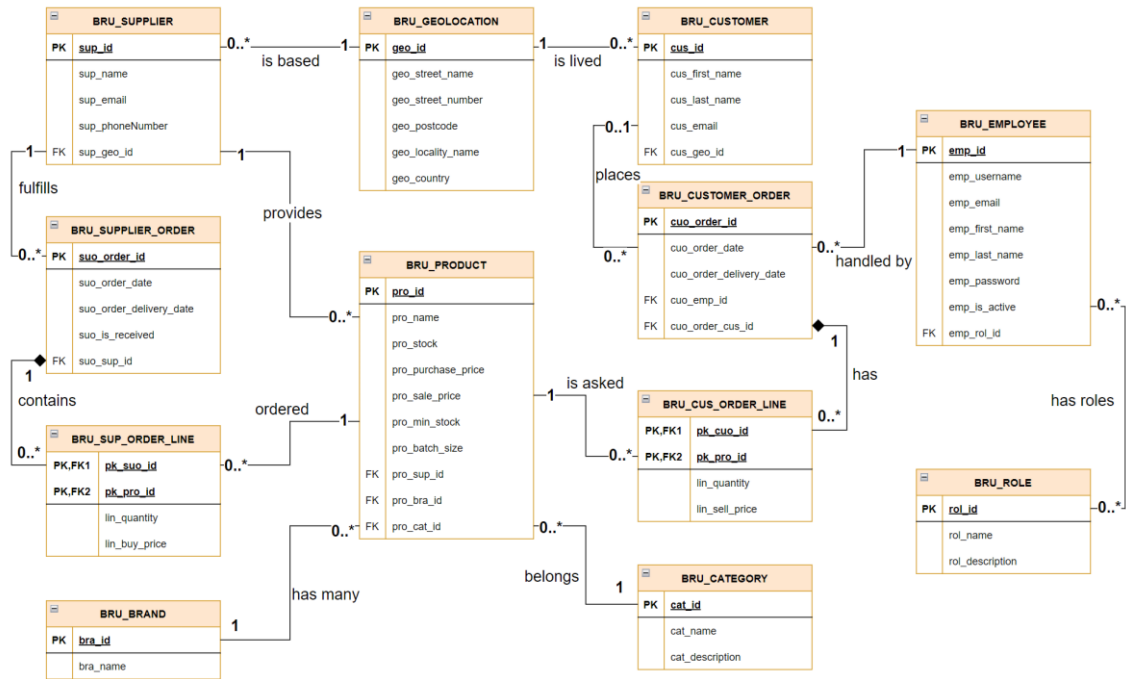


Figure 14. LDM diagram of the project. (self-made)

This stage was the last stage before starting to code and was concerned with selecting dependencies and creating the Java Spring environment.

Searching for dependencies manually in the Maven repository and creating the project structure by hand is possible. However, doing it this way is a waste of time. That is why the Spring Initializr has been made. It is a REST API web application that can create the skeleton of a Spring project and quickly find compatible dependencies. There are several ways to access this project, such as using the web application: <https://start.spring.io/>. It can also be included in another software, as with IntelliJ IDEA. (Walls 2022, chapter 1)

In the context of this project, due to the use of IntelliJ, I created my Spring project in the IDEA. After only a few clicks and providing information about the project, the coding environment is generated automatically. This does not take long and is easy to do. As mentioned in Chapter 3.2, during the whole development, the project has been saved in a GitHub repository, available in the appendix section.

```

<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>

  <parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>3.0.0</version>
    <relativePath/> <!-- lookup parent from repository -->
  </parent>

  <groupId>fi.haaga-helia</groupId>
  <artifactId>StockManager</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <name>StockManager</name>
  <description>Stock management system</description>

  <properties>
    <java.version>17</java.version>
  </properties>

  <dependencies>

    <!-- ===== -->
    <!-- Spring Boot -->
    <!-- ===== -->

    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-devtools</artifactId>
      <scope>runtime</scope>
      <optional>true</optional>
    </dependency>

    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-web</artifactId>
    </dependency>

    <!-- ===== -->
    <!-- Spring Database -->
    <!-- ===== -->

    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-data-jpa</artifactId>
    </dependency>
  </dependencies>

```

Figure 15. Spring application's POM. (self-made)

At this point, moving on to the next phase and starting coding was possible. I created the model Java classes during this first coding phase and added persistence. I based their creation on my LDM, which I modelled in one of the previous steps. The design of Java classes is basic and straightforward. However, when adding the JPA for persistence, some errors can occur, which can block the start of the server. According to Nader Soukouti (Haute Ecole de Gestion de Genève 2022), it is simpler and more efficient to create one entity at a time and check that there are no JPA-related problems by starting the server. It is a good practice that I have become accustomed to implementing and offers enormous time savings in the case of an error.

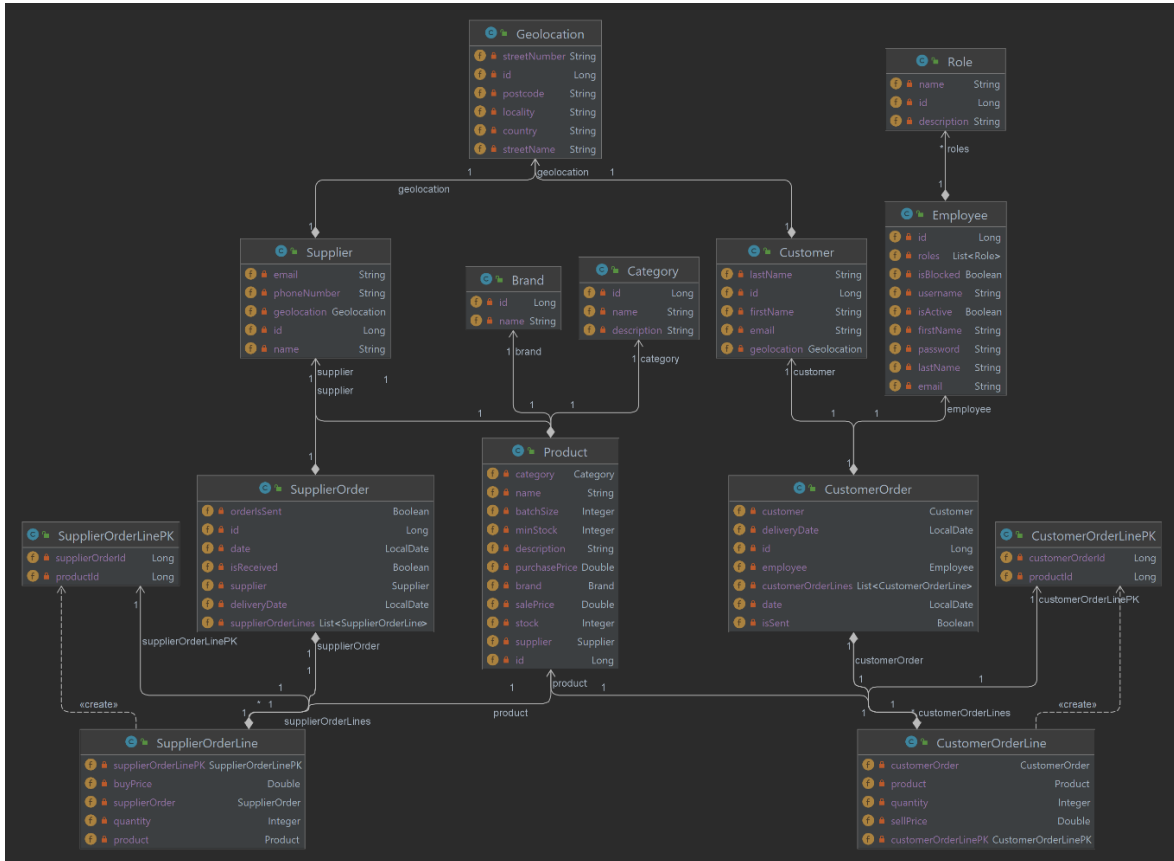


Figure 16. PDM diagram of the project generated with IntelliJ IDEA. (self-made)

As we can see on the PDM above, it represents the models of the project and their attributes. We can also see relationships between them.

When the creation and implementation of JPA entities were completed, I started to set up the security of the project. According to Laurentiu (2020, chapter 1), applications' security is increasingly taken seriously. It must be taken at the beginning of the project, as otherwise, it may lead to a lack of security and risks of failure – this is why I started implementing security at the beginning, and I always tried to keep it in mind. The first security element is encrypting sensitive data, such as passwords. In fact, user (employee) passwords are automatically encrypted using BCrypt, which Spring Security recommends. The second element of security that I established is the authentication of users. It means that employees can get a Jason Web Token (JWT), which will be used to access the resources of the REST API. It involves implementing several elements, such as a security configuration or a customised JWT. About the authorisation, I will discuss it in the phase about controllers.

