**STOCK MANAGEMENT SYSTEM**

**(Using Spring Boot and GCP)**

A Project Report Submitted

In Partial Fulfillment of the Requirements

for the Degree

of

MASTER OF COMPUTER APPLICATION

### **by**

Amit Kumar

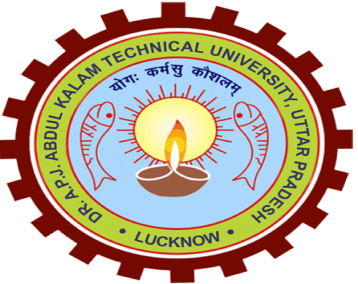
**(1900290140006)**

**Under the Supervision of**

Neelam Rawat

Associate Professor

**KIET Group of Institutions, Ghaziabad**

****

**to the**

**FACULTY OF MCA**

DR. APJ ABDUL KALAM TECHNICAL UNIVERSITY

(Formerly Uttar Pradesh Technical University) LUCKNOW

**May,2022**

**DECLARATION**

I hereby declare that the work presented in this report entitled “STOCK MANAGEMENT SYSTEM (using spring boot and GCP)", was carried out by me. I have not submitted the matter embodied in this report for the award of any other degree or diploma of any other University or Institute.

I have given due credit to the original authors/sources for all the words, ideas, diagrams, graphics, computer programs, experiments, results, that are not my original contribution. I have used quotation marks to identify verbatim sentences and given credit to the original authors/sources.

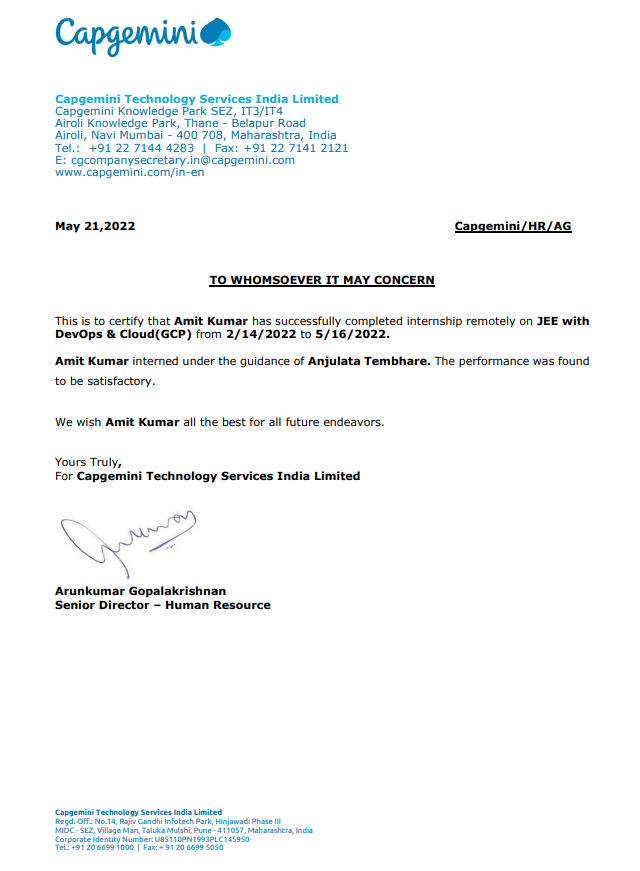
I affirm that no portion of my work is plagiarized, and the experiments and results reported in the report are not manipulated. In the event of a complaint of plagiarism and the manipulation of the experiments and results, I shall be fully responsible and answerable.

Name : Amit Kumar

Roll. No. : 1900290140006

Branch : Master of Computer Application

**TRAINING CERTIFICATE**

****

# **CERTIFICATE**

Certified that **Amit Kumar (**enrollment no 19002901400060) has carried out the project work presented in this thesis entitled **“Stock Management System (using Spring Boot and GCP )”** for the award of **Master of Computer Application** from Dr. APJ Abdul Kalam Technical University, Lucknow under my supervision. The thesis embodies results of original work, and studies are carried out by the student herself and the contents of the thesis do not form the basis for the award of any other degree to the candidate or to anybody else from this or any other University.

## Ms. Neelam Rawat External Examiner

(Internal Examiner)

Associate Professor

Dept. of Computer Applications

KIET Group of Institutions, Ghaziabad

# 

# **ABSTRACT**

Stock Management System on Cloud is an application based on various micro services. It deals with stock management for their database maintaining and generating report corresponding to the data is done on the basis of as per requirement is given.

So, we can say that it helps the management of stock and give exact database management of compony according to rules and regulation. It also helps in maintaining stock data and also display how many products are present in the stock and also gives the details of those products.

This application is designed to manage stock companies and organizations and also handle the sale and purchase of their products. The stock management system on Cloud includes different modules and features for adding, editing, viewing, and deleting items.

# 

# **ACKNOWLEDGEMENT**

Success in life is never attained single handedly. My deepest gratitude goes to my thesis supervisor, **Ms. Neelam Rawat** for her guidance, help and encouragement throughout my project work. Their enlightening ideas, comments, and suggestions.

Words are not enough to express my gratitude to **Dr. Ajay Kumar Shrivastava**, Professor and Head, Department of Computer Applications, for his insightful comments and administrative help at various occasions.

Fortunately, I have many understanding friends, who have helped me a lot in many critical conditions.

Finally, my sincere thanks goes to my family members and all those who have directly and indirectly provided me moral support and other kinds of help. Without their support, completion of this work would not have been possible in time. They keep my life filled with enjoyment and happiness.

Amit Kumar

Enrollment no-1900290140006

**TABLE OF CONTENT**

Declaration ii

Training Certificate iii

Certificate iv

Abstract v

Acknowledge vi

Table of content vii

List of tables ix

List of figures x

List of Abbreviation xii

Chapter 1 – Introduction 13-18

1.1 Project Description 13

1.2 Project Scope 14

1.3 Software/Tools 15

1.4 Hardware 17

1.5 Gaint Chart 18

Chapter 2 Literature Review 19-22

Chapter 3 Technical Feasibility 23-31

3.1 Introduction 23

3.2 Technical description 24

3.3 Technology used 29

Chapter 4 – backend Design 32-35

4.1 Data Dictionary 32

4.2 ER Diagram 33

4.3 Database Design 34

Chapter 5 – Code 36-56

Chapter 6 – Report 57-64

Chapter 7 – Testing 65-73

Chapter 8 – Limitation 74-76

8.1 Limitation 74

8.2 Future Scope 74

8.3 Future enhancement 75

Chapter 9 - Conclusion 77

Chapter 10 – References78-79

Resume 80

**LIST OF TABLES**

4.1 Data Dictionary 32

4.2 Stock Details 34

4.3 Supplied stock details 34

4.4 Order details 34

4.5 Delivery details 35

4.6 Report details 35

4.7 Order stock details 35

**LIST OF FIGURES**

* 1. Eclipse IDE 15
  2. Gradle 16
  3. Postgres 16
  4. Gaint Chart 18

4.1 ER diagram 33

6.1 Output 57

6.2 Add stock1 58

6.3 Add stock 2 58

6.4 Add stock3 59

6.5 find stock 1 59

6.6 find stock 2 60

6.7 find by supplier 1 60

6.8 find by supplier 2 61

6.9 find by supplier 3 61

6.10 Add by postman 1 62

6.11 Add By Postman 2 62

6.12 find by postman 1 63

6.13 find by postman 2 63

6.14 find by postman 3 64

7.1 test case 1 65

7.2 test case 2 66

7.3 test case result 1 67

7.4 test case result 2 68

7.5 test case result 3 69

7.6 test case result 4 70

7.7 test summary 71

7.8 add test 72

7.9 find by id 1 72

7.10 find by id 2 73

**LIST OF ABBREVIATIONS**

1. API – Application Programming Interface
2. CRUD – Create, Read, Update, Delete
3. IDE – Integrated Development Environment
4. OS – Operating System
5. HTTP – Hyper Text Transfer Protocol
6. URL – Uniform Resource Locator
7. RAM – Random Access Memory
8. MySQL – My Structured Query Language
9. POM – Project Object Model
10. JPA – Java Persistence API
11. K8s – Kubernetes
12. AWS – Amazon Web Services
13. RDS – Relational Database Service
14. IT – Information Technology
15. REST – Representational State Transfer
16. SOAP – Simple Object Access Protocol
17. JSON – JavaScript Object Notation

**CHAPTER 1**

**INTRODUCTION**

## PROJECT DESCRIPION

In today’s changing life style computer has become the most essential part of life. Most of the works being performed by the humans is now done by the computer The computer is being used in each and every field now a days.

I am developing software for a stock management and This software help in the stock management for their database maintaining and generating report corresponding to the data is done on the basis of as per requirement is given.

So, we can say that it helps the management of stock and give exact database management of compony according to rules and regulation. It also help in maintaining stock data and also display how many products are present in the stock and also gives the details of these products .

This software also gives or stores each and every information about orders.

This company uses a huge data base so for security of database we give the facility of backup and also recovery as per when company need it takes backup on hard disk.

This application is designed to manage stocks for companies and organizations and also handle the sale and purchase of their products. The stock management system includes different modules and features for adding, editing, viewing, and deleting items.

Usually, the manual stock management method run with pen and paper is not only labor-intensive but also time-consuming. This approach lacks a proper data organization structure, which can give rise to many risks associated with data mismanagement. This stock management project is a more efficient and improved approach to stock data management. It is much more secure and reliable than the manual method

## PROJECT SCOPE

The purpose of this project is to develop a java application. The objective of this process is as follows:

Stock management system is designed for the companies to easy maintain of their records and easy deal with the stock**.**

Usually, the manual stock management method run with pen and paper is not only labor-intensive but also time-consuming. This approach lacks a proper data organization structure, which can give rise to many risks associated with data mismanagement. This stock management project is a more efficient and improved approach to stock data management. It is much more secure and reliable than the manual method.

This software also gives or stores each and every information about orders. This company uses a huge data base so for security of database we give the facility of backup and also recovery as per when company need it takes backup on hard disk. It includes the purchasing of stock by customer , the customer can keep track of the records of the stock .

There are 3 actors in the project which are listed below:

1. Admin
2. Customer
3. Supplier

There are 5 Microservices in the project which are listed below:

1. Stock module
2. Supplied Stock module
3. Order Module
4. Report Module
5. Delivery Module

## SOFTWARE/TOOLS

While developing the stock management system application I have used various softwares and ID’s

**1.3.1 ECLIPSE**

Eclipse is an integrated development environment (IDE) used in computer programming. It contains a base workspace and an extensible plug-in system for customizing the environment. It is the second-most-popular IDE for Java development.

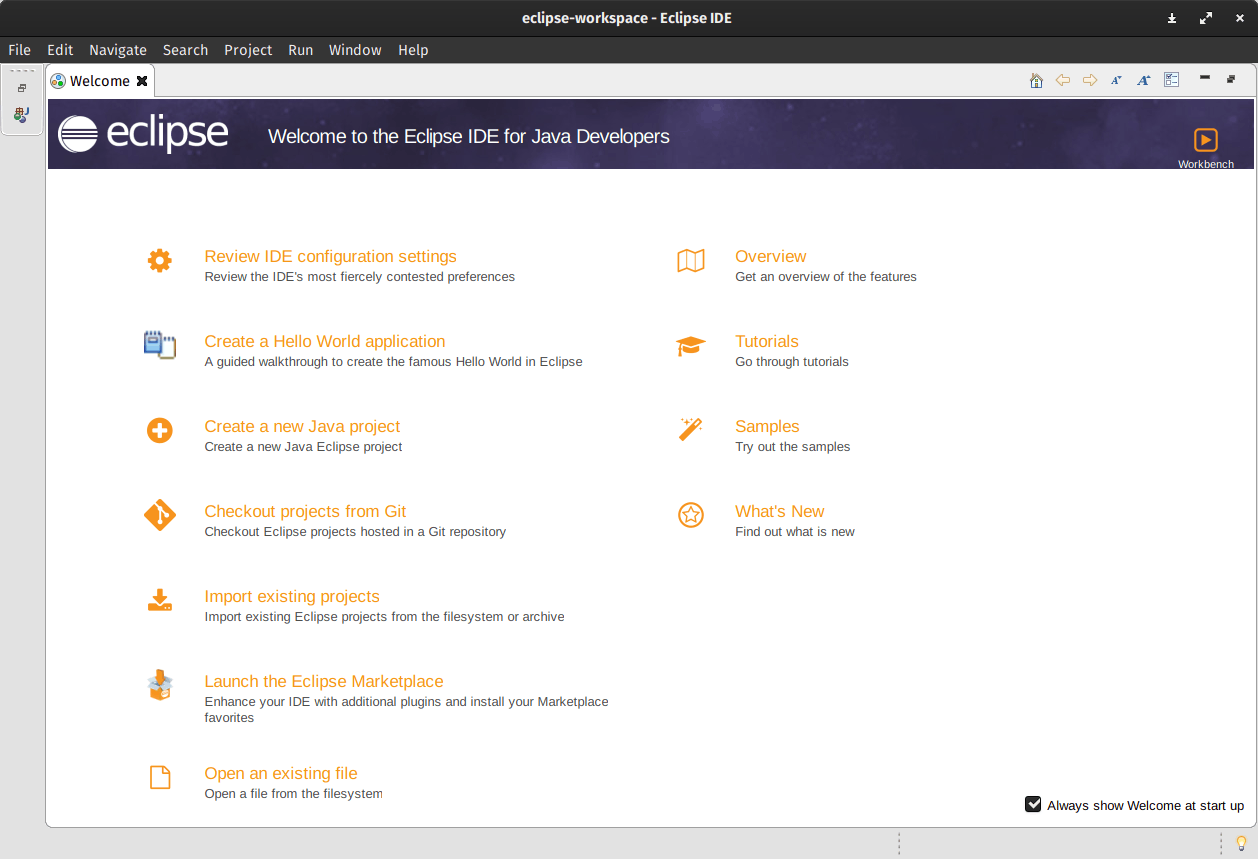


Figure 1.1(Eclipse IDE)

### 

### **GRADLE**

Gradle is a build automation tool known for its flexibility to build software. A build automation tool is used to automate the creation of applications. The building process includes compiling, linking, and packaging the code. The process becomes more consistent with the help of build automation tools.



Figure 1.2(Gradle)

* + 1. **POSTGRES**

PostgreSQL is a powerful, open source object-relational database system with over 30 years of active development that has earned it a strong reputation for reliability, feature robustness, and performance

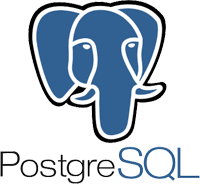


Figure 1.3 (PostgreSQL)

* + 1. **DOCKER**

Docker is an open source containerization platform. It enables developers to package applications into containers—standardized executable components combining application source code with the operating system (OS) libraries and dependencies required to run that code in any environment.

# **KUBERNETES**

Kubernetes is a portable, extensible, open source platform for managing containerized workloads and services, that facilitates both declarative configuration and automation. It has a large, rapidly growing ecosystem. Kubernetes services, support, and tools are widely available.

* 1. HARDWARE
* RAM - Minimum 8 GB RAM is required
* Processor - Intel i3/i5
* Operating System – Window10/Lubuntu
  1. GAINT CHART

## Gantt Chart

When creating a project schedule, the planner begins with a set of tasks (the work breakdown structure). If automated tools are used, the work breakdown is input as a task network. Effort, duration and start dates are input are each task network. As a consequence of this input, a timeline chart also called a Gantt chart is generated. A timeline chart is developed for entire project.

Gantt chart for project:



Figure 1.4 gaint chart

**CHAPTER 2**

**LITERATURE REVIEW**

[1] By using JAVA Programming approach relies on software for distributed batch and stream processing, as well as distributed storage. This chapter focuses on an oft-ignored angle of assuredness: performance assuredness. A significant pain point today is the inability to support reconfiguration operations, such as changing of the shard key in a shared storage/database system, or scaling up (or down) of the number of virtual machines (VMs) being used in a stream or batch processing system.

[2] By using Software Design approach discuss new techniques to support such reconfiguration operations in an online manner, whereby the system does not need to be shut down and the user/client-perceived behavior is indistinguishable regardless of whether a reconfiguration is occurring in the background, that is, the performance continues to be assured in spite of ongoing background reconfiguration. Next, we describe how to scale-out and scale-in (increase or decrease) the number of machines/VMs in cloud computing frameworks like distributed stream processing and distributed graph processing systems, again while offering assured performance to the customer in spite of the reconfigurations occurring in the background.

[3] By Using Fundamentals of Java Programming approach is the ability to support SLAs/SLOs (service-level agreements/objectives) such as deadlines. We present a new real-time scheduler that supports priorities and hard deadlines for Hadoop jobs. We implemented our reconfiguration systems as patches to several popular and open-source cloud computing systems, including MongoDB and Cassandra (storage), Storm (stream processing), LF Graph (graph processing), and Hadoop (batch processing).

