

**Spring Boot Application Using Three Layered Architecture  
in Java**

Project report submitted in partial fulfilment of the requirement  
for the degree of Bachelor of Technology

in

Computer Science and Engineering

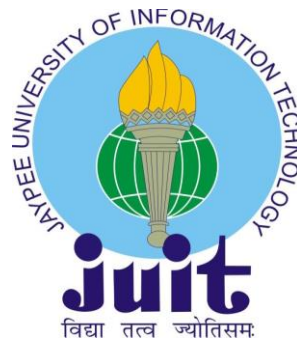
By

Sparsh Aggarwal (191332)

Under the supervision of

Dr. Shweta Pandit

to



Department of Computer Science & Engineering and  
Information Technology

**Jaypee University of Information Technology Wahnaghat,  
Solan-173234, Himachal Pradesh**

## CERTIFICATE

### Candidate's Declaration

I hereby declare that the work presented in this report entitled Spring Boot Application using three Layer Architecture in Java in partial fulfilment of the requirements for the award of the degree of Bachelor of Technology in Computer Science and Engineering submitted in the Department of Computer Science and Engineering, the Jaypee University of Information Technology Waknaghat is an authentic record of my own work carried out over a period from Feb 2023 to May 2023 under the supervision of Dr. Shweta Pandit (Assistant Professor, Department of Electronics & Communication Engineering) and Dr. Rajni Mohana (Associate Professor Department of Computer Science & Engineering and Information Technology).

The matter embodied in the report has not been submitted for the award of any other degree or diploma.



Sparsh Aggarwal, 191332

This is to certify that the above statement made by the candidate is true to the best of my knowledge.

Dr. Shweta Pandit  
Assistant Professor  
Department of Electronics & Communication Engineering

Dr Rajni Mohana  
Associate Professor  
Department of Computer Science & Engineering and Information Technology



Ujjawal Misra  
Director of Engineering  
ZopSmart  
Dated: 10-05-23

**JAYPEE UNIVERSITY OF INFORMATION TECHNOLOGY, WAKNAGHAT**

**PLAGIARISM VERIFICATION REPORT**

Date: .....

Type of Document (Tick):  PhD Thesis  M.Tech Dissertation/ Report  B.Tech Project Report  Paper

Name: \_\_\_\_\_ Department: \_\_\_\_\_ Enrolment No \_\_\_\_\_

Contact No. \_\_\_\_\_ E-mail. \_\_\_\_\_

Name of the Supervisor: \_\_\_\_\_

Title of the Thesis/Dissertation/Project Report/Paper (In Capital letters): \_\_\_\_\_

**UNDERTAKING**

I undertake that I am aware of the plagiarism related norms/ regulations, if I found guilty of any plagiarism and copyright violations in the above thesis/report even after award of degree, the University reserves the rights to withdraw/ revoke my degree/report. Kindly allow me to avail Plagiarism verification report for the document mentioned above.

**Complete Thesis/Report Pages Detail:**

- Total No. of Pages =
- Total No. of Preliminary pages =
- Total No. of pages accommodate bibliography/references =

(Signature of Student)

**FOR DEPARTMENT USE**

We have checked the thesis/report as per norms and found **Similarity Index** at .....(%). Therefore, we are forwarding the complete thesis/report for final plagiarism check. The plagiarism verification report may be handed over to the candidate.

(Signature of Guide/Supervisor)

Signature of HOD

**FOR LRC USE**

The above document was scanned for plagiarism check. The outcome of the same is reported below:

Copy Received on	Excluded	Similarity Index (%)	Generated Plagiarism Report Details (Title, Abstract & Chapters)	
	<ul style="list-style-type: none"> <li>• All Preliminary Pages</li> <li>• Bibliography/Images/Quotes</li> <li>• 14 Words String</li> </ul>		Word Counts	
<b>Report Generated on</b>			Character Counts	
		<b>Submission ID</b>	Total Pages Scanned	
			File Size	

Checked by  
Name & Signature

Librarian

.....

**Please send your complete thesis/report in (PDF) with Title Page, Abstract and Chapters in (Word File) through the supervisor at [plagcheck.juit@gmail.com](mailto:plagcheck.juit@gmail.com)**

## **ACKNOWLEDGEMENT**

Firstly, I express my heartiest thanks and gratefulness to almighty God for his divine blessing makes us possible to complete the project work successfully. I am really grateful and wish my profound my indebtedness to Supervisor **Dr. Shweta Pandit, Assistant Professor (SG)**, Department of ECE Jaypee University of Information Technology, Waknaghat. Machine Learning & keen interest of my supervisor in the field of “Machine Learning” to carry out this project. Her endless patience, scholarly guidance, continual encouragement, constant and energetic supervision, constructive criticism, valuable advice, reading many inferior drafts and correcting them at all stage have made it possible to complete this project.

I would like to express my heartiest gratitude to **Dr. Shweta Pandit**, Department of ECE and **Dr. Rajni Mohana**, Department of CSE for their kind help to finish my project.

I would also generously welcome each one of those individuals who have helped me straight forwardly or in a roundabout way in making this project a win. In this unique situation, I might want to thank the various staff individuals, both educating and non-instructing, which have developed their convenient help and facilitated my undertaking.

Finally, I must acknowledge with due respect the constant support and patients of my parents.

**Sparsh Aggarwal 191332.**

## **TABLE OF CONTENT**

<b>Title</b>	<b>Page No.</b>
<b>Certificate</b>	i
<b>Plagiarism Certificate</b>	ii
<b>Acknowledgement</b>	iii
<b>Table of Content</b>	iv
<b>List of Abbreviations</b>	v
<b>List of Figures</b>	vi
<b>Abstract</b>	vii
<b>Chapter-1 (Introduction)</b>	1-7
<b>Chapter-2 (Literature Survey)</b>	8-9
<b>Chapter-3 (System Development)</b>	10-28
<b>Chapter-4 (Experiments &amp; Result Analysis)</b>	29-47
<b>Chapter-5 (Conclusion)</b>	48-50
<b>References</b>	51-51

## **List of Abbreviations**

1. DAO: Data Access object
2. DTO: Data transfer object
3. REST: Representational State Transfer
4. API: Application Program Interface
5. UI: User Interface

## List of Figures

1. ZopSmart logo
2. Three-layer architecture in Spring boot application.
3. Categories table schema
4. Products table schema
5. Swagger UI for spring application
6. Posting a category
7. Retrieving a category
8. Deleting a category
9. Updating a category
10. Error in retrieving a category
11. Error in posting a category
12. Error in posting a existing category
13. Error in updating category
14. Posting a product
15. Retrieving a product
16. Retrieving a product of specific category
17. Updating a product
18. Deleting a product
19. Error in retrieving a particular product
20. Error in retrieving a product
21. Error in posting a product
22. Error in posting an existing product
23. Error in posting a product with negative price values
24. Performance analysis of the application

## ABSTRACT

A straightforward and scalable method for managing categories and items is offered by the Spring-Boot application. The programme offers three major APIs: one for retrieving all categories, one for retrieving all goods, and one for retrieving all items in a certain category. A Hello-World API is used to check whether the application is running and accessible at the beginning of the programme. First, stubs for the three major APIs are added before the programme is built. The functionality of these stubs is then evaluated piecemeal to make sure it matches expectations. After that, the application receives services, DAOs, and a database. The application's business logic is implemented by the services, and database communication is handled by the DAOs. For the controllers, services, and DAOs, test cases are created to validate the application's quality. These test cases are made to identify any potential flaws or problems before they have a chance to slow down the application's operation. To ensure that any problems that may arise can be quickly found and fixed, appropriate logging is also implemented with appropriate log levels. All of the classes and methods in the programme have the necessary documentation as well. Other developers should find it simple to comprehend the application's functionality and rapidly incorporate it into their own projects thanks to this documentation. For handling categories and items, the Spring-Boot application offers a strong and adaptable solution. It is the best option for any organisation that wants to manage a lot of items and categories because it is simple to use, easy to scale, and simple to maintain.



# CHAPTER 1

## INTRODUCTION

### 1.1 About Organization

Businesses may create an online presence using the wide range of tools provided by ZopSmart, an innovative technology supplier for the retail sector. With the aid of ZopSmart's tools, you can swiftly and effectively accomplish your goals, whether you're an e-commerce company aiming to improve your online business or a traditional brick-and-mortar store hoping to enter the digital market.

E-commerce has been increasingly popular in recent years, and the online market presents a wealth of business prospects. In order to build a solid basis for growth, ZopSmart can offer you the required technologies if you're trying to start your own internet business. With the help of ZopSmart's goods, numerous business owners can launch their own online retail outlets and benefit from the expansion possibilities in the online market.



*Figure 1: ZopSmart logo*

Whatever strategy a customer chooses to use, ZopSmart places a high priority on increasing productivity and efficiency for them in order to increase their online retail sales. clients can accomplish this by spreading the benefits among numerous clients, which allows them to get the best results with the least amount of work and expense. ZopSmart guarantees the entire satisfaction of all parties concerned, including clients and customers, by utilising cutting-edge concepts and technology.

## 1.2 Introduction

Developers may easily and effectively build highly stable and scalable web apps using the renowned Java-based framework Spring Boot. Because it makes it possible to quickly create apps with little setup, it has grown in favour among developers. The ability of Spring Boot to aid in the creation of APIs with a clear and user-friendly interface is one of its key features.

This project report explores the use of a three-layer design to create a Spring Boot API. The controller layer, service layer, and dao layer are the three separate layers that make up the three-layer architectural design pattern that divides the logical levels. This division of duties makes it possible to design apps in a more modular and scalable way. While ensuring the greatest levels of dependability and maintainability, this design pattern makes it easier and more flexible to create applications.

User input and output are managed by the controller layer. In order to manage requests and responses for a Spring Boot API, it plays a key function. Business logic for the application is located in the service layer, also called the business layer. Processing data and applying business rules are the responsibilities of this layer. In order to retrieve or save data, the dao layer—also referred to as the persistence layer—must be in contact with the database.

In a Spring Boot API, a three-layer architecture is used to create a high level of abstraction and modularity. By doing this, the scalability and maintainability of the application are ensured as it develops and increases over time. Debugging and testing are also made simpler.

The Spring Boot API will be used to develop a number of APIs for an e-commerce application. These will have APIs for locating all subcategories, all products, and all items within a specific category. Utilising a three-layer design, these APIs will be implemented.

RESTful endpoints for each API are found in the controller layer. Incoming requests must be processed via these endpoints, and the proper replies must be given. Class implementations for each API's business logic will be found in the service layer. With the help of these classes, data processing, business rules, and data layer functions will all be handled. Classes from the repository that engage in CRUD operations and database communication will be found in the dao layer.

The three-layer architecture is a useful design pattern for creating scalable and maintainable Spring Boot APIs, in conclusion. It is possible to achieve a high level of abstraction and modularity by breaking the programme up into three logical layers. This makes testing and debugging simpler. It is simple to develop APIs that fetch all categories, all products, and all items under a certain category using this design. Finally, a cloud server will host the RESTful API.

### **1.3 Problem Statement**

The API is designed to meet the need for a dependable and efficient way of storing and retrieving data linked to categories and items. A system that can efficiently and effectively manage product-related data, such as price, names, and classifications, is essential for firms that handle a significant number of products. Additionally, these companies require a quick and simple means to get this data for a variety of tasks like inventory management, sales analysis, and reporting.

Making an API that can successfully and effectively meet these needs is the difficult part. The API should enable endpoints for product creation and updating, give access to all product and category data, and filter it according to particular standards like category. Additionally, it should be built to handle numerous requests concurrently, guarantee data integrity, and offer great performance and scalability.

A three-layer architecture with separate controllers, services, and dao layers will be employed to create the API in order to address these issues. Requests and

replies to and from the API will be managed by the controller layer, while business logic and communication between the presentation and data levels will be managed by the service layer. All data-related duties, such as archiving and accessing details about items and categories, will fall within the purview of the data layer.

A popular Java-based web application development platform called Spring Boot will be used to generate the API. Spring Boot provides a variety of features that may be utilised to create a stable, scalable, and speedy API. SwaggerUI, an easy-to-use platform for developing interactive API documentation, will also be used in the API documentation.

The main goal is to develop an API that can satisfy the requirements of companies that deal with a variety of products and is also reliable, efficient, and capable of achieving those objectives. Excellent speed, data integrity, scalability, usability, and data security are required from the API.

## **1.4 Objectives**

By offering a concise and well-defined collection of RESTful APIs that enable quick and easy data access and modification, the main goal of this API is to promote user engagement with product and category data. The administration of product and category data is improved by the API's functionality for controlling mistakes and verifying data.

The API was created with modularity, scalability, and extensibility in mind, making it simple to incorporate into already-existing applications and systems. In order to guarantee that the code is tested, manageable, and reusable, it uses modern technologies and complies with long-standing industry standards. The API also emphasises the importance of user data security and incorporates the necessary authentication and authorisation steps.