However, the security of an application is not only authorisation but also other good practices. According to OWASP (s.a.), I avoided using non-parametrised queries and tried to be aware of common vulnerabilities and exposures (CVE). The project brings together several libraries which can have CVE. With Maven, it is easy to detect CVE on a dependency; the concerned dependency is highlighted in yellow. This project contains the “spring boot starter web” dependency, and this latter has two CVEs. Unfortunately, after trying several times to change the version, each included CVEs. Because of the importance of it in the project, I had to resign myself to keep it as it was.

```

<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-web</artifactId>
</dependency>
<!-- =====
<!-- Spring
<!-- =====
<dependency>
  <groupId>
  <artifactId>
</dependency>
<dependency>
  <groupId>
  <artifactId>hibernate-validator</artifactId>

```

Provides transitive vulnerable dependency maven:org.yaml:snakeyaml:1.33

- CVE-2022-41854 6.5 Out-of-bounds Write vulnerability with medium severity found
- CVE-2022-1471 9.8 Deserialization of Untrusted Data vulnerability with high severity found

Results powered by Checkmarx()

Show vulnerabilities info for maven:org.yaml:snakeyaml:1.33 Alt+Maj+Entrée More actions... Alt+Entrée

Tag name: artifactId
Description : The unique id for an artifact produced by the project group, e.g. maven-artifact.
Version : 3.0.0+

`xs:element` on www.w3.org

Figure 17. Spring boot starter web, CVE on project's dependency. (self-made)

The next stage was the longest due to the number of development methods. The latter concerns the creation of controllers, repositories, and services. The development of the controllers is based on the JPA models, which means that a user will call a specific controller method to interact with an entity. Therefore, each JPA model has its corresponding controller containing different methods. I developed each controller according to the principle of REST API learned in the “Implémentation de Service” course (Haute Ecole de Gestion de Genève, 2022). In addition, they are designed to accept and return DTO objects by marshalling them from a Java class to a JavaScript Object Notation (JSON) object and unmarshalling them (JSON to Java class). This ensures that unnecessary data is not exposed for security reasons. Controller methods returning lists of objects provide extended features, such as searching, sorting, and paging. In order to regulate access permissions to different methods, each uses the `@PreAuthorize` annotations, which allow specific roles to execute the action.

```

@GetMapping(produces = "application/json")
@PreAuthorize("hasAnyRole('ROLE_VENDOR', 'ROLE_MANAGER', 'ROLE_ADMIN')")
public @ResponseBody ResponseEntity<?> getProduct(@AuthenticationPrincipal Employee user,
                                                @RequestParam(required = false) String searchQuery,
                                                @PageableDefault(size = 10) Pageable pageable,
                                                @SortDefault.SortDefaults({
                                                    @SortDefault(sort = "name", direction = Sort.Direction.ASC)}) Sort sort) {
    try {
        log.info("message: "User {} is requesting all the products.", user.getUsername());
        Specification<Product> spec = null;
        if (searchQuery != null && !searchQuery.isEmpty()) {
            spec = (root, query, cb) -> cb.like(cb.lower(root.get("name")), "%" + searchQuery.toLowerCase() + "%");
        }
        pageable = PageRequest.of(pageable.getPageNumber(), pageable.getPageSize(), sort);
        Page<Product> products = pRepository.findAll(spec, pageable);
        if (products.getTotalElements() < 1) {
            log.info("message: "User {} requested all the products. NO DATA FOUND.", user.getUsername());
            ErrorResponse bm = new ErrorResponse(HttpStatus.NO_CONTENT.getReasonPhrase(), "message: "NO_PRODUCT_FOUND");
            return new ResponseEntity<>(bm, HttpStatus.NO_CONTENT);
        }
        List<ProductSimpleDTO> productsDTO = new ArrayList<>();
        for (Product product : products) {
            ProductSimpleDTO productSimpleDTO = ProductSimpleDTO.convert(product);
            createHATEOAS(productSimpleDTO);
            productsDTO.add(productSimpleDTO);
        }
        PageModel.PageMetadata pmd = new PageModel.PageMetadata(products.getSize(), products.getNumber(), products.getTotalElements());
        PageModel<ProductSimpleDTO> productDTOPage = PageModel.of(productsDTO, pmd);
        productDTOPage.add(LinkTo(ProductController.class).withRel("products"));
        log.info("message: "User {} requested all the products. RETURNING DATA.", user.getUsername());
        return new ResponseEntity<>(productDTOPage, HttpStatus.OK);
    } catch (Exception e) {
        log.info("message: "User {} requested all the products. UNEXPECTED ERROR.", user.getUsername());
        return new ResponseEntity<>(HttpStatus.INTERNAL_SERVER_ERROR);
    }
}

```

Figure 18. Example of a controller method to retrieve all the products. (self-made)

Intending to respond to requests, the controllers rely on the JPA repository to perform actions on the database. However, some of them may require implementing a more complex logic. In this case, I developed service classes. It applies to customer and supplier orders. Regarding security, according to OWASP (s.a.), the repositories and services only use parametrised queries to prevent SQL injection.

```

@Repository
public interface EmployeeRepository extends JpaRepository<Employee, Long> {
    Optional<Employee> findById(Long id);
    Page<Employee> findAll(Specification<Employee> spec, Pageable pageable);
    Boolean existsByEmail(String email); 4 usages
    Boolean existsByUsername(String username); 5 usages
    Optional<Employee> findByUsername(String username); 5 usages

    @Transactional 2 usages
    @Modifying(clearAutomatically = true)
    @Query(value = "UPDATE Employee e SET e.isActive = false, e.isBlocked = true WHERE e.id = :employeeId")
    void deactivateEmployeeById(@Param("employeeId") Long id);
}

```

Figure 19. Repository for the employee entity, used to perform database actions. (self-made)

Throughout the development process, I created tests to ensure the methods worked correctly. The tests cover the repositories, services, and controllers and are made in two ways. The first way consists of invoking the REST API using the Postman application and checking the result of the response visually. The second way is the unit tests using the JUnit5 library, and I directly coded them within the application.

Although writing unit tests may be tedious, they can be launched many times, and the response content is automatically checked. Furthermore, whatever the test's purpose, they are all independent, as they should not fail because of one another. Therefore, a database is dedicated to the tests and cleaned before each test.

When writing tests for a repository or service, it is enough to automatically wire the test entity manager and the concerned repository.

```

@ExtendWith(SpringExtension.class) no usages
@DataJpaTest
@TestPropertySource(locations = "classpath:application-test.properties")
@AutoConfigureTestDatabase(replace = AutoConfigureTestDatabase.Replace.NONE)
@Log4j2
public class GeolocationRepositoryTest {

    @Autowired 4 usages
    private GeolocationRepository gRepository;

    @Autowired 3 usages
    private TestEntityManager testEntityManager;

    @BeforeEach
    public void setUp() {
        EntityManager em = testEntityManager.getEntityManager();
        Query geolocationDelete = em.createQuery("DELETE Geolocation g");
        geolocationDelete.executeUpdate();
    }

    /**
     * This test is used to ensure that the geolocation repository can find the geolocation that corresponds to an id.
     */
    @Test
    public void findById() {
        // Initialization
        Geolocation geolocation = new Geolocation("Ratapihantie", "13", "00520", "Helsinki", "Finland");
        log.debug("GEOLOCATION TEST - FIND BY ID - New geolocation created: " + geolocation);
        EntityManager em = testEntityManager.getEntityManager();
        em.persist(geolocation);
        em.getTransaction().commit();
        log.debug("GEOLOCATION TEST - FIND BY ID - New geolocation saved: " + geolocation);
        // Execution
        Optional<Geolocation> geolocationOptional = gRepository.findById(geolocation.getId());
        // Verification
        log.debug("GEOLOCATION TEST - FIND BY ID - Geolocation verifications");
        assertTrue(geolocationOptional.isPresent());
        Geolocation geolocationFound = geolocationOptional.get();
        assertNotNull(geolocationFound);
        assertNotNull(geolocationFound.getId()); assertEquals(geolocation.getId(), geolocationFound.getId());
        assertNotNull(geolocationFound.getStreetName()); assertEquals(geolocation.getStreetName(), geolocationFound.getStreetName());
        assertNotNull(geolocationFound.getStreetNumber()); assertEquals(geolocation.getStreetNumber(), geolocationFound.getStreetNumber());
        assertNotNull(geolocationFound.getPostcode()); assertEquals(geolocation.getPostcode(), geolocationFound.getPostcode());
        assertNotNull(geolocationFound.getLocality()); assertEquals(geolocation.getLocality(), geolocationFound.getLocality());
        assertNotNull(geolocationFound.getCountry()); assertEquals(geolocation.getCountry(), geolocationFound.getCountry());
    }
}

