Web Application in Java using Three Layered Architecture

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

in

Computer Science and Engineering

By

PARUL SHARMA (191206)

UNDER THE SUPERVISION OF

Dr EKTA GANDOTRA

to

Department of Computer Science & Engineering and Information Technology

Jaypee University of Information Technology, Waknaghat, 173234, Himachal Pradesh, INDIA

CERTIFICATE

Candidate's Declaration

I hereby declare that the work presented in this report entitled Web Application in Java using three-layered structure 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 July 2022 to May 2023 under the supervision of Dr Ekta Gandotra (Associate Professor, Department of Computer Science & Engineering and Information Technology).

I also authenticate that I have carried out the above-mentioned project work under the proficiency stream data science.

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

[illegible]

Parul Sharma,191206

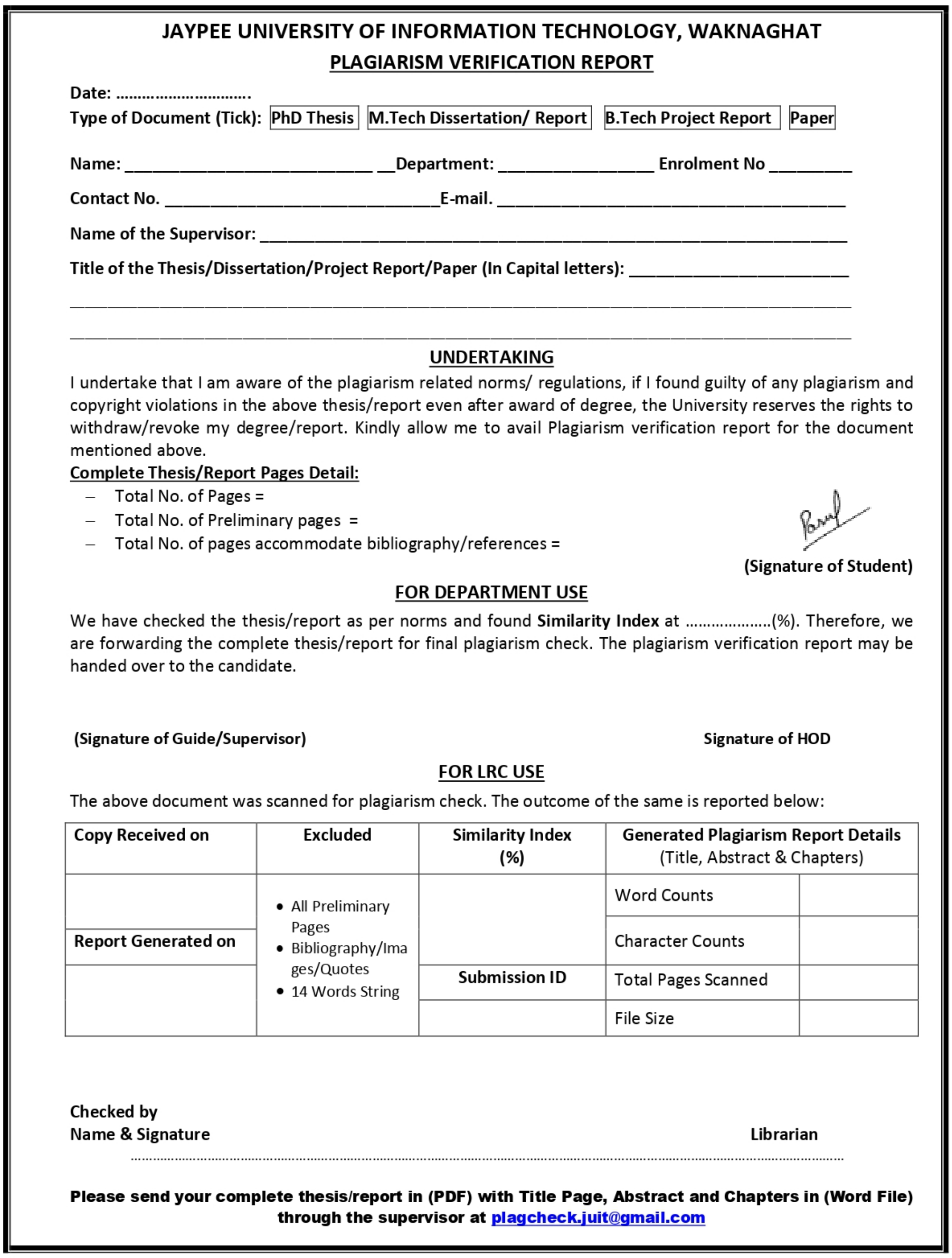
This is to certify that the above statement made by the candidate is true to the best of my knowledge.

Dr Ekta Gandotra
Associate Professor
Department of Computer Science & Engineering and Information Technology

[illegible]

(Signature)
Ujjawal Misra
Director of Engineering
Zopsmart
Dated: 8-May-2023

PLAGIARISM CERTIFICATE



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 =

[illegible]

(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)	
	• All Preliminary Pages • Bibliography/Images/Quotes • 14 Words String		Word Counts	
Report Generated on			Character Counts	
		Submission ID	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

ACKNOWLEDGEMENT

Firstly, I express our heartiest thanks and gratefulness to almighty God for his divine blessing to make it possible to complete the project work successfully. I am grateful and wish my profound indebtedness to Supervisor **Dr Ekta Gandotra**, **Associate Professor**, Department of CSE Jaypee University of Information Technology, Wakhnaghat. The deep knowledge & keen interest of my supervisor in the field of computer science helped me to carry out this project. Her endless patience, scholarly guidance, continual encouragement, constant and energetic supervision, constructive criticism, valuable advice, and reading many inferior drafts and correcting them at all stages have made it possible to complete this project.

I would like to express my heartiest gratitude to **Dr Ekta Gandotra, Department of CSE**, for her kind help to finish my project.

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

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

Parul Sharma (191206)

TABLE OF CONTENT

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

The process of building a web application is very straightforward, but testing, structuring, cleaning, and maintaining the code is difficult. To address this, we use a Three Layered Architecture in Java(SpringBoot).

The three layers are controller, service and dao/repository which are all independent of each other. The handler/controller layer sets the API endpoints and receives the request body and then parses anything that is required from that request. It then calls the service layer where all the logic of the program is defined, ensures that the response is in the required format and writes it to the response writer. This layer further communicates with the datastore/dao layer. It takes whatever it needs from the handler layer and then calls the datastore layer. The datastore layer is where all the data is stored. It can be any data storage. The dao layer is the only layer that communicates with the database. That is how we test each layer(unit testing using Mockito and JUnit) independently making sure that no layer affects the other.

Also after this, the web application was deployed using an AWS EC2 instance.

CHAPTER 1
INTRODUCTION

1.1 Introduction

An internship is a professional learning opportunity that provides an opportunity for students to excel and improve. Students can obtain new skills while exploring and furthering their careers through internships. My current internship at Zopsmart is described in this document. This internship report outlines the steps that helped me achieve a number of my stated objectives. I was given the role of Software Development Engineer Intern for my internship. During the first few months of the internship, we were required to master the Java programming language. I was given a live project after completing this learning procedure.

1.1.1 Job Description

In the application of engineering the goal are the making, implementation, and technical management of software is called software engineering. Software engineering was made to point to the problems that come with software development. When software surpasses timelines, budgets, and quality standards, issues occur. It makes certain that the software is created consistently, accurately, on schedule, within budget, and in accordance with the requirements. Software engineering has grown in importance in order to keep up with the users' continually changing needs and the environment in which the programme is anticipated to operate.

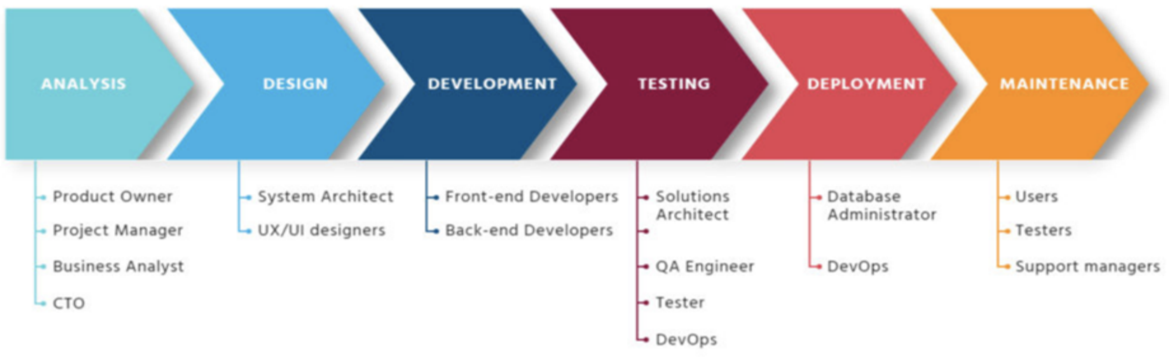


Fig 1: Software development cycle

Assisting with software design and development is the responsibility of the Software Engineer Trainee. They work together with other members of the team to develop secure and reliable software. Application development (code, programming), code debugging, and testing are the activities and responsibilities of this role. New software programmes, as well as detecting and addressing a variety of technical issues. Partnering with senior executives to identify issues and propose solutions.

1.1.2 Company

Modern retail technology provider ZopSmart offers all the resources needed to launch an online store. ZopSmart provides a set of tools that can help you get there quickly and effectively whether you're a conventional store trying to develop an omnichannel business or an online-only shop looking to grow an e-commerce business. E-commerce has been increasingly popular over time. And it is clear that the market for online stores offers a variety of possibilities. Let's say you're a shop looking to launch your own online business. ZopSmart can help companies create a strong foundation for their online growth using the many technologies at their disposal. Many suppliers can use the goods that ZopSmart offers to start their own online retail businesses. Additionally, there are several ways to make the experience better, such as installing a self-checkout system. According to the NRF 2020 consumer survey, more than 83% of shoppers believe convenience is more essential now than it was five years ago. Before we even understood what the phrase "coronavirus" meant, this survey was done in October 2019. Since shutdowns and social isolation have become the norm, convenience is no longer a "nice to have," but rather a "must have" quality. Online usage of Covid-19 has increased, and according to recent NRF polls, demand will increase even after the bulk of the population has received the vaccine.

Fig 2: Zopsmart Logo

1.1.3 Programming Language: Java

What is Java technology, and why do I need it?

Java is a computer platform for application development as well as an object-oriented, class-based, and concurrent programming language, which implies that numerous statements may be processed concurrently rather than sequentially. It is free to use and runs on all platforms.

Most programming languages need you to compile or interpret a programme before running it on your computer. The Java programming language is unique in that it compiles and interprets programmes. The compiler initially converts a programme into an intermediate language known as Java bytecodes, which are platform-independent codes processed by the Java platform's interpreter. On the computer, the interpreter parses and executes each Java bytecode command.

1.2 Problem Statement

To create a basic SpringBoot web application that implements CRUD operations on products and categories based on the three-layered architecture. Programs at each layer have their own unit test.

1.3 Objectives

The goal of the internship is to learn more about programming languages like Java and to create testable, structured, clean and maintainable web applications by using industry best practices. After the training, I was assigned to work on a real-world project i.e. Training and Upskilling for the company. This internship provides students with hands-on experience in order for them to gain a better understanding of the sector. The goal was to gain enough knowledge to be able to work efficiently on the project that had been assigned to us.

1.4 Motivation

To apply industry best practices and create a fast, scalable and secure web application.

1.5 Methodology

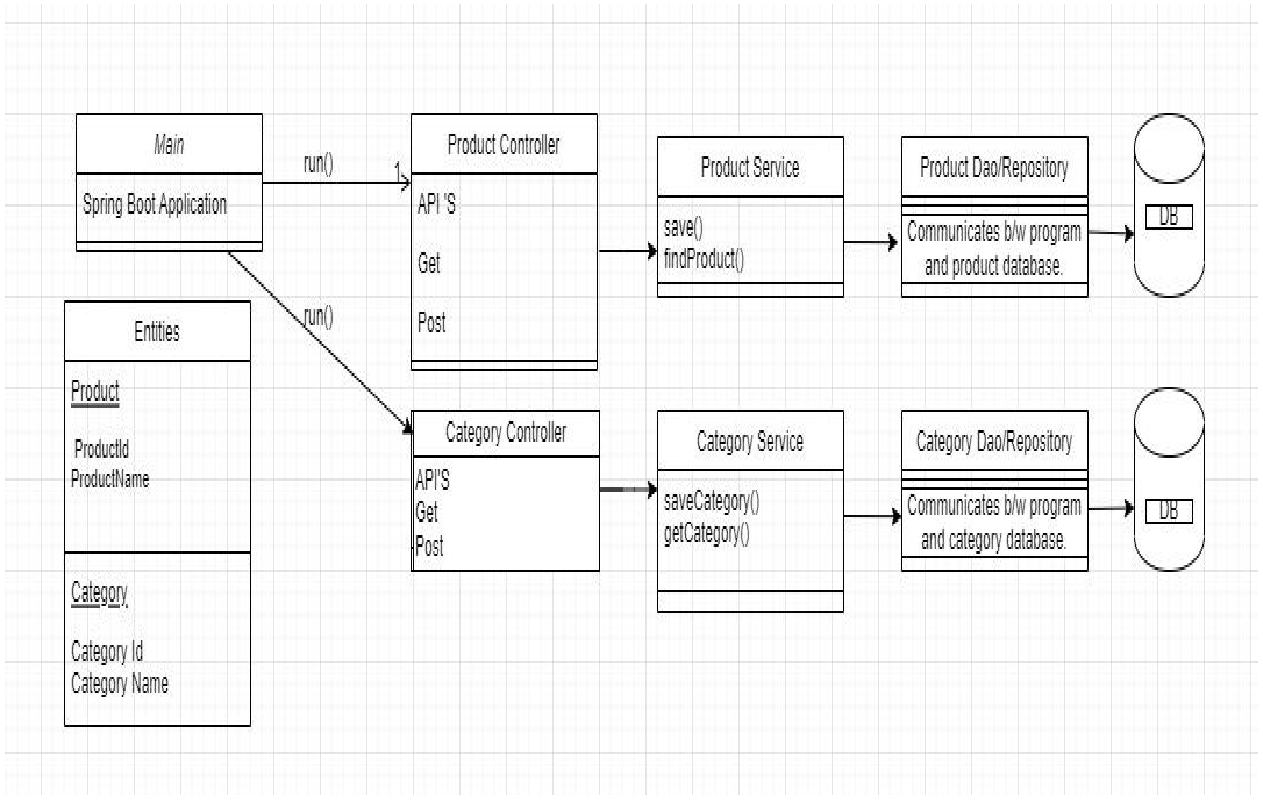


Fig 3:Spring Boot Application Methodology

Layered Architecture

Layers are independent of each other and communicate with each other through an interface. Basically, this helps us make our application modular, readable and maintainable.

This has 3 layers- the HTTP layer, the Service layer, and the Dao layer.

- **HTTP layer/controller:** validates query/path parameters, request body, and header checks.
- **Service layer:** Implements business logic and communicates with the dao layer.
- **Dao layer:** Implements database-level queries.

Each layer communicates with its previous/next layer using an interface(methods with input parameters and output types are defined). Testing of each layer is done by mocking its interface/DB/Server based on necessity.

1.6 Organisation

The remainder of the report is organised as follows: The literature survey on Java and other documentation is summarised in Chapter 2. The methodology and system development is described in Chapter 3. The experimental results and performance analysis based on the unit testing are shown in Chapter 4. The paper's conclusion and future scope are provided in Chapter 5.

Chapter-2

LITERATURE SURVEY

Popular Java framework Spring Boot offers a streamlined, opinionated approach to application development, making it easier to create web applications. We will examine the current state of research on Spring Boot web applications and talk about the key discoveries and contributions made in this field in this literature review.