[4] By Using Java in two semesters by approach has become the industry standard for rapid application deployment, scalable server support, mobile and distributed services, and it provides access to (theoretically) infinite resources. Unfortunately, researchers are still trying to converge towards cross-provider cloud computing frameworks to enable compatibility and seamless resource transition between cloud providers. Moreover, users are restricted to using the provider-specific pre-configured options of resources and services, irrespective of their current needs. At the same time, cloud services are provided as a direct service from the providers to the clients. This creates a segregated cloud market clientele, and non-negotiable pricing

[5] By using Learning Java with Games In this paper, we propose Java, a generic architecture for cloud composition and negotiated service delivery for cloud users. Jugo acts as a match-maker for service specifications from the users with the currently available assets from the cloud providers. The engagement of a middle-man as an opaque cloud service provider will create a better opportunity for cloud users to find cheaper deals, price-matching, and flexible resource specifications, with increased revenue and higher resource utilization for the cloud service providers.

[6] By using Computing and Software Science approach Many enterprises in industries start using Cloud Computing for their IT infrastructure services. This adoption of Cloud Computing is a part of the enterprise transformation which is the migration from a legacy IT environment to Cloud Computing. On the other hand, one of major targets is an industry solution which provides a critical business service to their end customers. This paper proposes Industry Cloud which is the enhanced design of Cloud Computing for industry solutions. It efficiently supports industry solutions for enterprise business requirements. The paper describes Industry Cloud with a requirement analysis of industry solutions, those adopted functions, and three use case scenarios in the electronics and retail industry. The contribution of the paper is the analysis of industry wide requirements, the definition of Industry Cloud with a common function among industry solutions and the usage with use case scenarios.

[7] By using Springboot approach the complexity of Cloud infrastructures is increasing every year, requiring new concepts and tools to face off topics such as process configuration and reconfiguration, automatic scaling, elastic computing and healthiness control. This paper presents a Smart Cloud solution based on a Knowledge Base, KB, with the aim of modeling cloud resources, Service Level Agreements and their evolution, while enabling the reasoning on cloud structures and implementing strategies of efficient smart cloud management and intelligence.

[8] By using CRUD approach, the solution proposed is composed of Smart Cloud Engine, SCE, the Knowledge Base, KB, and the Supervisor and Monitoring module for data acquisition. It can be easily integrated with any cloud configuration manager, cloud orchestra or, and monitoring tool, since the connections with these tools are performed by using REST calls.

[9] By using AWS approach, in addition to the huge number of dedicated servers deployed in data centers, there are billions of underutilized Personal Computers (PCs), usually used only for a few hours per day, owned by individuals and organizations worldwide. The vast untapped compute and storage capacities of the underutilized PCs can be consolidated as alternative cloud fabrics to provision broad cloud services, primarily infrastructure as a service

[10] By using SWAGGER approach to as "no data center" approach, complements the data center-based cloud provision model. In this paper, we present our opportunistic Cloud Computing system, called cu Cloud, that runs on scavenged resources of underutilized PCs within an organization/community. Our system demonstrates that the "no data center" solution indeed works. Besides proving our concept, model, and philosophy, our experimental results are highly encouraging.

[11] By using KUBERNETES approach Whatever one public cloud, private cloud or a mixed cloud, the users lack of effective security quantifiable evaluation methods to grasp the security situation of its own information infrastructure on the whole. This paper provides a quantifiable security evaluation system for different clouds that can be accessed by consistent API. The evaluation system includes security scanning engine, security recovery engine, security quantifiable evaluation model, visual display module and etc.

[12] By using DOCKER approach, The model composes of a set of evaluation elements corresponding different fields, such as computing, storage, network, maintenance, application security and etc. Each element is assigned a three tuple on vulnerabilities, score and repair method. The system adopts “One vote vetoed” mechanism for one field to count its score and adds up the summary as the total score, and to create one security view.

[13] By using PostgreSQL database approach We implement the quantifiable evaluation for different cloud users based on our AWS platform. It shows the dynamic security scanning score for one or multiple clouds with visual graphs and guided users to modify configuration, improve operation and repair vulnerabilities, so as to improve the security of their cloud resource.

[14] By using MySQL database approach, we move applications to the cloud is not only a technical decision but also a business-oriented decision, in which both business and technical factors (e.g., transformation effort multi-tenancy and auto-scaling enablement, scalability and extensibility) should be considered. However, existing approaches and tools do not support a consumable business-oriented cloud transformation decision to select more suitable transformation solution with the right cloud delivery model, services type, affordable transformation effort and etc.

[15] By using Gradle approach we introduce a practical three-step approach and a tool, CTA (Cloud Transformation Advisor) to enable decision makers to identify the most suitable cloud transformation solution to satisfy their business goals based on a well-structured cloud transformation knowledge base

**CHAPTER 3**

**TECHNICAL FEASBILITY**

## 3.1 INTRODUCTION

Feasibility of the system in an important aspect, which is to be considered. The system needs to satisfy the law of economic, which states that the maximum output should be yielded in minimum available resources.

A feasibility analysis evaluates the project’s potential for success; therefore, perceived objectivity is an essential factor in the credibility of the study for potential investors and lending institutions. There are five types of feasibility study separate areas that a feasibility study examines, described below.

* + 1. **Technical Feasibility**

This assessment focuses on the technical resources available to the organization. It helps organizations determine whether the technical resources meet capacity and whether the technical team is capable of converting the ideas into working systems. Technical feasibility also involves the evaluation of the hardware, software, and other technical requirements of the proposed system. As an exaggerated example, an organization wouldn’t want to try to put Star Trek’s transporters in their building currently, this project is not technically feasible.

**3.1.2. Economic Feasibility**

This assessment typically involves a cost/ benefits analysis of the project, helping organizations determine the viability, cost, and benefits associated with a project before financial resources are allocated. It also serves as an independent project assessment and enhances project credibility—helping decision-makers determine the positive economic benefits to the organization that the proposed project will provide.

**3.1.3. Legal Feasibility**

This assessment investigates whether any aspect of the proposed project conflicts with legal requirements like zoning laws, data protection acts or social media laws. Let’s say an organization wants to construct a new office building in a specific location. A feasibility study might reveal the organization’s ideal location isn’t zoned for that type of business.

That organization has just saved considerable time and effort by learning that their project was not feasible right from the beginning.

**3.1.4. Operational Feasibility**

This assessment involves undertaking a study to analyze and determine whether and how well the organization’s needs can be met by completing the project. Operational feasibility

studies also examine how a project plan satisfies the requirements identified in the requirements analysis phase of system development.

* + 1. Scheduling Feasibility

This assessment is the most important for project success; after all, a project will fail if not completed on time. In scheduling feasibility, an organization estimates how much time the project will take to complete.

When these areas have all been examined, the feasibility analysis helps identify any constraints the proposed project may face, including:

* Internal Project Constraints: Technical, Technology, Budget, Resource, etc.
* Internal Corporate Constraints: Financial, Marketing, Export, etc.
* External Constraints: Logistics, Environment, Laws, and Regulations, etc.

**3.2 TECHNOLOGY DESCRIPTION**

**3.2.1 SPRING BOOT**

Spring Boot provides a good platform for Java developers to develop a stand-alone and production-grade spring application that you can just run.

You can get started with minimum configurations without the need for an entire Spring configuration setup.

Spring Boot automatically configures your application based on the dependencies you have added to the project by using @EnableAutoConfiguration annotation. For example, if MySQL database is on your classpath, but you have not configured any database connection, then Spring Boot auto-configures an in-memory database.

The entry point of the spring boot application is the class contains **@**SpringBootApplication annotation and the main method.

Spring Boot automatically scans all the components included in the project by using **@**ComponentScan annotation.

**3.2.2 DOCKER**

Docker is an open source containerization platform. It enables developers to package applications into containers—standardized executable components combining application source code with the operating system (OS) libraries and dependencies required to run that code in any environment.

Docker is an open source [containerization](https://www.ibm.com/in-en/cloud/learn/containerization) platform. It enables developers to package applications into containers—standardized executable components combining application source code with the operating system (OS) libraries and dependencies required to run that code in any environment. Containers simplify delivery of distributed applications, and have become increasingly popular as organizations shift to cloud-native development and hybrid [multicloud](https://www.ibm.com/cloud/learn/multicloud) environments.

Developers can create containers without Docker, but the platform makes it easier, simpler, and safer to build, deploy and manage containers. Docker is essentially a toolkit that enables developers to build, deploy, run, update, and stop containers using simple commands and work-saving automation through a single API.

Docker also refers to [Docker, Inc.](https://www.docker.com/company) (link resides outside IBM), the company that sells the commercial version of Docker, and to the [Docker open source project](https://github.com/docker)(link resides outside IBM), to which Docker, Inc. and many other organizations and individuals contribute.

### **Notable Features of Spring Boot**

1. **Autoconfiguration:** Developers can automatically configure their Spring application. However, Spring Boot is also capable of changing the configuration based on the dependencies you list. For example, when you list “MySQL” as a dependency, it will configure your Spring application with the “MySQL connector” included. And if you want to add a custom configuration, you can create a class that overrides the default configuration for your “MySQL connector”.
2. **Standalone:** There’s no need to deploy your application to a web server. You simply enter the run command to start the application.
3. **Opinionated:** On the official page, we find that Spring Boot decides for you which defaults to use for the configuration. Also, it decides which packages to install for the dependencies you require. For example, if you include the Spring Boot starter “pom” for “JPA”, it will autoconfigure an in-memory database, a hibernate entity manager, and a simple data source. This is an example of an opinionated default configuration that you can override. While some developers might feel this is too opinionated, Spring Boot’s opinionated setup helps developers to get started quickly on their projects.

**3.2.3 KUBERNETES**

Kubernetes is a portable, extensible, open source platform for managing containerized workloads and services, that facilitates both declarative configuration and automation. It has a large, rapidly growing ecosystem. Kubernetes services, support, and tools are widely available.

Kubernetes, often abbreviated as “K8s”, orchestrates containerized applications to run on a cluster of hosts. The K8s system automates the deployment and management of cloud native applications using on-premises infrastructure or public cloud platforms. It distributes application workloads across a Kubernetes cluster and automates dynamic container networking needs. Kubernetes also allocates storage and persistent volumes to running containers, provides automatic scaling, and works continuously to maintain the desired state of applications, providing resiliency.

Kubernetes has many features that help orchestrate containers across multiple hosts, automate the management of K8s clusters, and maximize resource usage through better utilization of infrastructure. Important features include:

* **Auto-scaling**. Automatically scale containerized applications and their resources up or down based on usage
* **Lifecycle management**. Automate deployments and updates with the ability to:
  + Rollback to previous versions
  + Pause and continue a deployment
* **Declarative model**. Declare the desired state, and K8s works in the background to maintain that state and recover from any failures
* **Resilience and self-healing**. Auto placement, auto restart, auto replication and auto scaling provide application self-healing
* **Persistent storage**. Ability to mount and add storage dynamically
* **Load balancing**. Kubernetes supports a variety of internal and external load balancing options to address diverse needs
* **DevSecOps support**. [DevSecOps](https://tanzu.vmware.com/devsecops) is an advanced approach to security that simplifies and automates container operations across clouds, integrates security throughout the container lifecycle, and enables teams to deliver secure, high-quality software more quickly. Combining DevSecOps practices and Kubernetes improves developer productivity.

**3.2.4 GOOGLE CLOUD PLATEFORM**

Google Cloud is a suite of public cloud computing services offered by Google. The platform includes a range of hosted services for compute, storage and application development that run on Google hardware. Google Cloud services can be accessed by software developers, cloud administrators and other enterprise IT professionals over the public internet or through a dedicated network connection.

* Google Compute Engine, which is an infrastructure as a service (IaaS) offering that provides users with VM instances for workload hosting.
* Google App Engine, which is a platform as a service (PaaS) offering that gives software developers access to Google's scalable hosting. Developers can also use an SDK to develop software products that run on App Engine.
* Google Cloud Storage, which is a cloud storage platform designed to store large, unstructured data sets. Google also offers database storage options, including Cloud Datastore for NoSQL nonrelational storage, Cloud SQL for MySQL fully relational storage and Google's native Cloud Bigtable database.
* Google Kubernetes Engine (GKE), which is a management and orchestration system for Docker container and container clusters that run within Google's public cloud services. Google Kubernetes Engine is based on Kubernetes, Google's open source container management system.
* Google Cloud's operations suite, formerly Stackdriver, which is a set of integrated tools for monitoring, logging and reporting on the managed services driving applications and systems on Google Cloud.
* Serverless computing, which provides tools and services for event-based workload execution, such as Cloud Functions for creating functions that handle cloud events, Cloud Run for managing and running containerized applications and Workflows to orchestrate serverless products and APIs.
* Databases, which is a suite of database products delivered as completely managed services, including Cloud Bigtable for large-scale, low-latency workloads; Firestore for documents; CloudSpanner as a highly scalable, highly reliable relational database; and CloudSQL as a fully managed database for MySQL, PostgreSQL and SQL Server.

Google Cloud offers application development and integration services. For example, Google Cloud Pub/Sub is a managed and real-time messaging service that allows messages to be exchanged between applications. In addition, Google Cloud Endpoints enables developers to create services based on RESTful APIs and then make those services accessible to Apple iOS, Android and JavaScript clients. Other offerings include Anycast DNS servers, direct network interconnections, load balancing, monitoring and logging services.

**3.3TECHNOLOGY USED**

**3.3.1 SPRING BOOT**

Spring Boot provides a good platform for Java developers to develop a stand-alone and production-grade spring application that you can just run.

You can get started with minimum configurations without the need for an entire Spring configuration setup.

Spring Boot automatically configures your application based on the dependencies you have added to the project by using @EnableAutoConfiguration annotation. For example, if MySQL database is on your classpath, but you have not configured any database connection, then Spring Boot auto-configures an in-memory database.

The entry point of the spring boot application is the class Contains **@**SpringBootApplication annotation and the main method.

Spring Boot automatically scans all the components included in the project by using **@**ComponentScan annotation.

**3.3.2 DOCKER**

Docker is an open source containerization platform. It enables developers to package applications into containers—standardized executable components combining application source code with the operating system (OS) libraries and dependencies required to run that code in any environment.

Docker is an open source [containerization](https://www.ibm.com/in-en/cloud/learn/containerization) platform. It enables developers to package applications into containers—standardized executable components combining application source code with the operating system (OS) libraries and dependencies required to run that code in any environment. Containers simplify delivery of distributed applications, and have become increasingly popular as organizations shift to cloud-native development and hybrid [multicloud](https://www.ibm.com/cloud/learn/multicloud) environments.

Developers can create containers without Docker, but the platform makes it easier, simpler, and safer to build, deploy and manage containers. Docker is essentially a toolkit that enables developers to build, deploy, run, update, and stop containers using simple commands and work-saving automation through a single API.

**3.3.3 KUBERNETES**

Kubernetes is a portable, extensible, open source platform for managing containerized workloads and services, that facilitates both declarative configuration and automation. It has a large, rapidly growing ecosystem. Kubernetes services, support, and tools are widely available.

Kubernetes, often abbreviated as “K8s”, orchestrates containerized applications to run on a cluster of hosts. The K8s system automates the deployment and management of cloud native applications using on-premises infrastructure or public cloud platforms. It distributes application workloads across a Kubernetes cluster and automates dynamic container networking needs. Kubernetes also allocates storage and persistent volumes to running containers, provides automatic scaling, and works continuously to maintain the desired state of applications, providing resiliency.

**3.3.4 REST API**

A REST API (also known as RESTful API) is an application programming interface (API or web API) that conforms to the constraints of REST architectural style and allows for interaction with RESTful web services. REST stands for representational state transfer and was created by computer scientist Roy Fielding

**3.3.5 SWAGGER**

Swagger is a set of open source tools for writing REST-based APIs. It simplifies the process of writing APIs by notches, specifying the standards & providing the tools required to write beautiful, safe, performant & scalable APIs.

**3.3.6 GOOGLE CLOUD PLATEFORM**

Google Cloud is a suite of public cloud computing services offered by Google. The platform includes a range of hosted services for compute, storage and application development that run on Google hardware. Google Cloud services can be accessed by software developers, cloud administrators and other enterprise IT professionals over the public internet or through a dedicated network connection.