```

Figure 20. A part of the geolocation repository test class. (self-made)

As part of the controllers' test, I used the Mockito library provided in the Spring testing module. In addition, because of the project security, to execute the tests, it is mandatory to give a token. Otherwise, the request will be unauthorised.

```

public class GeolocationControllerTest {

    @Autowired
    private MockMvc mvc;

    @Autowired
    private RoleRepository roleRepository;

    @Autowired
    private EmployeeRepository employeeRepository;

    @Autowired
    private GeolocationRepository geoRepository;

    @Autowired
    private SupplierRepository supplierRepository;

    @Autowired
    private CustomerRepository customerRepository;

    private String token;

    @BeforeEach
    public void setUp() throws Exception {
        Role admin = new Role("ROLE_ADMIN", "ADMIN", "For the admins"); roleRepository.save(admin);
        String password = "1234";
        Employee employee = new Employee("mail@haaga.fi", "mail", "Main", "Haaga", new BCryptPasswordEncoder().encode(password), true, false);
        employeeRepository.save(employee);
        employee.setRoles(List.of(admin)); employeeRepository.save(employee);
        String requestBody = "{ \"username\": \"\" + employee.getUsername() + \"\", \"password\": \"\" + password + \"\"}";
        MvcResult mvcResult = mvc.perform(MockMvcRequestBuilders.post("/api/auth/login").accept(MediaType.APPLICATION_JSON).content(requestBody).header("Content-Type", MediaType.APPLICATION_JSON).andReturn());
        if (mvcResult.getResponse().getStatus() == 200) {
            AuthResponseDTO authResponseDTO = new Gson().fromJson(mvcResult.getResponse().getContentAsString(), AuthResponseDTO.class);
            token = "Bearer " + authResponseDTO.getToken();
        } else {
            throw new RuntimeException("AUTHENTICATION FAILED!");
        }
    }

    @Test
    public void getGeolocations() throws Exception {
        geoRepository.save(new Geolocation("Ratapihentie", 13, 00250, "Helsinki", "Finland"));
        geoRepository.save(new Geolocation("Campus Battelle, Rue de la Tambourine", 17, 1227, "Carouge", "Switzerland"));
        geoRepository.save(new Geolocation("Rte de Moutier", 14, 2800, "Delémont", "Switzerland"));

        mvc.perform(MockMvcRequestBuilders
            .get("/api/geolocations")
            .header("Authorization", token)
            .accept(MediaType.APPLICATION_JSON)
            .andDo(print())
            .andExpect(status().isOk())
            .andExpect(MockMvcResultMatchers.jsonPath("$.embedded.geolocationDTOList").exists())
            .andExpect(MockMvcResultMatchers.jsonPath("$.embedded.geolocationDTOList[0].id").isNotEmpty());
    }
}

```

Figure 21. Part of the geolocation's controller test class.

In order to test the functioning of the application as well as possible, there are almost three hundred tests that verify that each functionality is well implemented.

3.3.2 Deployment

The deployment of the project is undoubtedly the last phase of the project. The deployment consists of putting an application into production on a server so its end users can access it. This can be done on any platform, whether in the cloud, using services such as Heroku or on-premises servers. In order to demonstrate the deployment of the application, I decided to use an Apache Tomcat server directly on my machine. It allows me to avoid the cost of deploying the application in the cloud.

The deployment of the application must consider a few requirements. First, the project is based on Spring Boot version 3.0, which implies using JDK17 or higher. Furthermore, the server must be compatible with Jakarta EE, as for Apache Tomcat version 10. Finally, in a properties file, the project requires access to a MariaDB database, a configuration for cross-origin resource sharing (CORS), and settings for JWT.

```

# Profile (dev or prod)
spring.profiles.active=prod

# Database properties
spring.datasource.url=jdbc:mariadb://localhost:3306/stockmanagement
spring.datasource.username=root
spring.datasource.password=
spring.sql.init.mode=always
spring.jpa.defer-datasource-initialization=true
spring.datasource.driver-class-name=org.mariadb.jdbc.Driver
spring.jpa.generate-ddl=true
spring.jpa.hibernate.ddl-auto=update

# Cors properties
spring.security.cors.allowed-origins=http://localhost:3000
spring.security.cors.allowed-methods=GET,POST,PUT,DELETE
spring.security.cors.allowed-headers=Authorization,Content-Type

# JWT properties
jwt.expiration.duration=8
jwt.expiration.unit=10
jwt.secret=THIS_IS_A_BAD_SECRET

```

Figure 22. Example of a properties file to configure the application. (self-made)

As stated before, the deployment requires an external properties file. Therefore, I created the `stockmanager.xml` file at the following path: “tomcat/conf/Catalina/localhost”.

This file is used to specify the variables of the environment and, therefore, indicate the location of the properties file to Spring Boot.

```

<?xml version='1.0' encoding='utf-8'?>
<Context>
  <Environment name="spring.config.location" value="file:/C:/Users/matth/apache-tomcat-10.1.7/conf/Catalina/localhost/stockManager.properties" type="java.lang.String"/>
</Context>

```

Figure 23. `Stockmanager.xml` file to indicate the environment during deployment. (self-made)

In order to deploy the application itself, I designed it so that it could be deployed on Apache Tomcat without being modified. Therefore, I only needed to create a WAR (Web Application Resource) file of the application. I used the Maven lifecycle to get this, specifically the clean and install commands. The purpose of the first command is to clean the files from the destination package, where the WAR will be located. The second command aims to build the project, create an artifact, and deploy it to the local repository (m2) and the project's target directory. Due to the number of tests, the latter command may take a few minutes to be executed. Finally, I had to move the produced WAR file to the tomcat deployment folder at: “tomcat/webapps”.

After starting the Tomcat web server, the Spring application will be automatically initialised using the `Catalina.bat` file of the bin directory. Finally, the REST API is available using the URL:

http://IP_ADDRESS:8080/stockmanager/api

At this point, it is now possible to create a new employee by using the verb post on the endpoint: `/api/employees` and providing the new employee's data, including a password of at least eight characters.

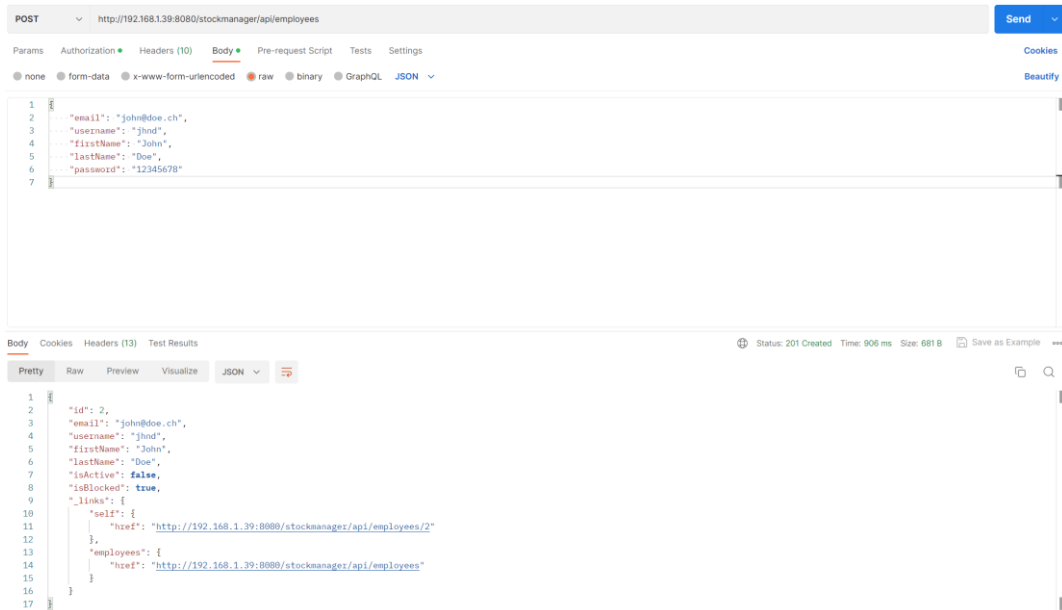


Figure 25. Creating a new employee. (self-made)

When creating an account, it will be automatically considered a vendor. However, it is impossible to use directly because it is deactivated and blocked by default, as shown in the picture above. To enable the use of the account, it is required to use a PUT request to the following endpoint, replacing the `{id}` with the user's id: `/api/employees/{id}/activate`. It is now possible to give an administrative role so that it can replace the default one. To provide the administrator role to the new account created, we can use the endpoint: `api/employees/{userId}/add-role/{roleId}`. The roles' identifiers are assigned as follows: administrator is one, manager is two, vendor is three.

