

THREE LAYERED ARCHITECTURE IN JAVA

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

in

Computer Science and Engineering/Information Technology

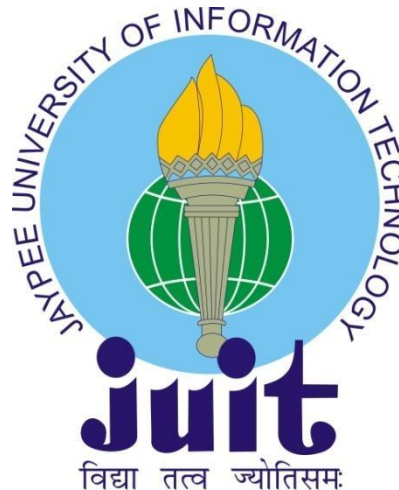
By

Ishita (191326)

Under the supervision of

Dr. Rajni Mohana

to



Department of Computer Science & Engineering and Information
Technology

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

Candidate's Declaration

This is to certify that the work which is being presented in the project report titled “Three Layer Architecture in Java” in partial fulfillment of the requirements for the award of the degree of B.Tech in Computer Science And Engineering and submitted to the Department of Computer Science And Engineering, Jaypee University of Information Technology, Waknaghat is an authentic record of work carried out by Ishita (191326) during the period from 7 Feb & 2022 to 29 May under the supervision of Dr.Rajni Mohana, Department of Computer Science and Engineering, Jaypee University of Information Technology, Waknaghat.

Ishita
(191326)

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

Dr.Rajni Mohana
Assistant Professor (SG)
Computer Science & Engineering and Information Technology
JUIT, Waknaghat
Dated: 10th May,2023



(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 =

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

I would like to thank Rashmi Singh, Manager of HR in Talent Acquisition, of Zopsmart Technologies, Bangalore for allowing me to do an internship within the organization.

I also would like to thank Mr. Ujjawal Misra, Mr. Prakhyat Saini, and all the people that worked along with me at Zopsmart Technologies, Bangalore for their patience and openness. They created an enjoyable working environment. I am extremely grateful to my supervisor, Dr.Rajni Mohana, Assistant Professor (SG) ,Department of CSE Jaypee University of Information Technology, Waknaghat, for her assistance. To complete this assignment, my supervisor has extensive knowledge and a deep interest in the subject of Cloud Computing. Her never-ending patience, intellectual direction, constant encouragement, constant and energetic supervision, constructive criticism, good suggestions, and reading many poor versions and fixing them at all stages made it possible to finish this job.

Ishita (191326)

Table Of Content

S. No	Contents	Page No.
1.	Candidate's Declaration	i
2.	Plagiarism Certificate	ii
3.	Acknowledgement	iii
4.	Table of Content	iv
5.	List of Figures	v
6.	List of Tables	vii
7.	Abstract	viii
8.	Chapter 1: Introduction	1
	1.1 Introduction	1
	1.2 Problem Statement	3
	1.3 Objectives	4
	1.4 Methodology	4
	1.5 About the Company	6
	1.6 Organization	8
9.	Chapter 2: Literature Survey	9
10.	Chapter 3: System Development	15
	3.1 Framework Used	15
	3.2 Technical Requirements	15
	3.2.1 IntelliJ	15
	3.2.2 Postgres	16
	3.2.3 Hardware Configuration	17
	3.2.4 Software Configuration	17
	3.3 Model Development	17
	3.3.1 Algorithm	18
	3.3.2 Deployment	19
11.	Chapter 4: Experiment and Result Analysis	22
	4.1 Api Request	22
	4.2 Exceptional Handling	26
	4.3 Validations	27
	4.4 Swagger	28
	4.5 Unit testing using Mockito	35
12.	Chapter 5: Conclusions	38
	5.1 Conclusion	38
	5.2 Future Scope	39
	5.3 Limitations	40

List Of Figures

Figure No.	Title	Page no.
Figure 1	Three Layer Architecture	3
Figure 2	Architecture Methodology	5
Figure 3	ZopSmart Logo	6
Figure 4	Intellij	16
Figure 5	Docker	20
Figure 6	Post request to add category	22
Figure 7	Post request to add product	23
Figure 8	Get request to get all categories	23
Figure 9	Get request to get product by category id	24
Figure 10	Patch request to update a product	25
Figure 11	Exceptional Handling	26
Figure 12	Validation	27
Figure 13	Swagger	28
Figure 14	Input to add a category	29
Figure 15	Output of adding a category	29
Figure 16	Edge case input	30
Figure 17	Edge case output	30
Figure 18	Output of get category	31
Figure 19	Input to create a product	32
Figure 20	Output of post a product	32
Figure 21	Edge case of productName	33
Figure 22	Output of edge case	33
Figure 23	Get product by product id	34
Figure 24	Get product by category id	35

Figure 25	Category controller test case	35
Figure 26	Product controller test case	36
Figure 27	Category service test case	36
Figure 28	Product service test case	37

List Of Tables

Table No.	Title	Page No.
Table 1	Hardware Requirement	17
Table 2	Software Requirement	17

Abstract

Developing a web application may seem like an easy task, but the difficulty lies in testing, organizing, refining, and sustaining the code. To tackle these challenges, we utilize the Three Layered Architecture approach, using Java programming language.

The three layers are Presentation layer, Business Logic Layer and Data Access layer which are independent of each other. The presentation layer is responsible for providing a user interface that enables users to interact with the application. It is the topmost layer of a multi-tier architecture, and it communicates with the underlying business logic layer to facilitate data processing and management. The business logic layer is responsible for implementing the application's core logic and rules. It acts as an intermediary between the presentation layer and the data storage layer, processing user requests and manipulating data to produce meaningful results. It is essential to design this layer in a way that is modular, scalable, and maintainable to ensure that the application functions efficiently and can evolve over time with changing business requirements. Data Access Layer provides a standardized way of accessing data from different sources, such as databases, files, and web services.

CHAPTER 1

INTRODUCTION

1.1 Introduction

Frameworks contain pre-written code that developers can reuse to construct applications or web applications without starting from scratch each time. Everything required to construct an app is provided or manifested as functions and classes in a comprehensive framework.

Frameworks written in a language are unique to that language. Frameworks can be used by calling their methods, inheriting them, and providing "callbacks", listeners, or other Observer pattern implementations. Framework is a collection of libraries that provide a skeletal architecture for the implementation of specific software. The abstraction allows for common design patterns to be easily reused while still allowing the specific details to be left to the developers.

It is a basic web application that implements **CRUD operations based on the three layered architecture**. Programs at each layer have their own unit test. There is also an implementation of middleware that authenticates the http request before sending it to the server. The three-tier architecture is a software architecture pattern that divides an application into three interrelated layers: the presentation layer, the business logic layer, and the data storage layer. Each layer is in charge of a distinct set of duties and has a well-defined interface for interacting with the other layers. Presentation layer displays the application's features and data to the user. It is used for interacting with the user. It gets the input from the user and displays the associated output. The business logic layer is in charge of processing data and enforcing the application's rules. It comprises all of the business logic of the application, including as calculations, validations, and decision-making procedures. To collect user input and retrieve and store data, the business logic layer connects with the display layer and the data storage layer. The business logic layer in Java is often implemented using Java classes. Data from a database or other data sources must be stored and retrieved via the data storage layer. It consists of elements like file systems, database servers, and web services. To send and receive data, the business logic layer and the data storage layer communicate. Frameworks like Hibernate, JPA, or JDBC are frequently used to implement Java's data storage layer.

ZopSmart has its own java framework that helps in implementing three layered architecture easily and efficiently. Rocket is an application framework built using enterprise Java 11 intended to create high performing, easily testable and reusable code. This framework uses various new techniques such as Plain old Java Object (POJO) and Dependency Injection(DI) to develop enterprise applications thereby removing the complexities involved while developing enterprise applications using Enterprise Java Beans (EJB). It is not open source.