By offering a dependable and effective API, the duty of developers who are designing applications that require product and category data is intended to be

made simpler. The API frees developers to concentrate on the core functionality of their apps by making data administration and retrieval simple. The RESTful design of the API makes it possible for it to be connected to other platforms and apps, making it a flexible and adaptable option for handling product and category data. In conclusion, the API offers a reliable and practical solution to both users and developers.

## **1.5 Methodology**

The Spring-Boot API was created through a methodical process that adhered to the listed criteria. The requirements collecting, planning, implementation, testing, and deployment stages were all parts of the development process.

Understanding the API's needs required first understanding the problem description and the project's main goal. After the requirements were established, a three-tier architecture for the API was created, consisting of a controller layer, a service layer, and a dao layer.

The API's implementation using the Spring-Boot framework came after the architecture was designed. Spring-Boot Controllers, Spring-Boot Services, and Spring-Boot Data Access Objects were used to implement the controller layer, service layer, and dao layer, respectively.

After the implementation was finished, the API underwent a rigorous testing process to make sure it complied with the specifications. Unit testing utilising frameworks like Mockito and JUnit5 was one of the testing techniques used. The API was set up using Docker containers and then deployed to an AWS EC2 server for public use when all problems had been fixed.

A functioning and scalable Spring-Boot API that met the requirements was successfully delivered using the method adopted. The adoption of a three-layer design enabled the maintenance and updating of the API and allowed for the clear separation of concerns. The usage of Docker containers and the Spring-

Boot framework, which allowed for quick development and simple deployment, also made the deployment process effective and streamlined. Therefore, the Spring-Boot API was created utilising a scientific and efficient process that resulted in a high-quality API that satisfied the requirements.

## **1.6 Organization of report:**

Chapter 1: The benefits of developing our Spring Boot application using a three-layered architecture are briefly discussed in Chapter 1 along with a brief review of our Spring Boot application. The issue statement, which forms the basis of the goals of our project, is presented together with an outline of the system's motivation and purpose. As part of the three-layered architecture, we also explore the construction process for a Spring Boot RESTful API for an e-commerce application.

Chapter 2: We undertake a literature review in Chapter 2 that compiles a number of papers and other materials that helped us build our RESTful API using the suggested three-layered architecture strategy. The review of the literature explores several challenges that were resolved, to differing degrees of success, during the project development phase utilising a variety of technologies, tools, languages, and approaches by diverse researchers. In addition, we talk about the strategy we used to build our application.

Chapter 3: The goal of Chapter 3 is to describe the step-by-step process we used to create the study setup. We started by looking through different research papers, documents, and e-commerce portals to compile a list of crucial elements that should be present in the API backend of our project. In this chapter, the tools and technology used in the process are the main topics.

Chapter 4: The findings and products of our RESTful API project are presented in Chapter 4 of the report. It seeks to show how accurate and successful our implementation is. This chapter also offers a thorough examination of several API use cases, emphasising the applications that could be made of them.

Overall, this chapter serves as a crucial end to our project, highlighting our accomplishments and our work's future directions.

Chapter 5: The report's study findings are summarised and concluded in Chapter 5, along with the project model's end product. It shows areas that can benefit from additional study and work. This chapter also presents creative work that was produced as a result of the analysis done, and it draws a conclusion from the outcomes.

## **CHAPTER-2**

### **LITERATURE SURVEY**

Two well-known frameworks for creating microservices, Spring Boot and Vert.x, are compared and contrasted in Christian Zepeda-Nez, et al.'s [1] study. In terms of memory utilisation, reaction time, and throughput, the authors assessed the frameworks' performance. Vert.x fared better than Spring Boot in all three parameters, especially memory use and reaction time, according to the results. The writers also go over each framework's benefits and drawbacks and offer suggestions for picking the best one based on the demands of a particular project.

The use of Spring Boot and ASP.NET Core for developing microservices architecture is compared in Md. Saiful Islam, et al.'s [2] study. The ease of use, performance, scalability, and maintainability of the frameworks are all evaluated by the writers. The findings indicate that while the performance and scalability properties of both frameworks are comparable, Spring Boot is more user-friendly and easier to maintain. The writers also go over each framework's benefits and drawbacks and offer suggestions for picking the best one based on the demands of a particular project.

In their study, Mohammed Alsharif et al. [3] analyse Spring Boot's function in the context of cloud-native Java programming. The advantages of Spring Boot are covered by the writers, including how simple it is to use, how modular it is, and how it supports different cloud platforms. Additionally, they look at how Spring Boot handles issues with distributed systems and containerization that arise with cloud-native programming. Spring Boot is a useful tool for cloud-native development, according to the authors, who also suggest using it to create microservices-based systems.

In the context of creating microservices, this paper [4] compares Spring Boot versus Node.js. By constructing a straightforward microservice using each and



running benchmarks, the study assesses the performance and scalability of each framework. The authors conclude that both Spring Boot and Node.js are viable for developing microservices, albeit each has advantages and disadvantages of its own. Better out-of-the-box support for microservices features, like service discovery, load balancing, and centralised configuration, is offered by Spring Boot. Contrarily, Node.js is lighter and better capable of handling more requests per second. The authors come to the conclusion that the selection of Spring Boot or Node.js for microservices development depends on the particular project requirements.

In the study paper [5], the Spring Boot framework for scalable web applications is evaluated in terms of performance. Using a benchmark application created to closely resemble a real-world online application, the authors assessed Spring Boot's performance in terms of response time, throughput, and scalability. With response times under 50 ms and throughput of more than 1000 requests per second, the study's results showed that Spring Boot performed well in terms of reaction time and throughput. The authors also discovered that Spring Boot was scalable, able to accommodate growing loads by providing more resources, such CPU and memory.

# CHAPTER-3

## SYSTEM DEVELOPMENT

### 3.1 Overview

This chapter's goal is to lay the theoretical groundwork needed to comprehend the report's content. It lists the technology and equipment used to build the project. Its goal is to give readers a thorough understanding of the underlying ideas behind spring boot RESTful APIs as well as the various tools and techniques used to create accurate applications.

### 3.2 Overall Design

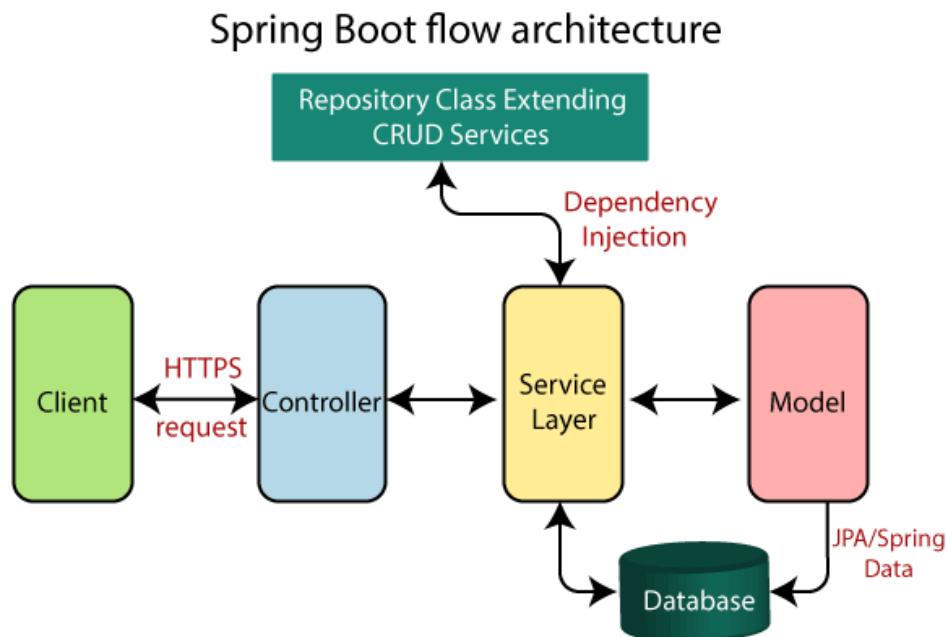


Figure 2: Three-layer architecture in Spring boot application

For creating Java-based apps, Spring Boot is a potent framework. Its primary strength is its ability to simplify application setup and configuration, relieving

developers of the responsibility of worrying about infrastructure setup. The Spring Framework, which has a variety of tools and modules for developing enterprise-level applications, is the main component of Spring Boot. Various dependencies and modules, such as web, data access, security, and others, are combined by the Spring Boot programme to produce the whole application. These modules are controlled by the configuration files for Spring Boot, which makes it simple to alter and customise the application. Spring Boot streamlines the development process overall by offering a streamlined and effective foundation for creating reliable and scalable applications.

### **3.3 Software & Environment**

The tools and technology needed to complete our project will be covered in this chapter's section. The hardware and software requirements for this project will also be covered.

#### **3.3.1 Software Requirements**

##### **Java**

Due to its dependability, security, and platform independence, Java is a widely used programming language that is favoured by developers. In the middle of the 1990s, James Gosling and his coworkers at Sun Microsystems invented Java, which has since developed into one of the most extensively used programming languages in the world. As an object-oriented programming language, Java relies on the idea of objects rather than just carrying out a series of instructions.

The fact that Java is platform independent is one of its main advantages. As a result, Java software may be run on any computer that has a Java Virtual Machine (JVM) installed, regardless of the machine's operating system. Because of this, Java has become quite popular for creating web apps and other types of software that must be able to function on a range of various systems. Security is yet another crucial aspect of Java. Java has a variety of security features to safeguard users from malicious code. When Java was created, security was a top priority. For instance, Java's "sandbox" security policy

restricts untrusted applications' access to crucial system resources like files and network connections.

Additionally well renowned for its reliability and toughness is Java. Programmes are more dependable and less prone to mistakes and crashes because to Java's sophisticated exception handling and type checking features. In addition, Java's integrated memory management system aids in avoiding memory leaks and other frequent programming problems that might lead to unsuccessful programmes.

There are many tools and resources available to help novice Java programmers, and Java has a huge and active development community. An environment that is more user-friendly for developers is provided by integrated development environments (IDEs), such as Eclipse and NetBeans. A set of tools for developing and testing Java programmes is called the Java Development Kit (JDK).

Java is utilised in a wide variety of applications, including games and mobile apps as well as enterprise software. Java is used by many large firms, including Google and Amazon, for their backend systems and other crucial infrastructure. Java is a popular choice for game development due to its performance and cross-platform interoperability. It is also frequently used for creating Android apps.

Although there are some significant changes, Java and C++ have a comparable syntax. Since variables in Java must be declared with a specific data type before they can be used, it is more highly typed than C++. Additionally, Java uses automatic garbage collection to manage memory, thus unlike C++, developers do not need manual memory allocation and deallocation.

Java has a large library of common classes and APIs, which is one of its most significant features. With the help of these libraries, programmers may easily create sophisticated apps with a wide range of features. Some of the most popular Java libraries are the Java Standard Library, which offers classes for

working with data structures, networking, and input/output; the Java Collections Framework, which offers a set of classes for working with collections of objects; and the JavaFX library, which offers a set of classes for developing advanced graphical user interfaces.

In summary, Java is a strong and adaptable programming language that is used by programmers all over the world for a variety of applications. For creating web applications and other software that must function on a range of various platforms, it is the best option due to its platform independence, security, and dependability. Java is likely to stay a popular programming language for many years to come thanks to its robust developer community and large library of standard classes and APIs.

### **Backend in Java**

The server-side logic that drives a web application or software system is created and maintained as part of backend development. Data storage, request processing, security administration, and response delivery are all handled by the backend. Because of its efficiency, scalability, and large library, Java is the perfect language for backend development.

Java is a powerful language that can be used to develop both web-based and non-web-based backend systems. Building effective backend systems is made simpler by Java's extensive array of libraries, tools, and frameworks. The most well-liked libraries and frameworks for Java backend development are listed below.

### **Spring Framework**

Popular Java framework for creating backend systems and online applications is called the Spring Framework. For developing Java-based enterprise applications, it offers a thorough programming and configuration model. Building scalable and effective backend systems is made simpler by Spring's

inclusion of technologies like inversion of control, aspect-oriented programming, and declarative transaction management.

### **Hibernate ORM**

A Java object-relational mapping (ORM) tool is called Hibernate. It makes it easier to convert Java objects into relational database tables. It is simpler to interact with databases in Java thanks to Hibernate's features like caching, lazy loading, and automated key generation.

### **Apache Struts**

Java web application developers can use Apache Struts, a well-liked open-source framework. It offers a model-view-controller (MVC) architecture for developing online applications. It is simpler to create reliable and scalable online applications with Struts' features, which include input handling, form validation, and error handling.

### **Apache Maven**

For Java applications, Apache Maven is a well-liked build automation tool. It offers a reliable and user-friendly build system that makes managing dependencies, building, and packaging Java programmes simpler. Building and managing Java projects is made simpler by Maven's features, which include build profiles, pluggable build processes, and dependency management.

### **Java Server Pages (JSP)**

JSP, or Java Server Pages, is a technology that allows programmers to create dynamic web pages in Java. The characteristics of JSP, such as scripting elements, custom tags, and expression language, make it easier to create dynamic and interactive web pages. Creating web-based backend systems in Java typically involves the usage of JSP.