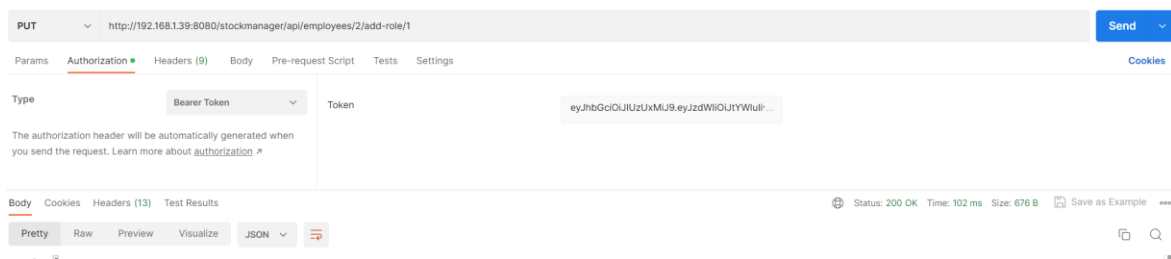


Figure 26. Adding the administrator role to the user with id two. (self-made)

The new account is now ready to be used as an administrator. In addition, it is now possible to deactivate the account named main, as below.

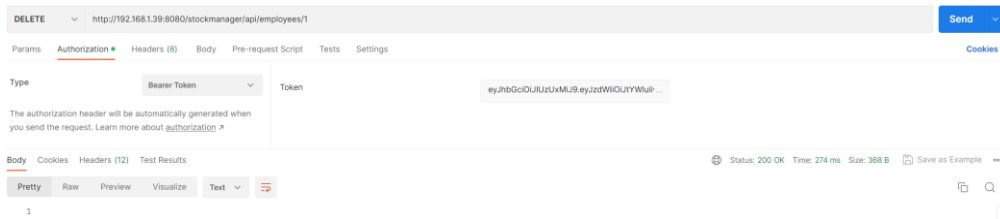


Figure 27. Disabling and blocking the user with id one. (self-made)

This manipulation implies that the “main” account must be reactivated before use. Whatever request is made with a blocked user, the server will return an HTTP code 401 (unauthorised).

3.4.2 Routine use

The application has multiple endpoints that allow the administration of employees, as above. In addition, of course, it also offers other stock and order management endpoints. The project’s GitHub repository contains a detailed file of the different endpoints and a description of each. You can find the project repository in the first appendix.

As mentioned, all API endpoints require a valid token to perform actions. Below are examples of requests that can be made when you are authenticated. These are made with a database containing sample data.

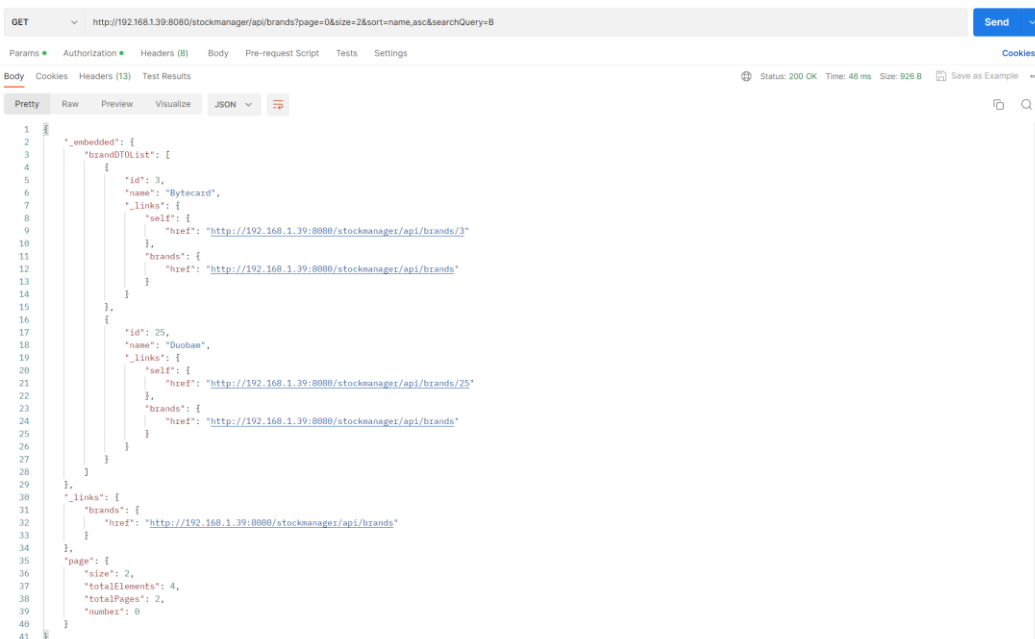


Figure 28. Example of search and sorting on all brands in the database. (self-made)

The figure above is an example of a GET request to search all the brands of the database having a “B” as the first letter of its name. The result list is sorted by name and paginated, with a size of two.

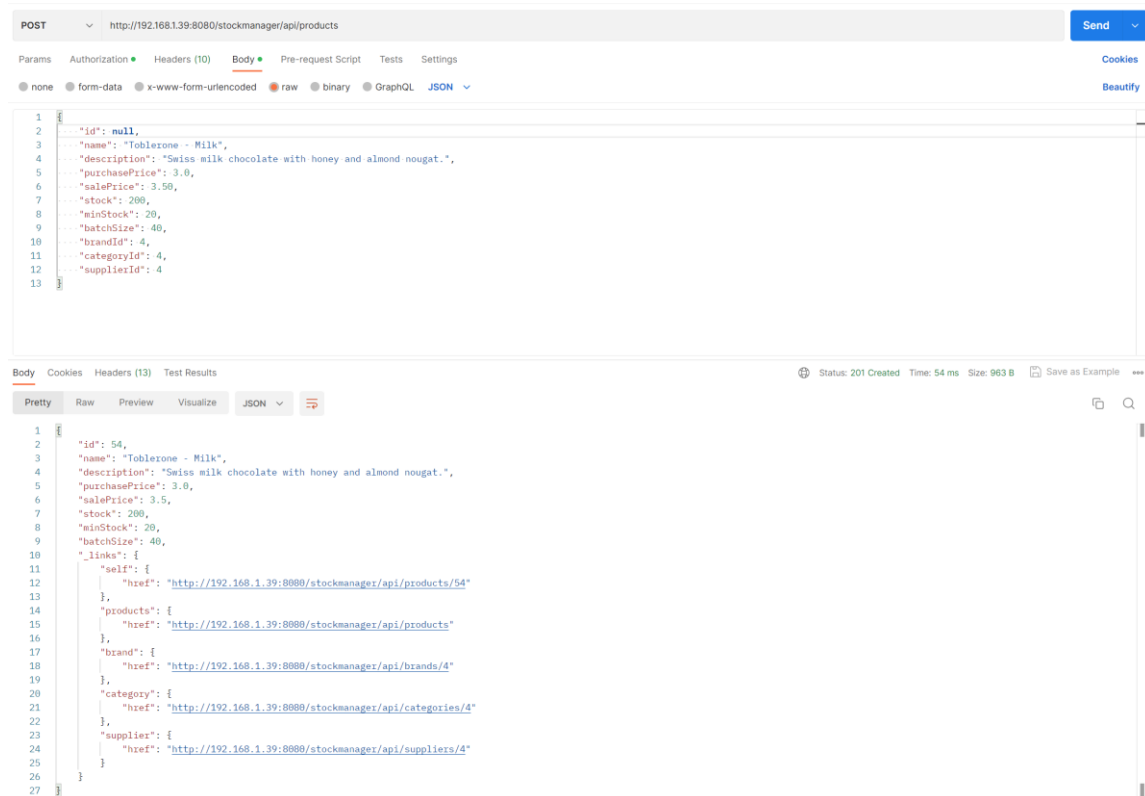


Figure 29. Example of the creation and registration of a new product. (self-made)

The figure above shows an example of a POST request to the REST API. The latter purpose is to save the new product in the database.

The following screenshots will simulate creating a command and a command line. Then we will send it and consult the decrement of the stock.