The framework mainly focuses on providing various ways to help you manage your business objects. It makes the development of Web applications much easier as compared to classic Java frameworks and Application Programming Interfaces (APIs), such as Java database connectivity(JDBC). It is a lightweight application framework used for developing enterprise applications. The Rocket framework can be considered as a collection of sub-frameworks, such as Core, HTTP, ORM. You can use any of these modules separately while constructing a Web application. The modules may also be grouped together to provide better functionalities in a Web application. The framework is loosely coupled because of dependency Injection.

The framework includes a health check and heartbeat mechanisms, enabling applications to monitor their own health status and respond accordingly. This helps maintain the availability and responsiveness of the application.

It supports caching to improve application performance by reducing repetitive computations and enhancing response times. File operations, cron scheduling, tracing, and performance monitoring features are also available, allowing developers to optimize their applications for better efficiency.

The framework integrates with a flexible logger, enabling developers to log application events, errors, and other relevant information for debugging and troubleshooting purposes. It also provides middleware support, allowing developers to add additional functionality to the request-response cycle, such as authentication, logging, rate limiting, and more. Furthermore, Rocket seamlessly integrates with metrics systems enabling the collection and analysis of application metrics. This helps developers monitor performance, identify trends, and make data-driven optimizations.

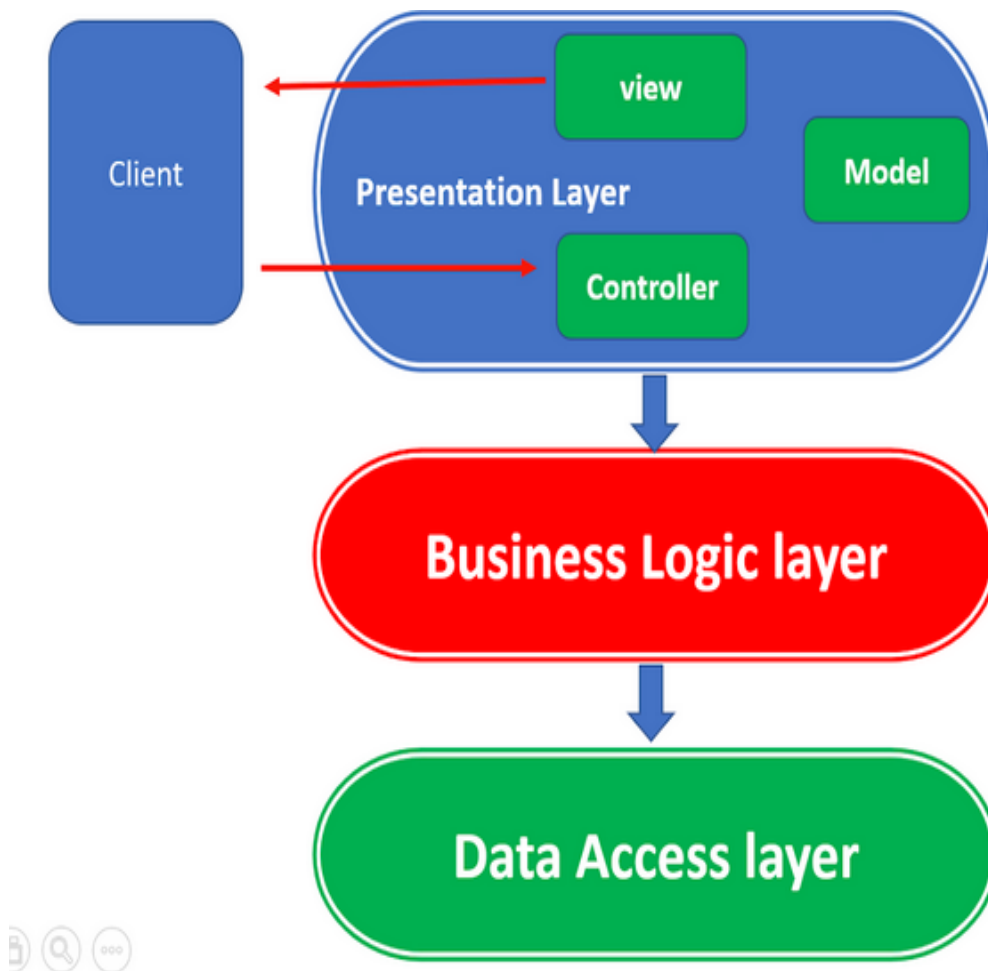


Figure 1. Three Layer Architecture

1.2 Problem Statement

When used in Java applications, the three-layered architecture is a significant software design pattern that can provide various advantages. A three-layered architecture may be required in Java for a number of important reasons, including:

1. Separate needs- The display, business logic, and data storage layers are clearly separated by the design from one another in terms of concerns. This makes it simpler to manage, test, and maintain the programme by enabling developers to concentrate on separately developing each layer.

2. Scalability and Reusability- One layer can be scaled up if it needs additional resources or processing power without affecting the other layers. Since each layer can be written as a

standalone module that can be utilized in other applications, the architecture promotes code reuse. This means that rather than starting from scratch, developers can save time and effort by reusing code from already existing modules.

3. Increased security- The architecture offers better security since critical data can be housed in a separate layer and protected using a variety of encryption techniques. As a result, even if one layer's security is compromised, the remaining layers are still secure.

1.3 Objective

To create testable, structured, clean and maintainable web applications by using industrial best practices. The aim is to simplify the task of developers who are creating applications that require product and category data by providing a dependable and efficient API. By simplifying data administration and retrieval, the API frees developers to focus on the primary functionality of their applications. The API is built according to RESTful principles, allowing it to be linked with other platforms and applications, making it a versatile and adaptable solution for managing product and category data. In summary, the API provides both users and developers with a trustworthy and effective solution.

1.4 Methodology

To develop web applications, work was done on linux environment. IntelliJ Idea is the tool used to write code, debug and check metrics of the application. Java is the language used to develop the framework. Since the framework is used to develop the backend of a web application, it doesn't need a UI. Three layered architecture was implemented to build the project and all the layers were independent of each other. The testing of the layers were done independently. The Spring-Boot API was created utilizing a methodical process that adhered to the listed requirements. The requirements collecting, planning, implementation, testing, and deployment stages were all part of the development process.

The first stage in the strategy was to comprehend the API's needs by looking at the problem description and the project's main goal. Following the definition of the requirements, a three-tier architecture for the API was created, with a controller layer, a service layer, and a dao layer.

The implementation of the API utilizing the Spring-Boot framework came after the

architecture had been created. Using Spring-Boot Controllers, Services, and Data Access Objects, the controller layer, service layer, and dao layer were individually implemented.

Once the implementation was complete, the API was rigorously tested to confirm that it satisfied the requirements. There were several testing techniques used, including unit testing with frameworks like Mockito and JUnit5. After all problems were fixed, the API was set up using Docker containers and deployed to an AWS EC2 server for public use. The chosen approach worked well in creating a scalable, useful Spring-Boot API that complied with the specifications. The three-layer strategy employed provided for a distinct separation of responsibilities and made it easy to maintain and upgrade the API. Furthermore, the deployment process was efficient and streamlined, thanks to the use of Docker containers and the Spring-Boot framework, which enabled rapid development and easy deployment.