## **Java Servlets**

Java developers can create server-side web applications by using the Java Servlet technology. It is simpler to develop scalable and effective online applications thanks to the functionality that servlets offer, like request handling, session management, and security management. Java backend systems with a web component are frequently built using servlets.

For building backend systems, many programmers turn to Java because of its strength and versatility. A variety of libraries, tools, and frameworks are available in Java that make it simpler to create scalable and effective backend systems. Because of its efficiency, scalability, and large library, Java is the perfect language for backend development. Java offers the functionality and tools necessary to develop reliable and scalable systems, whether you're constructing a web-based or non-web-based backend system.

## **Git Hub**

Version control and team communication are the main uses of the web-based platform GitHub in software development projects. It is a platform that is based on the Git distributed version control system and is well-liked for managing code repositories. Due to its extensive feature set and array of tools that promote collaboration and project management in the software development industry, GitHub is a platform that is heavily utilised by developers all over the world.

The ability to trace changes to code files over time is one of the major advantages of utilising GitHub. Developers may now quickly examine the code modifications that have been made, along with the developers' names and dates of implementation. When several developers are simultaneously working on the same codebase in large teams, this capability is especially helpful.

Utilising GitHub also has the primary advantage of promoting team member collaboration. Developers have the ability to divide the codebase into branches, which can then be worked on independently and merged back into the main

codebase when finished. This makes it possible for developers to work on various features or problem fixes concurrently without interfering with one another's efforts.

Additionally, a number of services are available on GitHub to assist with managing issues and bug reports. Issues can be created by users, assigned to team members, and tracked all the way to completion. It can be challenging to keep track of all the problems that develop in larger projects, so this is very helpful.

The flexibility of GitHub to interface with so many different other programmes and services is one of its most potent advantages. Along with a variety of different third-party services for jobs like code review, testing, and monitoring, this includes well-known technologies like Jenkins, Travis CI, and CircleCI for continuous integration and deployment.

Additionally, GitHub has a sizable developer community that supports open-source projects by contributing code, reporting bugs, and offering general assistance. Developers can benefit greatly from this community's abundance of knowledge and expertise on a variety of issues relating to software development.

Along with its main functions, GitHub also offers a number of extra tools and services. Project management tools, such as milestones and boards, which may be used to measure progress and organise tasks, are among them. Wikis enable developers to share expertise and describe their work.

GitHub is a very effective platform for organising software development projects overall. Any modern development team must have this tool because of its capacity to track changes, promote collaboration, and integrate with a wide range of other tools and services. It is a useful tool for developers of all levels



of experience due to its popularity and vibrant community of users and contributors.

### **Spring Boot**

The goal of the Spring Boot framework is to streamline the creation and deployment of web applications. It is the ideal choice for developers who want to work with Java since it provides a number of features and tools that are expressly made to make the process of developing, testing, and delivering applications simpler.

One of the major advantages of Spring Boot is that it has a pre-configured application context, saving developers time when setting up and configuring their applications. The application's business logic can be built instead, which is where they should concentrate.

The ability to deal with a variety of databases, including SQL, NoSQL, and even in-memory databases, is another crucial aspect of Spring Boot. As a result, it is an exceptionally versatile framework that can be applied to a variety of application kinds.

Spring Boot's capacity to assist developers in creating microservices is among its key advantages. A larger application is made up of microservices, which are discrete, tiny services. Compared to monolithic apps, which can overgrow and become challenging to manage over time, they are considerably easier to maintain and update because they are autonomous and small.

The fact that Spring Boot offers a variety of testing and debugging tools is another important advantage of the framework. As a result, bugs can be rapidly found and fixed by developers, making it possible to increase the dependability and performance of applications.

Installing the framework and configuring their development environment are prerequisites for beginning with Spring Boot. After that, they can start a new project and begin developing their application utilising a variety of tools and technologies.

One of the most well-liked tools for using Spring Boot is Spring Tool Suite, a potent IDE that offers a variety of functionality for creating and testing applications. It comes with a variety of templates and wizards that make it simple to start new projects.

Gradle is a build automation tool that is well-liked for use with Spring Boot and makes managing dependencies and creating applications simpler. It offers a variety of capabilities for testing and deploying apps, and is especially helpful for building large, sophisticated applications.

In addition to these tools, Spring Boot is compatible with a wide variety of libraries and frameworks. For instance, programmers can use Hibernate to interact with databases or the Spring Framework to create web applications.

All things considered, Spring Boot is a very potent framework that offers a variety of capabilities and tools for creating and delivering web applications. It offers a number of tools for testing and debugging applications and is particularly well suited for constructing microservices. Spring Boot is undoubtedly a framework you should think about using if you want to work with Java.

### **PostgreSQL**

An effective relational database management system is PostgreSQL, also referred to as Postgres. In 1996, the PostgreSQL Global Development Group initially made it available. It has a high level of scalability, dependability, and extensibility. Numerous businesses, from small start-ups to huge corporations, use PostgreSQL.

Support for sophisticated SQL and object-relational database features is one of PostgreSQL's distinguishing characteristics. It supports user-defined data types in addition to a wide range of built-in data types like arrays, hstore, and JSON. It is the best option for complex applications with various data requirements because of this.

The comprehensive support for concurrency and transactions that PostgreSQL offers is still another advantage. Multiple users can access the database simultaneously without encountering conflicts thanks to the system's usage of multi-version concurrency control (MVCC). This is accomplished by using transaction isolation levels, which can be adjusted to regulate the level of concurrency.

A variety of indexing and query optimisation strategies, including B-tree, hash, and GiST (Generalised Search Tree) indexing, are also supported by PostgreSQL. High-speed data retrieval is now possible, even for huge datasets.

Additionally, PostgreSQL has a solid track record of security. It offers capabilities like SSL encryption, row-level security, and database roles, enabling fine-grained control over data access. Additionally, it has a thriving community of developers who frequently publish updates and patches to fix security flaws.

In addition to being highly customisable, PostgreSQL has a vast array of extensions that can enhance the database's usefulness. From full-text search to geographic data processing, these enhancements address a wide range of use cases.

It is simple to connect to a PostgreSQL database when utilising one of the many well-known libraries that are available for Java backend applications that use PostgreSQL. One of the most well-liked ones is the Java Database Connectivity

(JDBC) API, which offers a common interface for using Java to access relational databases. Other object-relational mapping (ORM) frameworks, including Hibernate and MyBatis, offer higher-level abstractions for working with databases.

Strong PostgreSQL support is offered by Spring Boot, a well-liked Java platform for creating online applications. The configuration of a PostgreSQL data source is simple with Spring Boot, which also offers a variety of database-related features like support for JPA and transaction management.

In conclusion, PostgreSQL is a robust and trustworthy relational database management system with a variety of capabilities that make it well-suited for sophisticated applications. Because of its support for advanced SQL and object-relational database capabilities, transactions and concurrency, indexing and query optimisation, and security, it is a popular option for many applications. It is the best option for creating backend applications in the Java environment because of its extensive support for Java and the Spring Boot framework.

### **IntelliJ IDEA**

A well-liked integrated development environment (IDE) for Java developers is IntelliJ IDEA. It was developed by JetBrains and was made available in 2001. Both seasoned and inexperienced developers can benefit from IntelliJ IDEA's strong code analysis and refactoring features, which make it an invaluable tool.

The code editor in IntelliJ IDEA is a vital component. Code completion, syntax highlighting, and auto-formatting are just a few of the many functions that the editor offers. It also comes with a variety of navigational tools that make it simple to find and get around inside a codebase, like a file structure view and a search function.

Developers can step through their code and inspect variables and data structures at runtime using the robust debugger included in IntelliJ IDEA. The debugger

has features including call stack examination, expression evaluation, and breakpoint management.

Support for build automation tools like Maven and Gradle is another advantage of IntelliJ IDEA. Because it already has support for these tools, managing dependencies and creating projects is a breeze.

In addition, IntelliJ IDEA has a variety of plugins that expand its functionality. Plugins provide support for a wide range of technologies, including web frameworks like Spring and JavaServer Faces (JSF) and languages like Scala and Kotlin.

The integration of IntelliJ IDEA with other JetBrains tools, including ReSharper for .NET development and PyCharm for Python development, is one of the software's main advantages. This increases developer productivity and lowers the learning curve for new tools by enabling them to use a consistent set of tools across various projects and languages.

Developers may create code more quickly and with fewer errors thanks to a number of productivity enhancements included in IntelliJ IDEA. These include tools that assist programmers avoid common mistakes and enhance the quality of their code, such as code templates, live templates, and code inspections.

The support for version control programmes like Git and Subversion provided by IntelliJ IDEA is another asset. The management of code repositories and working with other developers are made simple by the built-in support for these systems.

IntelliJ IDEA offers robust support for frameworks like Spring Boot and Hibernate when creating Java backend applications. For these frameworks, it has capabilities like syntax highlighting and automatic code generation that make it simple to create and maintain large codebases.

In conclusion, IntelliJ IDEA is an effective and powerful IDE for Java developers. Because of its powerful code analysis and refactoring capabilities, highly configurable code editor, powerful debugger, support for build automation tools and plugins, and interaction with other JetBrains products, it is a great tool for developers of all skill levels. Because of its strong support for Java frameworks and version control systems, it is a fantastic choice for developing backend applications in the Java environment.

### **Postman**

Postman is a popular programme for administering and testing APIs (Application Programming Interfaces). Since its initial release in 2012, it has developed into an essential tool for the development of APIs.

Through Postman's user-friendly interface, developers can submit queries to APIs and receive responses. The HTTP methods GET, POST, PUT, DELETE, and PATCH are all compatible with it. Developers can quickly verify the functioning of their APIs with Postman, fix issues, and make sure the APIs are operating as intended.

Postman's capacity to save and group requests into collections is one of its most valuable features. The groups of similar requests known as collections can be shared and saved with other team members. since a result, productivity is increased and duplication of effort is decreased since engineers can work together on the same requests.

Additionally, Postman offers strong testing tools including automated testing, which enables developers to test APIs in a dependable and repeatable manner. This is accomplished by using test scripts written in JavaScript that can be automatically executed on a set of queries. This makes sure that APIs are dependable and consistent throughout time.

Postman also offers management and building of environments. Environments are collections of variables that can be used to categorise various settings, including development, staging, and production. Developers may now quickly move between several environments and test APIs in various scenarios thanks to this.

Additionally, Postman has a function called "Mock Servers" that lets programmers imitate a real API without the requirement for a backend implementation. This is helpful for developers who are working on an application's front end and need to test their API requests without access to a functional back end.

Postman's ability to integrate with other programmes and services like GitHub, Jira, and Slack is another key feature. This enables developers to collaborate more successfully with their team members and integrate Postman into their existing development workflow.

Postman is a powerful and essential tool for developers that use APIs in general. It is an essential stage in the design of APIs because of its user-friendly interface, support for automated testing, collections, environments, and integrations.

## **Swagger**

RESTful APIs can be created, documented, and used by developers with the help of the Swagger open-source software platform. Following its initial development by Tony Tam and his coworkers at Reverb Technologies, it is presently maintained by the OpenAPI Initiative (OAI). By providing a standardised interface for specifying API endpoints and activities, Swagger's major goal is to accelerate the design and implementation of APIs.

There are various parts that make up the Swagger framework, including the Swagger Editor, Swagger UI, and Swagger Codegen. Using the OpenAPI Specification (OAS) standard, developers can define and amend API specifications using the web-based Swagger Editor. The OAS format, which is

based on JSON or YAML syntax, is a standardised method of representing RESTful APIs. The Swagger Editor offers a feature-rich user interface that aids developers in creating and editing API specifications and offers immediate feedback on their accuracy.

The Swagger Editor's API specification can then be used to build client code and documentation, which can then be produced by the Swagger Codegen. Several computer languages, including Java, Python, and Ruby, can be created using the command-line tool known as the Swagger Codegen. The Swagger Codegen produces client libraries, server stubs, and documentation in addition to the code itself.

A user interface for testing and researching RESTful APIs is offered by the web-based Swagger UI tool. A user-friendly interface for developers to interact with the API endpoints is provided by the Swagger UI, which is automatically produced from the API specification. Developers can study the API documentation, test API endpoints, and observe sample requests and responses using the Swagger UI.

The ability to standardise how APIs are described and used is one of the key benefits of utilising Swagger. When endpoints and actions are provided using a standard format, it is simpler for developers to comprehend and use APIs made by other developers. As a result, developers may be able to interchange and reuse code more readily, which can boost productivity and shorten development timeframes.

The consistency and dependability of APIs are further enhanced by the implementation of Swagger. Developers can find and address issues with their APIs more easily when the definition of an API is clear. Because of this, APIs could be more reliable and durable and less likely to malfunction or create problems when used by other developers.



With its extensive selection of plugins and extensions, Swagger is also incredibly adaptable. These add-ons and extensions can be used to increase Swagger's functionality and integrate it with other programmes and frameworks. For instance, there exist plugins for Swagger that may be used with Spring Boot, Node.js, and other well-liked web development frameworks.