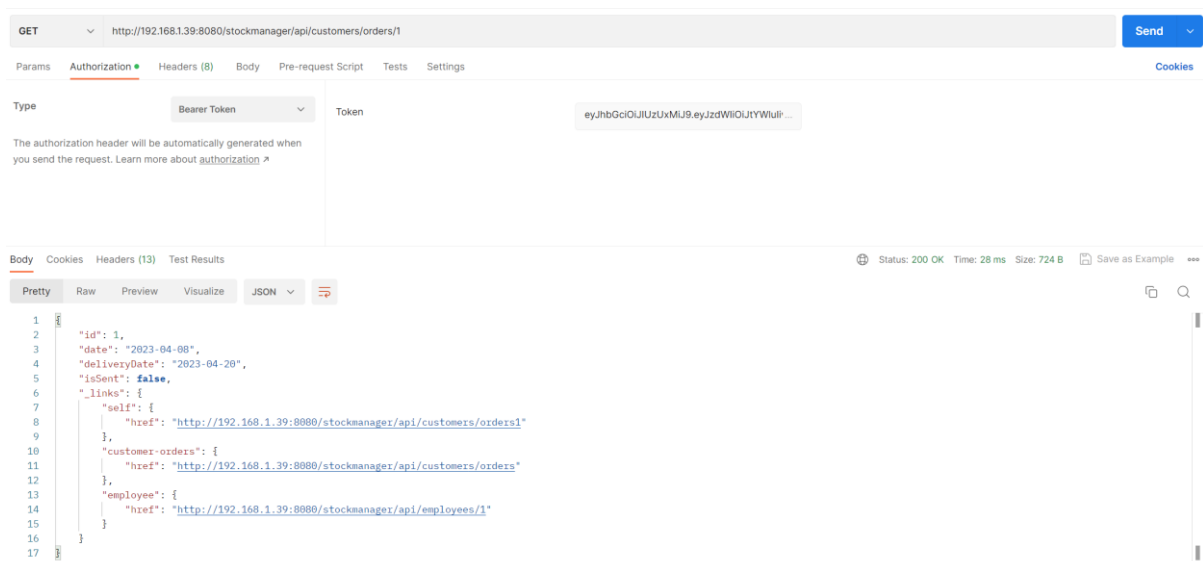


Figure 30. Creating a new customer order for today. (self-made)

In the previous figure, we have the creation of a new customer order. The customer order can be compared to a package box. It contains general information about an order, such as the delivery date.

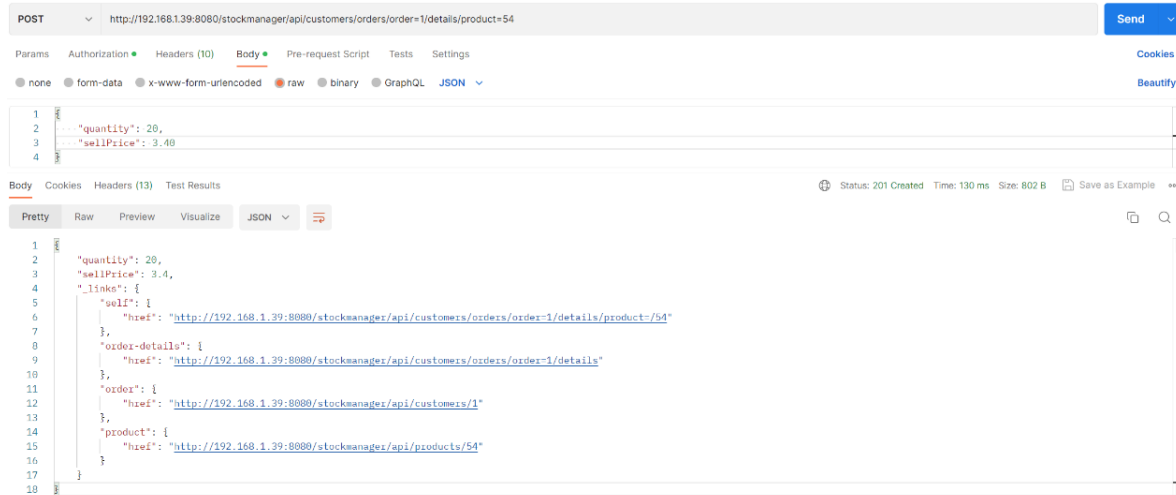


Figure 31. Creating a customer order line. (self-made)

On the other hand, the customer order line made in the previous image is the details between an order and one of the products ordered in this order. In other words, the customer order line contains the order id, product id, quantity, and sell price. An order can have one or multiple order lines, which these latter describe the content of an order.

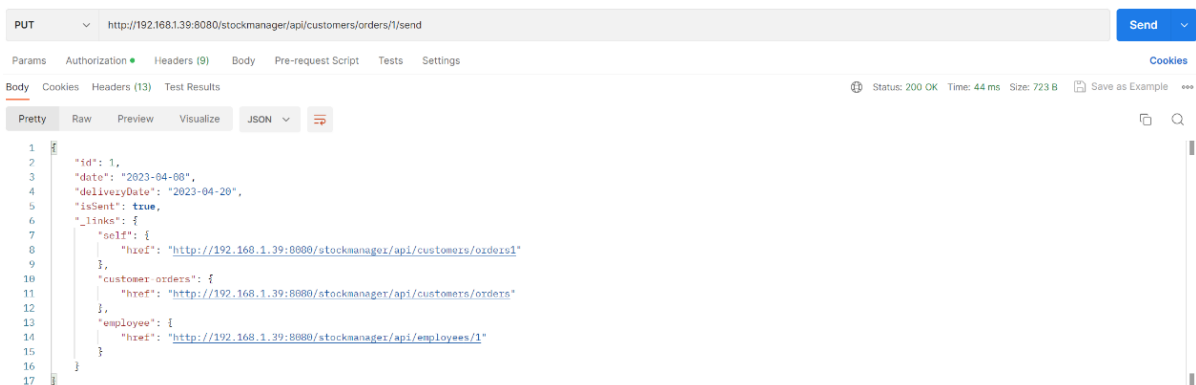


Figure 32. Sending the customer order line. (self-made)

This endpoint is used with the id of a customer order. By using it, it means that an order is sent to a customer. Therefore, the product related to this order by the order lines undergoes a decrease in stock.

However, it is essential to note that if a customer order is sent, it can no longer be modified. A customer order without any order lines cannot be considered as sent.

```

1 GET http://192.168.1.39:8080/stockmanager/api/products/54
2
3 {"id": 54,
4  "name": "Toblerone - Milk",
5  "description": "Swiss milk chocolate with honey and almond nougat.",
6  "salePrice": 3.5,
7  "stock": 180,
8  "_links": {
9    "self": {
10     "href": "http://192.168.1.39:8080/stockmanager/api/products/54"
11    },
12    "products": {
13     "href": "http://192.168.1.39:8080/stockmanager/api/products"
14    },
15    "brand": {
16     "href": "http://192.168.1.39:8080/stockmanager/api/brands/4"
17    },
18    "category": {
19     "href": "http://192.168.1.39:8080/stockmanager/api/categories/4"
20    }
21  }

```

Figure 33. Recognition of the decrease in product number 54. (self-made)

As a result of the previous steps, we should have a stock decrease. Remember, when we created the product, it had two hundred units. The image above shows that the stock now equals one hundred eighty. The difference corresponds to the quantity of the twenty products sold in the customer order.

With the application, it is also possible to manage supplier orders to refill the stocks of products. With the following examples, we will simulate the creation and reception of a supplier order.

```

1 POST http://192.168.1.39:8080/stockmanager/api/suppliers/orders
2
3 {"date": "2023-04-04",
4  "deliveryDate": "2023-04-08",
5  "orderIsSent": "false",
6  "isReceived": "false",
7  "supplierId": 4
}

```

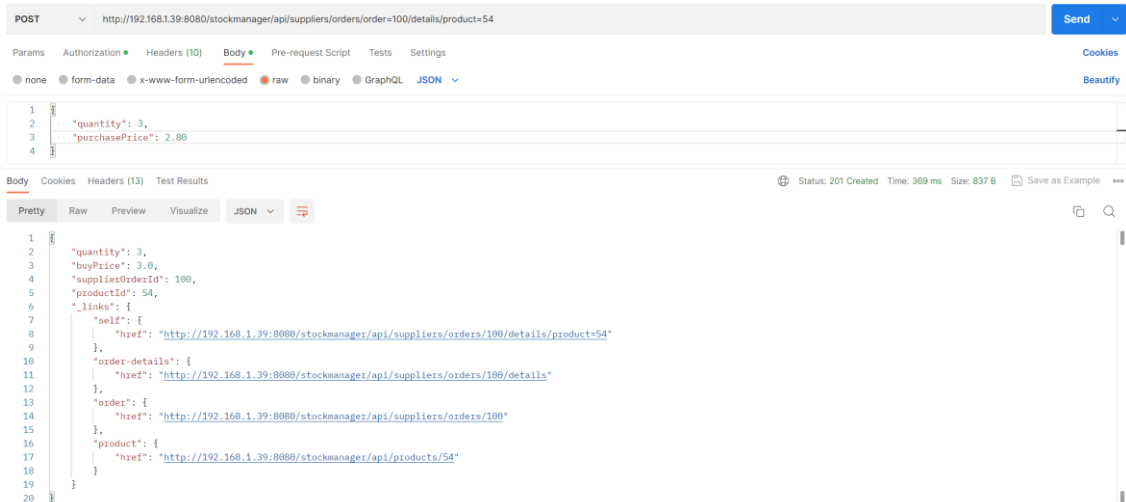
```

1 {"id": 100,
2  "date": "2023-04-04",
3  "deliveryDate": "2023-04-08",
4  "orderIsSent": false,
5  "isReceived": false,
6  "_links": {
7    "self": {
8     "href": "http://192.168.1.39:8080/stockmanager/api/suppliers/100"
9    },
10   "supplier-orders": {
11     "href": "http://192.168.1.39:8080/stockmanager/api/suppliers/orders"
12   },
13   "supplier": {
14     "href": "http://192.168.1.39:8080/stockmanager/api/suppliers/100"
15   },
16   "this-supplier-orders": {
17     "href": "http://192.168.1.39:8080/stockmanager/api/suppliers/100/orders"
18   }
19  }
20 }
21