**CHAPTER 4**

**BACKEND DESIGN**

**4.1 DATA DICTIONARY**

|  |  |  |
| --- | --- | --- |
| Microservice Name | Database Name | Table Name |
| Stock | **stockms** | **stock\_details** |
| Supplied Stock | **suppliedstockms** | **supplied\_stock\_details** |
| Order | **orderms** | **order\_details** |
| Report | **reportms** | **report\_details** |
| Delivery | **deliveryms** | **delivery\_details** |

Table 4.1

* 1. **ER diagram**

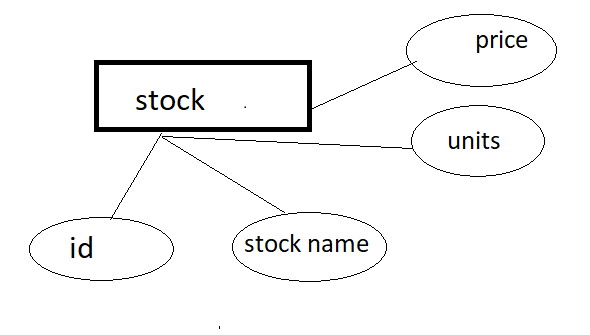
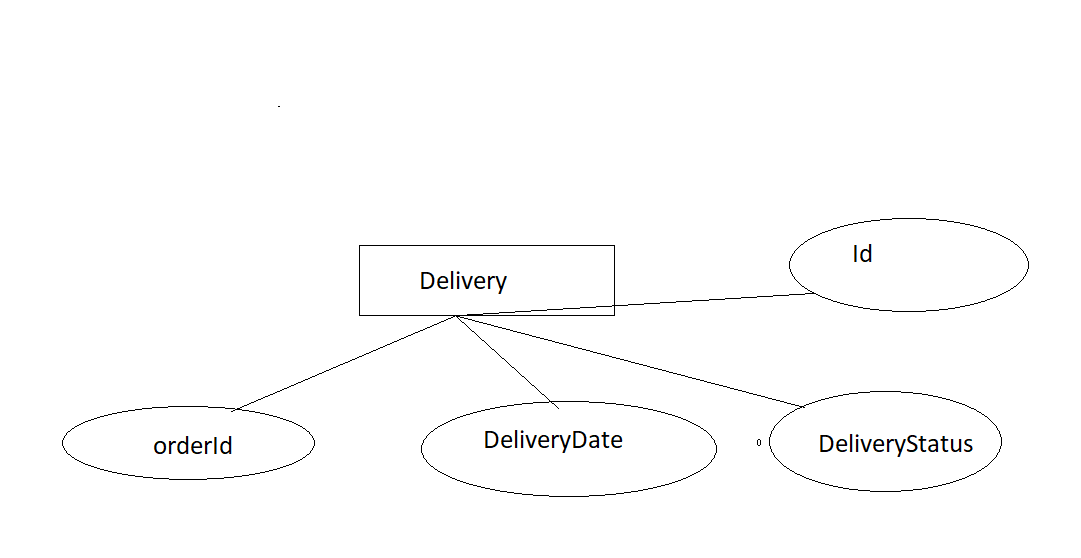
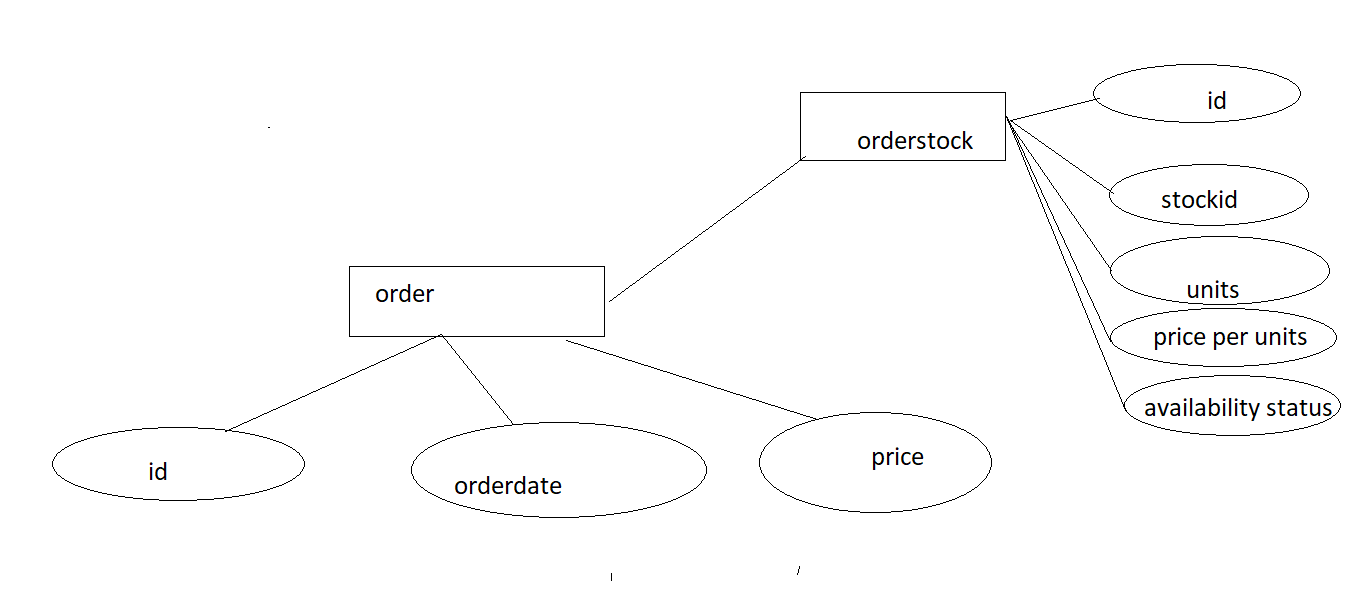
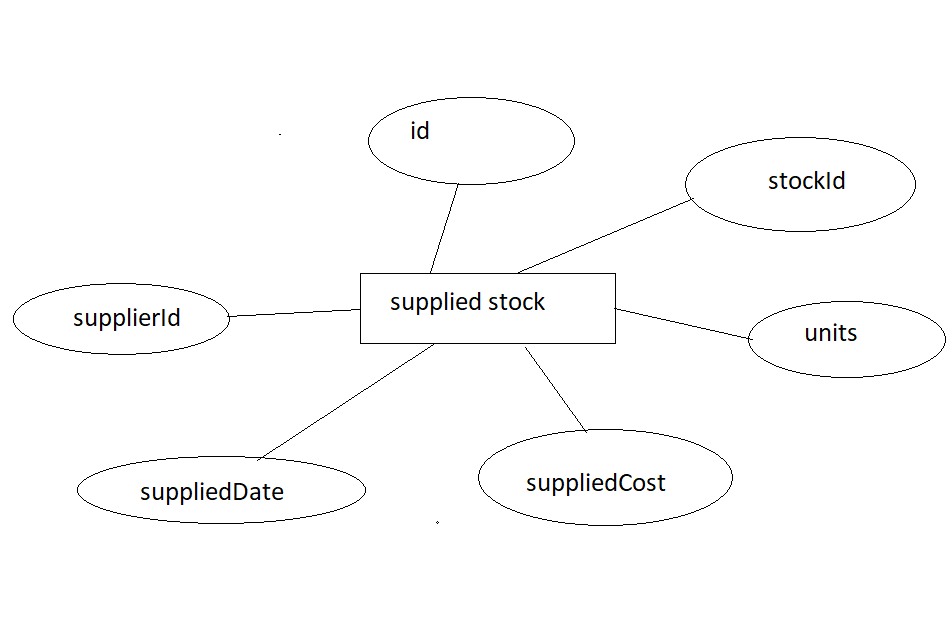
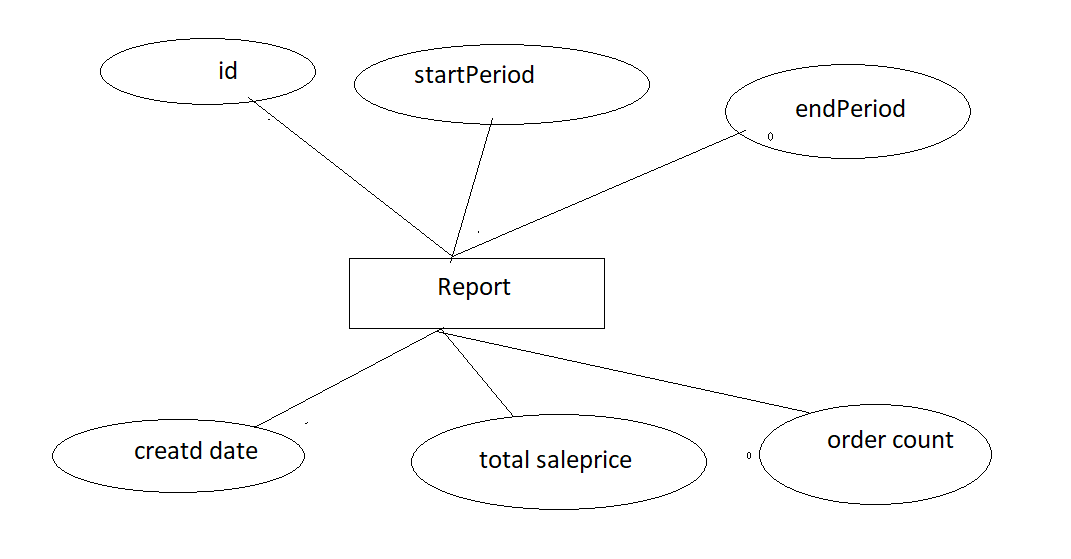
**** **** **** 

Figure 4.1(ER diagram)

* 1. **DATABASE DESIGN**
     1. **STOCK**

stock\_details

|  |  |
| --- | --- |
| Id | Int |
| Stock name | varchar(20) |
| Price | int |
| units | int |

Table 4.2

supplied\_stock\_details

|  |  |
| --- | --- |
| Id | Int |
| stockid | Int |
| supplierid | Int |
| units | Int |
| suppliedcost | Double |
| supplieddate | Date(yyyy/mm/dd) |

Table 4.3

order\_details

|  |  |
| --- | --- |
| Id | Int |
| Orderdate | Date(yyyy/mm/dd) |
| price | double |

Table 4.4

delivery\_details

|  |  |
| --- | --- |
| Id | Int |
| Orderid | Int |
| Deliverydate | Date(yyyy/mm/dd) |
| deliverystatus | Varchar(20) |

Table 4.5

report\_details

|  |  |
| --- | --- |
| Id | Int |
| Startperiod | Date(yyyy/mm/dd) |
| Endperiod | Date(yyyy/mm/dd) |
| Createddate | Date(yyyy/mm/dd) |
| Totalsaleprice | Double |
| ordercount | Int |

Table 4.6

order\_stock\_details

|  |  |
| --- | --- |
| Id | Int |
| Stockid | Int |
| Units | Int |
| Priceperunit | Int |
| Availabilitystatus | Varchar(20) |

Table 4.7

**CHAPTER 5**

**CODE**

SuppliedStockmsApplication.java

package com.trainingapps.stockapp.suppliedstockms;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

//import org.springframework.cloud.netflix.eureka.EnableEurekaClient;

import org.springframework.context.annotation.Bean;

import org.springframework.web.client.RestTemplate;

@SpringBootApplication

public class SuppliedstockmsApplication {

public static void main(String[] args) {

SpringApplication.run(SuppliedstockmsApplication.class, args);

}

@Bean

public RestTemplate restTemplate() {

return new RestTemplate();

}

}

SuppliedStockController.java

package com.trainingapps.stockapp.suppliedstockms.controller;

import java.util.List;

import javax.validation.Valid;

import javax.validation.constraints.Min;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.validation.annotation.Validated;

import org.springframework.web.bind.annotation.GetMapping;

import org.springframework.web.bind.annotation.PathVariable;

import org.springframework.web.bind.annotation.PostMapping;

import org.springframework.web.bind.annotation.RequestBody;

import org.springframework.web.bind.annotation.RequestMapping;

import org.springframework.web.bind.annotation.RequestParam;

import org.springframework.web.bind.annotation.RestController;

import com.trainingapps.stockapp.suppliedstockms.dto.AddSupplyStockRequest;

import com.trainingapps.stockapp.suppliedstockms.dto.SuppliedStockDetails;

import com.trainingapps.stockapp.suppliedstockms.exception.SuppliedStockNotFoundException;

import com.trainingapps.stockapp.suppliedstockms.service.ISuppliedStockService;

/\*\*

\* @author DELL

\* 3 rest end points have been created

\* 1.) localhost:8586/suppliedstock/add => for adding the supppliedstock details

\* 2.) localhost:8586/suppliedstock/findByStockId/{stockId} => for finding the suppplied stock by stockid

\* 3.) localhst:8586/suppliedstock/findBySupplierId => for findng by supplierId in given time frame

\*

\*/

@RequestMapping("/suppliedstock")

@RestController

@Validated

public class SuppliedStockController {

@Autowired

private ISuppliedStockService service;

@PostMapping("/add")

public SuppliedStockDetails addSuppliedStock(@RequestBody @Valid AddSupplyStockRequest request) {

return service.add(request);

}

//@RequestMapping(value="/findByStockId/{id}", method=RequestMethod.GET)

@GetMapping("/findByStockId/{stockId}")

public List<SuppliedStockDetails> findStockById(@PathVariable @Min(1) long stockId)

throws SuppliedStockNotFoundException {

List<SuppliedStockDetails> response = service.findSuppliedStockDetailsByStockId(stockId);

return response;

}

@GetMapping("/findBySupplierId")

public List<SuppliedStockDetails> findAll(@RequestParam Long supplierId, @RequestParam String startDate,

@RequestParam String endDate) {

List<SuppliedStockDetails> orders = service.findSuppliedStockBySupplierId(supplierId, startDate, endDate);

return orders;

}

}

DTO

AddSupplyStockRequest.java

package com.trainingapps.stockapp.suppliedstockms.dto;

import javax.persistence.Column;

import javax.validation.constraints.Min;

import javax.validation.constraints.NotNull;

/\*\*

\* @author DELL adding supplied stock

\* stockId => !< 1

\* supplierId => !< 1

\* suppliedCost => != null

\* units => !< 1

\*

\*/

public class AddSupplyStockRequest {

@Min(1)

@Column(nullable = false)

private long stockId;

@Min(1)

@Column(nullable = false)

private long supplierId;

@Column(nullable = false)

private String suppliedDate;

@Column(nullable = false)

private Double suppliedCost;

@Min(1)

@Column(nullable = false)

private int units;

public long getStockId() {

return stockId;

}

public void setStockId(long stockId) {

this.stockId = stockId;

}

public long getSupplierId() {

return supplierId;

}

public void setSupplierId(long supplierId) {

this.supplierId = supplierId;

}

public String getSuppliedDate() {

return suppliedDate;

}

public void setSuppliedDate(String suppliedDate) {

this.suppliedDate = suppliedDate;

}

public Double getSuppliedCost() {

return suppliedCost;

}

public void setSuppliedCost(Double suppliedCost) {

this.suppliedCost = suppliedCost;

}

public int getUnits() {

return units;

}

public void setUnits(int units) {

this.units = units;

}

}

SuppliedStockDetails.java

package com.trainingapps.stockapp.suppliedstockms.dto;

/\*\*

\* @author DELL

\* supplied stock details

\*

\*/

public class SuppliedStockDetails {

private long id;

private long stockId;

private long supplierId;

private String suppliedDate;

private Double suppliedCost;

private int units;// number of units supplied

public long getId() {

return id;

}

public void setId(long id) {

this.id = id;

}

public long getStockId() {

return stockId;

}

public void setStockId(long stockId) {

this.stockId = stockId;

}

public long getSupplierId() {

return supplierId;

}

public void setSupplierId(long supplierId) {

this.supplierId = supplierId;

}

public String getSuppliedDate() {

return suppliedDate;

}

public void setSuppliedDate(String dateText) {

this.suppliedDate = dateText;

}

public Double getSuppliedCost() {

return suppliedCost;

}

public void setSuppliedCost(Double suppliedCost) {

this.suppliedCost = suppliedCost;

}

public int getUnits() {

return units;

}

public void setUnits(int units) {

this.units = units;

}

}

Entity

SuppliedStock.java

package com.trainingapps.stockapp.suppliedstockms.entity;

import java.time.LocalDate;

import java.util.Objects;

import javax.persistence.Column;

import javax.persistence.Entity;

import javax.persistence.GeneratedValue;

import javax.persistence.Id;

import javax.persistence.Table;

/\*\*

\* @author DELL Entity class table name => SuppliedStockdetails primary field =

\* PrimaryField =>id => Auto generated

\*

\*/

@Table(name = "SuppliedStockdetails")

@Entity

public class SuppliedStock {

@GeneratedValue

@Id

private Long id;

@Column(nullable = false)

private Long stockId;

@Column(nullable = false)

private Long supplierId;

@Column(nullable = false)

private LocalDate suppliedDate;

@Column(nullable = false)

private Double suppliedCost;

@Column(nullable = false)

private int units;

public Long getId() {

return id;

}

public void setId(Long id) {

this.id = id;

}

public Long getStockId() {

return stockId;

}

public void setStockId(Long stockId) {

this.stockId = stockId;

}

public Long getSupplierId() {

return supplierId;

}

public void setSupplierId(Long supplierId) {

this.supplierId = supplierId;

}

public LocalDate getSuppliedDate() {

return suppliedDate;

}

public void setSuppliedDate(LocalDate suppliedDate) {

this.suppliedDate = suppliedDate;

}

public Double getSuppliedCost() {

return suppliedCost;