In conclusion, Swagger is an effective and adaptable framework for creating, consuming, and documenting RESTful APIs. The consistency and quality of APIs are improved by its standardised syntax for describing API endpoints and actions, and its user-friendly tools make it simple for developers to interact with and consume APIs. Swagger is a great option for developers wishing to create powerful and dependable APIs thanks to its broad selection of plugins and extensions.

### **3.3.2 Hardware Requirements**

- Ram: 8GB or higher,
- Storage: 500GB,
- CPU: 2GHz or faster, and
- Architecture: 32Bit or 64Bit

It's critical to keep in mind that the specified hardware and system requirements may change depending on the application's specific requirements and anticipated usage. Continuous system performance monitoring is advised, and any necessary hardware configuration changes should be made. The necessary hardware may become substantially more demanding for larger applications with heavy workloads and several concurrent users interacting with the API.

### **3.4 Implementation and Deployment**

A Spring Boot application must be configured, the data model must be established, and business logic must be implemented, among other processes. The fact that Spring Boot offers a large range of capabilities and components

that make many of these jobs easier is one of the main advantages of utilising it.

The programme must first be configured. In order to do this, you must define the application's attributes, including the server port and database connection information. It is simple to handle these properties because Spring Boot offers a centralised configuration mechanism. Both environment variables and properties files can be used to specify the properties.

The application's data model must then be established. In order to do this, Java classes that represent the data entities and their connections must be created. The Java Persistence API (JPA) and object-relational mapping (ORM) frameworks like Hibernate are supported by Spring Boot when working with relational databases.

Implementing the business logic is possible after defining the data model. In order to do this, Java code must be written to carry out the necessary data operations, such as record creation, reading, updating, and deletion. The Spring Data JPA repository abstraction, which offers a common interface for communicating with the database, is one of the tools offered by Spring Boot to make this process simpler.

The programme must expose a REST API for customers to use in addition to implementing the business logic. In addition to the Spring MVC framework and the Spring WebFlux reactive framework, Spring Boot offers a potent set of tools for creating RESTful APIs. Endpoint definition, request and response handling, and exception and error handling are all made simple by these frameworks.

A Spring Boot application must be configured, its data model must be defined, business logic must be implemented, a REST API must be exposed, and the application must be tested and deployed. Building contemporary, scalable, and

dependable applications is made simple by the large variety of capabilities and components that Spring Boot offers.

### 3.5 Database Schema

Two key tables—Category and Product—would make up the database design. The category name and category ID (the primary key) would be columns in the Category table. Columns for the product ID (primary key), product name, product description, product price, and category ID (foreign key referencing the Category table) would be found in the Product table.

**Categories table:**

Column Name	Data Type	Constraints
category_id	INTEGER	PRIMARY KEY, AUTO INCREMENT
category_name	VARCHAR(50)	NOT NULL

Figure 3: Categories table schema

**Products table:**

Column Name	Data Type	Constraints
product_id	INTEGER	PRIMARY KEY, AUTO INCREMENT
product_name	VARCHAR(50)	NOT NULL
price	DECIMAL(10, 2)	NOT NULL
category_id	INTEGER	FOREIGN KEY REFERENCES Categories(category_id)

Figure 4: Products table schema

A popular database schema design known as a "many-to-one relationship" illustrates a relationship between two entities in which one entity may have several instances of the other entity, but each instance of the other entity may only be related to one instance of the first entity. In this scenario, there are two entities: categories and products, where each product can only belong to one category but each category may have several products.

A number of restrictions have been added to the database structure in order to maintain data integrity and consistency. The use of foreign keys to prevent adding a product to a category that doesn't exist is one example of such a restriction. To ensure that no two categories or items have the same name or ID, the schema additionally incorporates distinctive restrictions.

# CHAPTER-4

## EXPERIMENTS & RESULT ANALYSIS

### 4.1 Overview

The API was effective in developing a large number of endpoints for data processing and database retrieval. To make sure the API was trustworthy and useful, extensive testing was done at every stage of the development process. Even when dealing with massive amounts of data, the API's rapid response time remained maintained. Logging was added to track the behaviour of the API in order to reduce performance overhead, and only pertinent data was captured by specifying the logging level.

The API was constructed using RESTful design principles, which made it easy for developers to comprehend and utilise. To give illuminating error messages in the event of any errors or exceptions, proper error handling was put in place. A three-layer design was employed to divide the presentation, service, and data access layers in order to guarantee the code's scalability and flexibility.

Low hardware needs for the API made it simple to install on a number of systems. By caching commonly used data and lowering the amount of database queries, the API's speed was boosted.

Overall, the API met all of the project's requirements and went above and beyond in terms of functionality and speed. It has the potential to be used in a number of applications and may be customised to match the unique requirements of diverse projects.

## 4.2 Experiment Results

The Spring-Boot application's API has been successfully deployed and tested. It has APIs that enable the retrieval of all items, categories, and goods falling under a given umbrella category. Services, DAOs, and a PostgreSQL database were then introduced into the system once the API stubs were gradually added and tested.

Test cases were written and ran for controllers, services, and DAOs to make that the API operates as expected. To guarantee that any problems or faults are quickly found and fixed, adequate logging was introduced at the necessary levels.

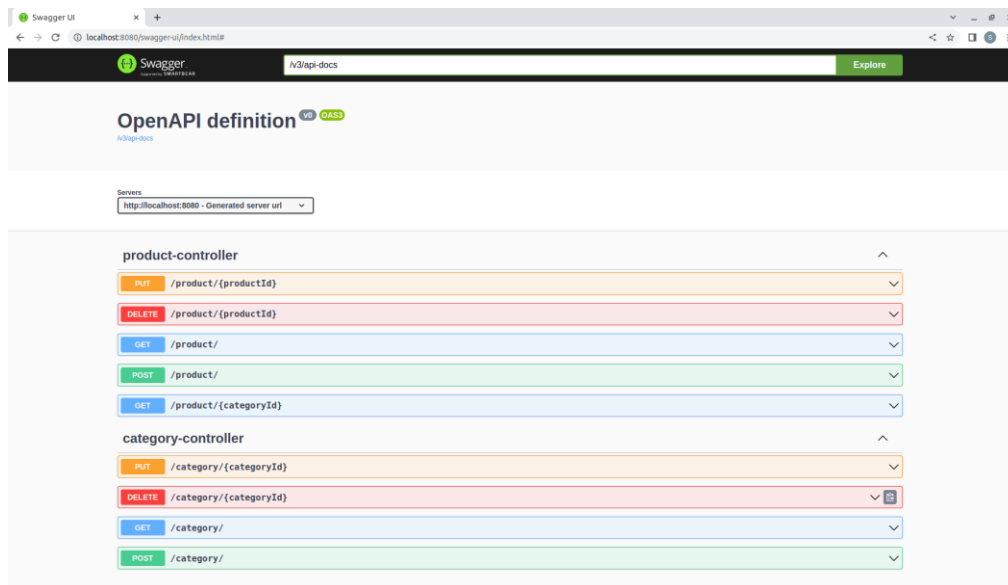
The API's response content can be changed as necessary and was made to interact with both request and response headers. For the client to receive the right status code, a response code may also be given.

Additional APIs were added to create and update goods, and these were thoroughly tested to verify they worked as intended. So the Spring-Boot application's API has been successfully implemented and thoroughly tested. It offers the essential resources for finding and changing information about products and categories.

## 4.3 Outputs at Various Stages

In this section, we'll look at how the Spring Boot RESTful API responded to various forms of user requests or data. We'll also keep track of how user requests that receive accurate exception answers from our RESTful API turn out. Our

API has been carefully developed to handle all errors while taking into account all potential user scenarios.



*Figure 5: Swagger UI for spring application*

Developers can interact with APIs and see their architecture using the open-source programme SwaggerUI. This web-based graphical user interface offers a framework for RESTful API testing and documentation. SwaggerUI makes API development and testing simpler by providing interactive API documentation that is simple to integrate into other applications or share with team members.

SwaggerUI is becoming more well-liked among API developers as a result of its simple interface and effective functionality. Because of this, programmers can quickly prototype and test their APIs to make sure they adhere to the necessary standards. Additionally, SwaggerUI offers a number of tools that help developers find and fix problems, making it a vital tool for developing dependable, high-quality APIs. SwaggerUI is more user-friendly than sending requests through the POSTMAN app.

In our implementation of the API, we made use of DTOs for sending requests; in this report, we'll talk more about how they're used. Since they enable communication between many application layers, including the controller, service, and data access levels, DTOs, also known as data transfer objects, are an essential component of contemporary API architecture. By separating data transmission from business logic with DTOs, an API can improve its performance, scalability, and maintainability. In large-scale applications, where changes to one component may have an impact on the entire system, this separation is especially crucial.

To ensure that only relevant data is provided and to improve efficiency while lowering the chance of data leakage or security flaws, we used DTOs in our API implementation to encapsulate the data being exchanged between the various layers of the application. DTOs also give us the ability to guarantee data consistency and adherence to a certain agreement or standard, which is essential when working with external systems, numerous teams, or microservices. Conflicts can be avoided and we can guarantee that the API is still functional and interoperable by defining a clear contract for data transport.

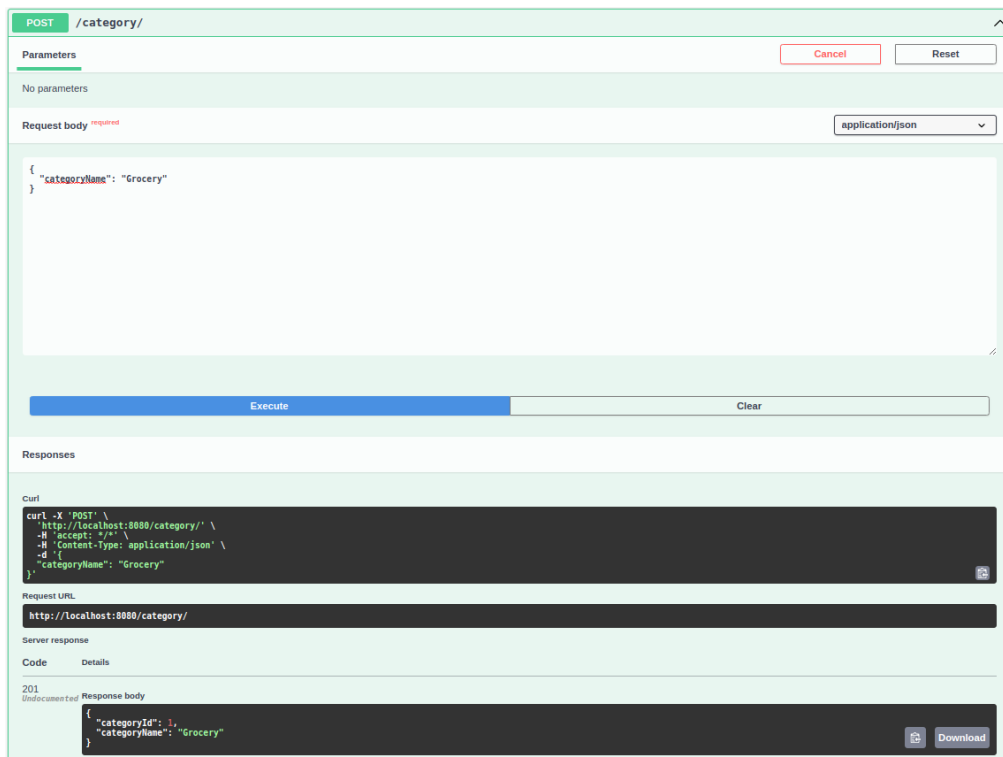


Figure 6: Posting a category



A POST request was made using our Swagger API, as we can see. POST requests are frequently used to add or insert new data into databases. In this instance, a new category needed to be added to our database. The newly created data is sent to us together with a response code, 201 (signifying that the creation was successful), after the request has been processed. We can determine from this response that using only the SwaggerUI, we successfully created a new category.