```

Figure 34. Create a new supplier order. (self-made)

The figure above shows an example of creating a supplier order. As for the customer order, a supplier one contains the primary information about an order. It includes the delivery date and also the sent and received status.



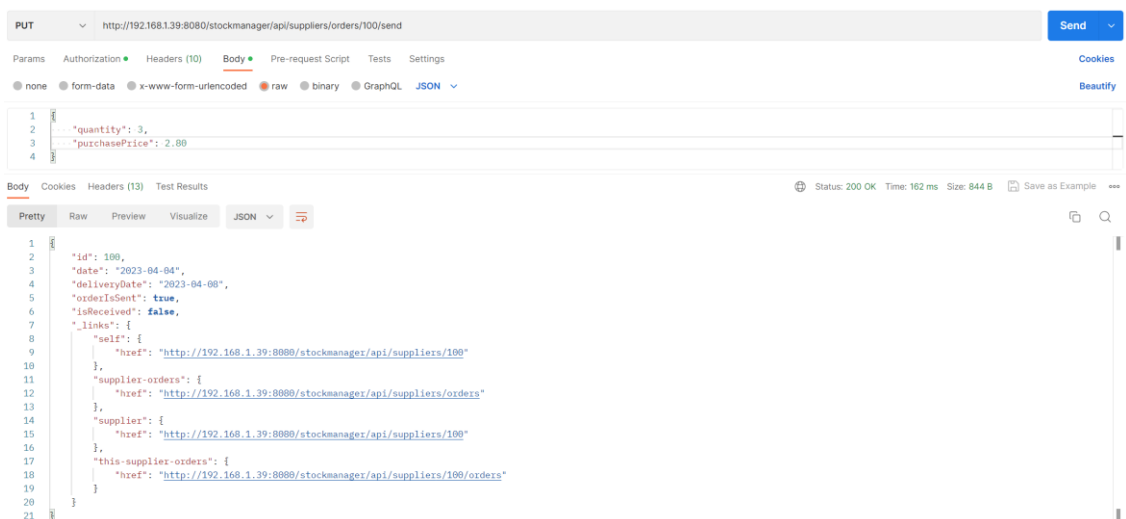
```

POST http://192.168.139.8080/stockmanager/api/suppliers/orders/order=100/details/product=54
Body
  none form-data x-www-form-urlencoded raw binary GraphQL JSON
  "quantity": 3,
  "purchasePrice": 2.80
Body
  Pretty Raw Preview Visualize JSON
  {
    "quantity": 3,
    "buyPrice": 3.0,
    "supplierOrderId": 100,
    "productId": 54,
    "_links": {
      "self": {
        "href": "http://192.168.139.8080/stockmanager/api/suppliers/orders/100/details/product=54"
      },
      "order-details": {
        "href": "http://192.168.139.8080/stockmanager/api/suppliers/orders/100/details"
      },
      "order": {
        "href": "http://192.168.139.8080/stockmanager/api/suppliers/orders/100"
      },
      "product": {
        "href": "http://192.168.139.8080/stockmanager/api/products/54"
      }
    }
  }
  
```

Figure 35. Create a new supplier order line. (self-made)

As for the customer order line, the order lines of supplier orders regroup the different products ordered in order. In this case, we create an order line for product fifty-four and the supplier order number one hundred.

The quantity is the most significant difference between the customer order lines and the supplier order lines. For a customer order line, the quantity field refers to the number of units the customer wants. On the other hand, for a supplier order line, the quantity is related to the number of batches ordered.

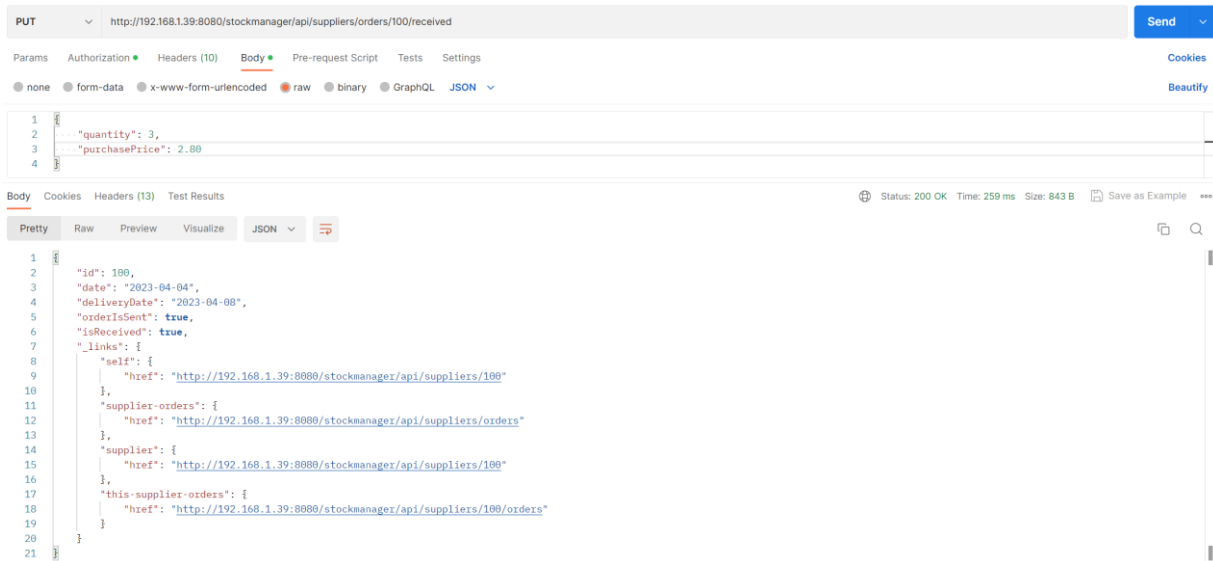


```

PUT http://192.168.139.8080/stockmanager/api/suppliers/orders/100/send
Body
  none form-data x-www-form-urlencoded raw binary GraphQL JSON
  "quantity": 3,
  "purchasePrice": 2.80
Body
  Pretty Raw Preview Visualize JSON
  {
    "id": 100,
    "date": "2023-04-04",
    "deliveryDate": "2023-04-06",
    "orderIsSent": true,
    "isReceived": false,
    "_links": {
      "self": {
        "href": "http://192.168.139.8080/stockmanager/api/suppliers/100"
      },
      "supplier-orders": {
        "href": "http://192.168.139.8080/stockmanager/api/suppliers/orders"
      },
      "supplier": {
        "href": "http://192.168.139.8080/stockmanager/api/suppliers/100"
      },
      "this-supplier-orders": {
        "href": "http://192.168.139.8080/stockmanager/api/suppliers/100/orders"
      }
    }
  }
  
```

Figure 36. Allows the order to be considered as sent to the supplier to avoid any future modification. (self-made)