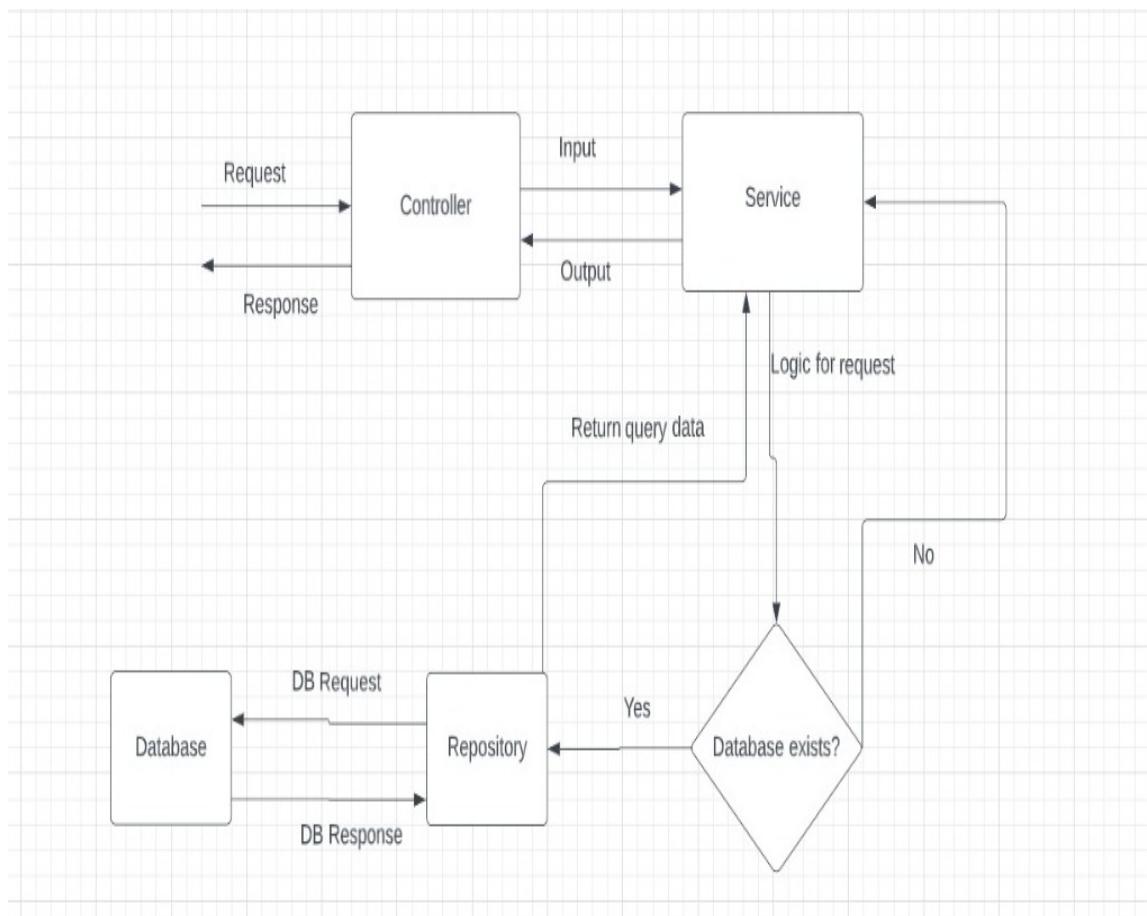


Figure 2. Three Layer Architecture Methodology

Three Layered Architecture -

The three layers are independent of each other and communicate with each other with the help of an interface.

Controller Layer- It interacts with the user for the input and output.

Service Layer- It has all the business logic written inside it and this layer further interacts with the Database layer.

Repository Layer- It interacts with the database where the data is stored like a table.

1.5 About the Company

ZopSmart is a software solution firm that offers you all the resources you need to launch an online store. ZopSmart's product line can assist you in building and running the ideal business. It provides a range of products, including, among others, Smart Store Eazy and Smart Payment Gateway. Zopsmart, a company that creates cutting-edge technologies for the retail sector, with clients ranging from small, independent furniture retailers to huge, international chains. An e-commerce platform, digital marketing, mobile commerce, automated logistics systems, management platforms, order management platforms, and internet of things (IoT) gadgets are just a few of the company's offerings. It delivers software solutions to some of the best businesses and has its own framework for operation.

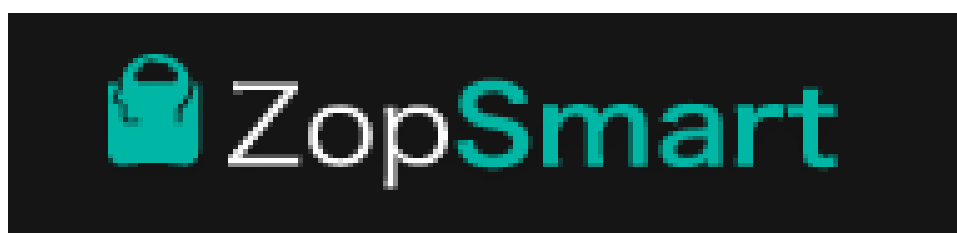


Figure 3. ZopSmart logo

ZopSmart Solutions

- **E-Commerce-** Prioritizing your consumers' requirements and preferences is vital if you want to give them an outstanding shopping experience. Implementing intuitive search capability,

which enables users to easily and quickly find what they're looking for, is one approach to accomplish this. Additionally, you may streamline the ordering procedure and make it easier for clients to complete their purchase by personalizing the product listing. It's critical to provide self-service return and reschedule options in order to further improve the consumer experience. Customers can quickly return products if they are dissatisfied with their purchase thanks to the ability to easily adjust orders in accordance with changing needs.

- **M-Commerce-** In the current digital era, it is essential to offer customers a seamless and pleasurable purchasing experience on mobile devices. Investing in a responsive website that can adapt to all screen sizes will guarantee that your clients have a wonderful shopping experience. Customers will be able to browse your online store on any device thanks to this, regardless of screen size or quality, and will have a consistent and optimized browsing experience. You can create native Android and iOS mobile applications that offer a top-notch experience to advance your mobile buying experience.
- **E-Wallet:** To enhance customer loyalty, consider utilizing an integrated wallet feature. With an integrated wallet, the checkout process becomes quicker and easier, which can lead to a more satisfying shopping experience for customers. Additionally, by offering refunds and wallet cashback, you can incentivize customers to make repeat purchases and further build their loyalty towards your brand. Furthermore, integrating wallet-credit as gift cards can be a cost-effective strategy for acquiring new customers while simultaneously increasing sales.
- **Order Management:** All of the orders will be displayed in a single interface, together with information about them like the client, the total, and the order status. Order statuses are updated as each step of the process is finished. Option to change the order's contents.
- **Operations:** Using the Store-manager mobile app to manage daily tasks will allow you to provide your customers with outstanding service at the lowest possible cost. Mobile inventory checker app to assure catalog accuracy. Picker is a smartphone app that makes picking and packaging effortless and effective. For the most effective delivery route, use the dispatch module. App for reliable order fulfillment, risk-free money collection, and carefully selecting returns.

- **Monitoring and Analytics:** Using real-time tracking to keep an eye on every aspect of your operation, you can manage your organization with little supervision. Alerts of deviations in the operations process that call for immediate action. Analytics with intelligence helps identify patterns and improve procedures. You can track down any problems with the help of thorough logs for each activity.

1.6 Organization

The rest of this report is organized as follows:

Chapter 1: In Chapter 1, we evaluate our Spring Boot application and briefly explore the advantages of constructing our Spring Boot application utilizing a three-layered design. A summary of the system's rationale and purpose is offered together with the issue statement, which serves as the foundation for our project's objectives.

Chapter 2: In Chapter 2, we conduct a literature research in which we collect several papers and other resources that assisted us in creating our RESTful API utilizing the advised three-layered architecture technique.

Chapter 3: The objective of Chapter 3 is to outline the detailed procedure we utilized to construct the study arrangement. To begin, we combed through many documents and e-commerce portals to build a list of essential components that ought to be included in the project's API backend. The primary subjects in this chapter are the equipment and technology employed in the procedure.