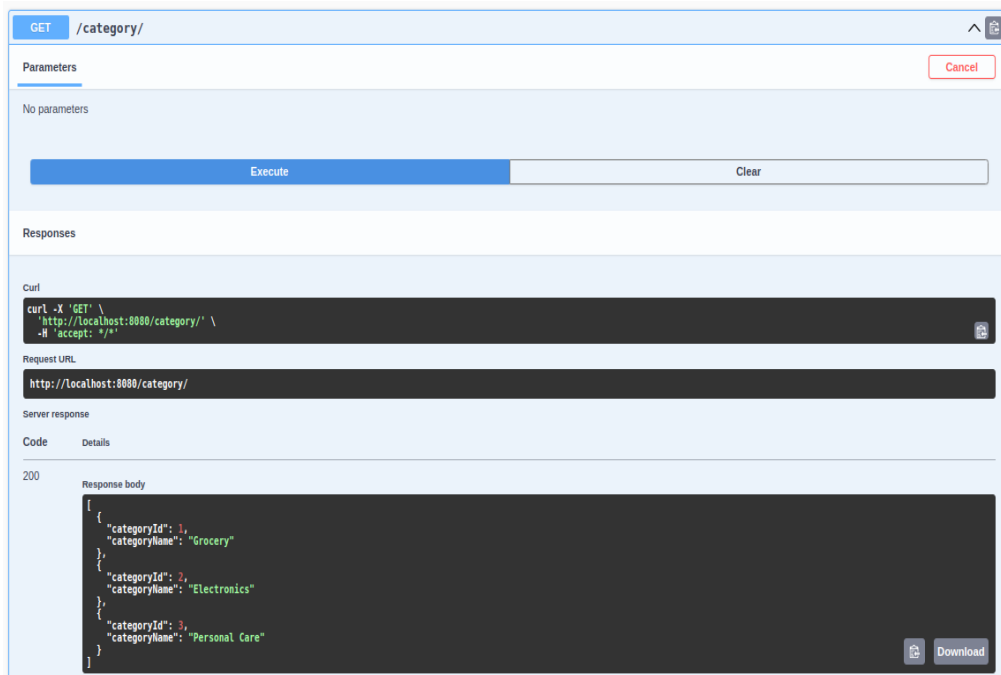


Figure 7: Retrieving a category

We can see from the execution of a GET request from our Swagger API that it retrieves all the data from the database and provides it to the user as a response. In this instance, the request was utilised to get access to every category that was kept in our database. After the programme has run, a response body containing the data that is available and the response code 200 (OK) are received. This demonstrates that utilising SwaggerUI's GET request can successfully fetch data from the database.

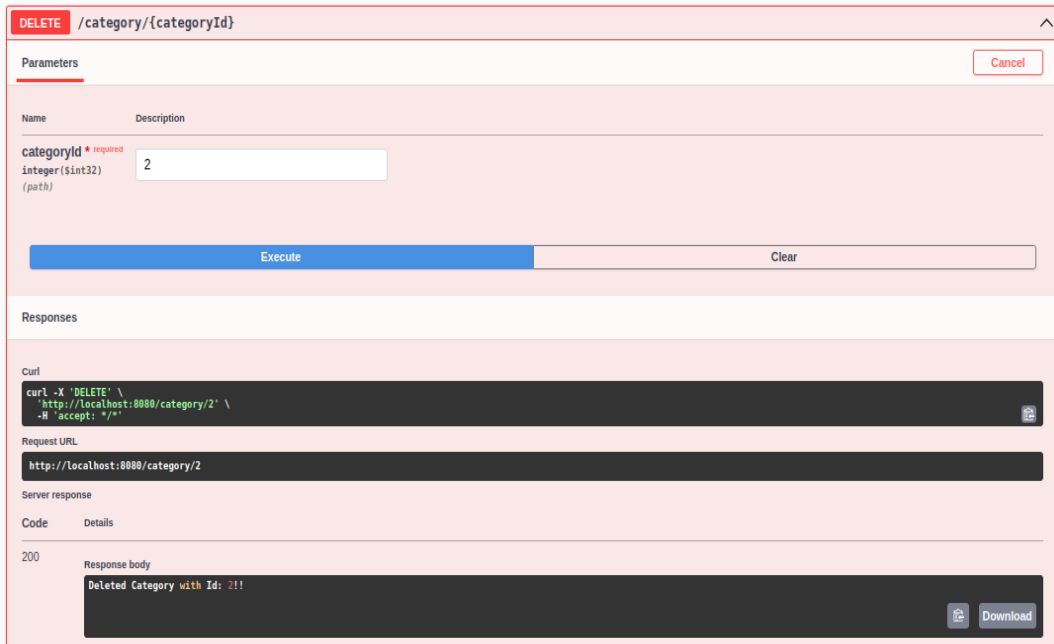


Figure 8: Deleting a category

When we look closer, we see that our swagger API sent out a DELETE request. We all are aware that records are deleted from databases via DELETE requests. Here, the request is particularly intended to delete a category from our database, and the path parameter contains the category's id. Following the request's execution, we get a response body confirming the category's successful deletion along with a 200 success response code.

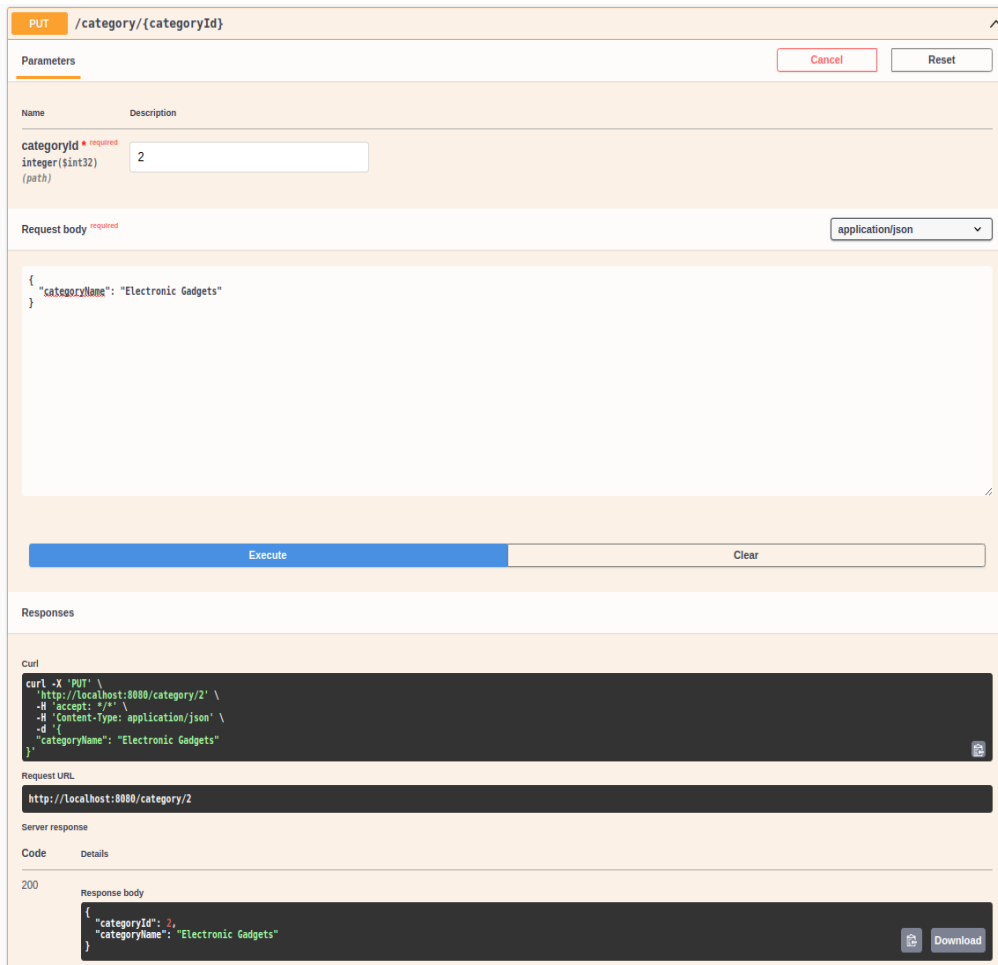


Figure 9: Updating a category

The Swagger API was used to carry out a PUT request in the example below, which is commonly used to update a database record. Here, the category ID was utilised as a route argument to update the name of a category in our database using the request. Following execution, the response body contains the updated data and a 200 response code, which denotes success. By using the PUT request through the Swagger UI, we can see from this answer whether we were successful in updating a record in the database.

It's time to assess our API's behaviour under unusual circumstances after looking at its typical inputs and outputs. By examining how our RESTful API manages unusual input from users, we will specifically investigate the exception handling for categories.

When attempting to update or create a category with a name that is on record in the database or when using null or empty name values, we will run tests to see how our API responds.

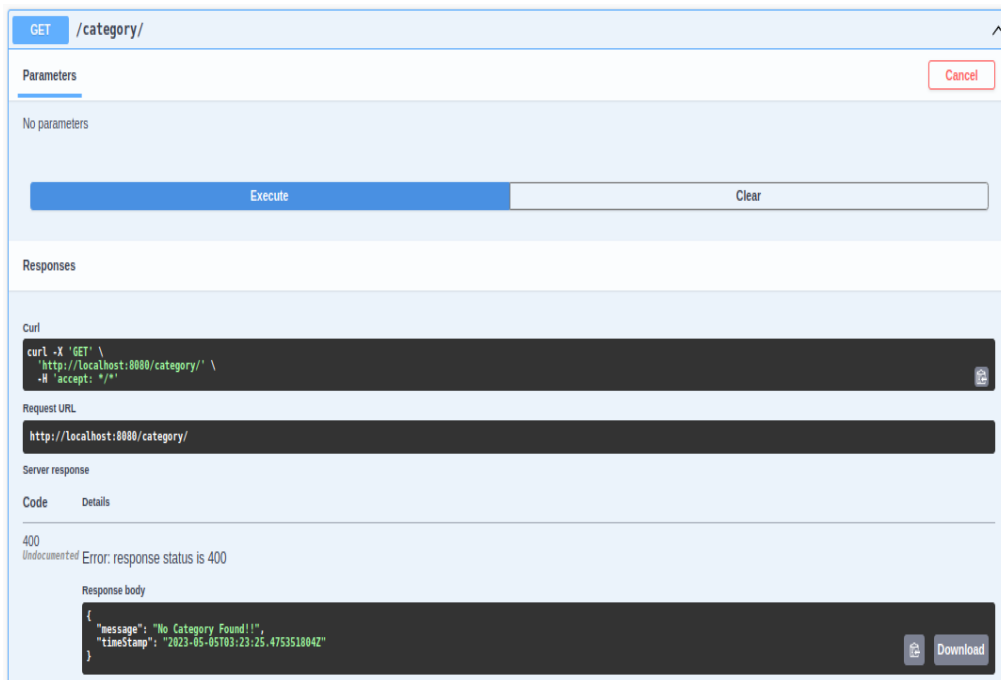


Figure 10: Error in retrieving a category

In order to retrieve all of the data from the database and provide it to the user as a response, we used a GET request from our Swagger API. We were trying to get all of the categories from our database in this example. However, we got a 400 (Error) response code and an error message in the body of the response. The category table in the database was empty, thus there was nothing to fetch, which is why this happened.

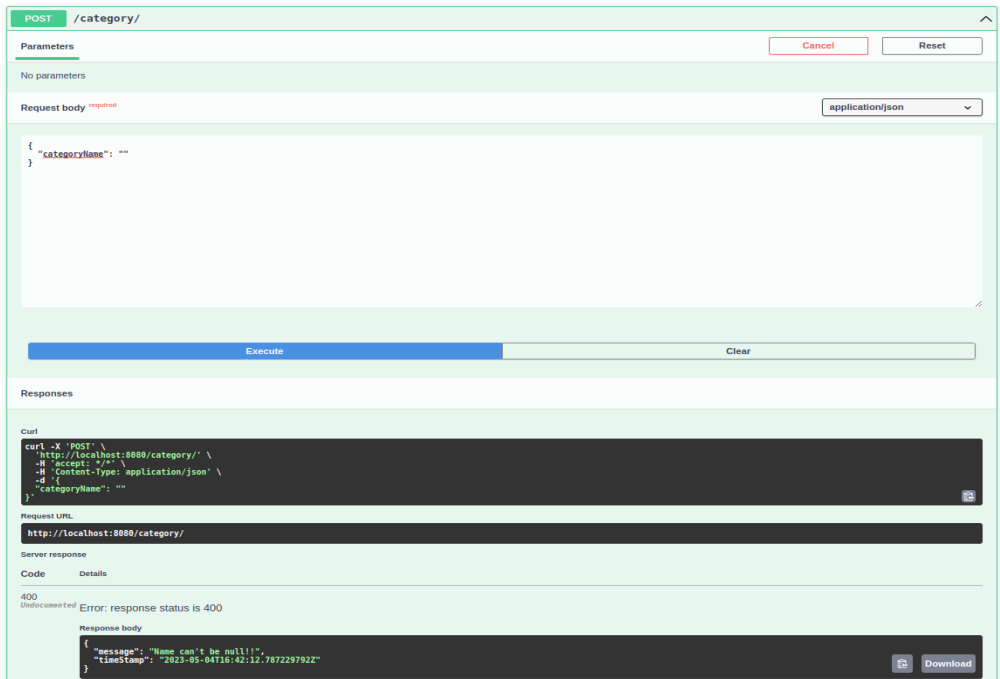


Figure 11: Error in posting a category

With an empty category name, we sent a POST request from our Swagger API. However, the response we got had a 400 response code, which denotes an error, and an error message.