Overview of Spring Boot Web Applications:

A web framework called Spring Boot, which is built on top of the Spring framework, aims to make the development of web applications simpler. Spring Boot applications are simple to build and require little configuration. The framework is well-liked by developers because it offers a variety of features like auto-configuration, embedded web servers, and support for different data sources.

1. Performance: Several studies have looked into how well Spring Boot web apps run. Overall, the outcomes have been encouraging, with studies claiming that Spring Boot apps perform on par with or even better than those built using alternative Java web frameworks. The performance of Spring Boot, Play, and Vert. x frameworks, for instance, was examined in a study by **Xu et al.** [1] and it was discovered that Spring Boot performed the best in terms of throughput and reaction time..

2. Security: Numerous research has looked into the security of Spring Boot apps because security is a crucial component of developing online applications. The studies have concentrated on a number of security-related topics, including secure coding techniques, authentication, and authorization. For instance, **Baker et al**.[2] examined the security flaws in Spring Boot apps and discovered that many of them were susceptible to popular attacks like SQL injection and cross-site scripting (XSS).

3. Development Process: Numerous research has also examined the creation of Spring Boot web apps. The ability of Spring Boot to expedite and simplify the development process is one of its main advantages. For instance, in research by **Aris et al.** [3], which created a backend application for a public complaint system and analysed the effects of Spring Boot on the development process, it was discovered that the time and effort needed to create web apps was decreased. The study noted that Spring Boot simplified the implementation of a microservices architecture.

4. Testing: Any online application's development process must include testing, and Spring Boot has various capabilities that make testing simpler. With an emphasis on unit testing, integration testing, and performance testing, several studies have examined the testing of Spring Boot applications. For instance, **Debnath et al.** [4] study examined the usage of the JUnit and Mockito frameworks for testing Spring Boot apps and discovered that they were successful in locating and correcting errors.

Popular web framework Spring Boot offers a number of features that make and speed up the creation of online applications. The performance, security, development process, and testing of the framework have all been the focus of several studies. The research's generally encouraging findings demonstrate Spring Boot's advantages for developing web applications.

Author(s)	**Title**	**Year**	**Methodology**	**Drawbacks**
Xu, L., Li, M., Chen, W., & Liu, X.	A Comparative Study on Performance of Web Application Frameworks for Java. Journal of Physics	2018	The goal of the study was to examine the effectiveness of Spring Boot, Play, and Vert. x, three Java web application frameworks. The authors used a benchmarking tool to gauge each framework's reaction time, throughput, and memory use. Vert. x had the best throughput and fastest reaction time, according to the data, whereas Spring Boot used the least memory.	The study excluded other well-known frameworks like JavaServer Faces (JSF), Struts, and JavaServer Pages (JSP) and only examined three web application frameworks: Spring Boot, Play, and Vert. x. The findings might not thus be transferable to other frameworks. The authors used a particular hardware and software setup to test the frameworks.
Oras Baker Quy Nguyen's	A Novel Approach to Secure Microservice Architecture from OWASP Vulnerabilities	2019	This is a guide on utilising Spring Framework and Spring Security Framework to develop security and design in microservices	The study may concentrate on certain flaws or attack patterns, which might not be an accurate representation of all possible dangers to microservice architectures. The suggested method might not be

				appropriate in all situations and might only be relevant to specific kinds of microservice architectures.
Hatma Suryotrisongko, Dedy Puji Jayanto, Aris Tjahyanto	Design and Development of Backend Application for Public Complaint Systems Using Microservice Spring Boot	2017	to create a web application using the spring-boot microservice architecture for a public complaint service.	More functionalities could be added to the website.
Debnath, S., Saha, R., & Sarkar	Empirical Analysis of JUnit and Mockito Frameworks for Java Applications Testing	2019	The study showed that unit testing for Java applications may be made more successful by combining JUnit with Mockito.	Tests took longer to execute thanks to Mockito. The authors countered that the advantages of better code coverage and fewer test cases exceeded the drawbacks of longer execution times.

Table 1: Comparison of Related Works

Chapter-3

SYSTEM DEVELOPMENT

3.1 Libraries/Frameworks Used

Java-It was created in 1995 by Sun Microsystems, a company that is now an Oracle subsidiary.

Java is both a platform and a programming language. Programming with Java is high-level, reliable, object-oriented, and secure.

SpringBoot-The Spring Framework serves as the foundation for the project known as Spring Boot. It offers a quicker and simpler way to install, set up, and execute both straightforward and web-based applications.

Postgres-A free and open-source relational database management system that emphasises flexibility and SQL compliance is PostgreSQL, sometimes referred to as Postgres.

Unit Testing:

Mockito-An open source Java testing framework known as Mockito was made available under the MIT Licence. For test-driven development or behaviour-driven development, the framework supports the generation of test double objects in automated unit tests.

JUnit- The Java programming language has a framework for unit testing called JUnit. Test-driven programming has benefited from JUnit.

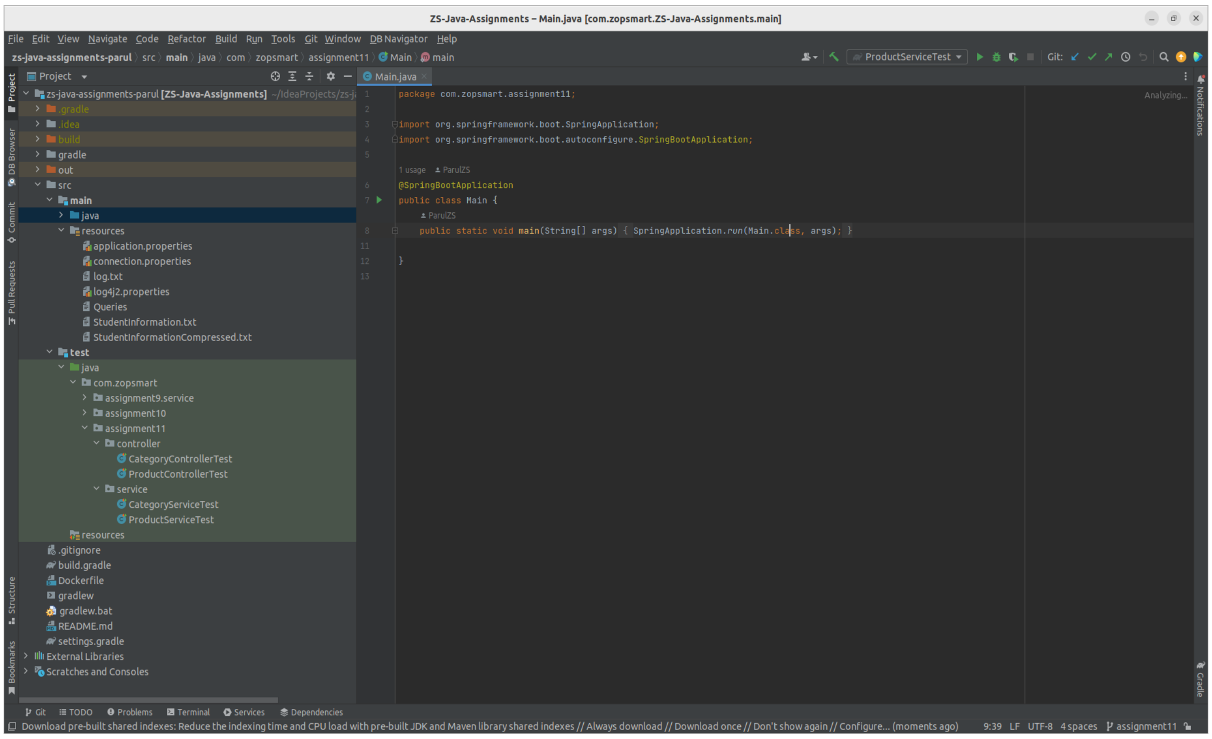
Swagger-Users may create, document, test, and consume RESTful web services with the aid of Swagger. It is applicable to both top-down and bottom-up API development methodologies.

MOVING FORWARD WITH JAVA

All the backend framework such as implementing HTTP request, sending a response to the server, writing program logic etc is written in Java.

3.2 Technical Requirements

- **IntelliJ** An integrated development environment (IDE) for creating software written in Java, Kotlin, Groovy, and other JVM-based languages is called IntelliJ IDEA. It is created by JetBrains (formerly IntelliJ), and it comes in both a proprietary commercial edition and an Apache 2 Licenced community edition. Both are applicable to the development of businesses.


ZS-Java-Assignments – Main.java [com.zopsmart.ZS-Java-Assignments.main]

```java
package com.zopsmart.assignment11;

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

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

}
```

Download pre-built shared indexes: Reduce the indexing time and CPU load with pre-built JDK and Maven library shared indexes // Always download // Download once // Don't show again // Configure... (moments ago)

Fig 4: IntelliJ Idea

- **Swagger** An open-source project called Swagger2 is used to provide the REST API documentation for RESTful web services. It offers a user interface for web browser access to our RESTful web services. You must include the following requirements in our build settings file in order to make Swagger2 available in a Spring Boot application.

- **Postgres** An effective open-source object-relational database system is PostgreSQL. It has been in active development for more than 15 years, and because of its tried-and-true design, it enjoys a solid reputation for dependability, data integrity, and accuracy.A database system with more than 30 years of ongoing development is PostgreSQL. Numerous data kinds, such as numbers, texts, dates, timestamps, and binary objects are supported. Additionally, user-defined functions and stored procedures are supported by PostgreSQL. With its scalability, PostgreSQL is a widely-liked option for business applications as well as for online applications. People frequently turn to PostgreSQL to handle challenging, large-scale data processing. This is due to the fact that PostgreSQL handles unusual database conditions better than MySQL when comparing the two databases. Compared to other database management systems, PostgreSQL offers greater functionalities. Additionally, because PostgreSQL is catalogue-driven in how it operates, it is extendable. In other words, it allows you to create data types and index types in addition to storing information about tables and columns.

Fig 5:PostgreSQL Logo

-GitHub Version control facilitates the management and tracking of coding changes in software projects. Version management is crucial when a software project expands.A developer replicates a portion of the source code (also known as the repository) while branching. The developer may then safely modify that section of the code without jeopardising the project's overall success. The developer can then integrate that code back into the primary source code to make it official after getting their particular portion of the code to perform properly. Then, all of these modifications are monitored and, if necessary, may be undone. Linus Torvalds developed the open-source version management system known as Git in 2005.

Git is a distributed version control system, which implies that every developer's computer has access to the whole codebase and history, making branching and merging simple. A for-profit organisation called **GitHub** provides a service for hosting Git repositories on the cloud. In essence, it makes it simpler for both individuals and teams to utilise Git for collaboration and version control. Because of GitHub's user-friendly design, even newbie programmers may benefit from Git. Without GitHub, utilising Git often necessitates a little more command-line experience and technical know-how.

3.2.1 Hardware Configuration

Processor: Intel Core i5-10310U CPU
RAM:16 GB
Hard Disk: 512 GB SSD
Monitor 13”
Mouse
Keyboard

Table 2: Hardware Configuration

3.2.2 Software Configuration

Operating System Ubuntu
Language Java
Runtime Environment JRE

Table 3: Software Configuration

3.3 Model Development

The domain model for the application must be created and defined for a Spring Boot web application. The entities and connections that make up the business logic of the application are represented by the domain model.

The processes for creating the model for a Spring Boot web application are listed below:

1. **Identify the entities**: Find the entities that make up the business logic of the application. The entities in an e-commerce platform, for instance, may include customers, orders, items, etc.

2. **Define the relationships**: Define the connections between the entities when you've recognised them. For instance, a consumer may place several orders and each purchase may include numerous goods.

3. **Create the entity classes**: Make the corresponding Java entity classes based on the recognisable entities and the connections between them. The fields, constructors, and methods required to represent the entities should be included in these classes.

4. **Define the repository interfaces:** Define the repository interfaces that will be applied to the entity class CRUD operations. These interfaces ought to build upon Spring Data JPA's JpaRepository interface.

5. **Implement the repository interfaces:** Apply Spring Data JPA to the repository interface implementation. As a result, you will be able to use straightforward method calls to carry out CRUD actions on the entity classes.

6. **Define the service interfaces**: Establish the service interfaces that will be utilised to implement the application's business logic. These interfaces should specify the procedures to be followed when interacting with the repositories and carrying out the required tasks.

7. **Implement the service interfaces:** Utilise the entity classes and repository interfaces to implement the service interfaces. You will then be able to implement the application's business logic.

8. **Define the controller classes:** Create the controller classes that will manage incoming HTTP requests, then call the relevant service methods to carry out the required tasks.

9. **Implement the controller classes**: Utilise the service interfaces to implement the controller classes. By doing so, you'll be able to manage incoming HTTP requests and deliver the necessary answers.

3.3.1 Algorithm

➔ **For Category API**

1. Create a Category class and fill it with necessary parameters like id and name.
2. To perform CRUD operations on categories, create a CategoryRepository interface that extends JpaRepositoryProduct, Long>.
3. Create a class called CategoryService to house all of the program's logic and allow it to communicate with the CategoryRepository.
4. Create a class called CategoryController to manage incoming requests and pass them on to CategoryService.
5. To retrieve all categories, define a method in the CategoryController.
6. In the ProductController, define a method for adding new categories.
7. Set up Spring Boot to utilise a suitable database, such as PostgreSQL.
8. Launch the application, then test the endpoints using a Postman or Swagger.

➔ **For Product API**

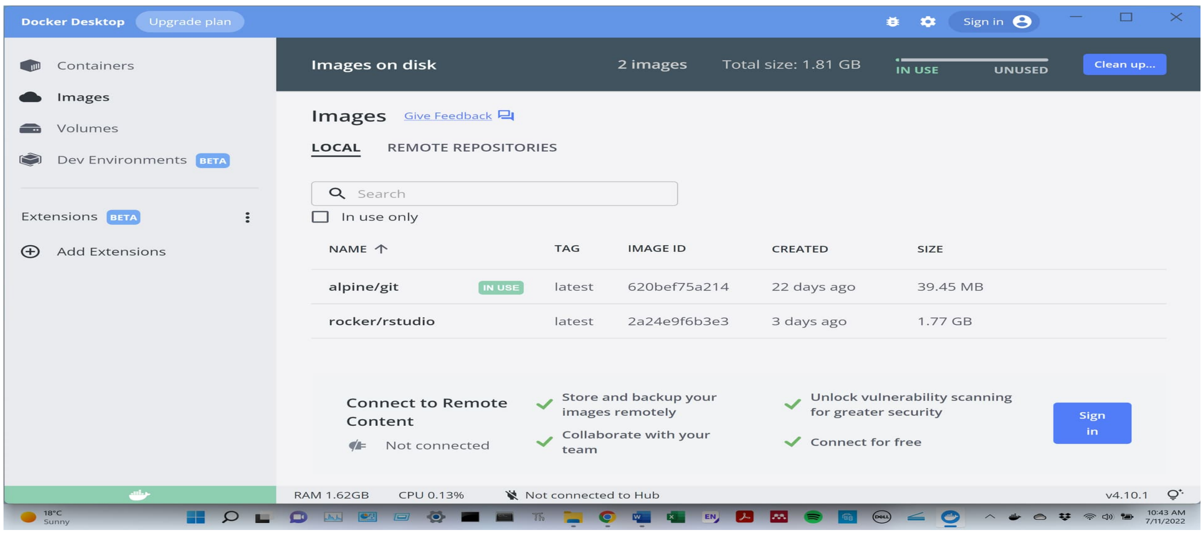
1. Create a Product class using Hibernate and populate it with pertinent fields like id, name, and category id that are mapped from the Category Table(Many to One).
2. To support CRUD operations on products, create a ProductRepository interface that extends JpaRepository.
3. Create a class called ProductService to house all of the program's logic and allow it to communicate with the ProductRepository.
4. Create a class called ProductController to manage incoming requests and give them off to the ProductService.
5. To retrieve every product, define a method in the ProductController.
6. In the ProductController, define a method for adding new products.
7. Create a product update function in the ProductController.
8. Call the service routines using the necessary logic.
9. Set up Spring Boot to utilise a suitable database, such as PostgreSQL.