Chapter 4: In Chapter 4 of the report, we present the results and outputs of our RESTful API project. It aims to demonstrate how precise and effective our implementation is. This chapter also provides a thorough analysis of a number of API use cases with an emphasis on potential applications.

Chapter 5: The final project model and a summary of the study results are included in Chapter 5 of the report. It highlights areas that would benefit from more research and effort. Additionally, the creative work that was created as a result of the study is presented in this chapter.

Chapter 2

Literature review

The Java Framework, developed by ZopSmart Technologies, incorporates various technologies to achieve its objectives. In this literature survey, we delve into the research and literature pertaining to the technologies employed in the Java Framework.

Java- Since its introduction in the middle of the 1990s, Java, a high-level programming language, has grown significantly in popularity. Java is a great language for novices to learn because of how simple its syntax is to understand and write. Furthermore, Java is platform-independent, allowing programmes written in it to operate without change on a variety of different operating systems[1]. Java is now one of the most popular programming languages for creating cross-platform apps as a result of this functionality. The success of Java is also largely due to its object-oriented programming approach. This architecture enables programmers to group their code into reusable objects, which makes managing and maintaining their programmes simpler. Automatic garbage collection is another feature that Java enables, which lessens the possibility of memory leaks and other memory-related problems. Java is also heavily utilized while creating mobile applications for Android-based devices. Because Android is built on top of the Java Virtual Machine (JVM), programmers may utilize Java to create robust, adaptable, and simple-to-maintain Android apps.

Java is very well-known for its security capabilities. Because it is intended to prevent harmful code from being executed, it is a popular choice for developing secure apps. This is accomplished by running Java code in a "sandbox" environment that limits the code's access to system resources. It also has a large ecosystem of libraries and frameworks. Because they provide ready-made solutions to common problems, these pre-written code modules can be used to save time when developing applications. Spring, Hibernate, and Struts are some popular Java frameworks. Java is very widely used in the business world. Many major organizations rely on Java-based apps for day-to-day operations, making it a vital talent for developers interested in working in the enterprise arena.

Gradle- Java applications can be created more easily thanks to Gradle, a well-liked build automation tool. Due to its adaptability and simplicity of usage, it has grown in popularity among developers since its initial introduction in 2007. One of Gradle's main benefits is its adaptability. Programmers can use its difficult scripting language to develop innovative logic and automate difficult processes. This means that developers can adapt their build process to meet the specific requirements of their application rather than being constrained by the restrictions of a predetermined build system. Gradle also offers a number of useful features that streamline the building process. It has a dependency management system, for instance, which automatically manages and downloads the application's dependencies. Developers are no longer required to manually acquire and maintain dependencies, which can be laborious and prone to mistakes.

Another essential trait is the ability to work on several projects at once. Due to this, developers are able to define and control massive projects with several modules and dependencies as a single entity. By ensuring that modifications to one module do not effect another, this makes maintaining big codebases simpler. Gradle is also very extensible, with a huge selection of plugins and extensions available right away to enhance the build process. Because of this, it is a very configurable tool that can be tailored to the unique needs of each Java project.

In conclusion, Gradle is a powerful and flexible build automation tool that accelerates the creation of Java applications. Developers frequently choose it because of its adaptability, simplicity, and potent capabilities. It is a helpful tool for managing large codebases because it supports multi-project builds and has a dependency management system.

Java Mock for testing- A simulated or artificial object used in software development to test the behavior of other objects or methods is referred to as a "Java mock." To put it another way, a Java fake is a stand-in for a real object that enables programmers to test their code in a supervised setting[2]. In order to evaluate interactions between objects, which might be challenging or time-consuming to test with real items, mock objects are frequently used. A mocking framework like Mockito or EasyMock can be used to construct a Java mock. Without writing complicated code, these frameworks let developers rapidly and easily construct mock objects.

To make sure that code is operating as expected, test-driven development (TDD) and behavior-driven development (BDD) practices frequently employ Java mocks. Developers can isolate particular sections of code and test them using mocks without affecting other components of the application. A Java mock, to put it simply, is a simulated object used in software development to test the functionality of other objects or methods. It is an effective tool for code testing and making sure that programmes are running properly.

Pub-Sub- Additionally, the Rocket framework utilizes various pub-sub systems for asynchronous messaging and event-driven architectures. These systems encompass Kafka, Avro, EventHub, EventBridge, and Amazon SNS. Extensive research has been conducted on these systems, exploring aspects like performance, scalability, reliability, and security.

In conclusion, thorough research has been conducted on the technologies employed in the Java Framework. The effectiveness of pub-sub systems as tools for software development and distribution has been thoroughly investigated. These technologies are incorporated into the Java platform, which offers a dependable and scalable platform for creating software systems such as web applications.

Redis- Keeping common in-memory data structures stored Redis is becoming increasingly popular. The primary benefit of Redis is its capacity to store and retrieve data from memory, enabling quick access to data. Strings, hashes, lists, sets, and sorted sets are among the data structures that Redis supports. Redis is a well-liked solution for numerous use cases, including caching, messaging, and real-time data processing, because to its adaptability. Redis enables data replication and sharding, providing fault tolerance and high availability.

Because Redis can store data in memory, it is also ideal for use cases where quick access to data is crucial. Redis has also been widely used by many different industries, including e-commerce, social networking, and gaming. Its fast performance and flexible data structures make it the ideal platform for applications that require real-time data processing and analysis.

Cassandra- Cassandra's ability to grow horizontally across several nodes, offering high availability and fault tolerance, is its main advantage. Cassandra excels in processing enormous volumes of data due to its distributed architecture, which makes it simple to add or remove nodes to accommodate changing data needs.

Cassandra's capacity to manage enormous amounts of data at quick read and write speeds is one of its main advantages. Cassandra is the best option for applications that need quick and effective data writes since it has a data model that is designed for workloads that involve a lot of writing. Finance, healthcare, and social media are just a few of the sectors where Cassandra has been widely implemented. It is the perfect platform for managing massive volumes of data in real-time due to its scalability, quick read and write rates, and distributed architecture.

Docker- Since its introduction in 2013, the well-known containerization platform Docker has grown in popularity. The ability to bundle and distribute software in containers is Docker's key feature. Because they separate software from the underlying system and guarantee that it operates consistently regardless of the host environment, containers are a quick and effective way to run programmes. The portability of Docker is one of its main advantages. Developers may more quickly test and deploy software across several platforms since Docker containers are simple to move between various environments[3]. A central repository for storing and distributing containers is another feature of Docker that makes it simpler for teams to work together on software development projects.

The scalability of Docker is an additional advantage. Applications may be scaled to suit changing demand because of Docker's simplicity in managing many containers. Additionally, developers can manage and automate container deployments thanks to Docker's interface with well-known orchestration technologies like Kubernetes. Docker also includes a number of tools and features that make container management simple. For instance, Docker Compose, a built-in orchestration tool, enables developers to create multi-container applications and control their deployment and scalability. Additionally, a variety of security features in Docker make it simple to create safe containerized programmes. For instance, it facilitates user namespaces, allowing containers to run with their own user and group IDs, thereby preventing unauthorized access to system resources.

Postman- The API (Application Programming Interface) platform Postman is a stand-alone tool for testing applications that creates, tests, designs, modifies, and documents APIs. For sending and examining HTTP requests and answers, it offers a straightforward Graphical User Interface. For testing purposes, utilizing Postman eliminates the need to create any HTTP client network code. Instead, we create collections of test cases and let Postman communicate with the API. Almost all of the functionality that a developer would want is

contained in this tool. This tool can transform the API into code for languages like JavaScript and Python as well as perform other HTTP requests including GET, POST, PUT, and PATCH.