}

public void setSuppliedCost(Double suppliedCost) {

this.suppliedCost = suppliedCost;

}

public int getUnits() {

return units;

}

public void setUnits(int units) {

this.units = units;

}

@Override

public int hashCode() {

return Objects.*hash*(id);

}

@Override

public boolean equals(Object obj) {

if (this == obj)

return true;

if (obj == null)

return false;

if (getClass() != obj.getClass())

return false;

SuppliedStock other = (SuppliedStock) obj;

return Objects.*equals*(id, other.id);

}

}

Manualtesting.java/FrontEnd.java

package com.trainingapps.stockapp.suppliedstockms.frontend;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.stereotype.Component;

import com.trainingapps.stockapp.suppliedstockms.dto.AddSupplyStockRequest;

import com.trainingapps.stockapp.suppliedstockms.dto.SuppliedStockDetails;

import com.trainingapps.stockapp.suppliedstockms.service.ISuppliedStockService;

/\*\*

\* @author DELL for manual testing runUI method is created and for display the

\* display method has been created

\*

\*/

@Component

public class ManualTesting {

@Autowired

private ISuppliedStockService service;

void runUI() {

try {

System.*out*.println("\*\*\*adding supplied Stock\*\*\*\*");

AddSupplyStockRequest requestData1 = new AddSupplyStockRequest();

requestData1.setStockId((long) 7);

requestData1.setSupplierId((long) 3);

requestData1.setSuppliedDate("05-04-2022");

requestData1.setUnits(5);

requestData1.setSuppliedCost(50000.0);

SuppliedStockDetails suppliedStock1 = service.add(requestData1);

AddSupplyStockRequest requestData2 = new AddSupplyStockRequest();

requestData2.setStockId((long) 3);

requestData2.setSupplierId((long) 4);

requestData2.setSuppliedDate("06-04-2022");

requestData2.setUnits(5);

requestData2.setSuppliedCost(10000.0);

SuppliedStockDetails suppliedStock2 = service.add(requestData2);

AddSupplyStockRequest requestData3 = new AddSupplyStockRequest();

requestData3.setStockId((long) 2);

requestData3.setSupplierId((long) 2);

requestData3.setSuppliedDate("07-04-2022");

requestData3.setUnits(5);

requestData3.setSuppliedCost(8000.0);

SuppliedStockDetails suppliedStock3 = service.add(requestData3);

System.*out*.println("\*\*\*Displaying the record\*\*\*\*");

display(suppliedStock1);

display(suppliedStock2);

display(suppliedStock3);

} catch (Exception e) {

System.*err*.println(e.getMessage());

}

}

void display(SuppliedStockDetails supplyStock) {

System.*out*.println(supplyStock.getId() + " - " + supplyStock.getStockId() + " - " + supplyStock.getSupplierId()

+ " - " + supplyStock.getSuppliedDate() + " - " + supplyStock.getUnits() + " - "

+ supplyStock.getSuppliedCost());

}

}

Service class

Suppliedstockserviceimpl.java

package com.trainingapps.stockapp.suppliedstockms.service;

import java.time.LocalDate;

import java.util.List;

import javax.transaction.Transactional;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.stereotype.Service;

import com.trainingapps.stockapp.suppliedstockms.dto.AddSupplyStockRequest;

import com.trainingapps.stockapp.suppliedstockms.dto.SuppliedStockDetails;

import com.trainingapps.stockapp.suppliedstockms.entity.SuppliedStock;

import com.trainingapps.stockapp.suppliedstockms.exception.SuppliedStockNotFoundException;

import com.trainingapps.stockapp.suppliedstockms.repository.ISuppliedStockRepository;

import com.trainingapps.stockapp.suppliedstockms.util.DateUtil;

import com.trainingapps.stockapp.suppliedstockms.util.SuppliedStockUtil;

/\*\*

\* @author DELL implementing the service interface

\*

\*/

@Transactional

@Service

public class SuppliedStockServiceImpl implements ISuppliedStockService {

@Autowired

private SuppliedStockUtil suppliedStockutil;

@Autowired

private DateUtil dateUtil;

@Autowired

private ISuppliedStockRepository repository;

@Override

public SuppliedStockDetails add(AddSupplyStockRequest request) {

SuppliedStock suppliedStock = newSuppliedStock();

suppliedStock.setStockId(request.getStockId());

suppliedStock.setSupplierId(request.getSupplierId());

LocalDate date = dateUtil.convertToDate(request.getSuppliedDate());

suppliedStock.setSuppliedDate(date);

suppliedStock.setSuppliedCost(request.getSuppliedCost());

suppliedStock.setUnits(request.getUnits());

suppliedStock = repository.save(suppliedStock);

SuppliedStockDetails details = suppliedStockutil.toSuppliedStockDetails(suppliedStock);

return details;

}

// created for unit testing to add the supplied stock details

public SuppliedStock newSuppliedStock() {

return new SuppliedStock();

}

// find supplied stock details by stockId method and returning in the form of List

@Override

public List<SuppliedStockDetails> findSuppliedStockDetailsByStockId(long stockId)

throws SuppliedStockNotFoundException {

List<SuppliedStock> list = repository.findByStockId(stockId);

if (list.isEmpty()) {

throw new SuppliedStockNotFoundException("Stock Not found");

}

List<SuppliedStockDetails> desired = suppliedStockutil.toListSuppliedStockDetails(list);

return desired;

}

// find supplied stock details by supplierId in given time frame method and returning in the form of List

@Override

public List<SuppliedStockDetails> findSuppliedStockBySupplierId(Long supplierId, String startDateText,

String endDateText) {

LocalDate startDate = dateUtil.convertToDate(startDateText);

LocalDate endDate = dateUtil.convertToDate(endDateText);

return findSuppliedStockBySupplierId(supplierId, startDate, endDate);

}

// for converting date text to local date format

public List<SuppliedStockDetails> findSuppliedStockBySupplierId(Long supplierId, LocalDate startDate,

LocalDate endDate) {

List<SuppliedStock> list = repository.findBySupplierIdInPeriod(supplierId, startDate, endDate);

List<SuppliedStockDetails> desired = suppliedStockutil.toListSuppliedStockDetails(list);

return desired;

}

}

Util class

package com.trainingapps.stockapp.suppliedstockms.util;

import java.util.ArrayList;

import java.util.Collection;

import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.stereotype.Component;

import com.trainingapps.stockapp.suppliedstockms.dto.SuppliedStockDetails;

import com.trainingapps.stockapp.suppliedstockms.entity.SuppliedStock;

@Component

public class SuppliedStockUtil {

@Autowired

DateUtil dateUtil;

public SuppliedStockDetails toSuppliedStockDetails(SuppliedStock suppliedStock) {

SuppliedStockDetails details = new SuppliedStockDetails();

details.setId(suppliedStock.getId());

details.setStockId(suppliedStock.getStockId());

details.setSupplierId(suppliedStock.getSupplierId());

details.setSuppliedCost(suppliedStock.getSuppliedCost());

String dateText = dateUtil.convertToText(suppliedStock.getSuppliedDate());

details.setSuppliedDate(dateText);

details.setUnits(suppliedStock.getUnits());

return details;

}

public List<SuppliedStockDetails> toListSuppliedStockDetails(List<SuppliedStock> list) {

List<SuppliedStockDetails> desired = new ArrayList<>();

for (SuppliedStock stock : list) {

SuppliedStockDetails response = toSuppliedStockDetails(stock);

desired.add(response);

}

return desired;

}

}

Dependencies

plugins {

id 'org.springframework.boot' version '2.5.3'

id 'io.spring.dependency-management' version '1.0.11.RELEASE'

id 'java'

id 'war'

}

group = 'com.trainingapps'

version = '0.0.1-SNAPSHOT'

sourceCompatibility = '1.8'

repositories {

mavenCentral()

}

ext {

set('springCloudVersion', "2020.0.3")

}

dependencies {

implementation 'org.springframework.boot:spring-boot-starter'

implementation 'org.springframework.boot:spring-boot-starter-web'

implementation 'org.springframework.boot:spring-boot-starter-data-jpa'

implementation 'org.postgresql:postgresql:42.3.3'

implementation 'org.springframework.boot:spring-boot-starter-validation'

runtimeOnly 'org.springdoc:springdoc-openapi-ui:1.5.9'

testImplementation 'org.springframework.boot:spring-boot-starter-test'

implementation 'org.springframework.boot:spring-boot-starter-actuator'

providedRuntime 'org.springframework.boot:spring-boot-starter-tomcat'

implementation 'org.springframework.cloud:spring-cloud-starter-kubernetes:1.1.6.RELEASE'

implementation 'org.springframework.cloud:spring-cloud-starter-kubernetes-config:1.1.6.RELEASE'

}

dependencyManagement {

imports {

mavenBom "org.springframework.cloud:spring-cloud-dependencies:${springCloudVersion}"

}

}

tasks.named('test') {

useJUnitPlatform()

}

Date util

package com.trainingapps.stockapp.suppliedstockms.util;

import java.time.LocalDate;

import java.time.format.DateTimeFormatter;

import org.springframework.stereotype.Component;

/\*\*

\* @author DELL Date format "dd-MM-yyyy" converting date to string and

\* string to date

\*

\*/

@Component

public class DateUtil {

final String pattern = "dd-MM-yyyy";

private DateTimeFormatter format = DateTimeFormatter.*ofPattern*(pattern);

// convert String to LocalDate format

public LocalDate convertToDate(String text) {

LocalDate date = LocalDate.*parse*(text, format);

return date;

}

// convert LocalDate to String format

public String convertToText(LocalDate date) {

String text = date.format(format);

return text;