10. Launch the application, then test the endpoints using a Postman or Swagger-compatible tool.

3.3.2 Deployment

Docker:

An open platform for creating, distributing, and executing programmes is Docker. You may divide your apps from your infrastructure with the help of Docker, allowing for rapid software delivery. You can manage your infrastructure using Docker in the same manner that you manage your apps. You may drastically shorten the time between developing code and executing it in production by utilising Docker's methodology for shipping, testing, and deploying code rapidly. The ability to bundle and operate a programme in a loosely separated environment known as a container is provided by Docker. You may execute several containers concurrently on a single host thanks to the isolation and security. You can execute applications without depending on what is already installed on the host because containers are small and come with everything you need to run them. Sharing containers while working is simple, and you can make sure that everyone you share with receives the same container that operates in the same way.



NAME		TAG	IMAGE ID	CREATED	SIZE
alpine/git	IN USE	latest	620bef75a214	22 days ago	39.45 MB
rocker/rstudio		latest	2a24e9f6b3e3	3 days ago	1.77 GB

Fig 6: Docker Desktop

One typical method for executing and maintaining apps in a cloud environment is to deploy a Docker container on an **AWS EC2 instance**. Here is a quick rundown of the phases in this procedure:

1. **Create an AWS EC2 instance:** For this, the proper EC2 instance type must be chosen, the instance must be created and configured, and the security groups and key pairs must be set up.

2. **Install Docker on the EC2 instance:** A technology called Docker makes it possible to create, transport, and operate programmes inside containers. You may either download and manually install Docker on the EC2 instance using the package management.

3. **Build the Docker container image:** A Dockerfile, or script containing instructions for constructing the Docker image, must be made. The base image must be specified, the application code must be added, and any relevant customizations must be set up.

4. **Push the Docker image to a container registry:** The Docker image has to be uploaded to a container registry before you can deploy it to the EC2 instance. Popular container registries include Docker Hub, Amazon ECR, and Google Container Registry.

5. **Pull the Docker image on the EC2 instance:** You may use the Docker CLI to pull the Docker image onto the EC2 instance after it is made accessible in the container registry.

6. **Run the Docker container:** The "docker run" command may then be used to launch the Docker container on the EC2 instance while supplying any required environment variables and port mappings.

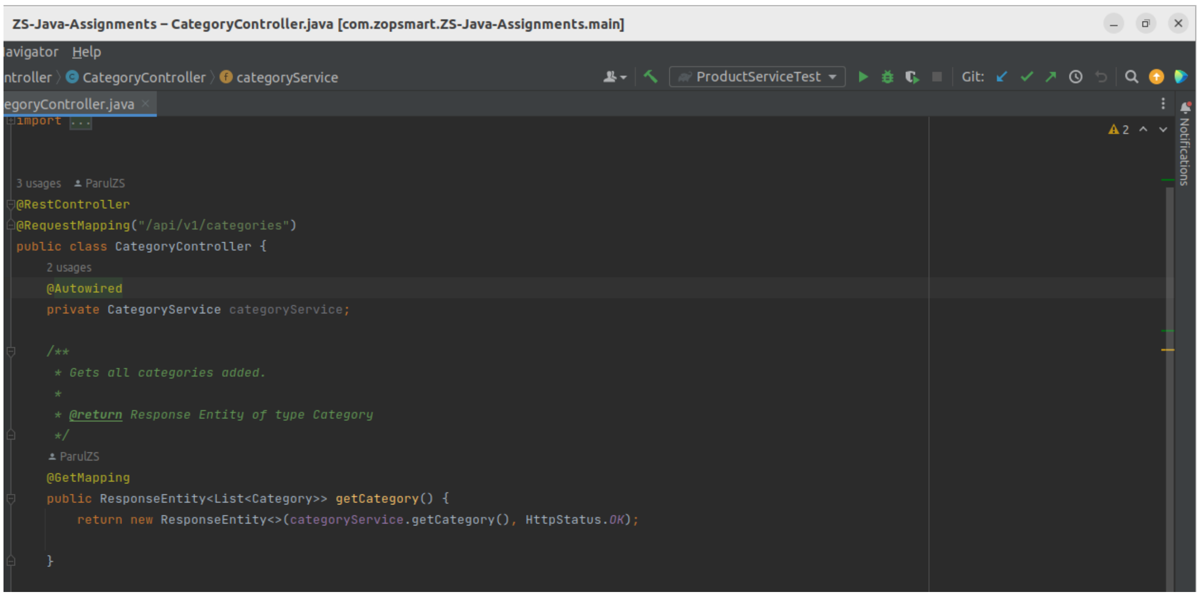
Chapter-4
EXPERIMENTS & RESULT ANALYSIS

4.1 API Requests:

Since they are now an integral part of daily life, APIs are used in practically all of our online activities. When a developer calls the server by adding an endpoint to a URL, an API request is made.The points of contact where an API interacts with a different system are referred to as API endpoints. An endpoint identifies the place from which an API may access the resources it requires. A server must be contacted for information before an API may begin to function. An endpoint is the channel of communication that APIs utilise to transmit a request and describe the location of a specific resource. It is crucial in specifying the precise locations of resources that may be accessed and in ensuring the smooth operation of any software that interacts with it.

Different types of HTTP requests:

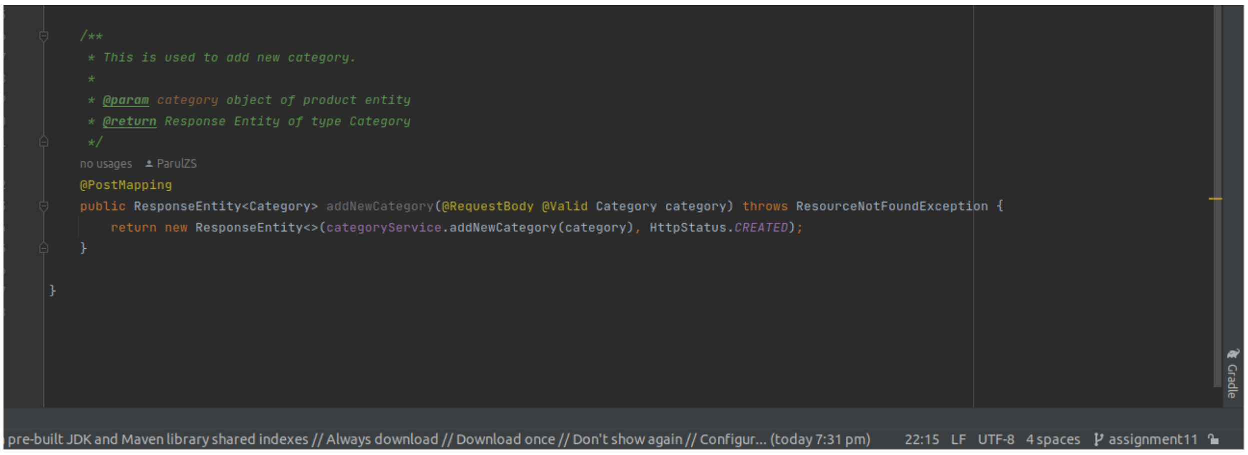
GET Request: Data from a server is retrieved using it. The server responds to a GET request by returning the desired information in the response body.


ZS-Java-Assignments – CategoryController.java [com.zopsmart.ZS-Java-Assignments.main]
avigator Help
ntroller 〉 CategoryController 〉 categoryService
ProductServiceTest
Git:
egoryController.java
Notifications

```
import ...


3 usages  ParulZS
@RestController
@RequestMapping("/api/v1/categories")
public class CategoryController {
    2 usages
    @Autowired
    private CategoryService categoryService;

    /**
     * Gets all categories added.
     *
     * @return Response Entity of type Category
     */
    ParulZS
    @GetMapping
    public ResponseEntity<List<Category>> getCategory() {
        return new ResponseEntity<>(categoryService.getCategory(), HttpStatus.OK);

    }
```

Fig 7: Get Request

POST Request: In the request body of a POST request, the client delivers information that the server can utilise to build or update resources.



```
    /**
     * This is used to add new category.
     *
     * @param category object of product entity
     * @return Response Entity of type Category
     */
    no usages  ParulZS
    @PostMapping
    public ResponseEntity<Category> addNewCategory(@RequestBody @Valid Category category) throws ResourceNotFoundException {
        return new ResponseEntity<>(categoryService.addNewCategory(category), HttpStatus.CREATED);
    }

}
```

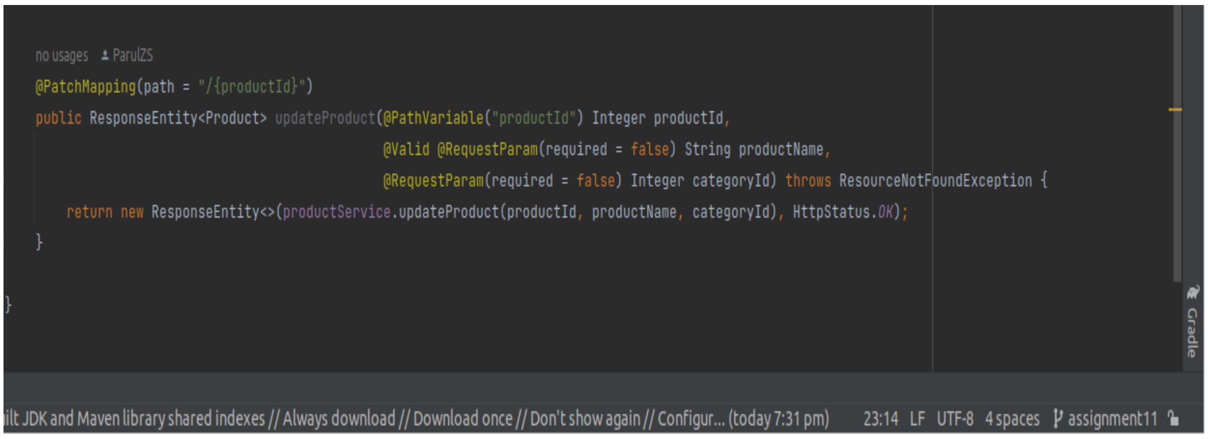
pre-built JDK and Maven library shared indexes // Always download // Download once // Don't show again // Configur... (today 7:31 pm)   22:15 LF UTF-8 4 spaces assignment11

Fig 8: POST Request

PUT Request: In the request body of a PUT request, the client delivers information that the server can utilise to update the requested resource.

DELETE Request: A DELETE request instructs the server to remove the given resource when it is made by a client.

PATCH Request: In a PATCH request, the client transmits data that the server can utilise to change a portion of the requested resource. When only a few fields of a resource need to be updated rather than the full resource, this is helpful.

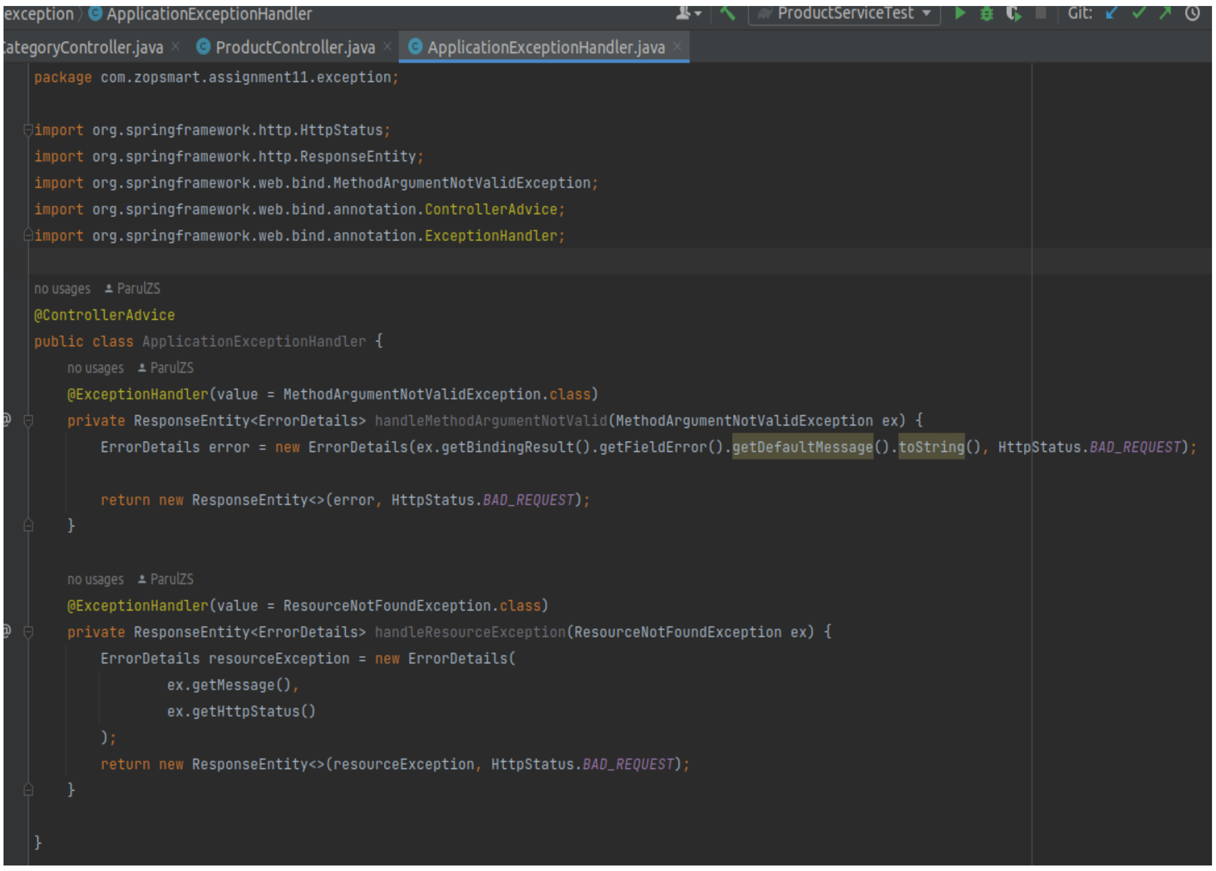


```
    no usages  ParulZS
    @PatchMapping(path = "/{productId}")
    public ResponseEntity<Product> updateProduct(@PathVariable("productId") Integer productId,
                                                 @Valid @RequestParam(required = false) String productName,
                                                 @RequestParam(required = false) Integer categoryId) throws ResourceNotFoundException {
        return new ResponseEntity<>(productService.updateProduct(productId, productName, categoryId), HttpStatus.OK);
    }

}
```

ilt JDK and Maven library shared indexes // Always download // Download once // Don't show again // Configur... (today 7:31 pm)   23:14 LF UTF-8 4 spaces assignment11

Fig 9: Patch Request

4.2 Exception Handling:



exception / ApplicationExceptionHandler

ProductServiceTest Git:

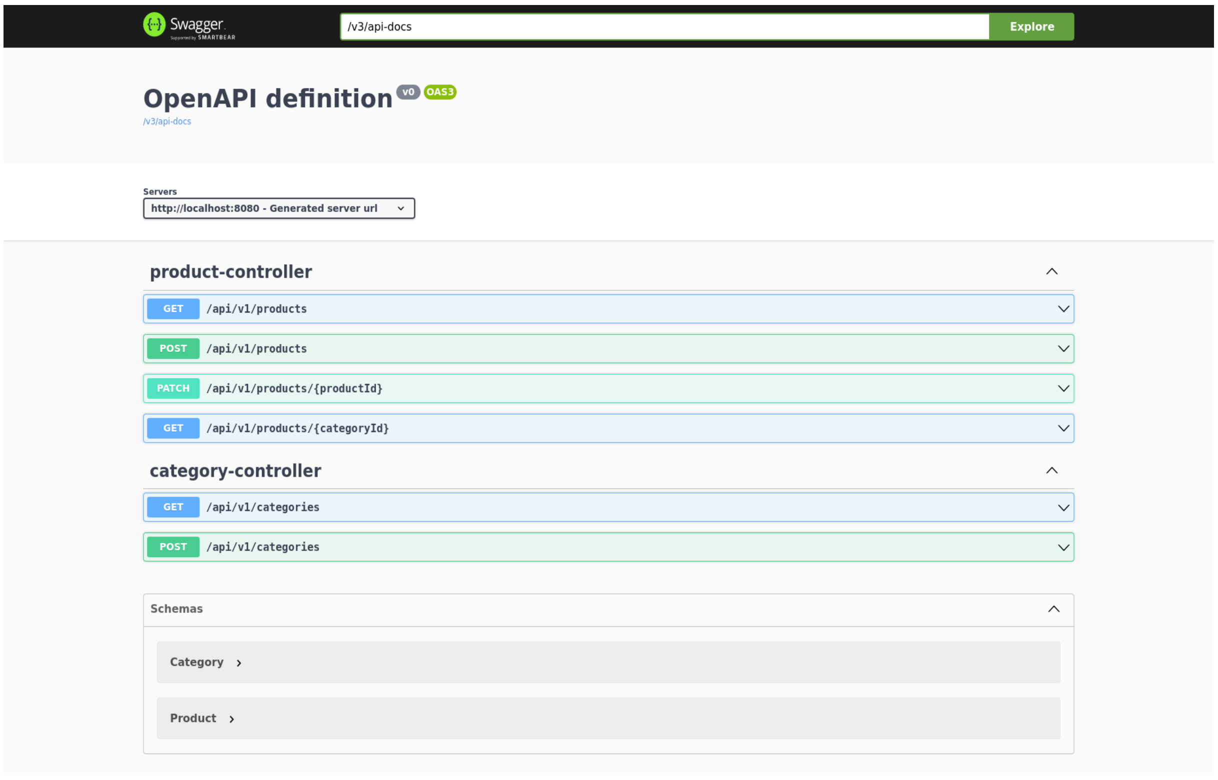
ategoryController.java × ProductController.java × ApplicationExceptionHandler.java ×

```java
package com.zopsmart.assignment11.exception;

import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.MethodArgumentNotValidException;
import org.springframework.web.bind.annotation.ControllerAdvice;
import org.springframework.web.bind.annotation.ExceptionHandler;

no usages  ParulZS
@ControllerAdvice
public class ApplicationExceptionHandler {
    no usages  ParulZS
    @ExceptionHandler(value = MethodArgumentNotValidException.class)
    private ResponseEntity<ErrorDetails> handleMethodArgumentNotValid(MethodArgumentNotValidException ex) {
        ErrorDetails error = new ErrorDetails(ex.getBindingResult().getFieldError().getDefaultMessage().toString(), HttpStatus.BAD_REQUEST);

        return new ResponseEntity<>(error, HttpStatus.BAD_REQUEST);
    }

    no usages  ParulZS
    @ExceptionHandler(value = ResourceNotFoundException.class)
    private ResponseEntity<ErrorDetails> handleResourceException(ResourceNotFoundException ex) {
        ErrorDetails resourceException = new ErrorDetails(
                ex.getMessage(),
                ex.getHttpStatus()
        );
        return new ResponseEntity<>(resourceException, HttpStatus.BAD_REQUEST);
    }

}
```

Fig 10: Exception Handling

We have a great deal of freedom when it comes to managing exceptions thanks to the @ExceptionHandler annotation. To begin with, all we have to do to utilise it is to annotate a method with the @ExceptionHandler annotation, either in the controller itself or in a @ControllerAdvice class:

4.3 Swagger



/v3/api-docs

OpenAPI definition

/v3/api-docs

Servers

http://localhost:8080 - Generated server url

product-controller

GET /api/v1/products

POST /api/v1/products

PATCH /api/v1/products/{productId}

GET /api/v1/products/{categoryId}

category-controller

GET /api/v1/categories

POST /api/v1/categories

Schemas

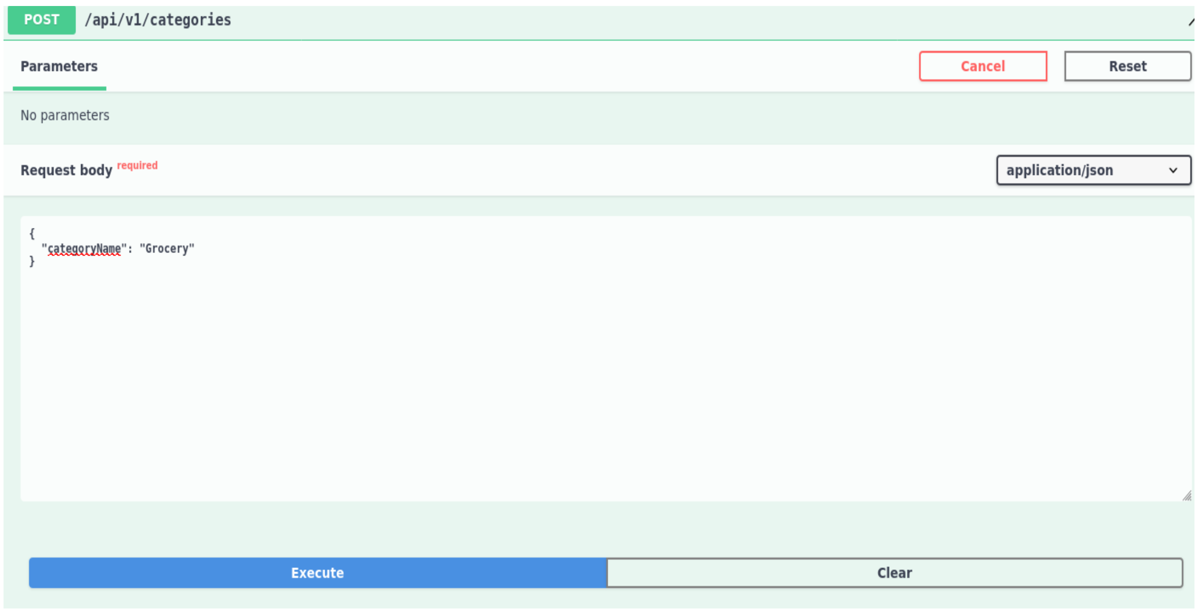
Category

Product

Fig 11: Swagger

Category API

Create a Category



POST /api/v1/categories

Parameters

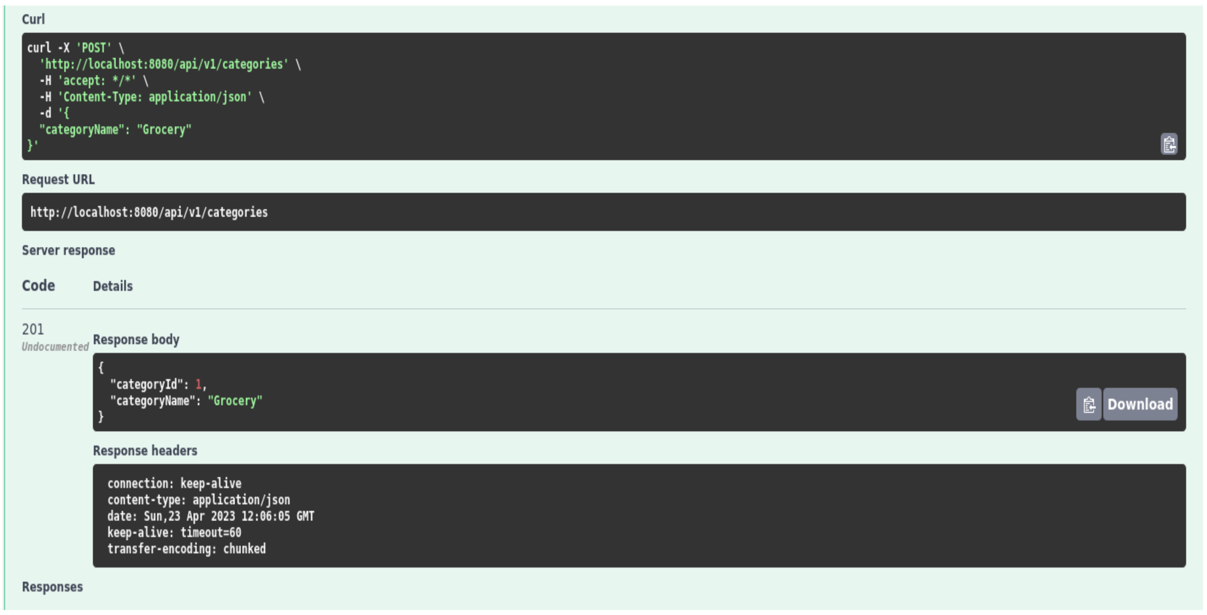
No parameters

Request body required

application/json

```
{
  "categoryName": "Grocery"
}
```

Fig 12: Create Category (Post Request)


Curl

```
curl -X 'POST' \
  'http://localhost:8080/api/v1/categories' \
  -H 'accept: */*' \
  -H 'Content-Type: application/json' \
  -d '{
  "categoryName": "Grocery"
}'
```

Request URL

```
http://localhost:8080/api/v1/categories
```

Server response

Code Details

201 Undocumented

Response body

```
{
  "categoryId": 1,
  "categoryName": "Grocery"
}
```

Response headers

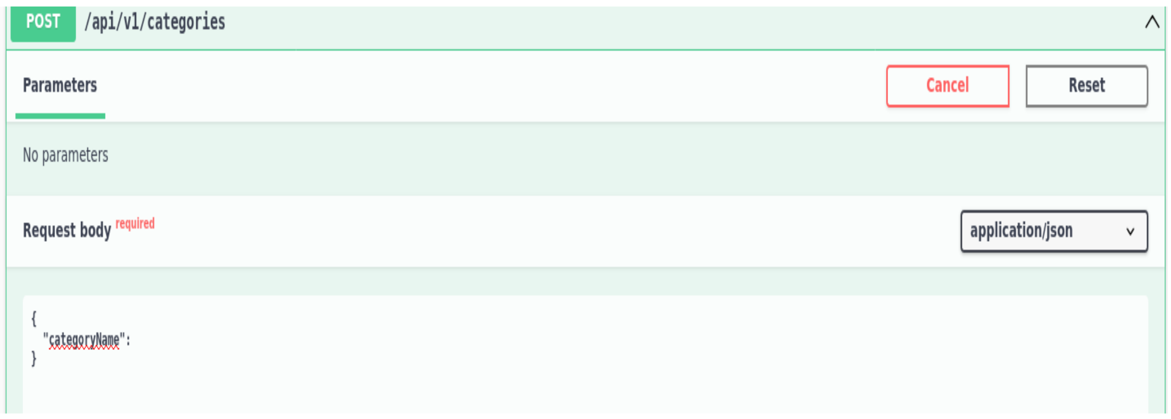
```
connection: keep-alive
content-type: application/json
date: Sun,23 Apr 2023 12:06:05 GMT
keep-alive: timeout=60
transfer-encoding: chunked
```

Responses

Fig 13: Output For Create Category

Edge Cases:

1. If the category name in the request body is empty or null.


POST /api/v1/categories

Parameters

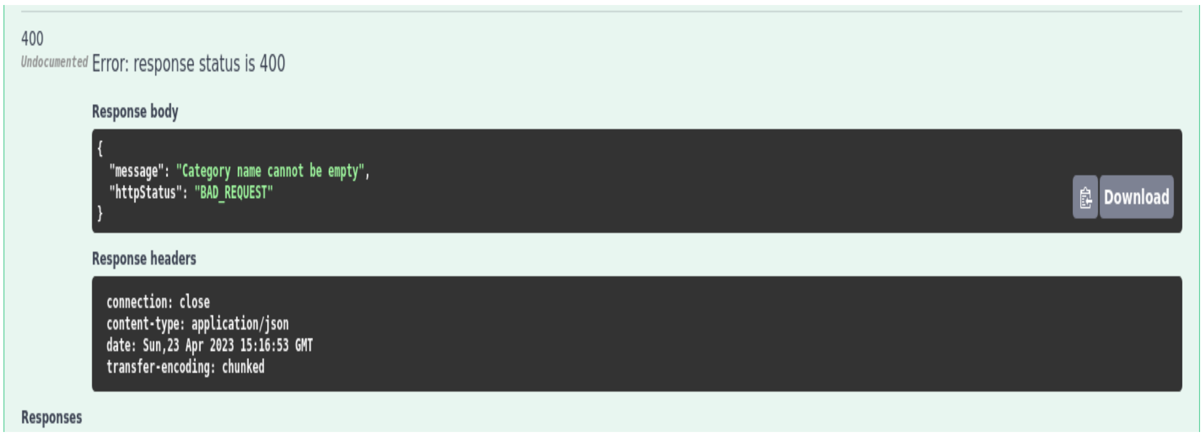
No parameters

Request body required

application/json

```
{
  "categoryName":
}
```

Fig 14: Edge Case (Category name is null for creating category)


400
Undocumented Error: response status is 400

Response body

```
{
  "message": "Category name cannot be empty",
  "httpStatus": "BAD_REQUEST"
}
```

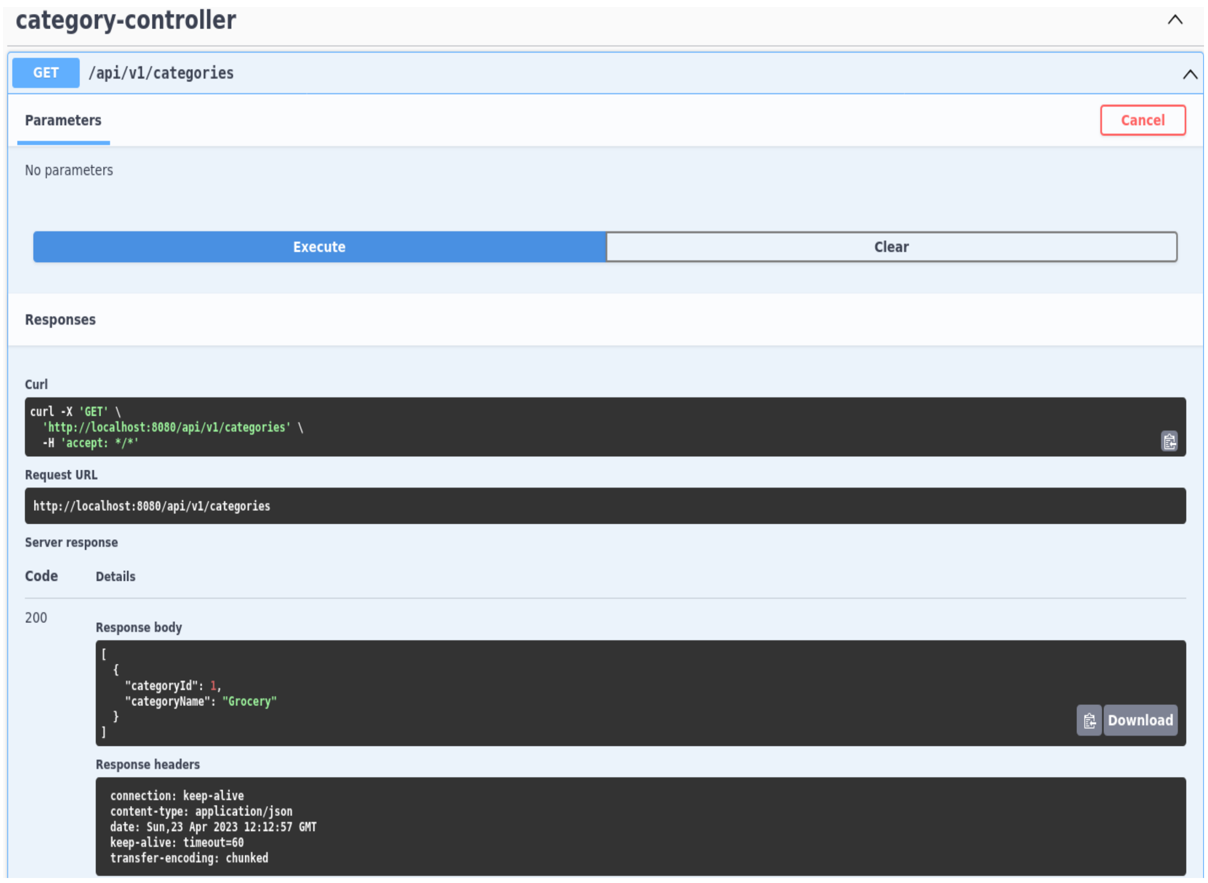
Response headers

```
connection: close
content-type: application/json
date: Sun,23 Apr 2023 15:16:53 GMT
transfer-encoding: chunked
```