Postman's rich feature set and ability to work with various technologies make it a valuable tool for developers working with the java framework. The automation features of Postman are another important aspect. API testing can be automated by developers using Postman, which can save testing time and increase testing precision. A crucial tool for DevOps organizations, Postman may also be linked with well-known CI/CD programmes like Jenkins and Travis CI.

Additionally, Postman offers a large selection of pre-built integrations with well-known APIs like Twilio, Stripe, and Slack. Developers may easily create and test APIs that interface with these well-known services thanks to these integrations. Postman's simplicity of use is one of its key advantages. The platform's user-friendly interface makes it straightforward and intuitive for developers to construct and test APIs without needing much technical knowledge. Additionally, Postman offers thorough support materials and documentation to aid developers in getting up and running quickly.

Swagger- For creating, outlining, and testing RESTful APIs, many people use Swagger. We will look into Swagger's numerous characteristics, advantages, and disadvantages in this literature review, as well as its uses in a variety of sectors. The ability of Swagger to produce interactive documentation for APIs is one of its key advantages. This documentation offers developers a simple interface for exploring and comprehending API endpoints, parameters, and answers. Java, Python, and JavaScript are just a few of the programming languages that Swagger supports for code creation. Through less manual coding needed to create an API, this can hasten the development process.

Swagger enables API design and administration functions in addition to documentation and code creation. For instance, Swagger enables developers to specify and enforce API standards, such as parameter formats and naming conventions. The quality and consistency of APIs throughout an organization may be enhanced as a result. Swagger also includes a number of useful tools for testing and troubleshooting APIs. It comes with an in-built testing framework that allows developers to rapidly and simply test their APIs to ensure that they are working properly. Furthermore, numerous debugging tools, such as logging and tracing support, are provided, making it simple to locate and fix API issues. Swagger also offers API

versioning, which is a useful feature. Developers can declare a variety of API versions using Swagger, each with its own set of endpoints and parameters. This allows for easy backward compatibility while providing extra features and capabilities.

Git and Github- Developers can follow changes to their code over time using the distributed version control system known as Git. It offers tools like branching, merging, and tagging that let developers manage many versions of a project and work together on code. Git's speed and efficiency, which enable developers to work on vast codebases without experiencing productivity hiccups, are among its key advantages. In contrast, GitHub is a web-based platform that offers hosting for Git repositories in addition to extra tools for teamwork and project management. Developers can monitor issues and defects on GitHub, share their code with others, and participate in open-source projects.

Chapter 3

System Development

3.1 Framework Used

SpringBoot

A well-liked open-source framework called Spring Boot is used to create web apps in Java. It is frequently used because of how simple it is to use, how quickly it can be developed, and how well it can interface with other Spring frameworks. The framework offers a subjective method for creating applications, allowing the developer to concentrate on business logic rather than setup. By cutting down on the amount of time needed for setup, configuration, and deployment, Spring Boot increases developer productivity[4]. Along with these built-in capabilities and connectors, the framework offers starter packs and auto-configuration, which streamline development and cut down on external dependencies. Due to its optimized application startup and runtime behavior, it had higher performance. The framework also has integrated caching, which can enhance the performance of applications even more. It also offers a testing-friendly environment with support for end-to-end, unit, and integration testing. The framework offers several tools and modules, including JUnit and Mockito, that make testing easier and raise code quality.

3.2 Technical Requirements

3.2.1. IntelliJ

For Java developers, IntelliJ IDEA is a well-liked integrated development environment (IDE). It was created by JetBrains and is renowned for its strong features, approachable interface, and comprehensive support for a variety of technologies. Another significant advantage of IntelliJ is its support for version control systems. It supports Git, Subversion, and other version control systems, which makes it simple to manage code changes and communicate with other developers. Additionally, it supports code reviews and pull requests, enabling teams to study and combine code changes more quickly.

A number of software development and deployment tools are also included with IntelliJ. It supports widely used deployment platforms like Docker and Kubernetes as well as build tools like Maven and Gradle. In a variety of settings, from local development machines to cloud-based production environments, this makes it easier to create and deliver software.

One of IntelliJ's most useful features is the ability for plugins. With thousands of them freely available, it offers a robust ecosystem for plugins that enables developers to increase the IDE's functionality. It is simple to modify the IDE to meet the unique requirements of a project thanks to the plugins for well-known frameworks, code analysis tools, and productivity tools that are included.

A variety of productivity tools are also included in IntelliJ in addition to these characteristics, which can aid developers in working more effectively. The use of keyboard shortcuts, code templates, and a variety of code completion and suggestion tools can all help programmers create code more quickly and with fewer mistakes.

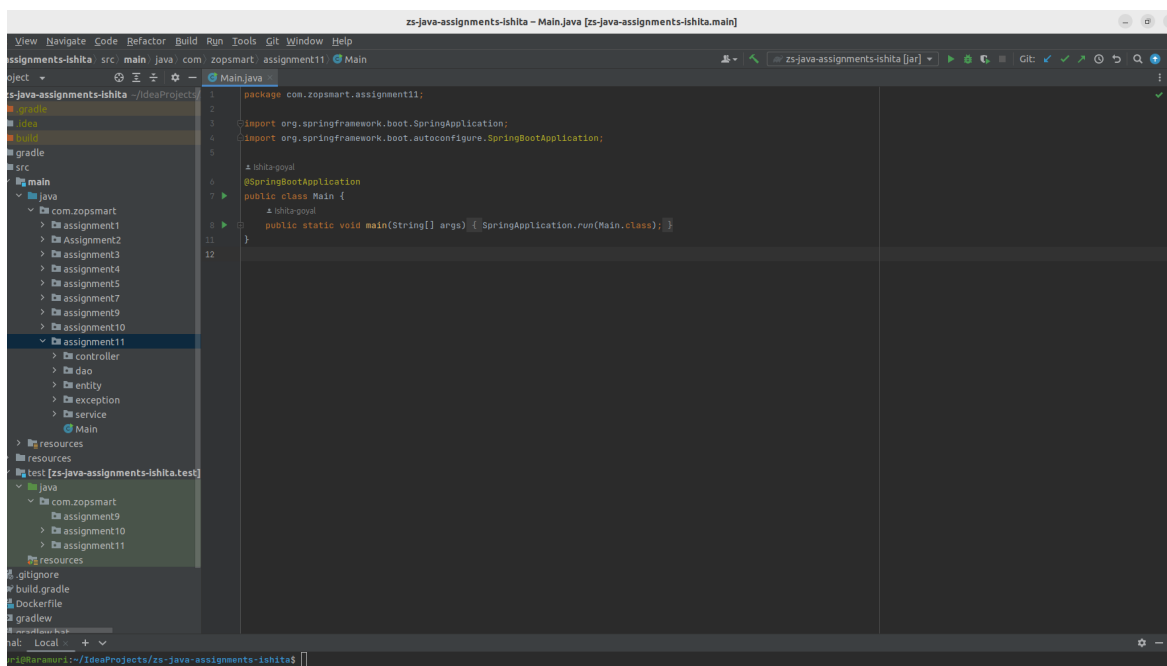


Figure 4. IntelliJ Idea

3.2.2. Postgres

With features like transaction support, crash recovery, and data integrity checks, PostgreSQL is created to be very dependable. Replication is also supported, enabling high availability and failover[5]. PostgreSQL supports huge databases and high concurrency since it is made to be scalable. Additionally, partitioning is supported, enabling dispersed databases and better performance. It is intended to be extendable and supports the use of unique data types,

operators, and functions. Additionally, it supports procedural languages like PL/SQL and PL/Python, enabling the development of sophisticated stored procedures.