**CHAPTER 6**

**REPORT**

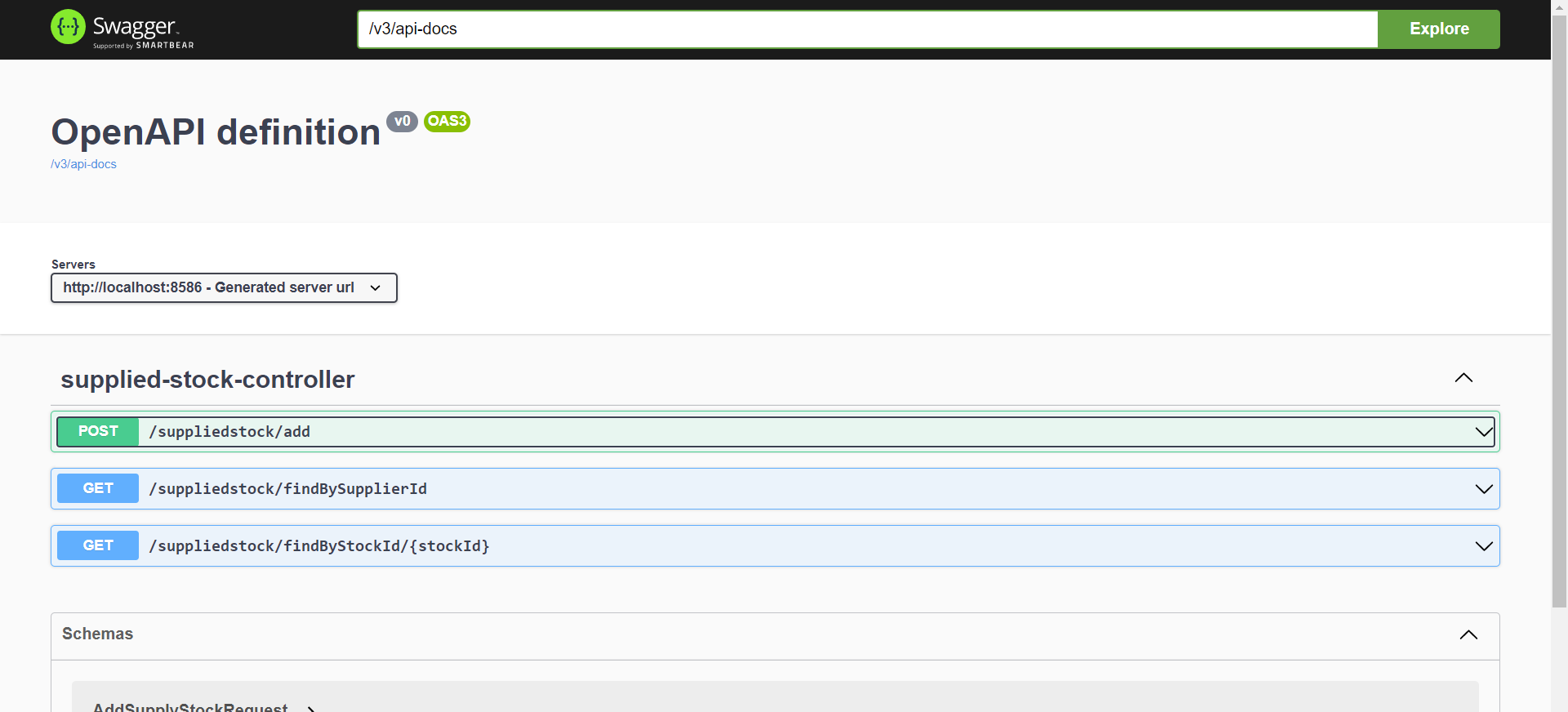
****

Figure 6.1

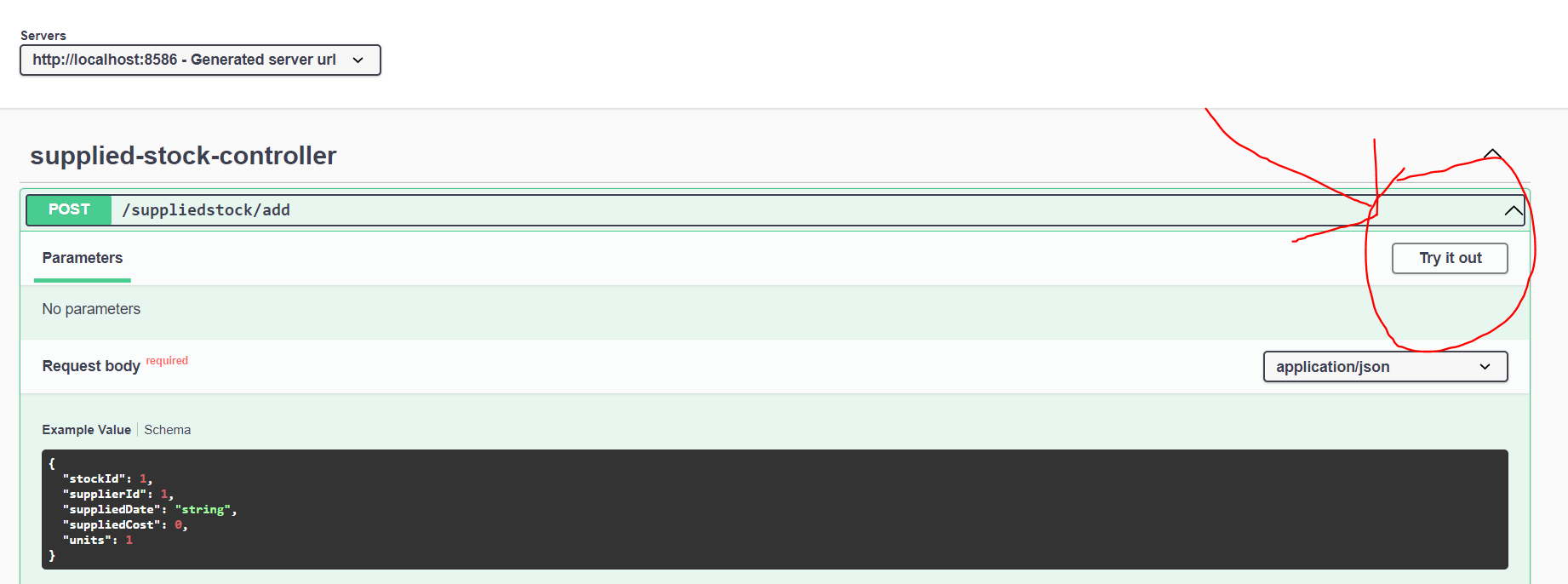


Figure 6.2

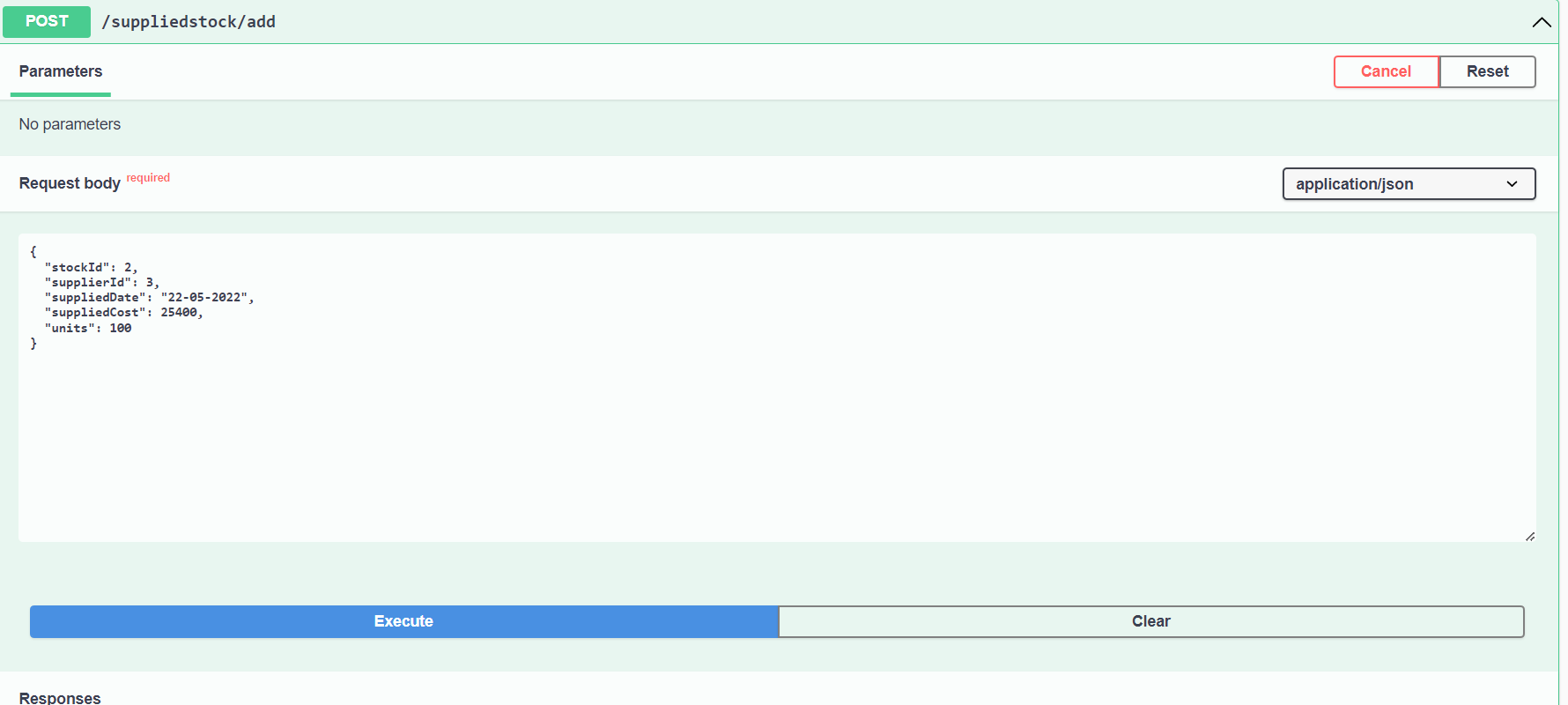


Figure 6.3

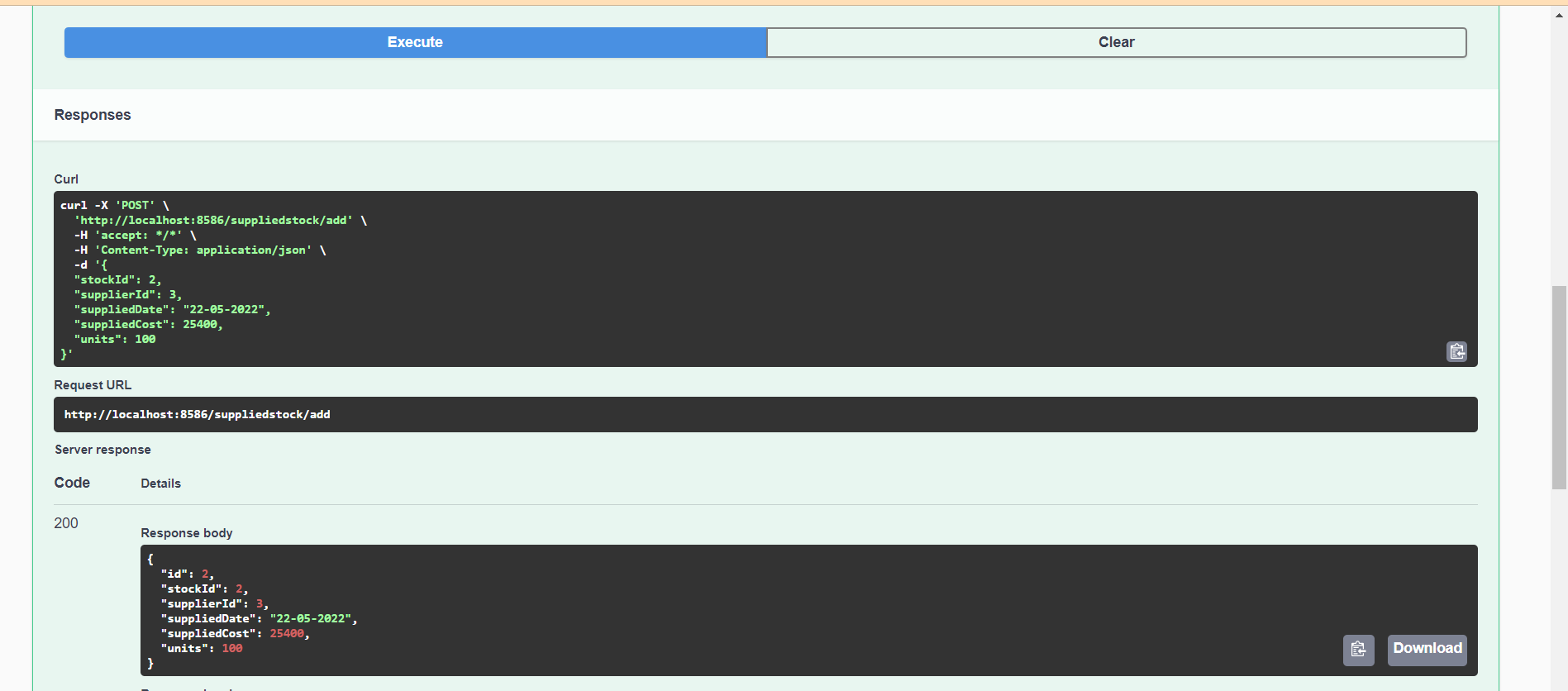


Figure 6.4

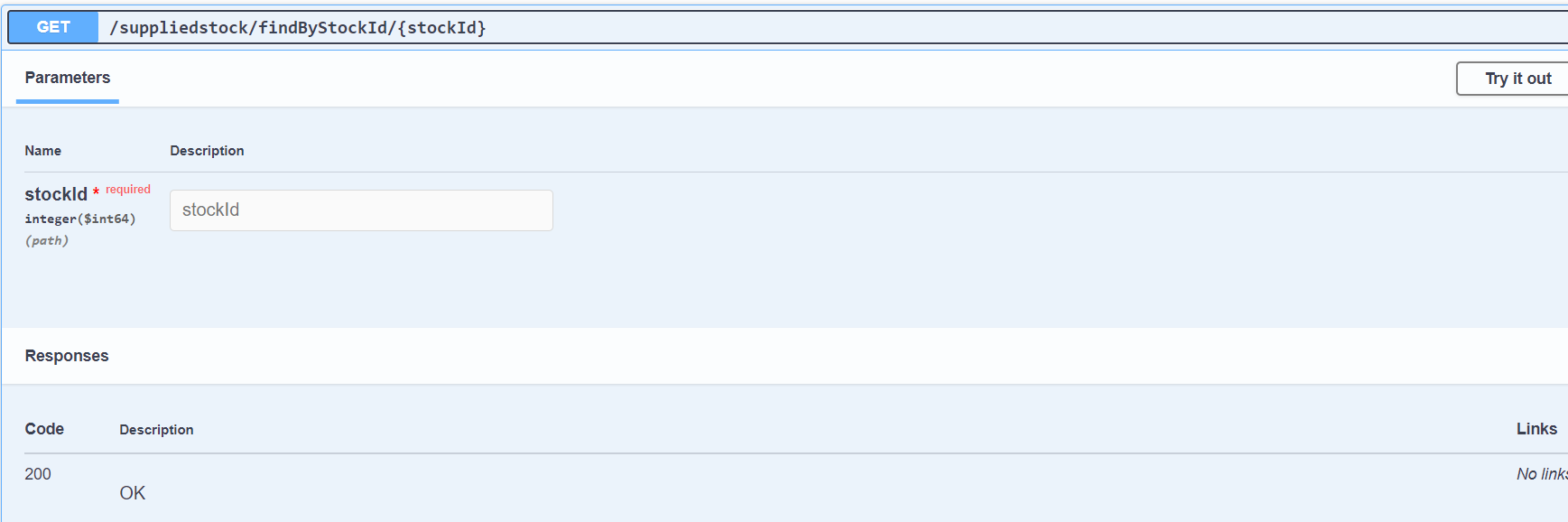


Figure 6.5

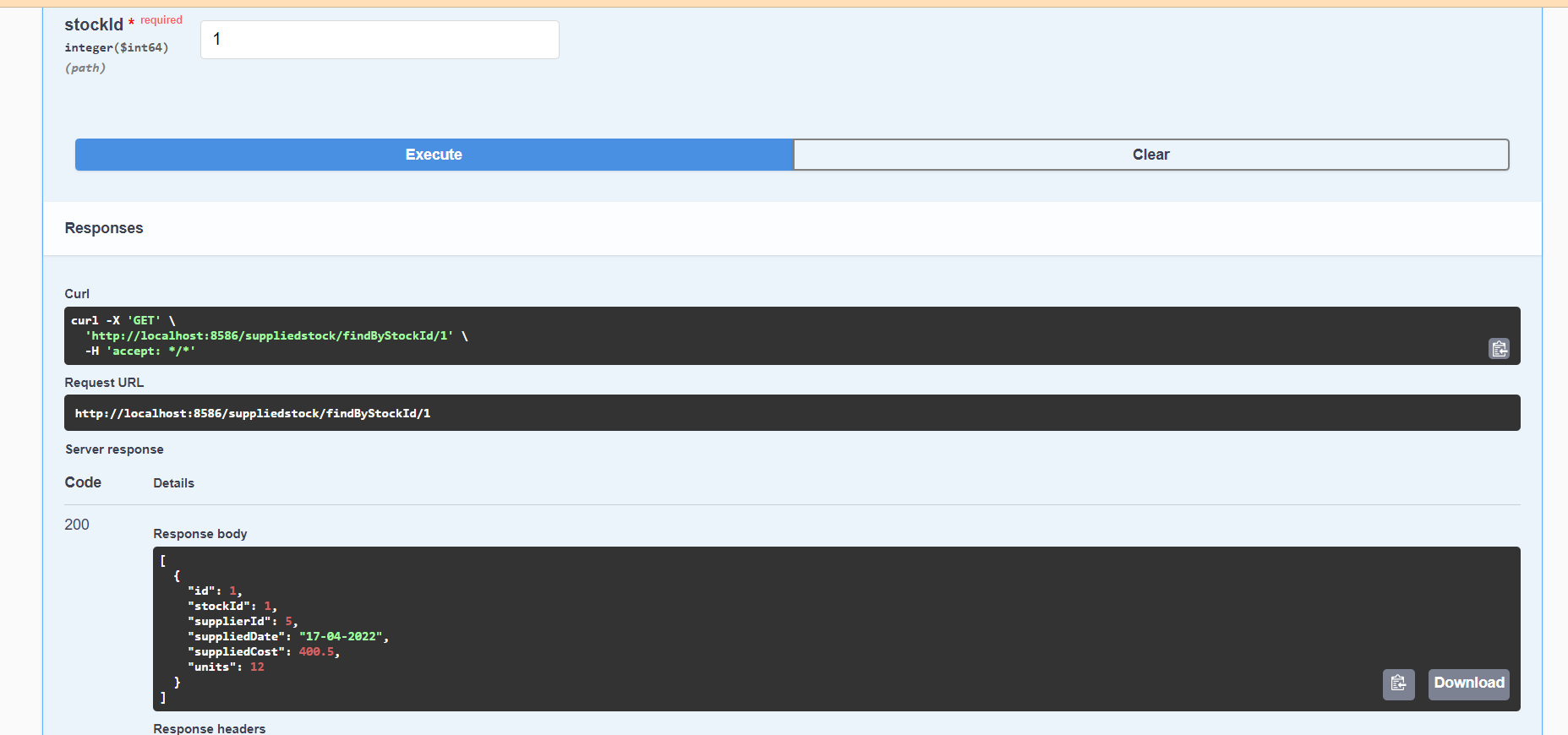


Figure 6.6

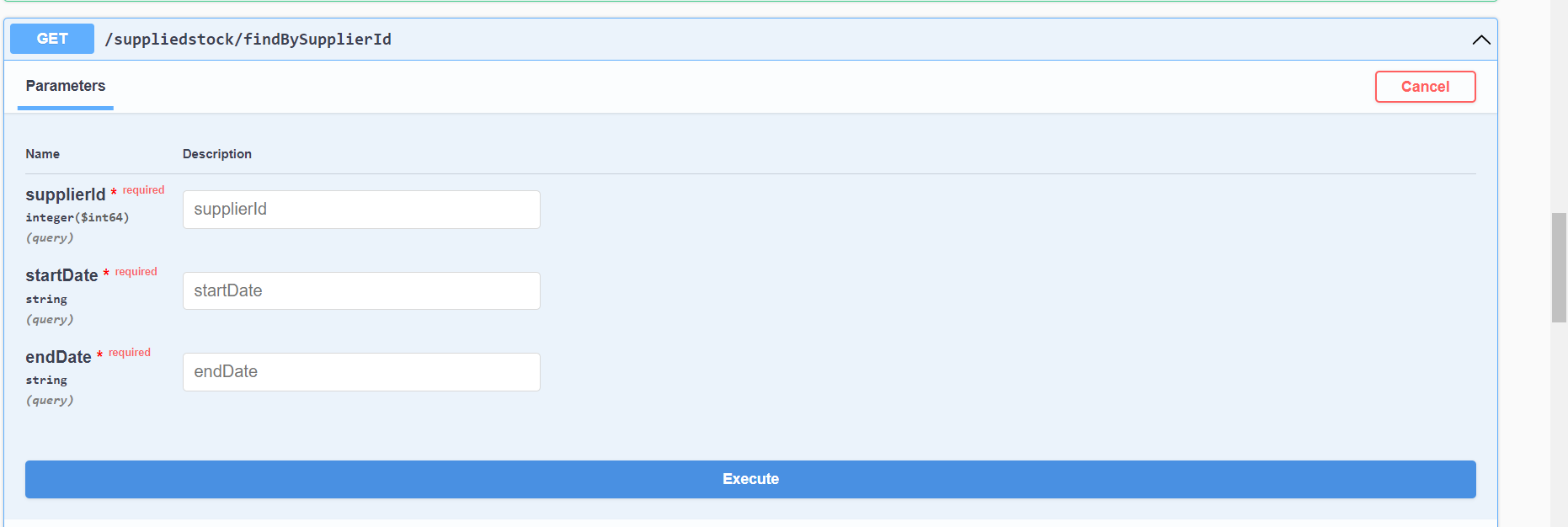


Figure 6.7

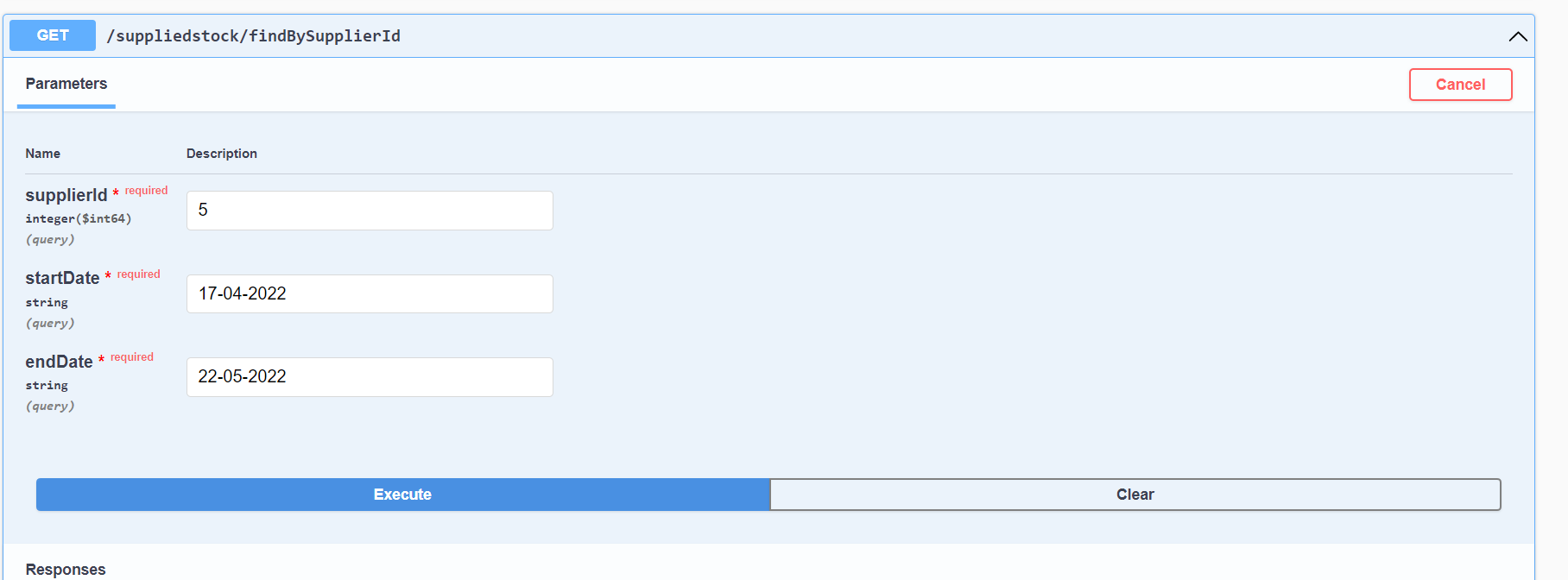


Figure 6.8

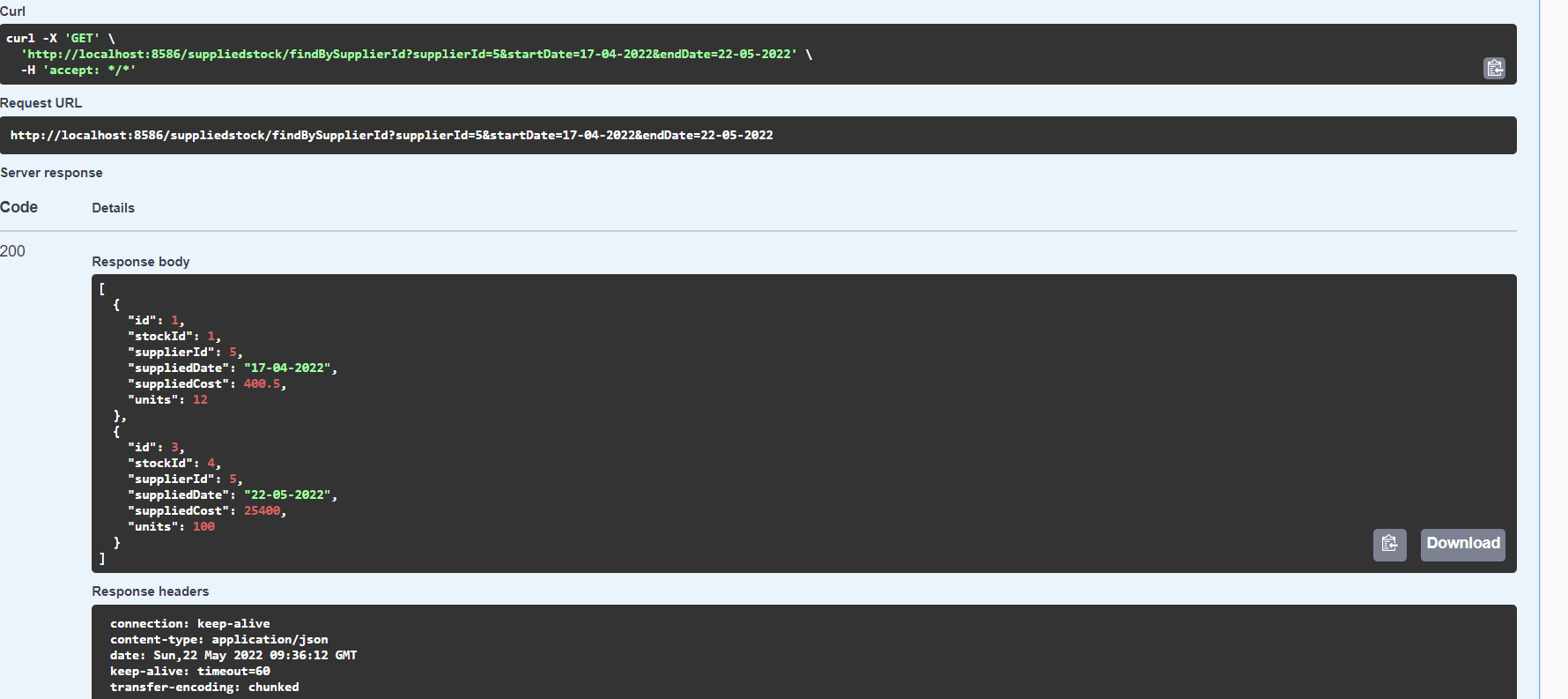


Figure 6.9

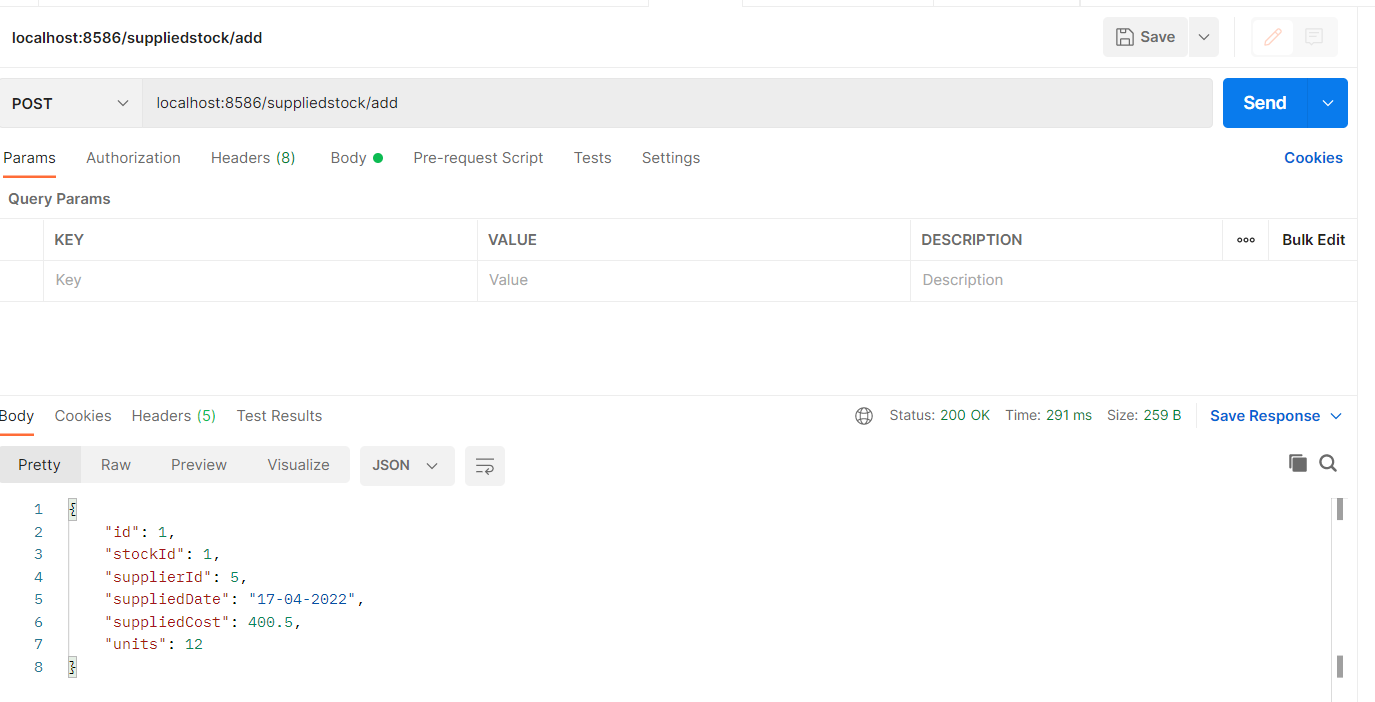


Figure 6.10

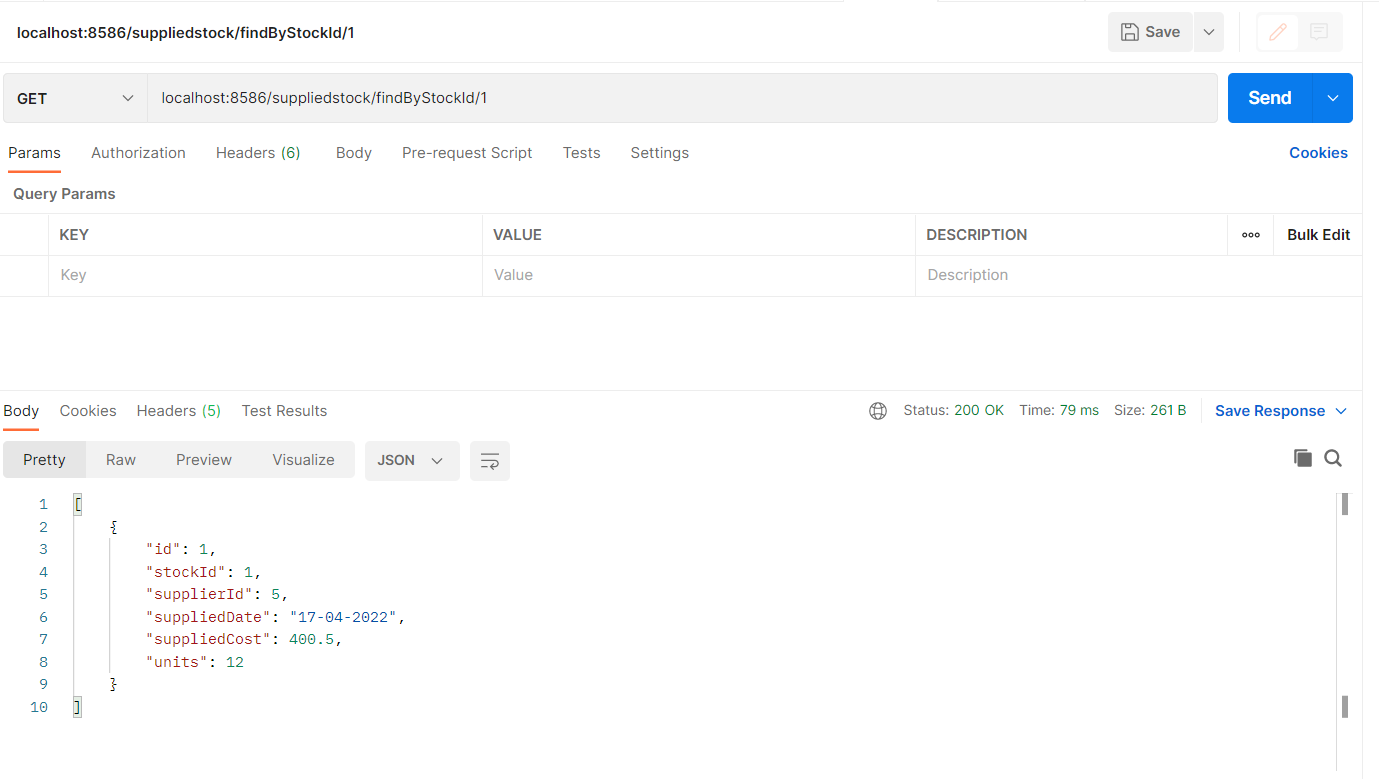


Figure 6.11

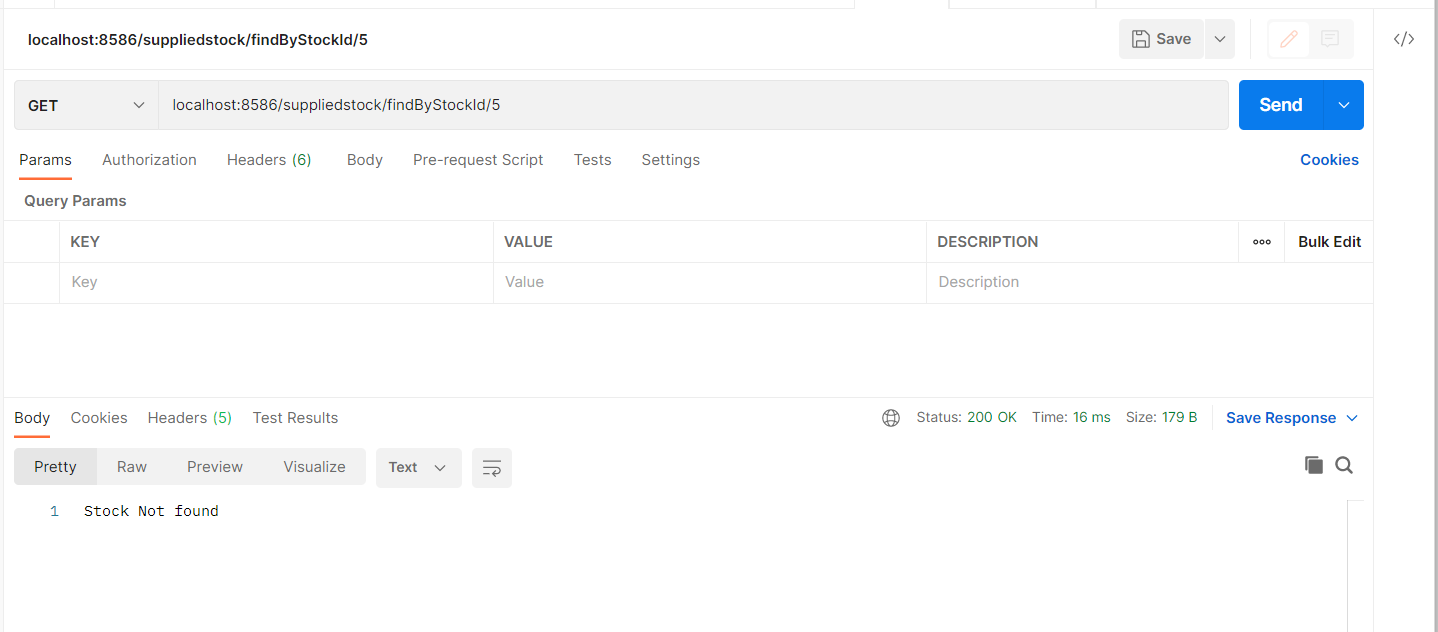


Figure 6.12

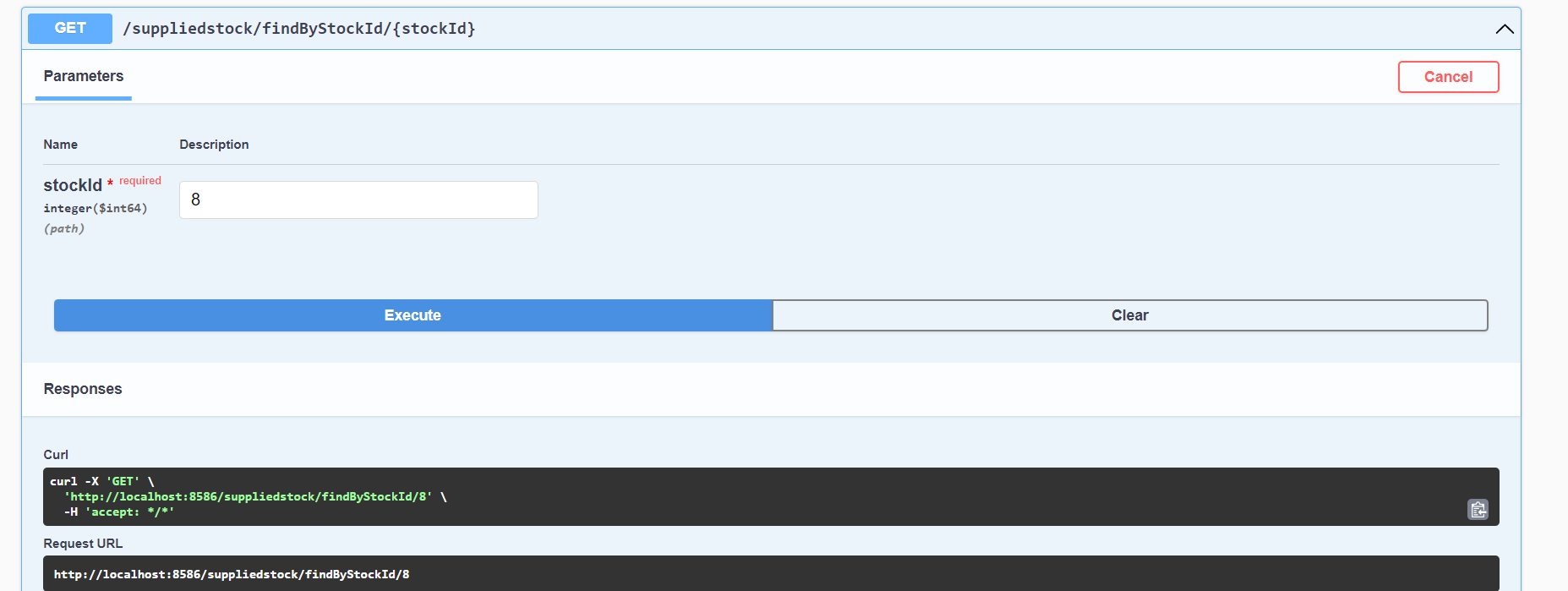


Figure 6.13



Figure 6.14

**CHAPTER 7**

**TESTING**

****

Figure 7.1

****

Figure 7.2

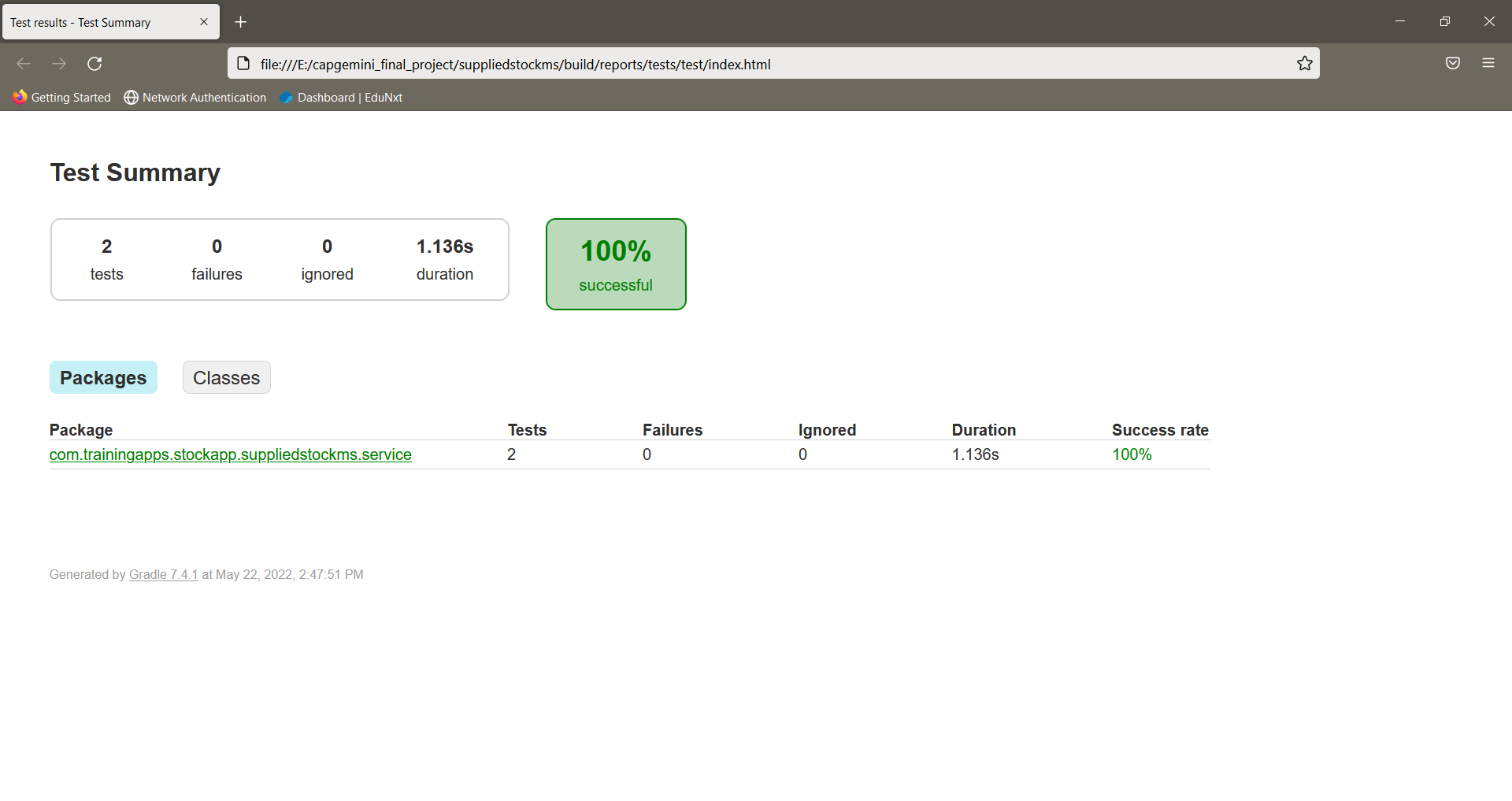
****

Figure 7.3

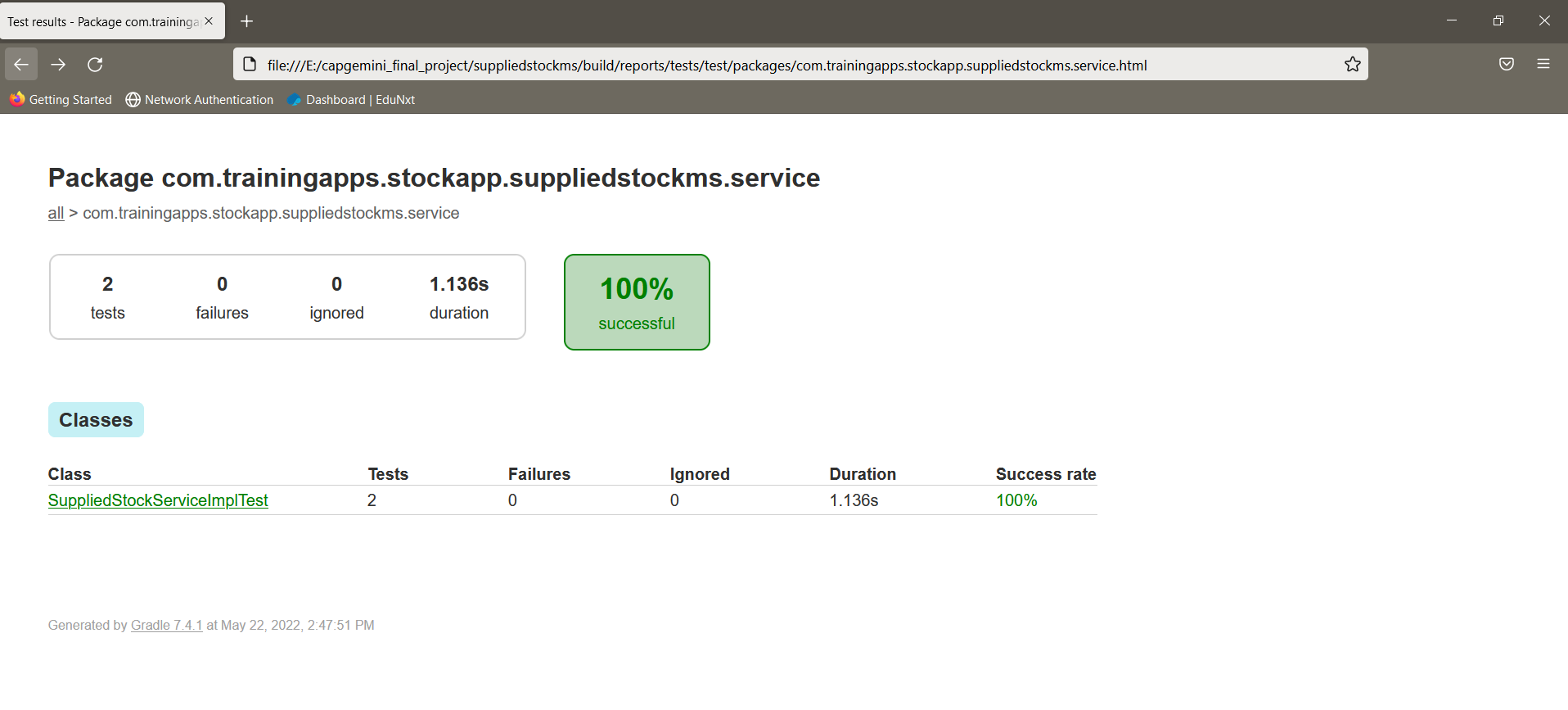
****

Figure 7.4

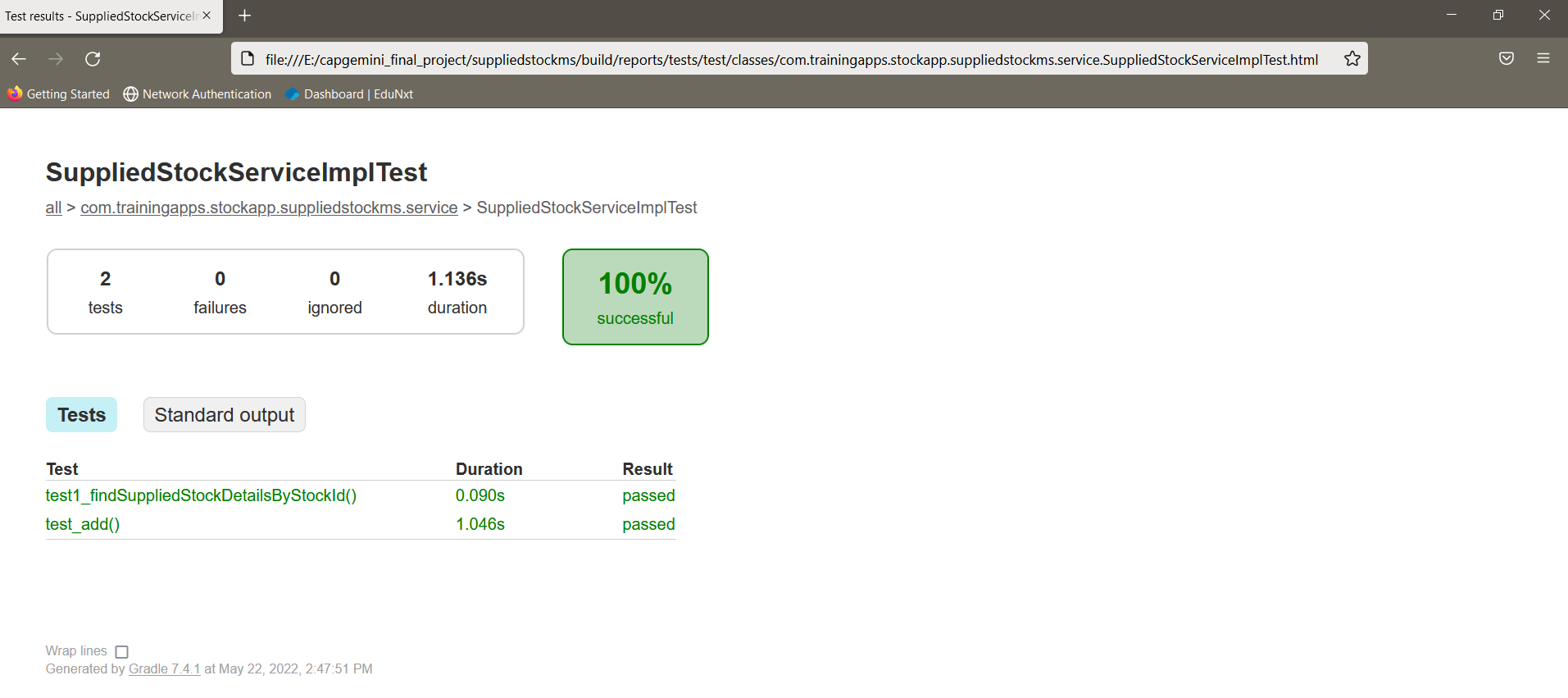
****

Figure 7.5

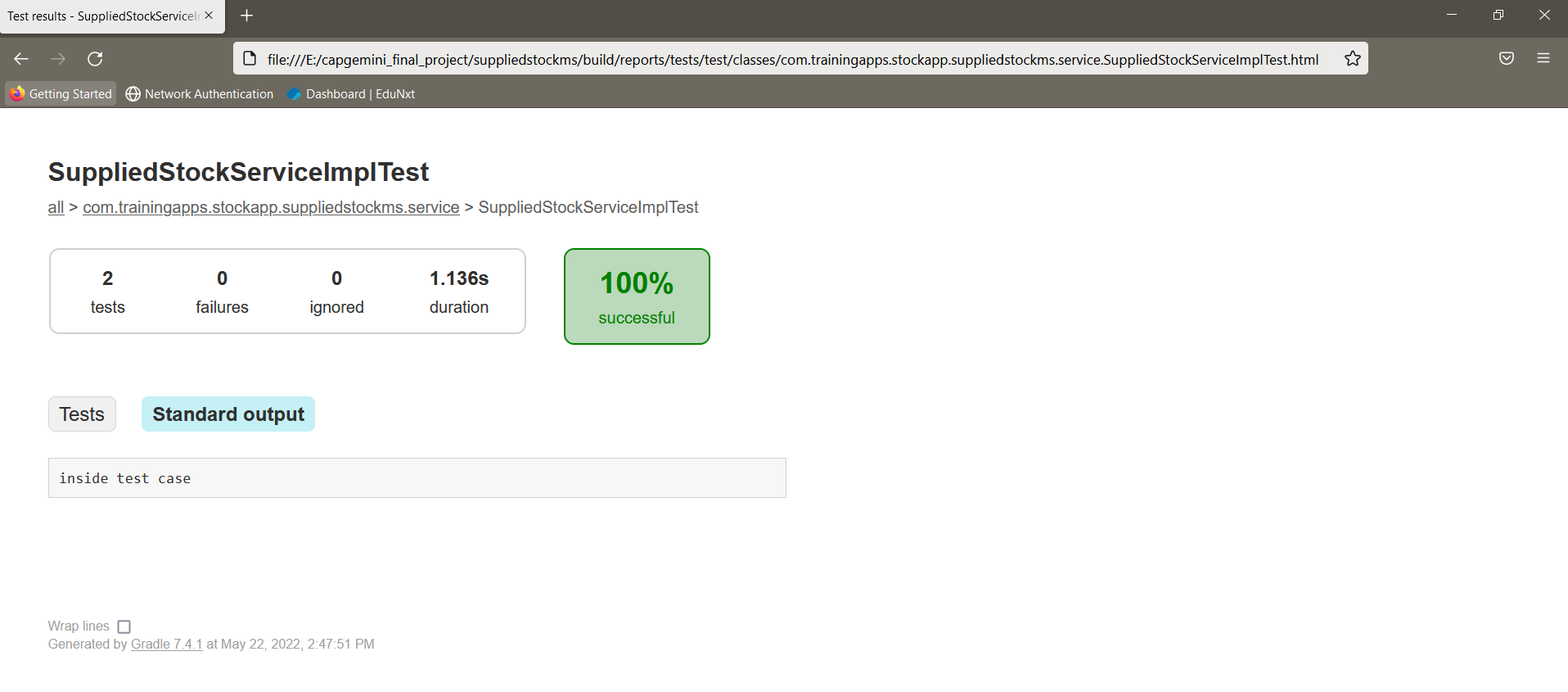
****

Figure 7.6

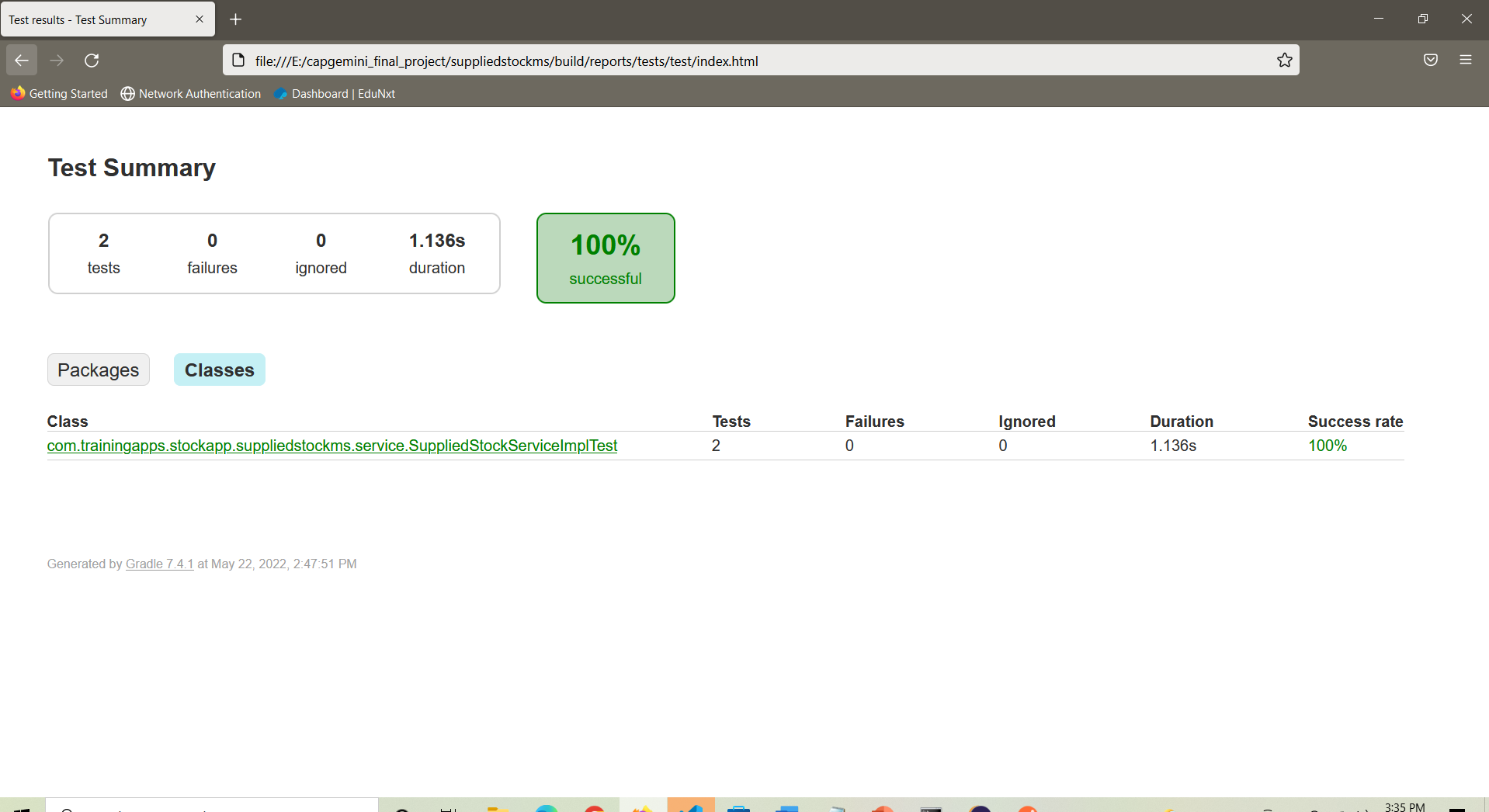
****

Figure 7.7

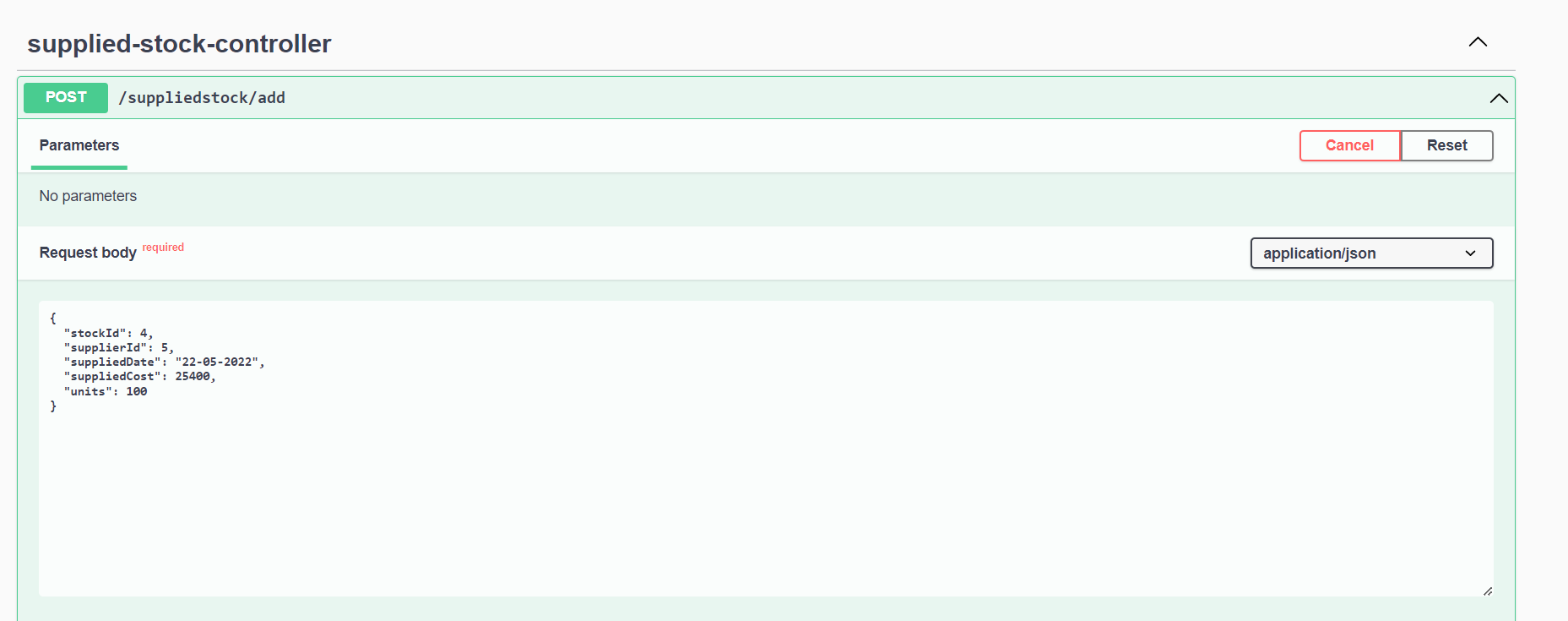
****

Figure 7.8

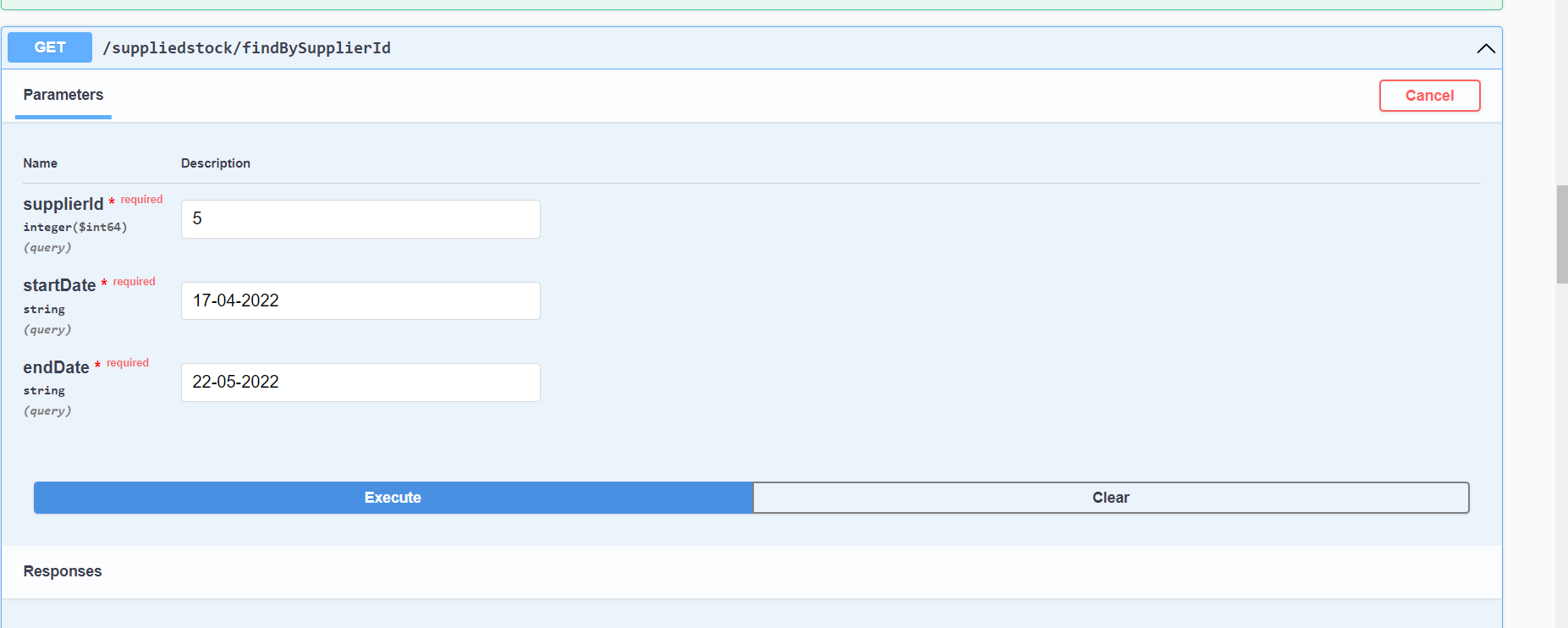
****

Figure 7.9

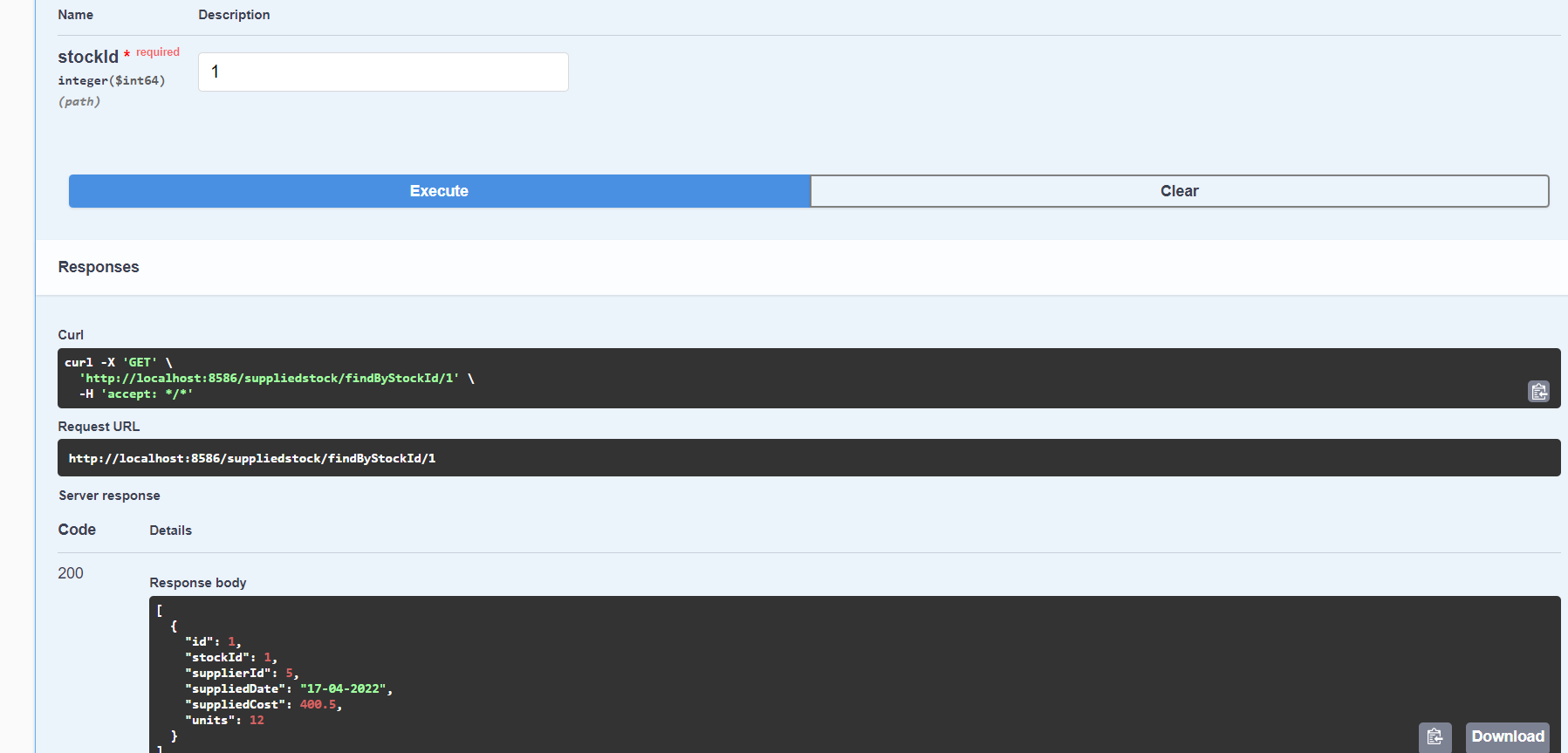
****

Figure 7.10

**CHAPTER 8**

**LIMITATION**

**8.1 Limitation**

An API is a set of definitions and protocols for building and integrating application software. It’s sometimes referred to as a contract between an information provider and an information user—establishing the content required from the consumer (the call) and the content required by the producer (the response).