This endpoint is used for the system to consider a supplier order as sent. In case it is sent, it cannot be modified anymore. There is, therefore, no impact on stock values.



```

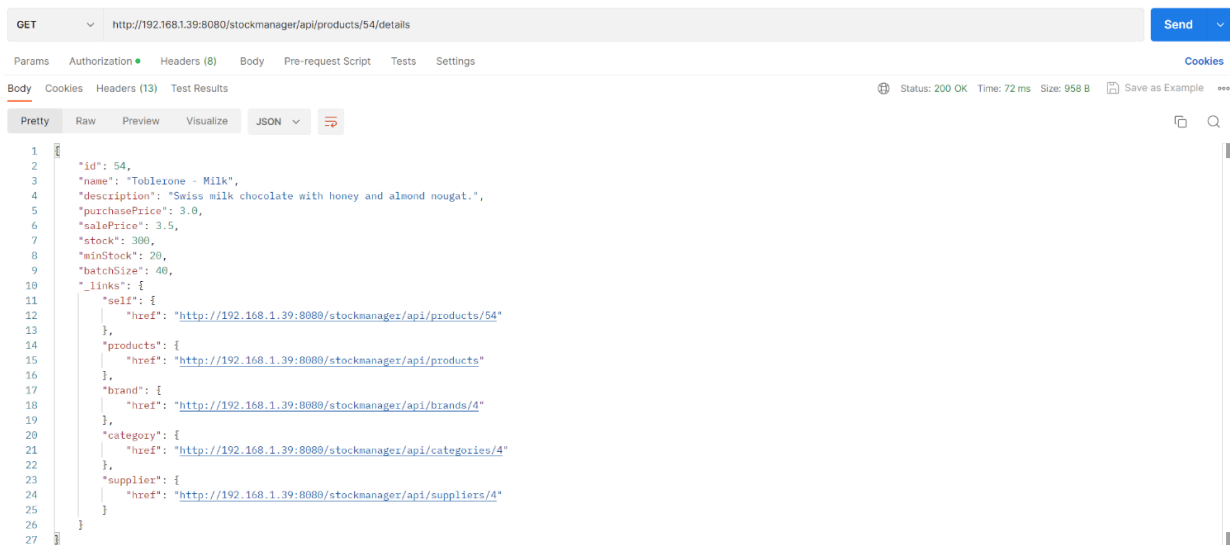
PUT http://192.168.1.39:8080/stockmanager/api/suppliers/orders/100/received
{
  "quantity": 3,
  "purchasePrice": 2.80
}

Status: 200 OK Time: 259 ms Size: 843 B
{
  "id": 100,
  "date": "2023-04-04",
  "deliveryDate": "2023-04-08",
  "orderIsSent": true,
  "isReceived": true,
  "_links": {
    "self": {
      "href": "http://192.168.1.39:8080/stockmanager/api/suppliers/100"
    },
    "supplier-orders": {
      "href": "http://192.168.1.39:8080/stockmanager/api/suppliers/orders"
    },
    "supplier": {
      "href": "http://192.168.1.39:8080/stockmanager/api/suppliers/100"
    },
    "this-supplier-orders": {
      "href": "http://192.168.1.39:8080/stockmanager/api/suppliers/100/orders"
    }
  }
}

```

Figure 37. Notes the supplier order as received and increments the stock. (self-made)

The purpose of the endpoint in the figure above is to enable the system to consider a supplier order as received. When the receive status of an order is changed, it automatically increments the product stocks. As mentioned, the stock is incremented by multiplying the batch size of a product and the quantity specified in the order line.



```

GET http://192.168.1.39:8080/stockmanager/api/products/54/details
Status: 200 OK Time: 72 ms Size: 958 B
{
  "id": 54,
  "name": "Toblexone - Milk",
  "description": "Swiss milk chocolate with honey and almond nougat.",
  "purchasePrice": 3.0,
  "salePrice": 3.5,
  "stock": 300,
  "minStock": 20,
  "batchSize": 40,
  "_links": {
    "self": {
      "href": "http://192.168.1.39:8080/stockmanager/api/products/54"
    },
    "products": {
      "href": "http://192.168.1.39:8080/stockmanager/api/products"
    },
    "brand": {
      "href": "http://192.168.1.39:8080/stockmanager/api/brands/4"
    },
    "category": {
      "href": "http://192.168.1.39:8080/stockmanager/api/categories/4"
    },
    "supplier": {
      "href": "http://192.168.1.39:8080/stockmanager/api/suppliers/4"
    }
  }
}

```

Figure 38. Recognition of the increase in inventory for product fifty-four. (self-made)

Finally, the figure above shows us the information about a product in detail. As you can see, the product's stock has been incremented by one hundred and twenty units. It is due to the order, including three batches of forty units for product fifty-four.

4 Discussions

This thesis aims to develop an open-source back-end application for a stock management system. The application is intended to be used by small and medium-sized companies. The purpose is to provide a low-cost application that avoids losses due to bad management of the product stocks. An essential point of the thesis is that it was developed by myself and, therefore, without the involvement of a third-party. However, although the outcome can be used without any modification, it is not recommended in its current state. It provides the core business processes, but it is more relevant to adapt the application to the company's needs. Moreover, the outcome includes only a REST API to communicate with other systems. It implies that a front-end development is required to make it human-usable and user-friendly.

According to Chapter 1.1, the thesis aims to develop the back-end application using the Spring Framework. The latter manages the core of the application and orchestrates its smooth working. The Spring application includes a REST API, which can communicate with other systems, such as a front-end. In addition, a MariaDB database is responsible for storing the processed data of the API. Combining these elements is intended to allow processing of various objects, such as products, orders, suppliers, and customers.

The realisation of this thesis started in December 2022 with the subject proposal. The leading development of the thesis began in January 2023. Seventy per cent of the thesis was reached around the middle of April of the same year. Approximately one month later, this thesis was completed and submitted.

As mentioned, this thesis outcome is designed to help small and medium-sized companies in stock management. This domain is a crucial element in customer satisfaction and therefore plays a vital role in the evolution of a company. Through this thesis, I wanted to develop an application that helps avoid poor stock management. Unfortunately, the latter can lead to monetary losses. Since the project was developed by myself, I decided only to include the core concepts of a stock management system. This makes the application generic and easily adaptable to the needs of each company. In addition, the project is open-source, which allows any third-party to make changes to it as they wish. Moreover, this characteristic enables companies to use the system at lower costs.

Chapter 1.3 of the thesis describes the quality measures and determines whether the thesis is successful or not. These latter are divided into two categories, mandatory and optional. As a reminder, the mandatory criteria require that the application provide CRUD operations through a REST API for the brands, categories, products, suppliers, customers, and orders.

Each API endpoint should have its basic unit test and protection against SQL injections. It is also included that the application can be easily adapted to the needs of each company.

To determine that these criteria are met, I set up several verifications. Firstly, the application and the database have been designed generically to make it flexible. In this sense, the database scheme is handled by the Spring application. The Java classes and entities can be easily modified to adapt, at the same time, the structure of the application and database. Moreover, new controllers, repositories, and services can be easily added to the application to manage more business processes. Secondly, whether it be the database or the application itself, both can process the main objects of a stock manager by including them in their scheme. Thirdly, the application provides endpoints through a REST API to enable CRUD operations. Each endpoint has comments in the code and has been tested several times on Postman. However, this testing technique is unreliable long-term, so the application contains basic unit tests of each endpoint, repository, and service. Finally, to prevent SQL injections, the application uses parametrised queries. The endpoints were also tested for SQL injections from Postman throughout the project's development.

It was also important for me that the application meets the optional criteria. Those criteria include creating documentation for each API endpoint and developing extensive unit tests. Regarding safety, the API should be secured by integrating other measures such as CSRF or CORS. In order to fulfil these criteria, the application has comments for each method created, as well as documentation for each endpoint. The application documentation (see the first appendix) also explains how to deploy the project quickly. The Spring HTTP filter chain is used to secure the application, includes a CORS configuration, and requires a CSRF token for each request. These measures are tested in the extensive unit tests of the application. Indeed, approximately three hundred tests have been developed to ensure the application is working correctly. They verify that every action leads to the desired result.

To summarise, through this project, I created a secure REST API using the Spring Framework. Furthermore, due to the project's adaptability and ease of deployment, I believe it could be implemented with a few modifications in a real-life case. Given the verification measures to meet the success criteria and the implemented features, I think the project can be considered successful.

In general, during the thesis development, I did not have many problems. However, to have a relevant thesis, I wanted to use as much as possible reliable and academic sources. Searching for these sources took a lot of time, especially at the beginning when I didn't have the right tools in hand. In terms of developing the application, I had three main problems. The first one concerns the migration from Java EE to Jakarta EE. Indeed, some of the helpers (like Stack Overflow) are still based on Java EE. So, it is sometimes difficult to find the correspondence for Jakarta EE.

The second problem concerns the differences between the four and five versions of JUnit. Indeed, some mechanisms were changed, so the configuration of the test classes had to be changed. In addition, I wanted the test database to be externalised. I was, therefore, often confronted with problems during the execution of my tests. The last one concerns the common vulnerabilities and exposures of the Spring Boot dependency. I have spent much time trying to solve this problem but in vain. The dependency is essential to the project, so I could not replace or remove it.