3.2.3 Hardware Configuration

Table 1. Hardware Configuration

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

3.2.4 Software Configuration

Table 2. Software Configuration

Operating System Ubuntu
Language Java
Runtime Environment JRE

3.3 Model Development

Developing a model for a Spring Boot application involves several steps, which include:

1. **Creating the Model:** The creation of the application's data model is the first step in the model development process. In order to store and manipulate data, it is necessary to determine the entities and relationships that will be used. For example- for an ecommerce store's catalog , it would have a variety of categories and projects. The relation between them would be a category can have a variety of products like electronics can have mobile, laptops, television etc.
2. **Selecting a database for storing data :** Selecting a data storage place that works with Spring Boot is the next step. Relational databases like PostgreSQL and MySQL as well as NoSQL

databases like MongoDB and Cassandra are popular options. This project is using postgres for the management of data.

3. Create Entity class - Create the relevant Java entity classes in accordance with the recognised entities and their connections. These classes should contain the necessary constructors, methods, and fields to represent the entities.

4. Creating Dao interface- The ways in which a user interacts with a repository are specified by its interfaces. These interfaces can be built manually or with the aid of programmes like Spring Data JPA. Following that, the built-in support for database access in Spring Boot can be used to build the repository APIs.

5. Implementing Dao interface- 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. Creating Service class The service layer interacts with the dao layer and fetches and post data into the database with the help of crud api. The service layer stores the business logic of the application. It specifies the procedures to be followed when interacting with the repositories and carrying out the required tasks.

7. Creating Controller class - It interacts with the user and performs functions based on what the user demands. It interacts with the service class and api are defined in the controller class. It will manage incoming HTTP requests, then call the relevant service methods to carry out the required tasks.

8. Testing - It is crucial to test the model after it has been developed and put into use to make sure it functions as intended. Unit tests, integration tests, or end-to-end tests can all be used to do this. JUnit, Mockito, and other tools are available for testing.

3.3.1 Algorithm

1. Create a class called Category Entity that defines the category's properties, including its id, name, description, and any other pertinent fields.
2. Make a CategoryRepository interface that extends JpaRepository and includes methods for CRUD operations such findAll(), findById(), save(), deleteById(), and others.

3. Making a CategoryService interface will allow you to specify the business logic for categories' methods. This could provide tools for adding, reading, updating, and deleting categories
4. Create a class called CategoryController that will handle HTTP requests and responses for operations related to categories. In order to provide CRUD operations on the Category resource, it should implement RESTful endpoints .
5. The Create, Read, Update, and Delete methods should be used: In order to conduct CRUD operations on the Category entity, these methods should make use of the repository and service layers.
6. To prevent bad data from being stored, make sure that the proper data validation is done.
7. Run tests on each endpoint to ensure that they produce the desired results. An automated testing framework like JUnit is used.
8. Launch the application, and test the endpoints using swagger or postman.

3.3.2 Deployment

The foremost step before deploying an application is to create the jar of the file and create a docker image. Docker image is necessary so that the application can easily run on any of the applications. Since Docker images are self-contained, they have all the configurations and dependencies required to run an application. By doing this, the programme will function reliably across various contexts and platforms.

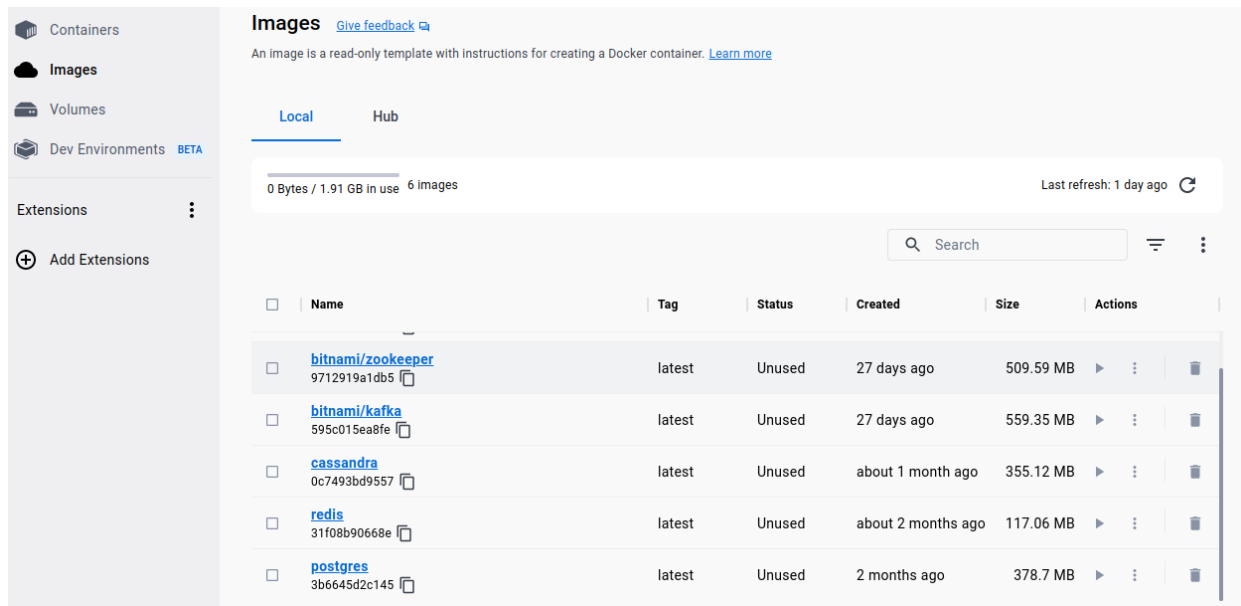


Figure 5. Docker

To deploy the application, AWS is used. Following steps are followed to deploy an application.

1. Create an EC2 instance- This requires selecting the appropriate EC2 instance type, creating and configuring the instance, configuring the security groups, and configuring the key pairs. Configure inbound and outbound rules and security groups to set up access to your application.
2. Install Docker on the instance- Inside containers, applications may be created, transported, and run thanks to a technology called Docker. Using the package management, you may manually download and install Docker on the EC2 instance.
3. Build image of docker - It is necessary to create a Dockerfile, which is a script containing instructions for building the Docker image. The application code must be added, the base image must be provided, and any necessary customizations must be configured. Before deploying the Docker image to the EC2 instance, the image must be submitted to a container registry. Docker Hub, Amazon ECR, and Google Container Registry are a few well-known container registries.
4. Building the spring boot application - A build tool like Maven or Gradle can be used to build your Spring Boot application. Make sure to create a WAR file for your application's packaging. The image is then pushed onto the docker hub and pulled back on the ec2 instance.

5. Pull docker on ec2 instance - After the Docker image is made available in the container registry, you can use the Docker CLI to pull it onto the EC2 instance.
6. Run docker container - You can use the Docker CLI to pull the Docker image onto the EC2 instance after it becomes accessible in the container registry.

Chapter 4

Experiment and Result Analysis

4.1 API Requests-

An API (Application Programming Interface) request is a web service request that enables two different software applications to connect with one another. To get or send data from one system to another, API calls are widely utilised. A common method for transferring data over the internet, the HTTP protocol, is used to submit API inquiries. The most popular HTTP methods for API inquiries are GET, POST, PUT, and DELETE. To initiate an API request, you must first know the endpoint of the API that you want to use, as well as any required parameters or headers. Once you have this information, you may send the request to the API using a programming language or a tool like Postman.

Different types of HTTP request-

1. Post

```
no usages  ↳ Ishita-goyal
@PostMapping
public ResponseEntity<Category> addCategory(@RequestBody @Valid Category category) throws ResourceException {
    logger.info("Add a category ");
    Category saveCategory = categoryService.addNewCategory(category);
    return new ResponseEntity<>(saveCategory, HttpStatus.CREATED);
}
}
```

Figure 6. Post request to add a category

```

/**
 * Adds product and sends returns status
 *
 * @param product object of product entity
 * @return response entity
 */
no usages  ▲ Ishita-goyal
@PostMapping
public ResponseEntity<Product> addProduct(@RequestBody @Valid Product product) throws ResourceException {
    logger.info("Add a product ");
    Product saveProduct = productService.addNewProduct(product);
    return new ResponseEntity<>(saveProduct, HttpStatus.CREATED);
}

```

Figure 7. Post request to add a product

The post request is made to add data to the database. Post request is made to add categories and products to the respective table and map them to each other.