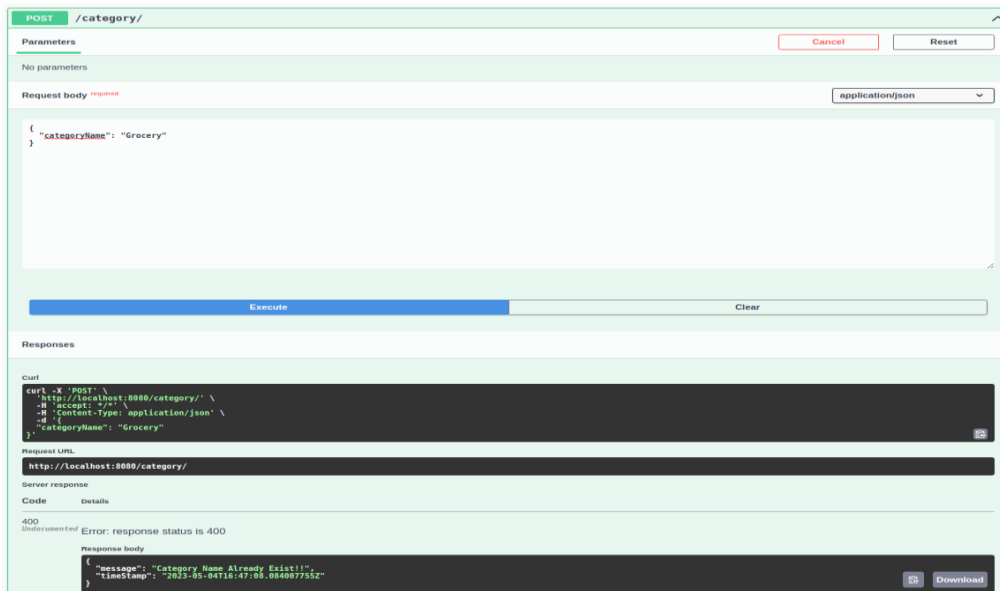


Figure 12: Error in posting a existing category

We can see from the POST call we made using our Swagger API that the category name was already present in the database. The response code was 400, which denotes an error, and the response body contained an error message as a result.

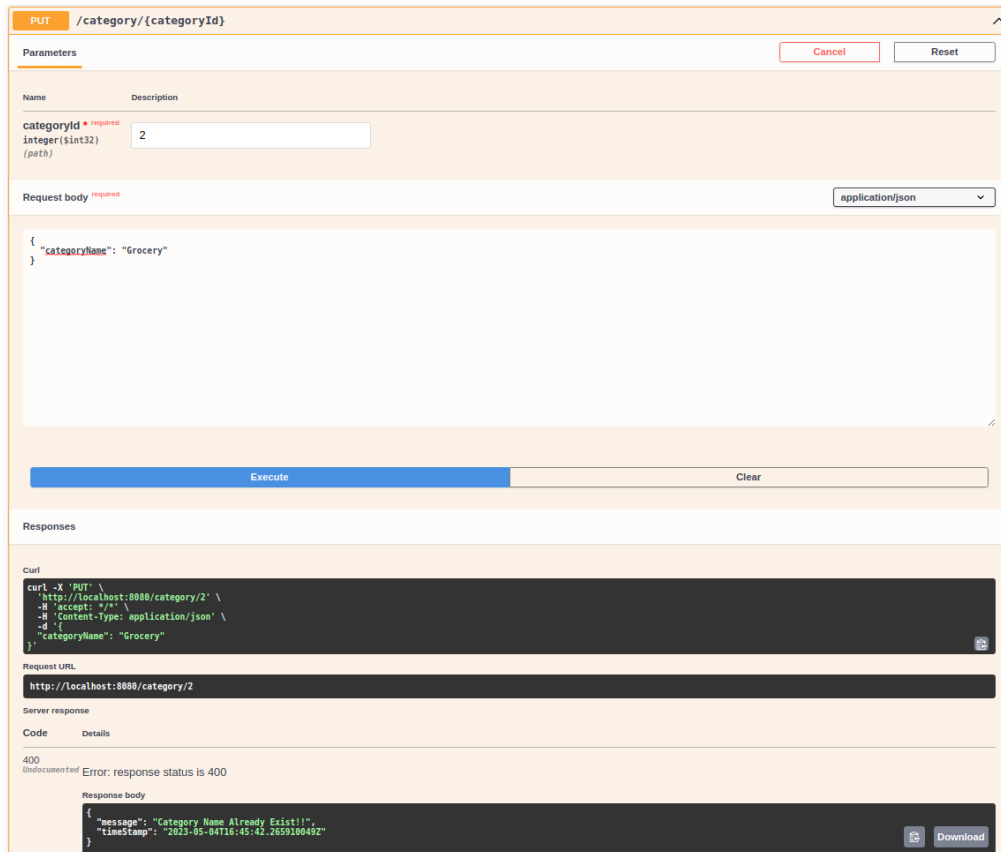


Figure 13: Error in updating a category

By sending the category ID as a path parameter in a PUT request from our Swagger API, we can see that the request is meant to update a category name in our database. The response body, on the other hand, shows an error message with a response code of 400 (for Error). The reason for this is that a category with the same name already exists.

After successfully creating a number of categories in our PostgreSQL database, we can now go on to creating products for the various categories that are available.

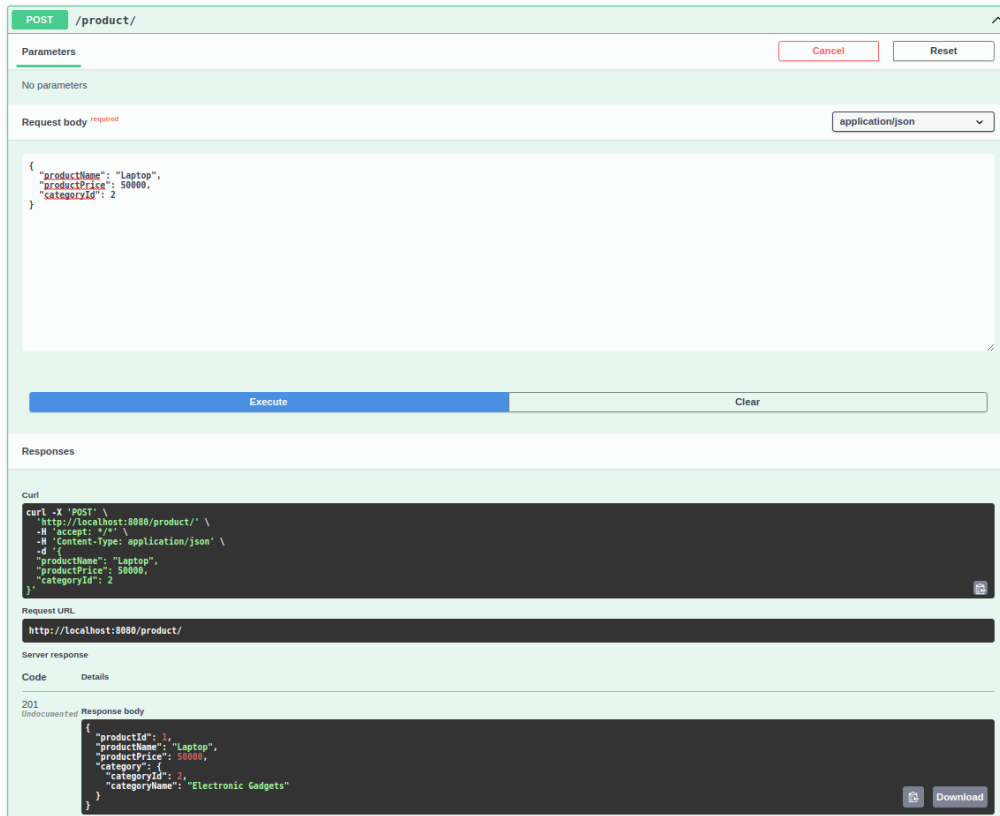


Figure 14: Posting a product

Here, we can see that a POST request was sent from our Swagger API to the database to add a new product. The user-provided category ID/type, price, and product name were all included in the request body. After the request was processed, we got the created data in the response body and the response code 201, which meant the creation was successful.

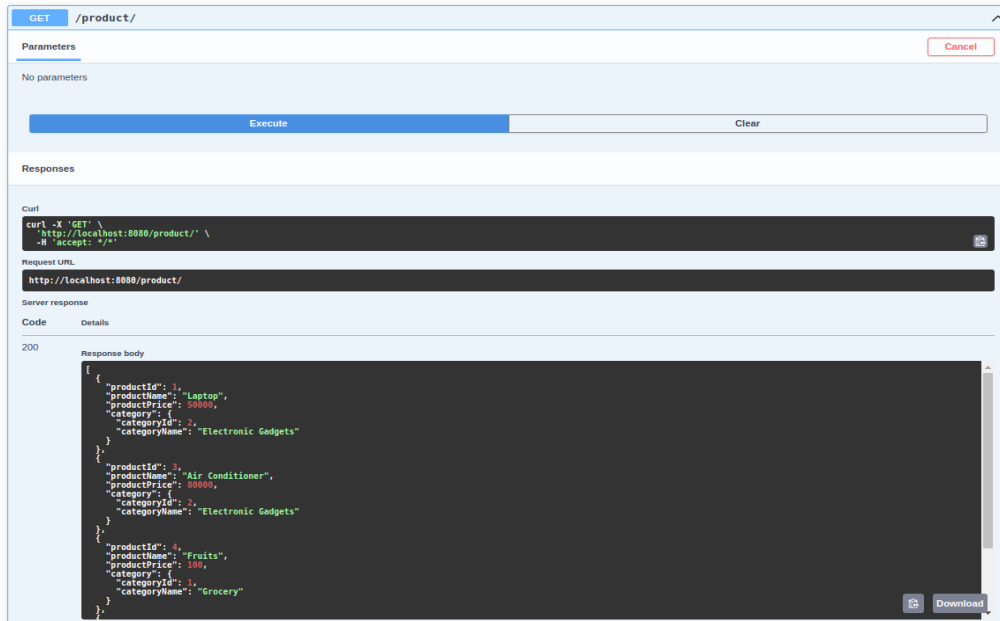


Figure 15: Retrieving a product

To get access to all the products kept in our database, we sent a GET request over our Swagger API. The goal of this request was to retrieve all of the available goods. The obtained data is contained in the response body, and the response code is 200 (signifying success).

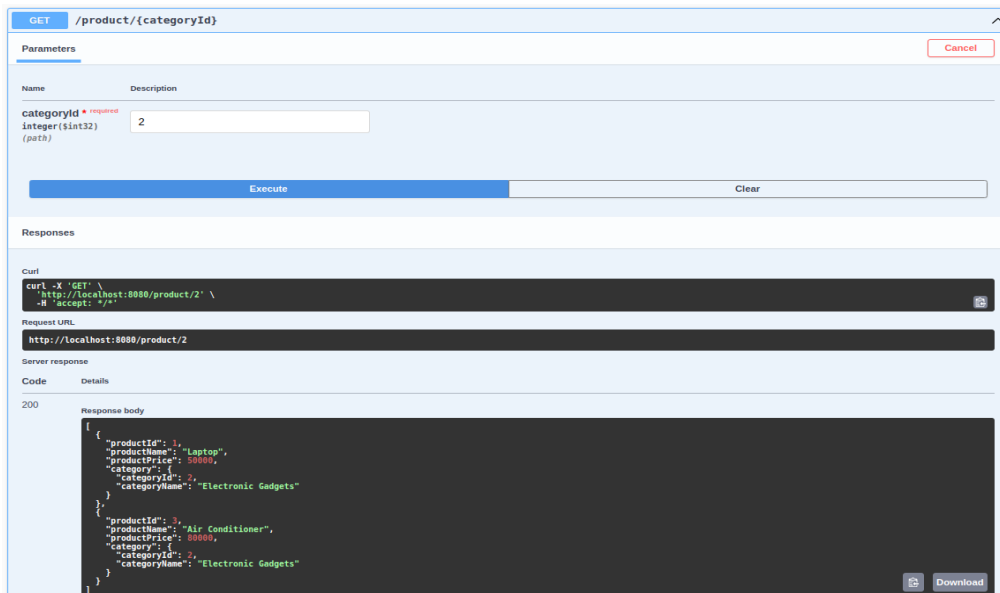


Figure 16: Retrieving a product of specific category



It is clear that we used our Swagger API to make a GET call to get all items from a particular category from our database (the category ID was supplied as a path argument). Following execution, we get the data that was retrieved together with a response code of 200 (which denotes success).