Responses

Fig 15: Output Edge Case (Category name is null for creating category)

Get All Categories


category-controller

GET /api/v1/categories

Parameters

No parameters

Responses

Curl

```
curl -X 'GET' \
  'http://localhost:8080/api/v1/categories' \
  -H 'accept: */*'
```

Request URL

```
http://localhost:8080/api/v1/categories
```

Server response

Code Details

200

Response body

```
[
  {
    "categoryId": 1,
    "categoryName": "Grocery"
  }
]
```

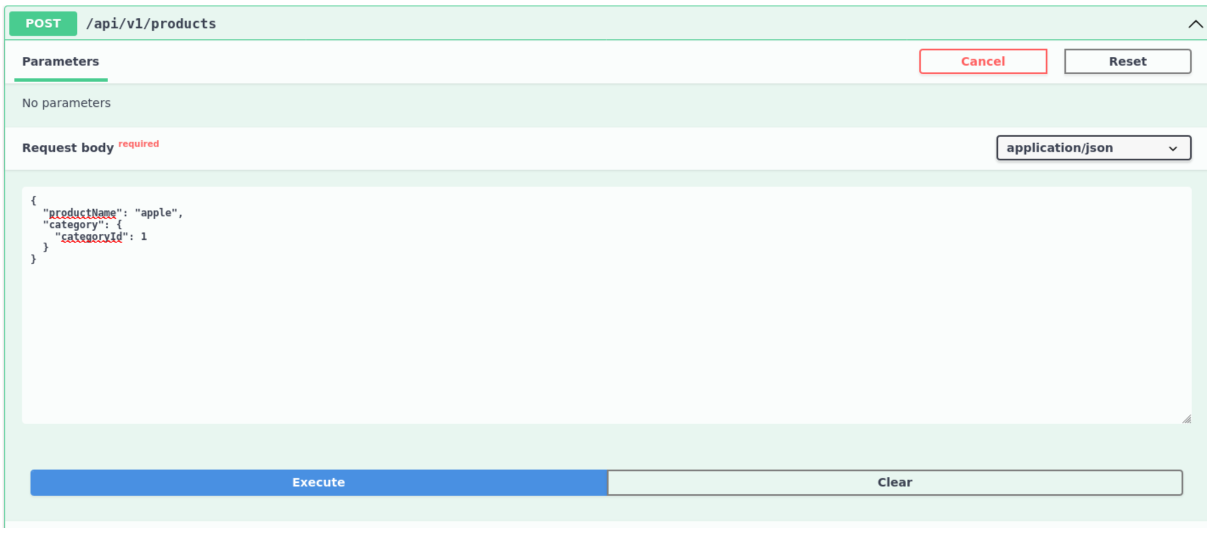
Response headers

```
connection: keep-alive
content-type: application/json
date: Sun,23 Apr 2023 12:12:57 GMT
keep-alive: timeout=60
transfer-encoding: chunked
```

Fig 16: Get All Categories (Get Request)

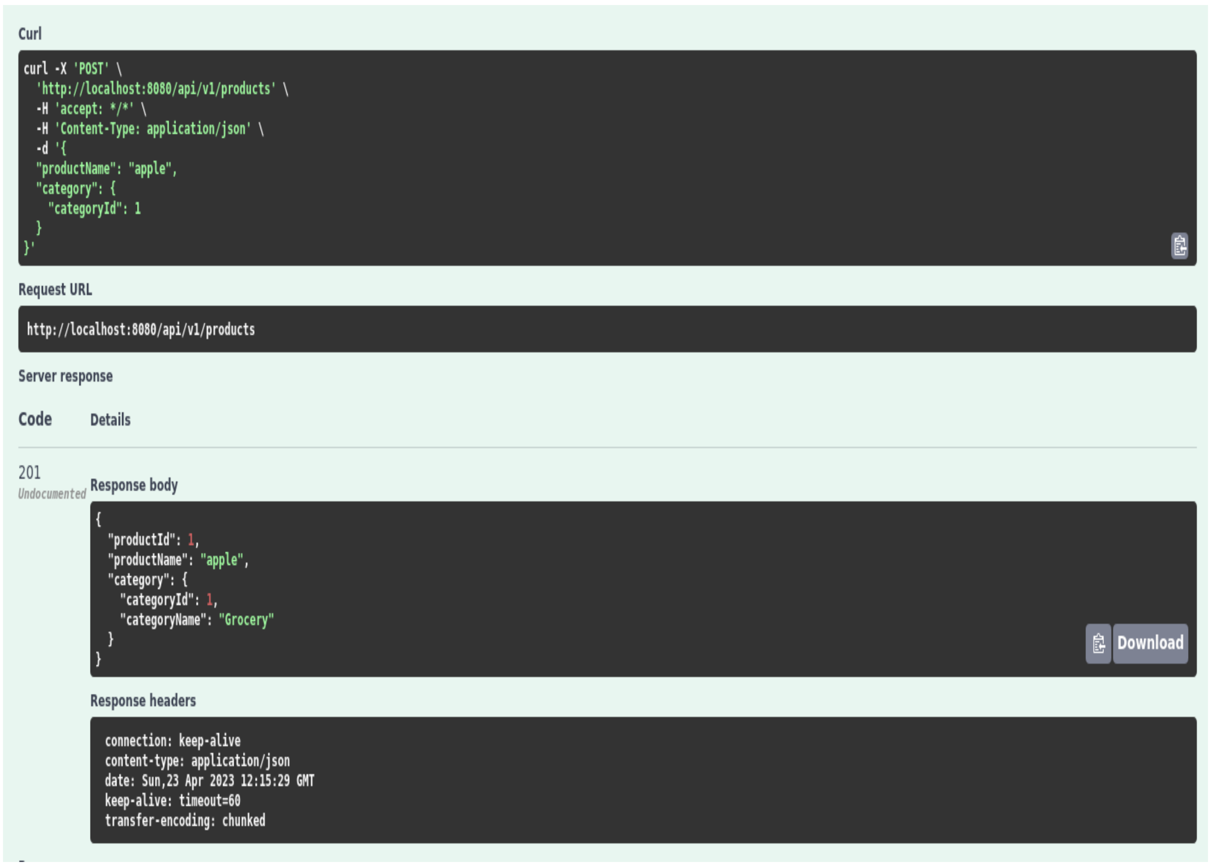
Products API

Create a new product



```
{
  "productName": "apple",
  "category": {
    "categoryId": 1
  }
}
```

Fig 17: Create Product (Post Request)


Curl

```
curl -X 'POST' \
  'http://localhost:8080/api/v1/products' \
  -H 'accept: */*' \
  -H 'Content-Type: application/json' \
  -d '{
  "productName": "apple",
  "category": {
    "categoryId": 1
  }
}'
```

Request URL

```
http://localhost:8080/api/v1/products
```

Server response

Code	Details
201 *Undocumented*	Response body

```
{
  "productId": 1,
  "productName": "apple",
  "category": {
    "categoryId": 1,
    "categoryName": "Grocery"
  }
}
```

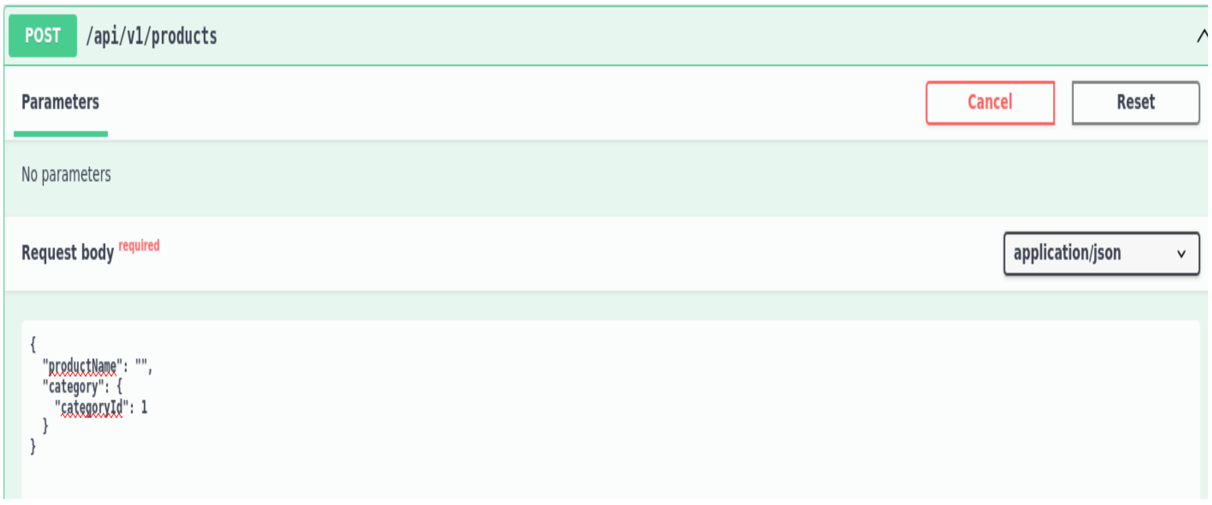
Response headers

```
connection: keep-alive
content-type: application/json
date: Sun,23 Apr 2023 12:15:29 GMT
keep-alive: timeout=60
transfer-encoding: chunked
```

Fig 18: Output Create Product (Post Request)

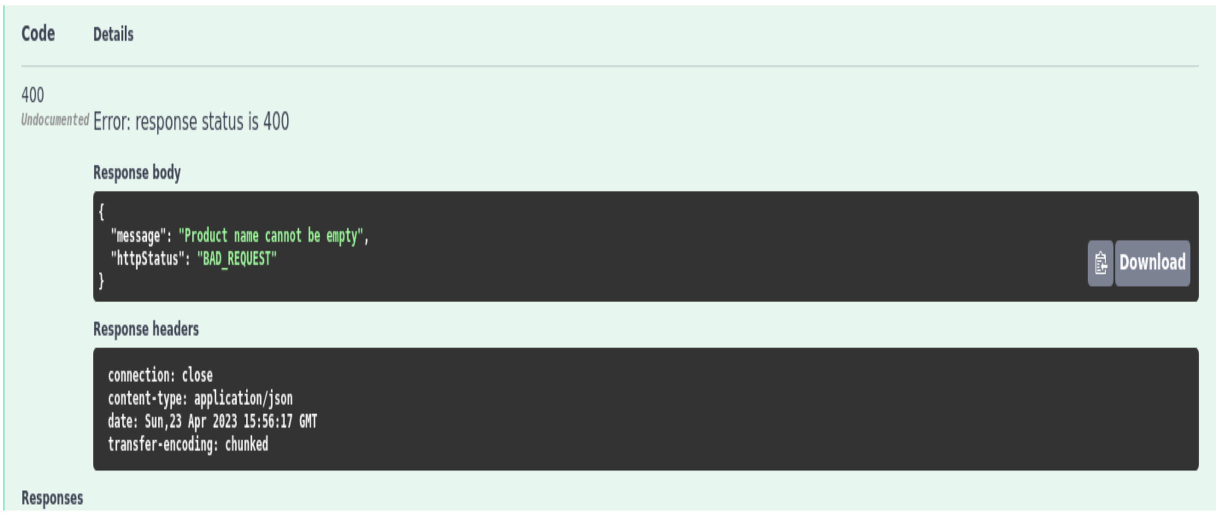
Edge Cases:

1. If the product name in the request body is empty or null.



```
{
  "productName": "",
  "category": {
    "categoryId": 1
  }
}
```

Fig 19: Edge Case Create Product (Product name is null)



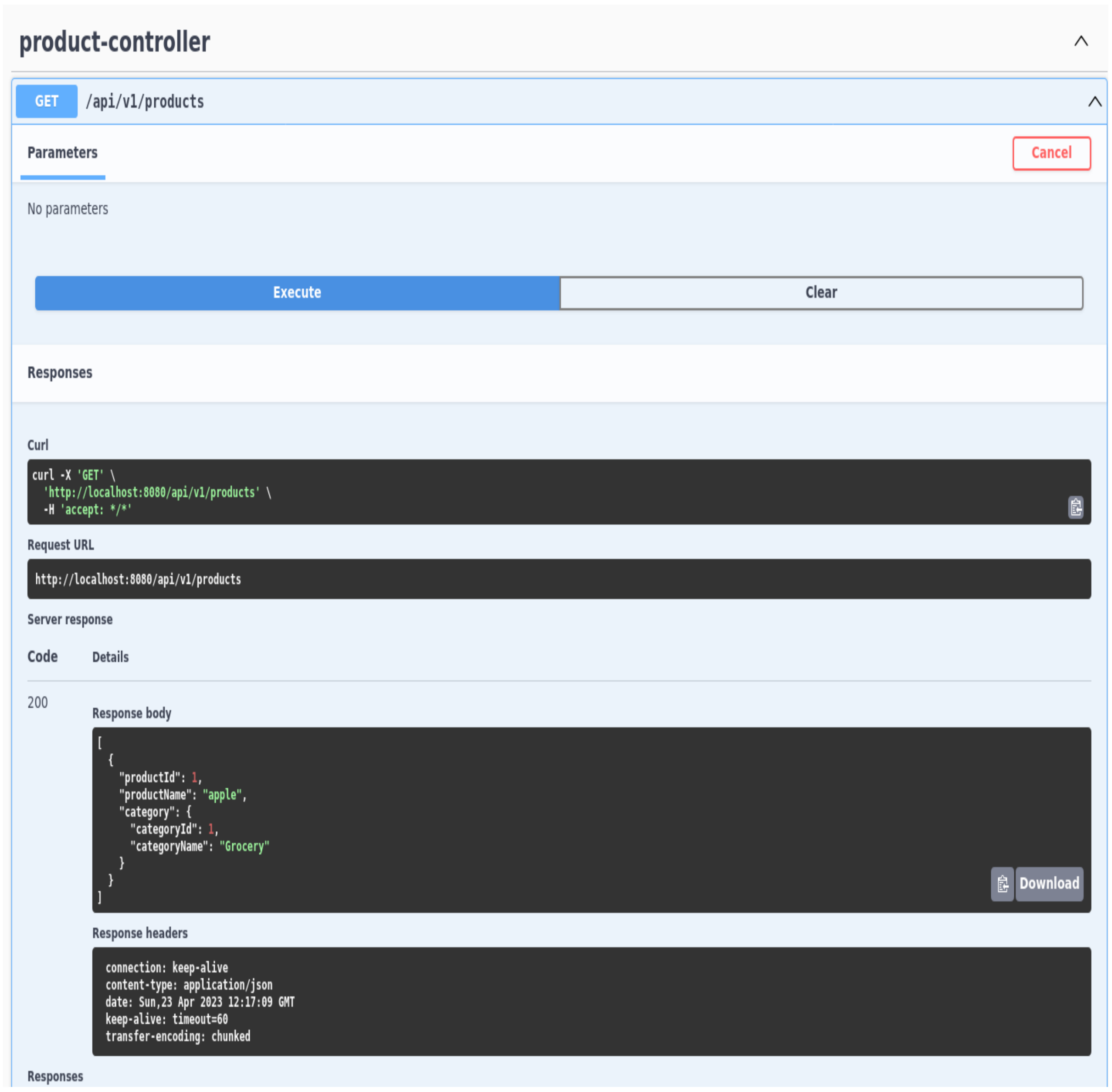
Error: response status is 400

```
{
  "message": "Product name cannot be empty",
  "httpStatus": "BAD_REQUEST"
}
```

```
connection: close
content-type: application/json
date: Sun,23 Apr 2023 15:56:17 GMT
transfer-encoding: chunked
```