2. Get

```

/**
 * Get all categories
 * @return ResponseEntity
 * @throws ResourceException Custom Exception
 */
no usages  ▲ Ishita-goyal
@GetMapping
public ResponseEntity<List<Category>> getCategory() throws ResourceException {
    logger.info("All categories are : ");
    return new ResponseEntity<List<Category>>(
        (List<Category>) categoryService.getCategory(), HttpStatus.OK);
}

```

Figure 8. GET request to get all categories


```

    Ishita-goyal
    @GetMapping
    public ResponseEntity<List<Product>> getProduct() throws ResourceException {
        logger.info("All product are ");
        return new ResponseEntity<List<Product>>((List<Product>) productService.getProduct(), HttpStatus.OK);
    }

    /**
     * get products based on category id
     * @param categoryId id of category
     * @return response entity
     * @throws ResourceException custom exception
     */
    no usages Ishita-goyal
    @GetMapping("/{categoryId}")
    public ResponseEntity<List<Product>> getProductByCategoryId(@PathVariable Integer categoryId) throws ResourceException {
        logger.info("Products for categoryId " + categoryId + " are");
        return new ResponseEntity<>(productService.getProductByCategoryId(categoryId), HttpStatus.OK);
    }

```

Figure 9. Get request to get all product and get by category id

Get request is used to fetch items from the database. For the product, get calls are being made in two ways. First get call is getting all the products from the database and second get call is getting the products based on specified product id since the category id and product id are mapped to each other.

3. Put

In RESTful APIs, a PUT request is a sort of HTTP request method used to update or replace an existing resource with new data. In other words, a PUT request is used to change the content of a server resource.

4. Delete

It is used to delete a resource from the server in RESTful APIs. A DELETE request, in other words, is used to remove a resource from the server. To make a DELETE request, the client

sends an HTTP request to the server containing the URL of the resource to be deleted. The server gets the request and deletes the requested resource.

5. Patch

```
/**
 * It is used to update the product
 * @param productId product id
 * @param productName product name
 * @return response entity that has the status
 * @throws ResourceException Custom exception
 */
no usages 1 Ishita-goyal
@PatchMapping(path = "{productId}")
public ResponseEntity<Product> updateProduct(@PathVariable("productId") Integer productId,
                                             @RequestParam(required = false) String productName,
                                             @RequestParam(required = false) Integer categoryId) throws ResourceException {
    logger.info("update product ");
    return new ResponseEntity<>(productService.updateProduct(productId, productName, categoryId), HttpStatus.OK);
}
```

Figure 10. Patch request to update a product

A patch request is used to make modifications to an existing resource rather than replacing it completely. To make a patch request, the client makes an http request to the server that includes the url of the resource as well as instructions on how to edit the resource. The server gets the request and makes the necessary changes to the resource. patch requests, unlike put requests, only change the parts of a resource indicated in the request. Patch requests are handy when you wish to make minor changes to a resource without having to replace the entire resource.

4.2 Exceptional Handling

```
no usages 1 Ishita-goyal
@ControllerAdvice
public class ApplicationExceptionHandler {

no usages 1 Ishita-goyal
@ExceptionHandler(value = MethodArgumentNotValidException.class)
private ResponseEntity<ErrorResponse> handleMethodArgumentNotValid(MethodArgumentNotValidException ex) {
    ErrorResponse apiException = new ErrorResponse(
        ex.getBindingResult().getFieldError().getDefaultMessage(),
        HttpStatus.BAD_REQUEST
    );
    return new ResponseEntity<>(apiException, HttpStatus.BAD_REQUEST);
}

no usages 1 Ishita-goyal
@ExceptionHandler(value = ResourceException.class)
private ResponseEntity<ErrorResponse> handleResourceException(ResourceException ex) {
    ErrorResponse apiException = new ErrorResponse(
        ex.getMessage(),
        ex.getHttpStatus()
    );
    return new ResponseEntity<>(apiException, HttpStatus.BAD_REQUEST);
}
}
```

Figure 11. Exceptional Handling

The `@ExceptionHandler` annotation gives us a lot of leeway when it comes to handling exceptions. To begin, all we need to do is annotate a method with the `@ExceptionHandler` annotation, either in the controller itself or in a `@ControllerAdvice` class, to use it.

4.3 Validations

```

  Ishita-goyal *
@Entity
@Table(name = "category")
@JsonIgnoreProperties({"hibernateLazyInitializer"})
public class Category {
    3 usages
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private int categoryId; @Column(unique = true)
    @NotEmpty(message = "name not valid")
    @NotBlank(message = "name not valid")
    private String categoryName;

    no usages  Ishita-goyal
    public Category() {
    }

    no usages  Ishita-goyal
    public Category(int categoryId, String categoryName) {
        this.categoryId = categoryId;
        this.categoryName = categoryName;
    }
}

```

Figure 12. Validations

Validations and annotations are added to make the fields and class to act a certain way. NotEmpty annotation confirms that a field cannot be left empty with adding it to the database. Id annotations referring to the id would be added automatically on posting a product or category to the database.

4.4 Swagger

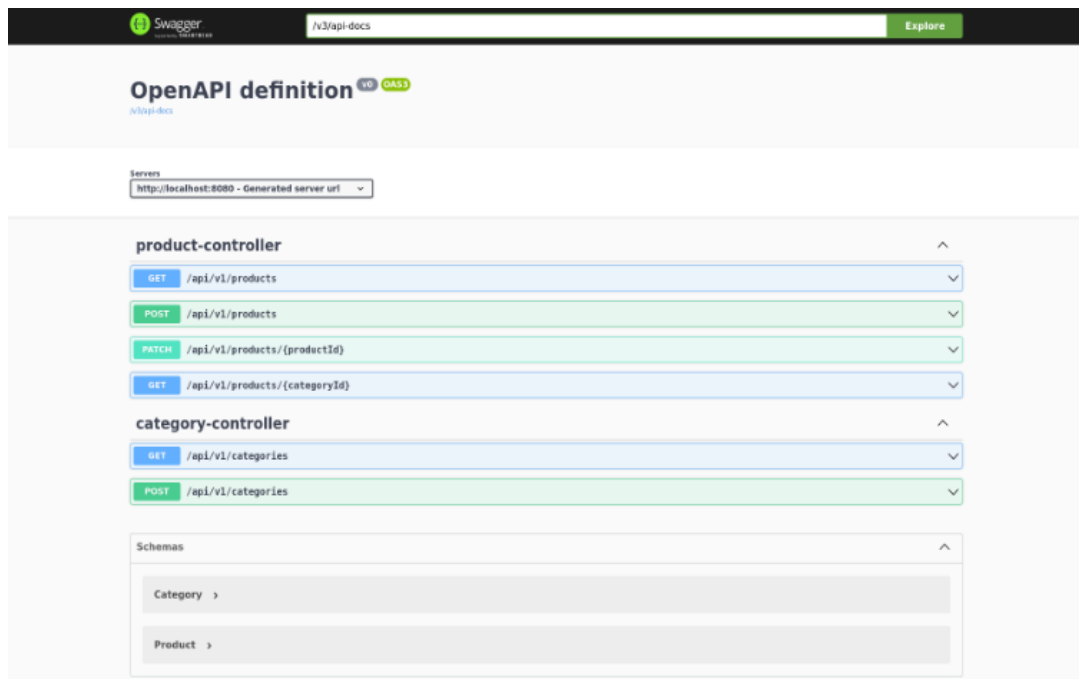


Figure 13. Swagger

Swagger has all the endpoints for the product and category. Users can hit any end point and from the controller it will call the service layer and then it will call the dao layer.