In this REST API currently we’re performing only CRUD operations on these microservices i.e. this microservice will provide users to perform GET, PUT, POST, DELETE operations on the database through the defined controllers. Which means a user can use this API for getting all the details of the existing products, or can add a new product and details related to the product, or update the details of the existing product, or can delete an existing product. Beside these we can also perform various operations using microservices.

Other than this some service which can be added on this API are PATCH method for making partial changes to an existing resource, HEAD method for a response identical to that of a GET request, but without the response body, LINK method to add meta information (Object head information) to an object, without touching the object’s content, and many more methods.

**8.2 Future Scope**

**REST** overcomes many of the disadvantages of **SOAP**, such as the need for clients to know the operation semantics as a pre-requisite for its use, or the use of different ports for different types of notifications. In addition, **REST** can handle many resources, while **SOAP** needs many operations to accomplish that.

* It is usually simple to build and adapt.
* Low use of resources.
* Process instances are created explicitly.
* With the initial URI, the client does not require routing information.
* Clients can have a generic ‘listener’ interface for notifications.

While **SOAP** focuses on the design of distributed applications, **REST** does so with scalability and large-scale performance for distributed hypermedia systems.

* **Scalability**. This protocol stands out due to its scalability. Thanks to the separation between client and server, a product may be scaled by a development team without much difficulty.
* **Flexibility and portability.** With the indispensable requirement for data from one of the requests to be properly sent, it is possible to perform a migration from one server to another or carry out changes on the database at any time. Front and back can therefore be hosted on different servers, which is a significant management advantage.