Fig 20: Output Edge Case Create Product (Product name is null)

Get All Products



```
curl -X 'GET' \
  'http://localhost:8080/api/v1/products' \
  -H 'accept: */*'
```

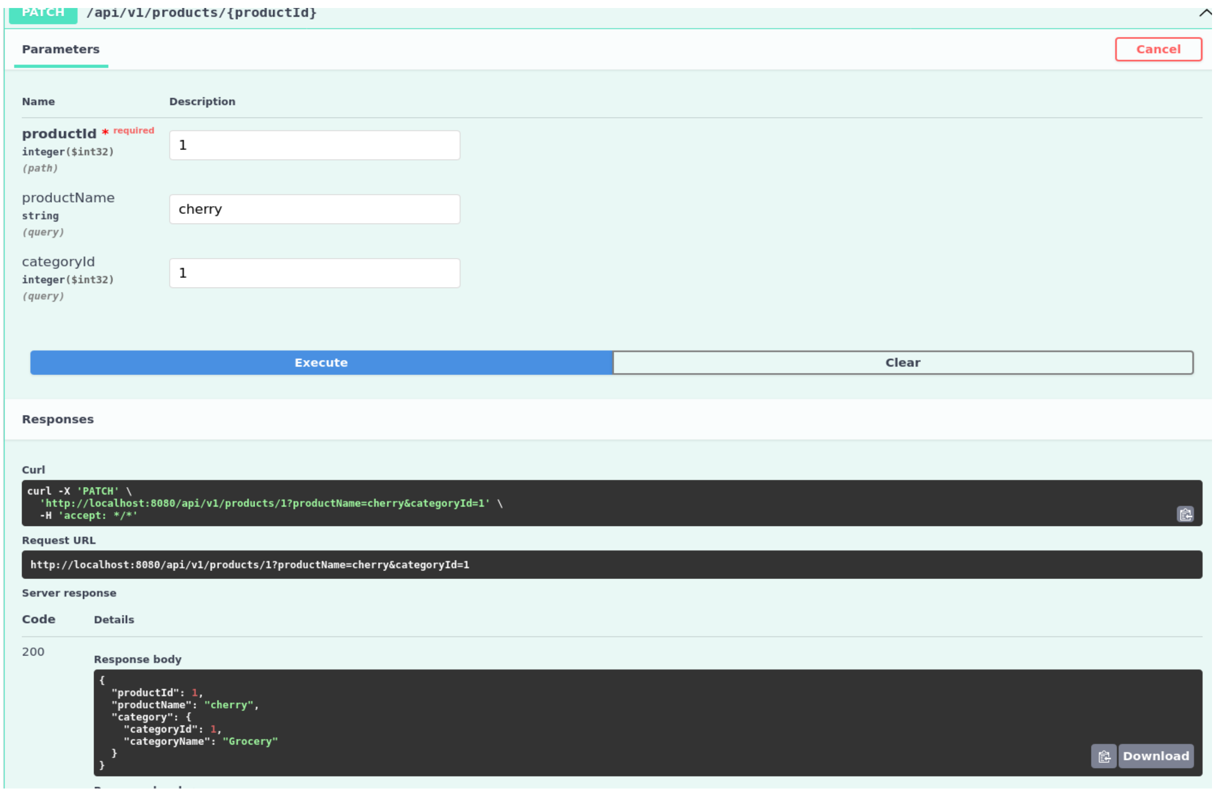
```
http://localhost:8080/api/v1/products
```

```
[
  {
    "productId": 1,
    "productName": "apple",
    "category": {
      "categoryId": 1,
      "categoryName": "Grocery"
    }
  }
]
```

```
connection: keep-alive
content-type: application/json
date: Sun,23 Apr 2023 12:17:09 GMT
keep-alive: timeout=60
transfer-encoding: chunked
```

Fig 21: Get All Products (Get Request)

Update a Product



productId * required
integer($int32)
(path)
1

productName
string
(query)
cherry

categoryId
integer($int32)
(query)
1

Curl

```
curl -X 'PATCH' \
  'http://localhost:8080/api/v1/products/1?productName=cherry&categoryId=1' \
  -H 'accept: */*'
```

Request URL

```
http://localhost:8080/api/v1/products/1?productName=cherry&categoryId=1
```

Server response

Code Details

200

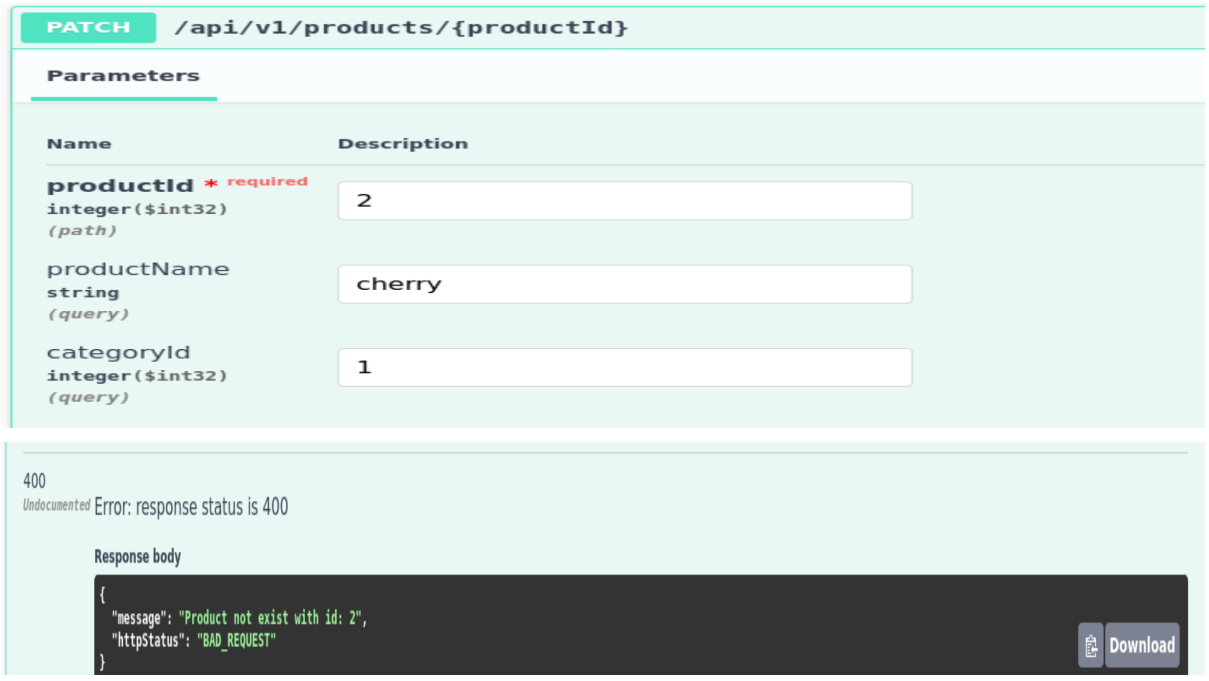
Response body

```
{
  "productId": 1,
  "productName": "cherry",
  "category": {
    "categoryId": 1,
    "categoryName": "Grocery"
  }
}
```

Fig 22: Update Product (Patch Request)

Edge Cases:

1. If the product id does not exist



productId * required
integer($int32)
(path)
2

productName
string
(query)
cherry

categoryId
integer($int32)
(query)
1

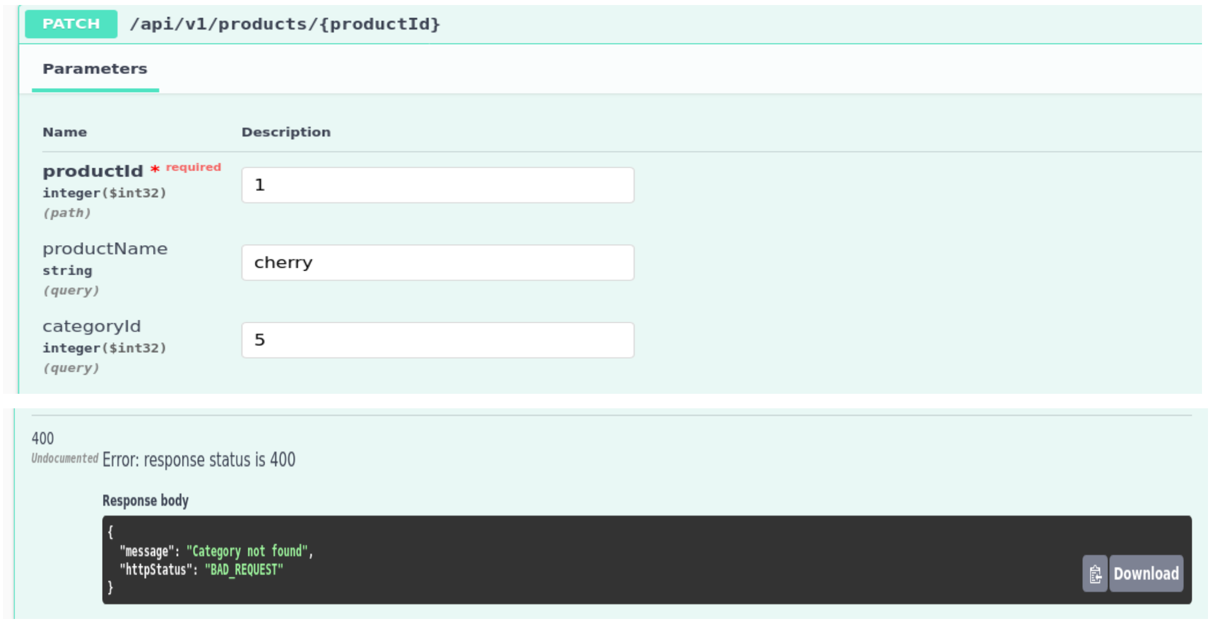
400
Undocumented Error: response status is 400

Response body

```
{
  "message": "Product not exist with id: 2",
  "httpStatus": "BAD_REQUEST"
}
```

Fig 23: Edge Case Update Product (Product id does not exist)

2. If category id does not exist



PATCH /api/v1/products/{productId}

Parameters

Name	Description
productId * required integer($int32) *(path)*	1
productName string *(query)*	cherry
categoryId integer($int32) *(query)*	5

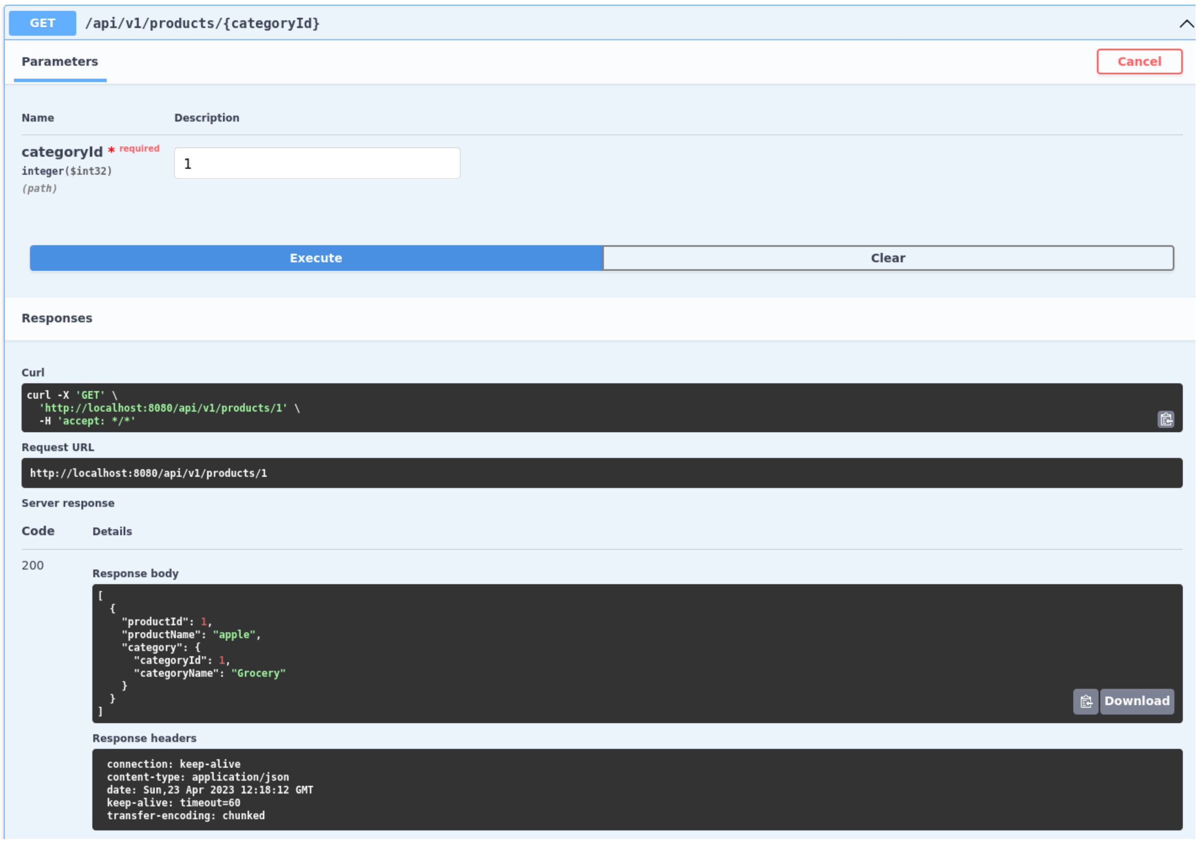
400
Undocumented Error: response status is 400

Response body

```
{
  "message": "Category not found",
  "httpStatus": "BAD_REQUEST"
}
```

Fig 24: Edge Case Update Product (Category id does not exist)

Get Product By a Category Id



GET /api/v1/products/{categoryId}

Parameters

Name	Description
categoryId * required integer($int32) *(path)*	1

Responses

Curl

```
curl -X 'GET' \
  'http://localhost:8080/api/v1/products/1' \
  -H 'accept: */*'
```

Request URL

```
http://localhost:8080/api/v1/products/1
```

Server response

Code	Details
200	Response body

```
[
  {
    "productId": 1,
    "productName": "apple",
    "category": {
      "categoryId": 1,
      "categoryName": "Grocery"
    }
  }
]
```