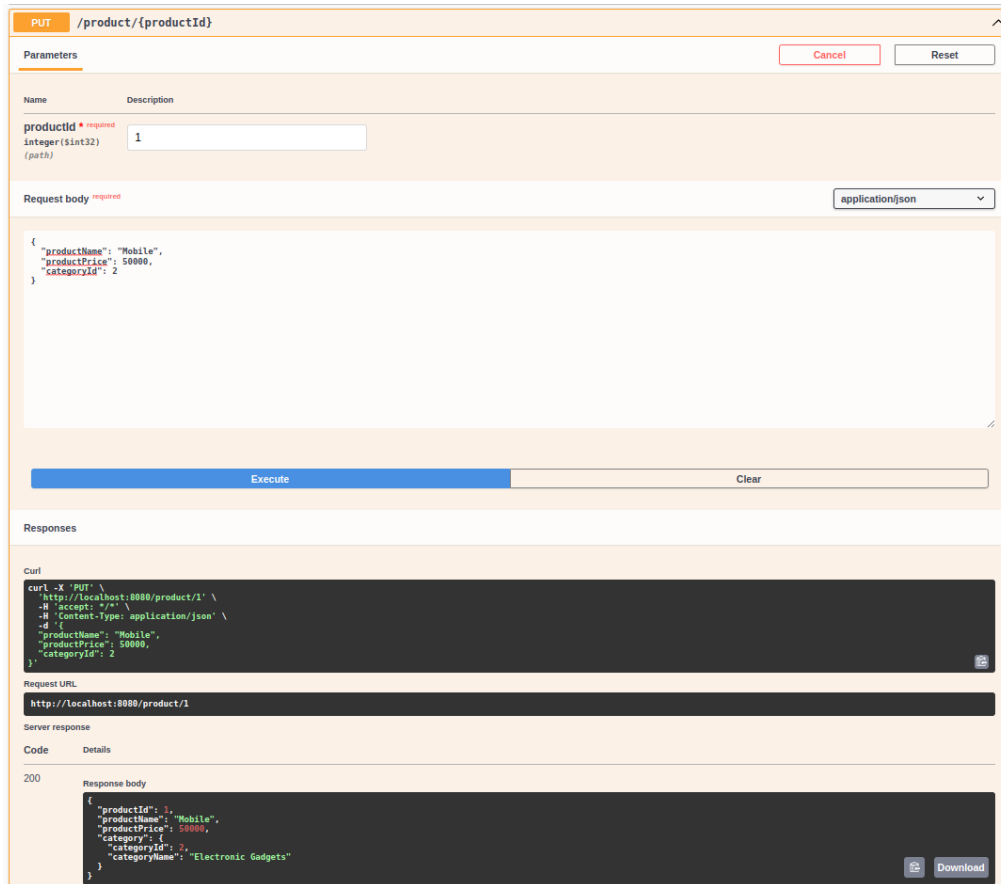


Figure 17: Updating a product

In order to update a product in the database, we issued a PUT request from our Swagger API. The product ID that needs to be updated is included in the request as a path parameter. Following execution, we got the modified data in the response body and the response code 200 (OK). This demonstrates that our PUT request using the Swagger UI successfully updated a record in the database.

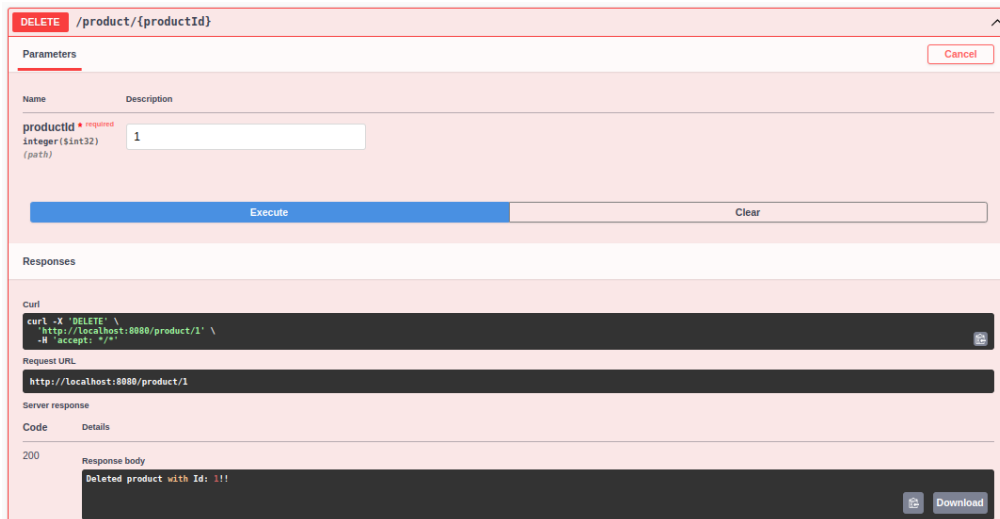


Figure 18: Deleting a product

To remove a product from our database, our swagger API sent a DELETE request. The product's id must be eliminated as a path parameter for this request. After the command has been carried out, a response body containing a message verifying the deletion and a response code of 200 (OK) is received. Because of the cascade property that has been configured, when a category is deleted from the database, all of the goods that belong to that category are also deleted.

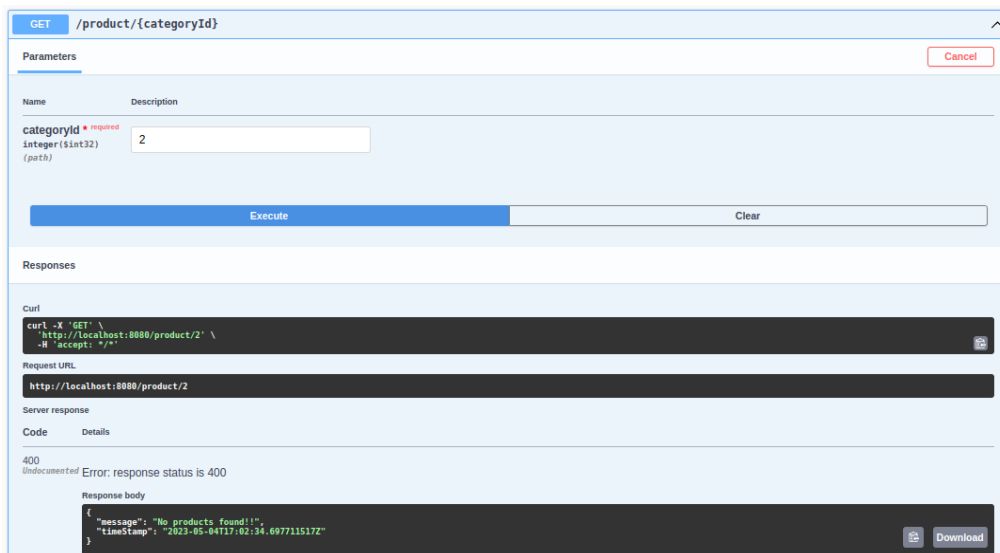


Figure 19: Error in retrieving a particular product

To obtain all goods connected to a particular category ID supplied as a path argument, we carried out a GET call over our Swagger API. But we discovered that the category was already gone from the database. Despite this, we got a response with the response code 400 (for ERROR), which said there were no goods available.

We have noted the typical product inputs and API answers in a variety of situations. Let's look at how our API responds to abnormal situations now. This means checking our RESTful API for erroneous user inputs and confirming the handling of exceptions for individual goods.

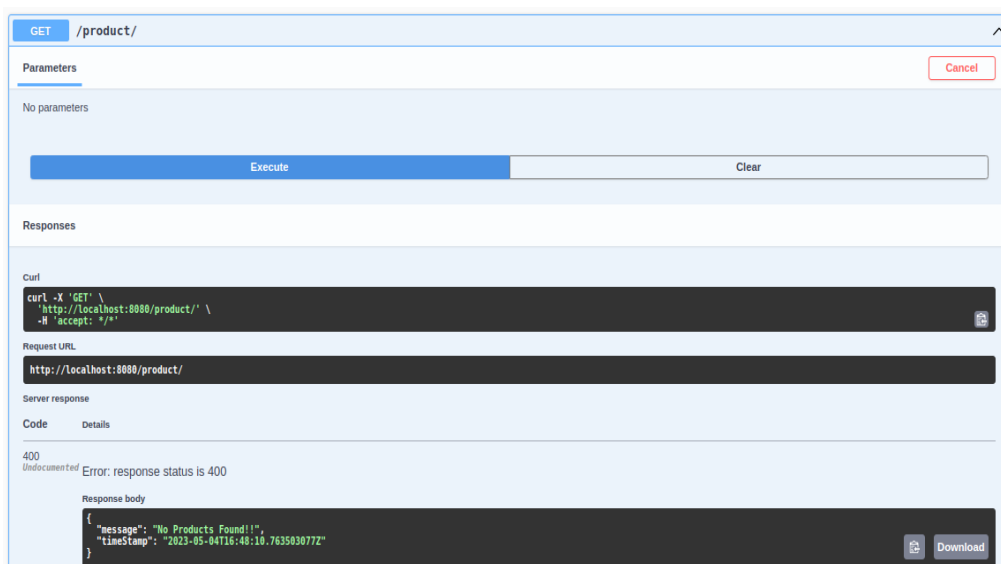


Figure 20: Error in retrieving a product

For the purpose of retrieving all the items, we sent a GET request over our Swagger API. The answer body we received, however, contained a notice stating that no items were accessible and a response code of 400, signifying an error because no products were present in our database.

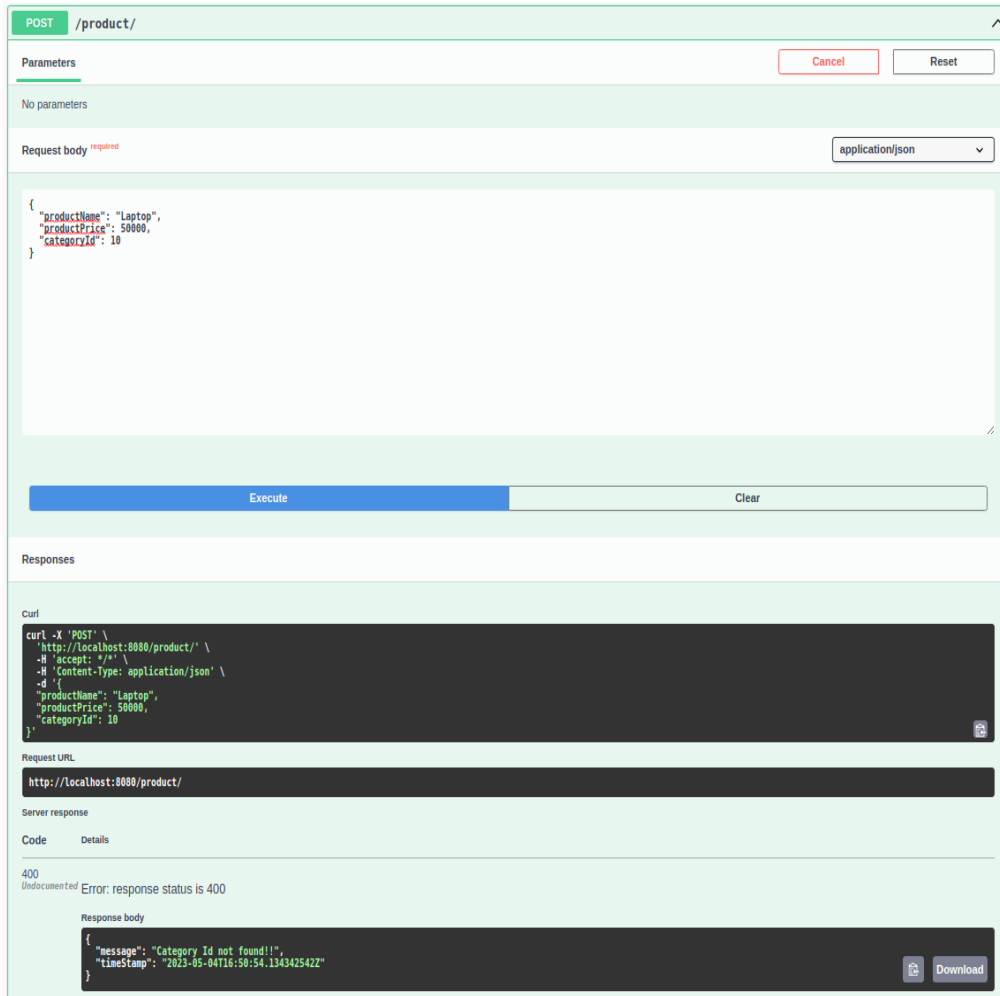


Figure 21: Error in posting a product

To add a new product to our database, we issued a POST request using our Swagger API. The user supplied the request body, which included the product name, price, and category ID or type. Nevertheless, after submitting the request, we got an error message with the generated data and a 400 (for error) response code. This is due to the fact that no category matching the specified ID is present in the database.