* **Independence.** With the separation between client and server, the protocol makes it easy for developments across a project to take place independently. In addition, the RESTAPI adapts at all times to the working syntax and platform. This offers the opportunity to use multiple environments while developing.

**8.3 Future Enhancement**

Primarily, this API provides users to perform CRUD operations or microservices on the database. So, in future other operations or microservices can be added on. Some of the enhancements which can be applied are as follows:

PATCH method for making partial changes to an existing resource

HEAD method for a response identical to that of a GET request, but without the response body

LINK method to add meta information (Object head information) to an object, without touching the object’s content

UNLINK method to decouple the item, to remove a link, to delink

PURGE, COPY, HEAD, LOCK, UNLOCK, PROPFIND, etc microservices can be implemented in the existing REST API, which will eventually enhance the performance of the API and become more useful for the users.

**CHAPTER 9**

**CONCLUSION**

Our main task is to develop this software to make the easy maintain of records. Create the docker image and use that on Kubernetes to assemble of the modules integrate in containers using Kubernetes microservices . And later deploy that on Google cloud platform.

**CHAPTER 10**

**REFERENCES**

JAVA Programming by Rajiv Chopra - <https://elib4u.ipublishcentral.com/product/java-programming-50073021>

# Introduction to Software Design with Java by Martin P. Robillard - <https://link.springer.com/book/10.1007/978-3-030-24094-3>

# Fundamentals of Java Programming by Mitsunori Ogihara - <https://link.springer.com/book/10.1007/978-3-319-89491-1>

Java in Two Semesters by Quentin Charatan, Aaron Kans - <https://link.springer.com/book/10.1007/978-3-319-99420-8>

# Learning Java with Games by Chong-wei Xu - <https://link.springer.com/book/10.1007/978-3-319-72886-5>

Computing and Software Science by Bernhard Steffen,  Gerhard Woeginger - <https://link.springer.com/book/10.1007/978-3-319-91908-9>

# Concise Guide to Software Testing by Gerard O'Regan - <https://link.springer.com/book/10.1007/978-3-030-28494-7>

Dependable Software Engineering. Theories, Tools, and Applications by Nan Guan, Joost-Pieter Katoen,  Jun Sun - <https://link.springer.com/book/10.1007/978-3-030-35540-1>

Human-Centered Software Engineering by Cristian Bogdan, Kati Kuusinen, Marta Kristín Lárusdóttir, Philippe Palanque, Marco Winckler - <https://link.springer.com/book/10.1007/978-3-030-05909-5>

Information and Software Technologies by Robertas Damaševičius, Giedrė Vasiljevienė - <https://link.springer.com/book/10.1007/978-3-030-30275-7>

# Model-Driven Engineering and Software Development

By Luís Ferreira Pires, Slimane Hammoudi, Bran Selic - <https://link.springer.com/book/10.1007/978-3-319-94764-8>

New Opportunities for Software Reuse by Rafael Capilla, Barbara Gallina, Carlos Cetina - <https://link.springer.com/book/10.1007/978-3-319-90421-4>

Requirements Engineering: Foundation for Software Quality by Erik Kamsties,  Jennifer Horkoff, Fabiano Dalpiaz - <https://link.springer.com/book/10.1007/978-3-319-77243-1>

Software Analysis, Testing, and Evolution by Lei Bu, Yingfei Xiong - <https://link.springer.com/book/10.1007/978-3-030-04272-1>

Software Failure Investigation by  Jan Eloff, Madeleine Bihina Bella - https://link.springer.com/book/10.1007/978-3-319-61334-5

**RESUME** 