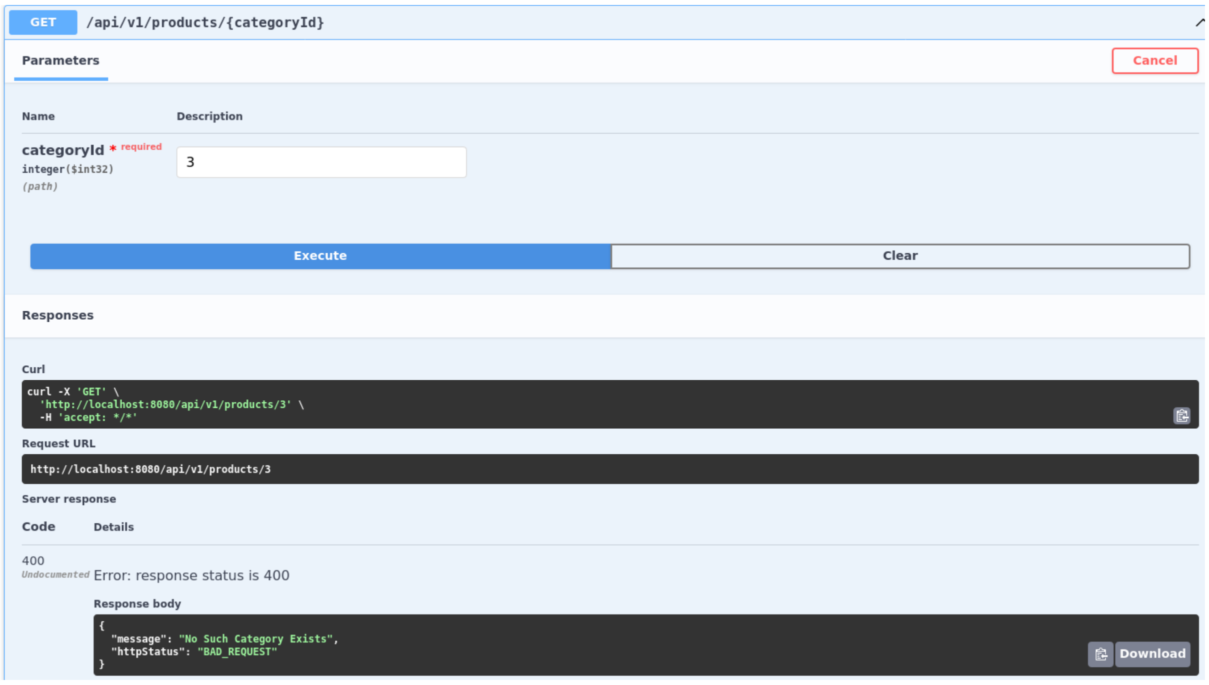
Response headers

```
connection: keep-alive
content-type: application/json
date: Sun,23 Apr 2023 12:18:12 GMT
keep-alive: timeout=60
transfer-encoding: chunked
```

Fig 25: Update Product (Patch Request)

Edge Cases:

1. If no category Id exists



Curl

```
curl -X 'GET' \
  'http://localhost:8080/api/v1/products/3' \
  -H 'accept: */*'
```

Request URL

```
http://localhost:8080/api/v1/products/3
```

Server response

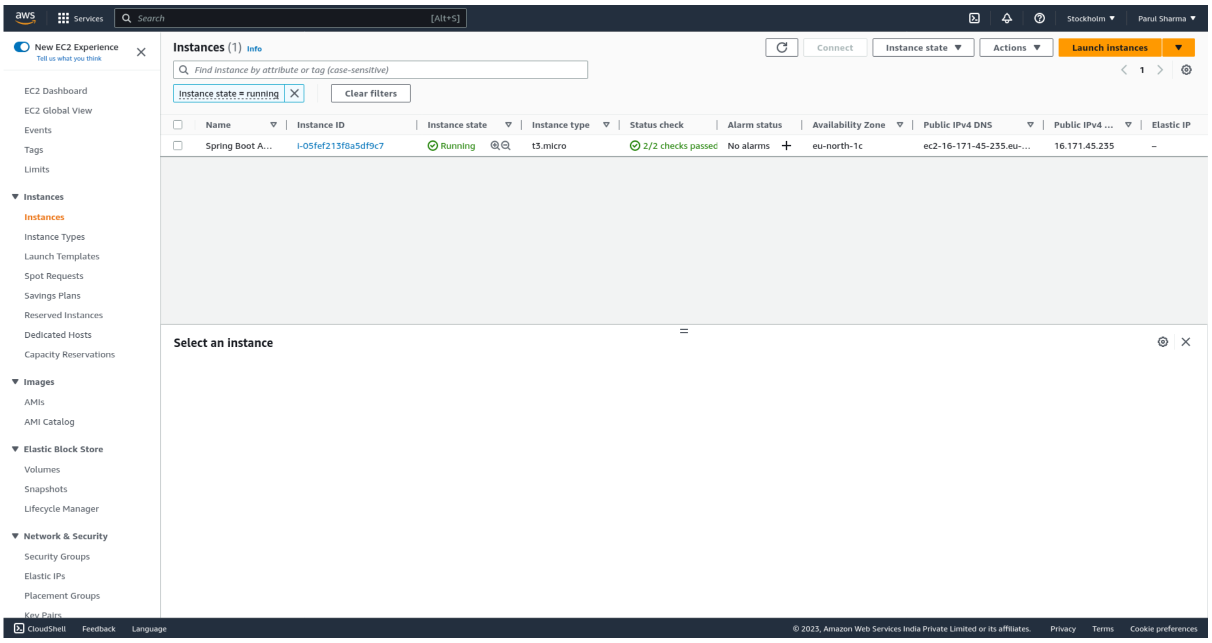
Code	Details
400 *Undocumented*	Error: response status is 400

Response body

```
{
  "message": "No Such Category Exists",
  "httpStatus": "BAD_REQUEST"
}
```

Fig 26: Edge Case Get Product (Category id does not exist)

4.4 Deployment on AWS

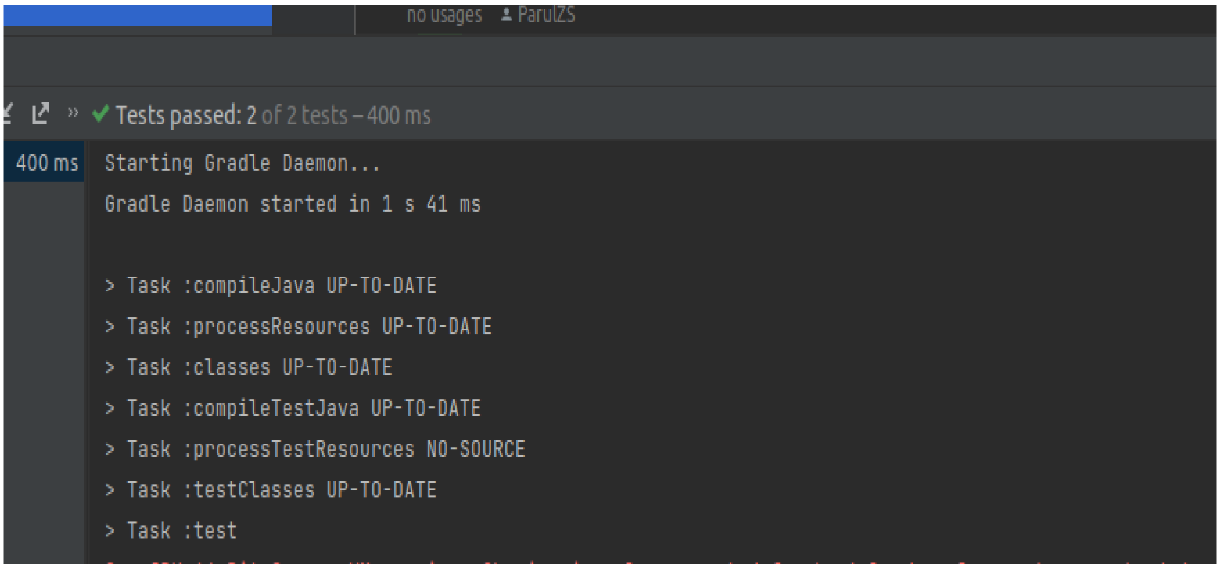


Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone	Public IPv4 DNS	Public IPv4 ...	Elastic IP
Spring Boot A...	i-05fef213f8a5df9c7	Running	t3.micro	2/2 checks passed	No alarms	eu-north-1c	ec2-16-171-45-235.eu-...	16.171.45.235	–

Fig 27: AWS deployment

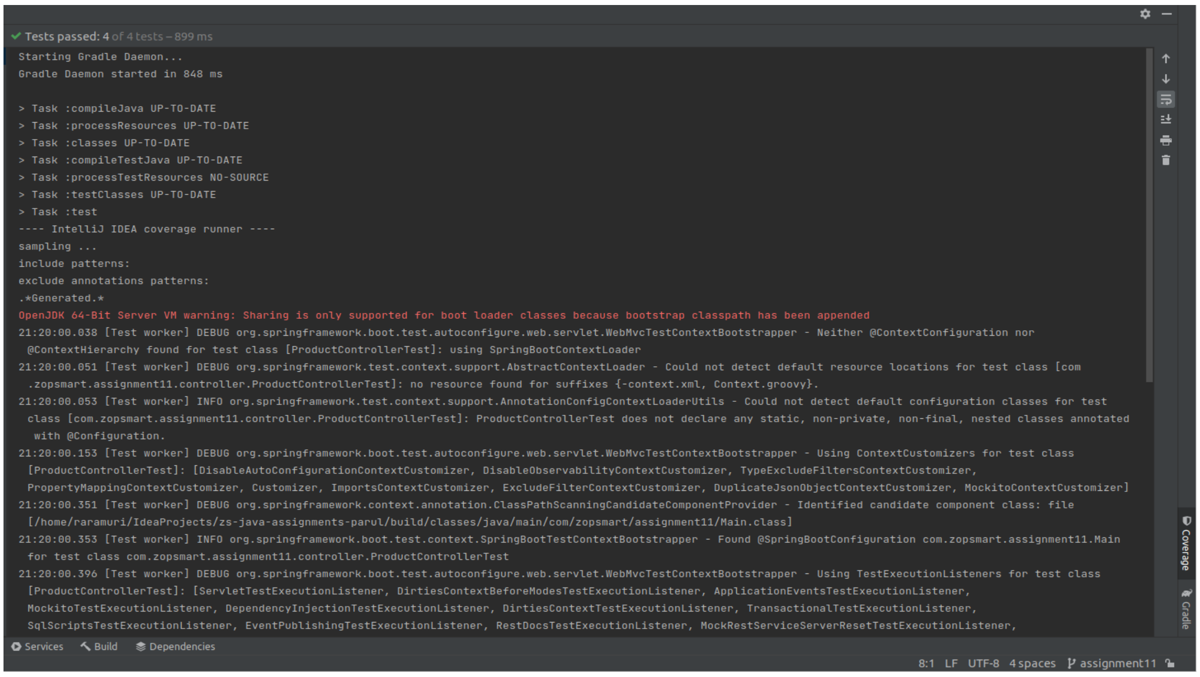
4.5 Unit Testing using Mockito and Junit5

Test for Controller Class:



```
no usages  ParulZS

Tests passed: 2 of 2 tests – 400 ms

400 ms  Starting Gradle Daemon...
        Gradle Daemon started in 1 s 41 ms

        > Task :compileJava UP-TO-DATE
        > Task :processResources UP-TO-DATE
        > Task :classes UP-TO-DATE
        > Task :compileTestJava UP-TO-DATE
        > Task :processTestResources NO-SOURCE
        > Task :testClasses UP-TO-DATE
        > Task :test
```

Fig 28: Test for Category Controller Class

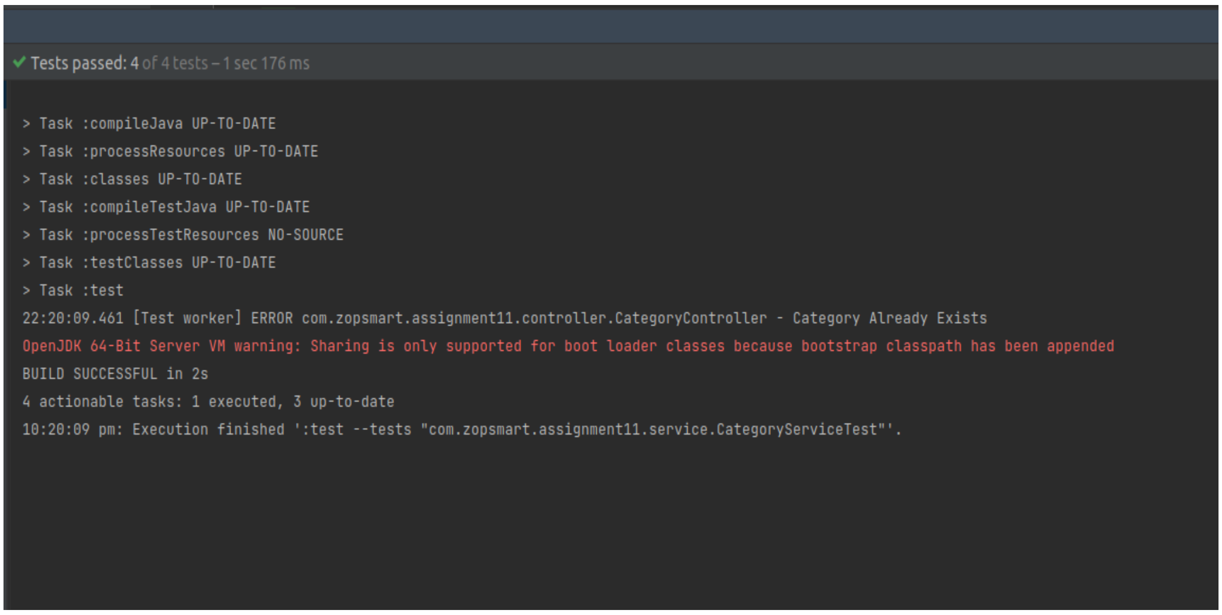


```
Tests passed: 4 of 4 tests – 899 ms
Starting Gradle Daemon...
Gradle Daemon started in 848 ms

> Task :compileJava UP-TO-DATE
> Task :processResources UP-TO-DATE
> Task :classes UP-TO-DATE
> Task :compileTestJava UP-TO-DATE
> Task :processTestResources NO-SOURCE
> Task :testClasses UP-TO-DATE
> Task :test
---- IntelliJ IDEA coverage runner ----
sampling ...
include patterns:
exclude annotations patterns:
.*Generated.*
OpenJDK 64-Bit Server VM warning: Sharing is only supported for boot loader classes because bootstrap classpath has been appended
21:20:00.038 [Test worker] DEBUG org.springframework.boot.test.autoconfigure.web.servlet.WebMvcTestContextBootstrapper - Neither @ContextConfiguration nor
 @ContextHierarchy found for test class [ProductControllerTest]: using SpringBootContextLoader
21:20:00.051 [Test worker] DEBUG org.springframework.test.context.support.AbstractContextLoader - Could not detect default resource locations for test class [com
 .zopsmart.assignment11.controller.ProductControllerTest]: no resource found for suffixes {-context.xml, Context.groovy}.
21:20:00.053 [Test worker] INFO org.springframework.test.context.support.AnnotationConfigContextLoaderUtils - Could not detect default configuration classes for test
 class [com.zopsmart.assignment11.controller.ProductControllerTest]: ProductControllerTest does not declare any static, non-private, non-final, nested classes annotated
  with @Configuration.
21:20:00.153 [Test worker] DEBUG org.springframework.boot.test.autoconfigure.web.servlet.WebMvcTestContextBootstrapper - Using ContextCustomizers for test class
 [ProductControllerTest]: [DisableAutoConfigurationContextCustomizer, DisableObservabilityContextCustomizer, TypeExcludeFiltersContextCustomizer,
 PropertyMappingContextCustomizer, Customizer, ImportsContextCustomizer, ExcludeFilterContextCustomizer, DuplicateJsonObjectContextCustomizer, MockitoContextCustomizer]
21:20:00.351 [Test worker] DEBUG org.springframework.context.annotation.ClassPathScanningCandidateComponentProvider - Identified candidate component class: file
 [/home/raramuri/IdeaProjects/zs-java-assignments-parul/build/classes/java/main/com/zopsmart/assignment11/Main.class]
21:20:00.353 [Test worker] INFO org.springframework.boot.test.context.SpringBootTestContextBootstrapper - Found @SpringBootConfiguration com.zopsmart.assignment11.Main
 for test class com.zopsmart.assignment11.controller.ProductControllerTest
21:20:00.396 [Test worker] DEBUG org.springframework.boot.test.autoconfigure.web.servlet.WebMvcTestContextBootstrapper - Using TestExecutionListeners for test class
 [ProductControllerTest]: [ServletTestExecutionListener, DirtiesContextBeforeModesTestExecutionListener, ApplicationEventsTestExecutionListener,
 MockitoTestExecutionListener, DependencyInjectionTestExecutionListener, DirtiesContextTestExecutionListener, TransactionalTestExecutionListener,
 SqlScriptsTestExecutionListener, EventPublishingTestExecutionListener, RestDocsTestExecutionListener, MockRestServiceServerResetTestExecutionListener,
```