In the event that the project has to be redone, I would like to improve or change some points. Firstly, I would like to have more time to develop the project. This does not necessarily mean over a more extended period but being more available each day for the project. However, a more comprehensive development phase would have allowed me to conduct interviews. This would have allowed me to have the point of view of experts, which would have allowed me to implement more specific business processes. This lack may be one of the weaknesses of the thesis, although I tried to mitigate it with the project's adaptability. Another point I would change is related to the writing part of the thesis. When I started writing it, I was not well organised in managing my sources. Only after meeting with my advisor I understand how to manage them correctly. I would have saved a few work hours if I had been more systematic.

In general terms, I estimate that the thesis has a relatively solid foundation. The application does not have many prerequisites to work. Having a MariaDB database, at least the JDK 17 and Maven, is enough to war the project and deploy it through a web server. However, the weakness is its usability due to the REST API. The latter is not intended to be directly used by a human. For proper use, it is necessary to have a front-end application that enables interaction between the API and humans. Moreover, I advise the user companies to adapt the application to their needs first. It would allow them to use the application's benefits fully. Despite these defaults, the application can be easily adapted and deployed, which makes it easy to use.

I see the future of the application as follows. First of all, in the near future, I would like to develop a front-end to make up for this lack. This will probably be based on the React Framework. At that time, new features can also be created on the back-end, such as PDF generation for orders. However, another future for the application may be emerging. Indeed, the open-source characteristic of the project allows anyone to participate in the project's development as they wish.

The development of this thesis was very instructive for me, as I improved many of my skills. Firstly, this project is the most prominent IT project I have ever managed alone. I have therefore learned how to manage a project altogether. It enables me to discover and learn how to plan and organise a project in its entirety. In fact, I was alone to work on the project to ensure the excellent implementation of each feature. Project management also includes dealing with internal and external issues.

Indeed, I had to be able to minimise their impact by mitigating them.

Secondly, I also enhanced my skills in database development and management. The database was a crucial part of the project. A poor implementation of the latter could have led to a delay in development or even failure. The latter had to meet multiple constraints to ensure that the project would be scalable and able to consider and manage with primary data required by the system. In terms of application development with the Spring Framework, I strengthen my skills and knowledge of the general functioning of the framework. This also applies to the components I used for the thesis development. For example, I learned how to integrate and manage elements, such as JUnit5 or take care of the application security through the application. Moreover, the application provides a REST API that must meet several requirements as the previous two points. Until this thesis project, I could never implement all these elements together and ensure the smooth working of an application.

Finally, I also improved my general skills, which are used daily. These include researching reliable sources, critical thinking, and written and verbal communication.

Sources

- Apache Maven. 2023. Maven. URL: <https://maven.apache.org/index.html>. Accessed: 20 February 2023.
- Baeldung. 2020. Ant vs. Maven vs. Gradle. URL: <https://www.baeldung.com/ant-maven-gradle>. Accessed: 20 February 2023.
- Baeldung. 2022. Defining JPA Entities. URL: <https://www.baeldung.com/jpa-entities>. Accessed 21 February 2023.
- Baeldung. 2022. Maven dependencyManagement vs. dependencies Tags. URL: <https://www.baeldung.com/maven-dependencymanagement-vs-dependencies-tags>. Accessed: 21 February 2023.
- Baeldung. 2022. Why Choose Spring as Your Java Framework. URL: <https://www.baeldung.com/spring-why-to-choose>. Accessed: 26 February 2023.
- Baeldung. 2022. Introduction to Project Lombok. URL: <https://www.baeldung.com/intro-to-project-lombok>. Accessed: 1 March 2023.
- Bauer, C., King, G., Gary G. 2016. Java Persistence with Hibernate. Manning Publications. E-book. Accessed: 22 February 2023.
- Cosmina, I. 2018. Java for Absolute Beginners. Apress. E-book. Accessed: 17 February 2023.
- Cosmina, I., Harrop, R., Ho, C. & Schaefer, C. 2017. Pro Spring 5. Apress. E-book. Accessed: 24 February 2023.
- Craig, Walls. 2022. Spring in Action, Sixth Edition. Simon & Schuster. E-book. Accessed: 25 March 2023.
- Fol, P. 26 October 2021. Spring vs. the World: Comparing Spring Boot Alternatives. URL: <https://www.jrebel.com/blog/spring-boot-alternatives>. Accessed: 24 February 2023.
- Fredrich, T. 2013. RESTful Services Best Practices. URL: <https://www.restapitutorial.com/re-sources.html>. Accessed: 27 February 2023.
- Gilad, Bracha, J. G. & Gosling, J. 2005. Java™ Language Specification, Third Edition. Addison-Wesley Professional. E-book. Accessed: 17 February 2023.

GitHub. 2022. NASAWorldWind/WorldWindJava. URL: <https://github.com/NASAWorldWind/World-WindJava>. Accessed: 17 February 2023.

Haute Ecole de Gestion de Genève – University of Applied Sciences and Arts Western Switzerland 2022. Course “Implémentation de Services” by Stefan Behfar and Nader Soukouti. Accessed: 26 March 2023.

IBM. s.a. What is inventory management? URL: <https://www.ibm.com/topics/inventory-management>. Accessed: 18 April 2023.

Johnson, HA. 2017. Trello. J Med Libr Assoc. Hanover (NH). URL: <https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5370621/>. Accessed: 18 March 2023.

Kaluža, M. Kalanj, M. & Vukelić, B. 7 March 2019. A Comparison of Back-end Frameworks For Web Application Development. Hrčak University of Zagreb University Computing Centre. UNR: <https://hrcak.srce.hr/en/clanak/321176>. Accessed: 25 February 2023.

Keith M., Nardone, M. & Schincariol, M. 2018. Pro JPA 2 in Java EE 8. Apress. E-book. Accessed: 23 February 2023.

Laurentiu S. 2020. Spring Security in Action. Manning Publications. E-book. Accessed: 01 April 2023.

Lohr, S. April 2009. In Sun, Oracle Sees a Software Gem. The New York Times. URL: <https://www.nytimes.com/2009/04/21/technology/companies/21sun.html>. Accessed: 16 February 2023.

Lombok. s.a. Lombok features. URL: <https://projectlombok.org/features/>. Accessed: 1 March 2023.

OpenClassrooms. 2022. Learn About Agile Project Management and Scrum. ULR: <https://open-classrooms.com/fr/courses/4544621-learn-about-agile-project-management-and-scrum/4544628-discover-the-waterfall-and-iterative-approaches-to-software-development>. Accessed: 03 Mai 2023.

Oracle. 2009. Oracle Buys Sun Microsystems. URL: <https://www.oracle.com/corporate/press-release/oracle-buys-sun-042009.html>. Accessed: 16 February 2023.

Oracle. 2013. 37. Introduction to the Java Persistence API. URL: <https://docs.oracle.com/javase/7/tutorial/persistence-intro001.htm#BNBQA>. Accessed: 22 February 2023.

Oracle. 2013. The Java EE 6 Tutorial. Redwood City. URL: <https://docs.oracle.com/javase/6/tutorial/doc/javaeetutorial6.pdf>. Accessed: 22 February 2023.

- Oracle. 2021. Oracle No-Fee Terms and Conditions (NFTC). URL: <https://www.oracle.com/downloads/licenses/no-fee-license.html>. Accessed: 16 February 2023.
- OWASP. 2021. OWASP TOP 10. URL: <https://owasp.org/Top10/>. Accessed: 14 March 2023.
- OWASP. s.a. Java Spring developers @ OWASP. URL: <https://owasp.org/dev-pages/java/spring/index.html>. Accessed: 27 March 2023.
- OWASP. s.a. Rest Security Cheat Sheet. URL: https://cheatsheetseries.owasp.org/cheatsheets/REST_Security_Cheat_Sheet.html. Accessed: 28 February 2023.
- Späth P. 2019. Beginning Jakarta EE. Apress. E-book. Accessed: 20 April 2023.
- Spring. 2023. What Spring can do. URL: <https://spring.io/>. Accessed: 24 February 2023.
- Spring. 2019. Spring Framework Documentation. URL: <https://docs.spring.io/spring-framework/docs/5.1.8.RELEASE/spring-framework-reference/index.html>. Accessed: 26 February 2023.
- Sushevich A. Jun 2020. 8 Best Popular Projects on Java. Medium. URL: <https://medium.com/java-revisited/8-best-popular-projects-on-java-e1a663ab3cc1>. Accessed: 17 February 2023.
- Trello. s.a. What is Trello, Learn Features, Uses and More. URL: <https://trello.com/tour>. Accessed: 18 March 2023.
- Trello. s.a. About Trello. URL: <https://trello.com/about>. Accessed: 18 March 2023.

Appendices

Appendix 1. GitHub repository

Application repository URL: <https://github.com/MatthieuBruh/StockManager>