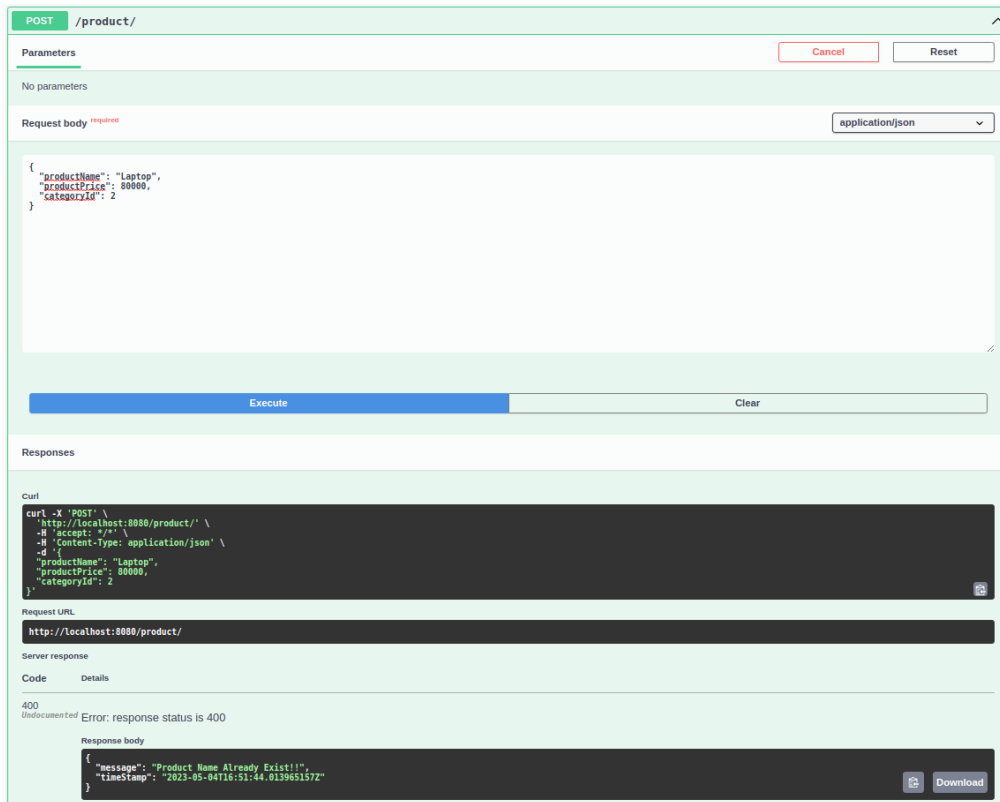


Figure 22: Error in posting an existing product

With the user-provided product name, price, and category id, we hoped to add a new product to our database after processing a POST request from the swagger API. However, because a product with the same name already existed in our database, the request returned a response code of 400 (which denotes an error), along with an error message and the produced data.

The API also deals with the exception that results from creating a product without a name. It shows a necessary message. All of these exceptions are also addressed in the PUT request to guarantee adequate data validation of user input and prevent the introduction of unnecessary data into our database, which could result in a number of errors.

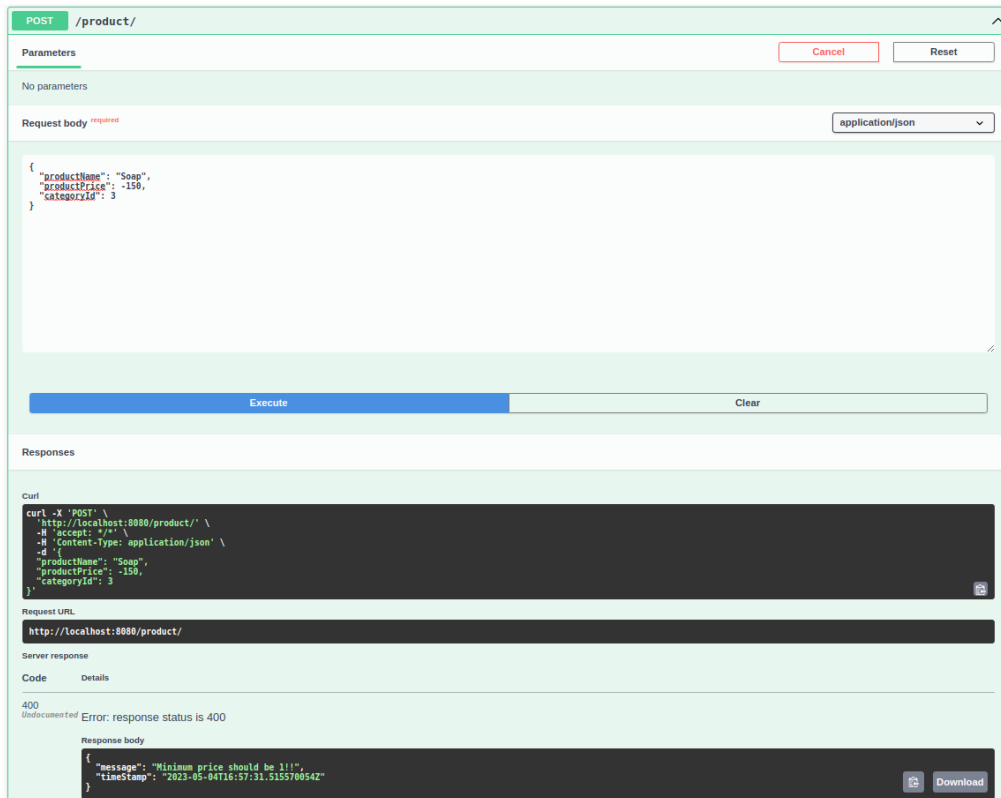
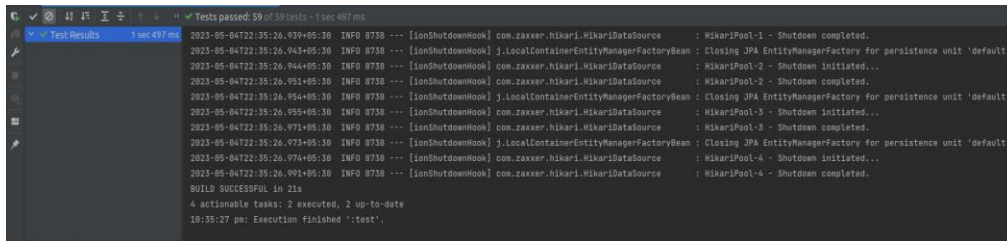


Figure 23: Error in posting a product with negative price values

Using our Swagger API, we conducted a POST request to add a new product to the database with the information provided in the request body, including the product name, price, and category ID or type. Although the user attempted to create a product with a negative pricing value, which is not permitted according to the validation established in our API, we received an error message along with the created data and a response code of 400 (showing an error) when we executed the request.

## 4.4 Performance Analysis

A complete unit test coverage was performed, and it was found that all 59 tests that were developed for a range of situations and scenarios passed and were given the status "PASSED". Both Mockito and JUnit5 were used to develop the tests, allowing for thorough testing of various functionality and ensuring that the code was working as intended without any unexpected failures or defects.



```
Tests passed: 59 of 59 tests - 1 sec 497 ms
Test Results 1 sec 497 ms
2023-05-04T22:35:26.939+05:30 INFO 8738 --- [onShutdownHook] com.zaxxer.hikari.HikariDataSource : HikariPool-1 - Shutdown completed.
2023-05-04T22:35:26.943+05:30 INFO 8738 --- [onShutdownHook] j.LocalContainerEntityManagerFactoryBean : Closing JPA EntityManagerFactory for persistence unit 'default'
2023-05-04T22:35:26.944+05:30 INFO 8738 --- [onShutdownHook] com.zaxxer.hikari.HikariDataSource : HikariPool-2 - Shutdown initiated...
2023-05-04T22:35:26.951+05:30 INFO 8738 --- [onShutdownHook] com.zaxxer.hikari.HikariDataSource : HikariPool-2 - Shutdown completed.
2023-05-04T22:35:26.954+05:30 INFO 8738 --- [onShutdownHook] j.LocalContainerEntityManagerFactoryBean : Closing JPA EntityManagerFactory for persistence unit 'default'
2023-05-04T22:35:26.955+05:30 INFO 8738 --- [onShutdownHook] com.zaxxer.hikari.HikariDataSource : HikariPool-3 - Shutdown initiated...
2023-05-04T22:35:26.971+05:30 INFO 8738 --- [onShutdownHook] com.zaxxer.hikari.HikariDataSource : HikariPool-3 - Shutdown completed.
2023-05-04T22:35:26.973+05:30 INFO 8738 --- [onShutdownHook] j.LocalContainerEntityManagerFactoryBean : Closing JPA EntityManagerFactory for persistence unit 'default'
2023-05-04T22:35:26.974+05:30 INFO 8738 --- [onShutdownHook] com.zaxxer.hikari.HikariDataSource : HikariPool-4 - Shutdown initiated...
2023-05-04T22:35:26.991+05:30 INFO 8738 --- [onShutdownHook] com.zaxxer.hikari.HikariDataSource : HikariPool-4 - Shutdown completed.
BUILD SUCCESSFUL in 21s
4 actionable tasks: 2 executed, 2 up-to-date
10:35:27 pm: Execution finished 'test'.
```

Figure 24: Performance analysis of the application

# CHAPTER-5

## CONCLUSIONS

### 5.1 Conclusion

Using Spring Boot, we created a REST API for this project that has a large number of endpoints for diverse uses. It enables retrieving all products, categories, and things that fall within a specific category. Our API has three layers that make up its architecture: a controller layer, a service layer, and a DAO layer. Data retrieval and storage are handled by the DAO layer's interactions with the database, while business logic is handled by the service layer. We also developed test cases for the controller, service, and DAO levels to make sure our API is operating correctly.

In conclusion, we have successfully created a dependable and scalable REST API using Spring Boot. The API's three-layer design makes it simple to update and maintain the codebase as needed. We have effectively stored and retrieved enormous volumes of data thanks to the use of a PostgreSQL database. The portability of the API, which enables it to be utilised in a variety of platforms and applications, including e-commerce and inventory management systems, is one of its key advantages.. The design of specialised interfaces for diverse applications is made simple by the ability to easily retrieve all categories, all commodities, and all items under a given category.

Another useful feature of the API is its logging functionality, which simplifies the tracking and debugging of mistakes in complicated programmes. The codebase's possible performance bottlenecks can be found by using the logging function.

To make using the API easier, request and response headers have been introduced. Now that developers can send sensitive information in the headers,



including authentication tokens, the API is more secure. Building a solid API also requires having the ability to determine a response code. It can be used by developers to tell the client application whether the request was successful or not.

The development of a product creation and update API has increased the functionality of the API. In order to save time and effort, developers can now create and change products programmatically.

In conclusion, our Spring Boot-based REST API provides a versatile, scalable, and trustworthy platform for developing a variety of apps. It's a great option for developers because to its three-layer structure, PostgreSQL database usage, test cases, logging features, Request and Response Headers, and product creation and updating capabilities. We are sure that this API will support developers in producing top-notch applications that cater to their clients' needs.

## **5.2 Future Scope**

There are many possible improvements that might be made to the RESTful API developed in this project, which could better its performance, security, and usability for businesses and developers looking to accomplish their objectives more effectively.

The security of the API is a crucial area for development, and it might be reinforced by adding authentication and authorisation mechanisms. A user's level of access to the API would be determined by permission, while authentication would ensure that only authorised users are able to access it.

Another essential element that can increase API efficiency is caching, which lowers the amount of database requests necessary. Response times could be sped up and scalability increased by using a caching system.

Filtering and data pagination are critical for large databases. Clients could only get the data they require thanks to filtering and pagination capabilities, which would cut down on network traffic and improve performance overall.

For developers that want to use the API, clear and thorough documentation is also essential. The API should have comprehensive documentation that describes how to use it, what data is available, and its capabilities and restrictions.

A further strategy to stop abuse, improve efficiency, and guarantee fair use by all clients is rate restriction, which limits the volume of API traffic.

By interacting with other APIs, the API's value and potential for collaboration may be expanded. Users would benefit from an API that is simple to use and simple to combine with other APIs.

Last but not least, software updates might be distributed rapidly and efficiently using the automated process known as continuous integration and deployment (CI/CD). Without sacrificing availability, a CI/CD approach might provide smooth API upgrades.

## References

- [1] C. Zepeda-Núñez et al., "Performance evaluation of Spring Boot and Vert.x in microservices," 2019 IEEE/ACS 16th International Conference on Computer Systems and Applications (AICCSA), Abu Dhabi, United Arab Emirates, 2019, pp. 1-8, doi: 10.1109/AICCSA47632.2019.8812115.
  
- [2] M. S. Islam et al., "A Comparative Study on Implementing Microservices Architecture with Spring Boot and ASP.NET Core," 2020 International Conference on Computer, Communication, Chemical, Materials and Electronic Engineering (IC4ME2), Khulna, Bangladesh, 2020, pp. 1-6, doi: 10.1109/IC4ME248511.2020.9292276.
  
- [3] C. Basu and K. Singh, "The Role of Spring Boot in the Era of Cloud Native Java," 2019 IEEE International Conference on Cloud Computing (CLOUD), Milan, Italy, 2019, pp. 400-404, doi: 10.1109/CLOUD.2019.00069.
  
- [4] S. Singh, R. Singh and S. S. Saini, "A Comparative Study of Spring Boot and Node.js for Developing Microservices," 2019 3rd International Conference on Computing Methodologies and Communication (ICCMC), Erode, India, 2019, pp. 605-610, doi: 10.1109/ICCMC.2019.8712615.
  
- [5] S. S. M. M. Kamal, M. A. Hossain and M. A. R. Amin, "Performance Evaluation of Spring Boot for Scalable Web Applications," 2018 4th International Conference on Advances in Electrical Engineering (ICAEE), Dhaka, 2018, pp. 1-6, doi: 10.1109/ICAEE.2018.8629004.