Fig 29: Test for Product Controller Class

Tests for Services Class:

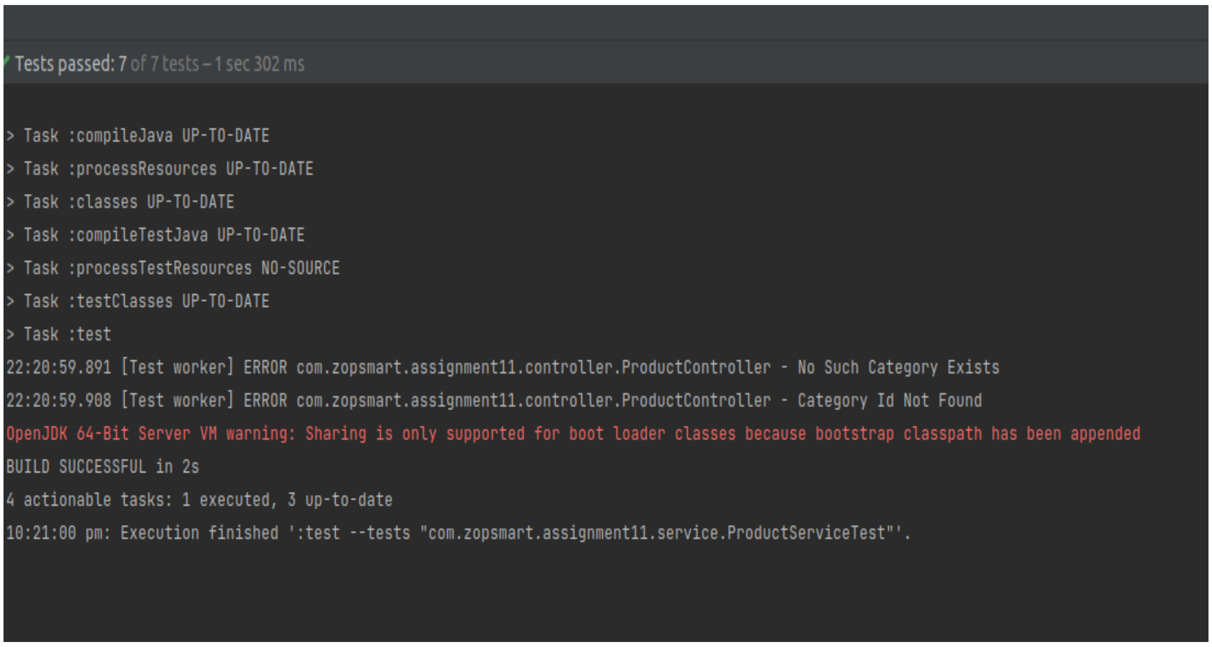
- Category Service



```
✓ Tests passed: 4 of 4 tests – 1 sec 176 ms

> Task :compileJava UP-TO-DATE
> Task :processResources UP-TO-DATE
> Task :classes UP-TO-DATE
> Task :compileTestJava UP-TO-DATE
> Task :processTestResources NO-SOURCE
> Task :testClasses UP-TO-DATE
> Task :test
22:20:09.461 [Test worker] ERROR com.zopsmart.assignment11.controller.CategoryController - Category Already Exists
OpenJDK 64-Bit Server VM warning: Sharing is only supported for boot loader classes because bootstrap classpath has been appended
BUILD SUCCESSFUL in 2s
4 actionable tasks: 1 executed, 3 up-to-date
10:20:09 pm: Execution finished ':test --tests "com.zopsmart.assignment11.service.CategoryServiceTest"'.
```

Fig 30: Test for Category Service Class

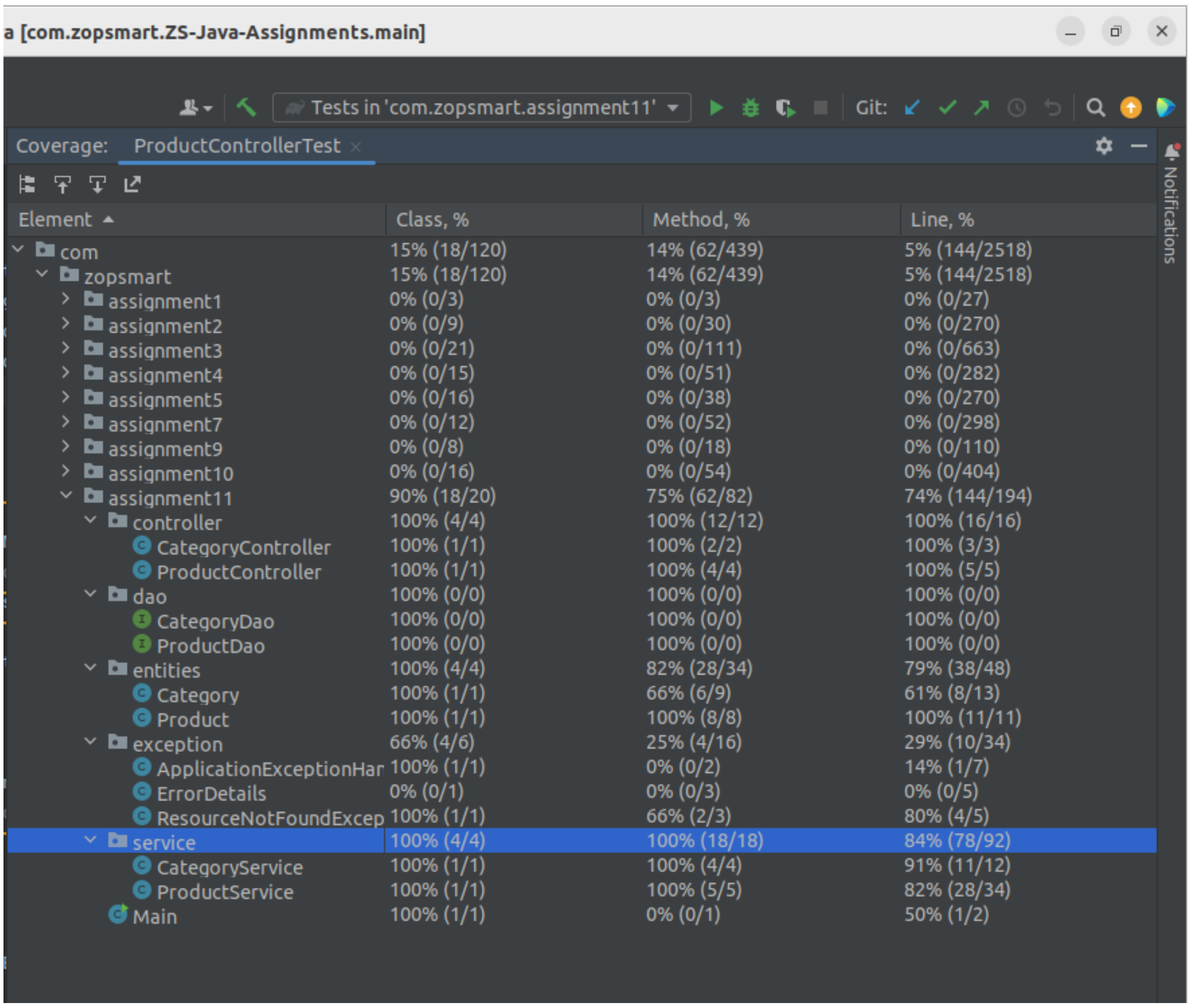
- Product Service



```
✓ Tests passed: 7 of 7 tests – 1 sec 302 ms

> Task :compileJava UP-TO-DATE
> Task :processResources UP-TO-DATE
> Task :classes UP-TO-DATE
> Task :compileTestJava UP-TO-DATE
> Task :processTestResources NO-SOURCE
> Task :testClasses UP-TO-DATE
> Task :test
22:20:59.891 [Test worker] ERROR com.zopsmart.assignment11.controller.ProductController - No Such Category Exists
22:20:59.908 [Test worker] ERROR com.zopsmart.assignment11.controller.ProductController - Category Id Not Found
OpenJDK 64-Bit Server VM warning: Sharing is only supported for boot loader classes because bootstrap classpath has been appended
BUILD SUCCESSFUL in 2s
4 actionable tasks: 1 executed, 3 up-to-date
10:21:00 pm: Execution finished ':test --tests "com.zopsmart.assignment11.service.ProductServiceTest"'.
```

Fig 31: Test for Product Service Class

Test Coverage:


a [com.zopsmart.ZS-Java-Assignments.main]

Tests in 'com.zopsmart.assignment11'
Git:
Coverage: ProductControllerTest
Notifications

Element	Class, %	Method, %	Line, %
com	15% (18/120)	14% (62/439)	5% (144/2518)
zopsmart	15% (18/120)	14% (62/439)	5% (144/2518)
assignment1	0% (0/3)	0% (0/3)	0% (0/27)
assignment2	0% (0/9)	0% (0/30)	0% (0/270)
assignment3	0% (0/21)	0% (0/111)	0% (0/663)
assignment4	0% (0/15)	0% (0/51)	0% (0/282)
assignment5	0% (0/16)	0% (0/38)	0% (0/270)
assignment7	0% (0/12)	0% (0/52)	0% (0/298)
assignment9	0% (0/8)	0% (0/18)	0% (0/110)
assignment10	0% (0/16)	0% (0/54)	0% (0/404)
assignment11	90% (18/20)	75% (62/82)	74% (144/194)
controller	100% (4/4)	100% (12/12)	100% (16/16)
CategoryController	100% (1/1)	100% (2/2)	100% (3/3)
ProductController	100% (1/1)	100% (4/4)	100% (5/5)
dao	100% (0/0)	100% (0/0)	100% (0/0)
CategoryDao	100% (0/0)	100% (0/0)	100% (0/0)
ProductDao	100% (0/0)	100% (0/0)	100% (0/0)
entities	100% (4/4)	82% (28/34)	79% (38/48)
Category	100% (1/1)	66% (6/9)	61% (8/13)
Product	100% (1/1)	100% (8/8)	100% (11/11)
exception	66% (4/6)	25% (4/16)	29% (10/34)
ApplicationExceptionHar	100% (1/1)	0% (0/2)	14% (1/7)
ErrorDetails	0% (0/1)	0% (0/3)	0% (0/5)
ResourceNotFoundExcep	100% (1/1)	66% (2/3)	80% (4/5)
service	100% (4/4)	100% (18/18)	84% (78/92)
CategoryService	100% (1/1)	100% (4/4)	91% (11/12)
ProductService	100% (1/1)	100% (5/5)	82% (28/34)
Main	100% (1/1)	0% (0/1)	50% (1/2)

Fig 32: Test Coverage for web application

Testing is done using mockito and Junit 5 which increases the code coverage and is helpful in unit testing.Also there exists two types of testing i.e. integration testing and unit testing.In this project we focussed on unit testing only.

Chapter-5

CONCLUSIONS

5.1 Conclusion

In conclusion, the three-layered Spring Boot web application is an effective and scalable solution to create online applications. It is simpler to maintain and adjust distinct components of the programme without impacting the others when the application is divided into display, service, and persistence layers. User interface is handled by the controller layer, business logic is handled by the service layer, and data storage is handled by the repository layer. This enhances the application. Additionally, Spring Boot offers a variety of tools and features, such as built-in support for several databases and web technologies, dependency injection, and auto-configuration, to streamline the development process.
In conclusion three layered architectures have increased the effectiveness of the web applications.

5.2 Future Scope

The project or web application is monolithic ,hence we can convert it into microservices which has many advantages like:

1. Autonomous In a microservices architecture, each service component may be created, deployed, run, and scaled independently of the functionality of other services.

2. Specialised: Each service is designed with a distinct set of skills and a focus on a certain issue. The service may eventually become further divided into smaller services as it scales over time. When a system is dealing with a spike in demand, it aids teams in determining the size of infrastructure requirements, the worth of a product, and how to maintain dependability.

3. Easy Deployment: Continuous integration and delivery are made possible by microservices, making it simple to test out new ideas or roll back if something goes wrong. The cheap cost of failure promotes software updates, fosters development, and shortens the time it takes to sell new features.

In monolithic application ensure these practices:

Integration: Using messaging and APIs, a monolithic programme may be connected to various platforms and services. Integration is more crucial than ever because of the emergence of APIs and microservices.

Maintenance: Maintenance becomes difficult when the size of monolithic application increases. For best results enable practices such as clean code, testing, and continuous integration and delivery.

5.3 Applications Contributions

1. **E-commerce:** E-commerce websites are required to be highly scalable, and strong security is suitable for Spring Boot web applications. Also, it requires managing high traffic on the site, carrying out safe transactions, and delivering a flawless user experience.

2. **Healthcare:** Patient portals, telemedicine platforms, and electronic health records (EHR) systems may all be created in the healthcare industry using Spring Boot web apps. The standard of hospitals and healthcare is increased, real-time services between doctors and patients, and secure retain patient data.

3. **Finance:** Banking and financial applications including online banking systems, mobile banking apps, and investment management can be done with the help of Spring Boot web applications in the financial sector. The application can safely manage transactions and offer real-time data analysis services.

4. **Logistics:** Supply chain management systems, transportation management systems, and warehouse management systems may all be built using Spring Boot web applications. These systems can efficiently manage inventories, plan routes and delivery schedules, and enable real-time tracking of shipments.

5. **Social networking:** Social networking systems for social media, online communities, and collaboration may be created with Spring Boot web applications.

REFERENCES

[1] L. Xu, M. Li, W. Chen, and X. Liu, "A Comparative Study on Performance of Web Application Frameworks for Java," Journal of Physics: Conference Series, vol. 1065, no. 5, p. 052022, 2018.

[2] O. Baker and Q. Nguyen, "A novel approach to secure microservice architecture from OWASP vulnerabilities," in 2019 International Conference on Recent Advances in Business, Management and Information Technology (ICRABMIT), pp. 1-6, IEEE, 2019.

[3] H. Aris, N. Yahya, M. A. Ishak, and M. Z. Yusoff, "Design and development of a backend application for public complaint system," Journal of Telecommunication, Electronic and Computer Engineering, vol. 9, no. 2-9, pp. 29-33, 2017.

[4] S. Debnath, R. Saha, and S. Sarkar, "Empirical Analysis of JUnit and Mockito Frameworks for Java Applications Testing," International Journal of Computer Applications, vol. 182, no. 26, pp. 25-31, 2019.

[5] ZopSmart, "ZopSmart," 2021. [Online]. Available: https://zopsmart.com/.

[6] Oracle, "Java SE - Downloads | Oracle Technology Network | Oracle," Oracle, 2021. [Online]. Available: https://www.java.com/.

[7] JetBrains, "IntelliJ IDEA: The Java IDE for Professional Developers by JetBrains," JetBrains, 2021. [Online]. Available: https://www.jetbrains.com/idea/.