Category Api- Create a category-

The screenshot shows a REST client interface for a POST request to the endpoint `/api/v1/categories`. The interface includes a "Parameters" section with "No parameters" listed, and a "Request body" section where the content type is set to `application/json`. The request body contains the following JSON:

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

At the bottom of the interface, there are "Execute" and "Clear" buttons.

Figure 14. Input to add category

The screenshot displays the output of a REST client interface. It includes the following sections:

- Curl:** A terminal window showing the curl command used to execute the request:

```
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:** A section with a "Code" of 201 and a "Details" link.
- Response body:** A terminal window showing the JSON response:

```
{
  "categoryId": 1,
  "categoryName": "Grocery"
}
```
- Response headers:** A terminal window showing the 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:** A section at the bottom for viewing the response history.

Figure 15. Output to get category

Edge case

If the category name is empty -

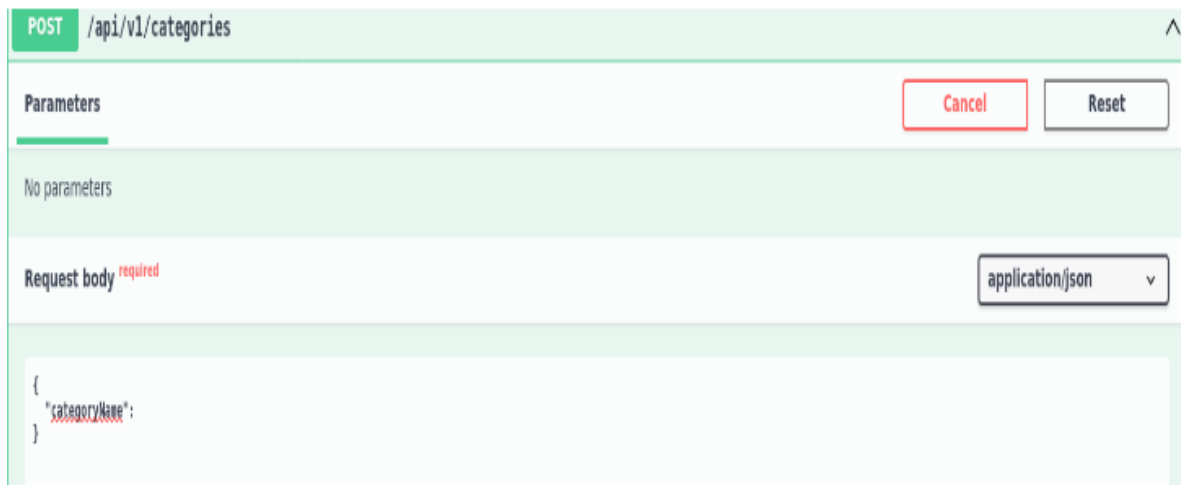


Figure 16. Edge case

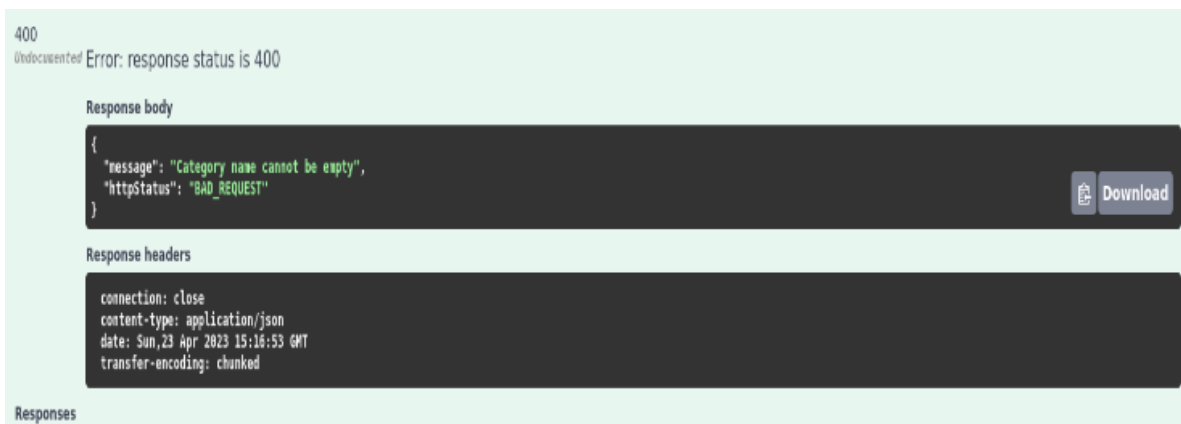


Figure 17. Edge case in case category name is empty

Get all categories-

The screenshot shows a REST client interface with the following sections:

- Method and URL:** GET /api/v1/categories
- Parameters:** No parameters. A "Cancel" button is visible.
- Buttons:** "Execute" and "Clear" buttons.
- Responses:**
 - Curl:**

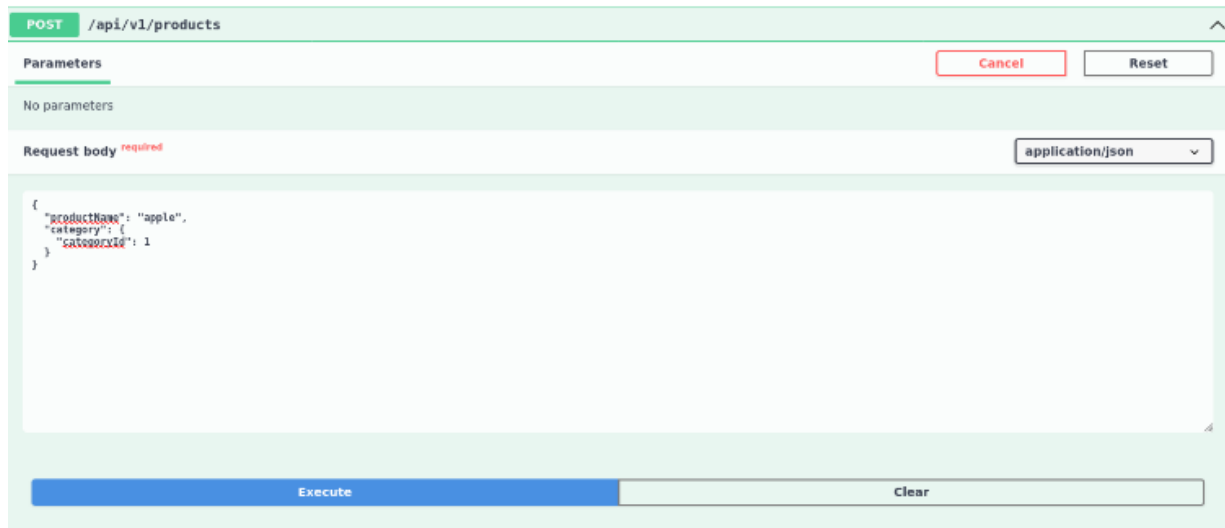
```
curl -X 'GET' \  
'http://localhost:8080/api/v1/categories' \  
-H 'accept: */*'
```
 - Request URL:** http://localhost:8080/api/v1/categories
 - Server response:**

Code	Details
200	<p>Response body</p> <pre>[{ "categoryId": 1, "categoryName": "Grocery" }]</pre> <p>Response headers</p> <pre>connection: keep-alive content-type: application/json date: Sun, 23 Apr 2023 12:12:57 GMT keep-alive: timeout=60 transfer-encoding: chunked</pre>

Figure 18. Output of get request

Do a get http request to get all the categories present in the database at that moment.

Product Api- Create a new product

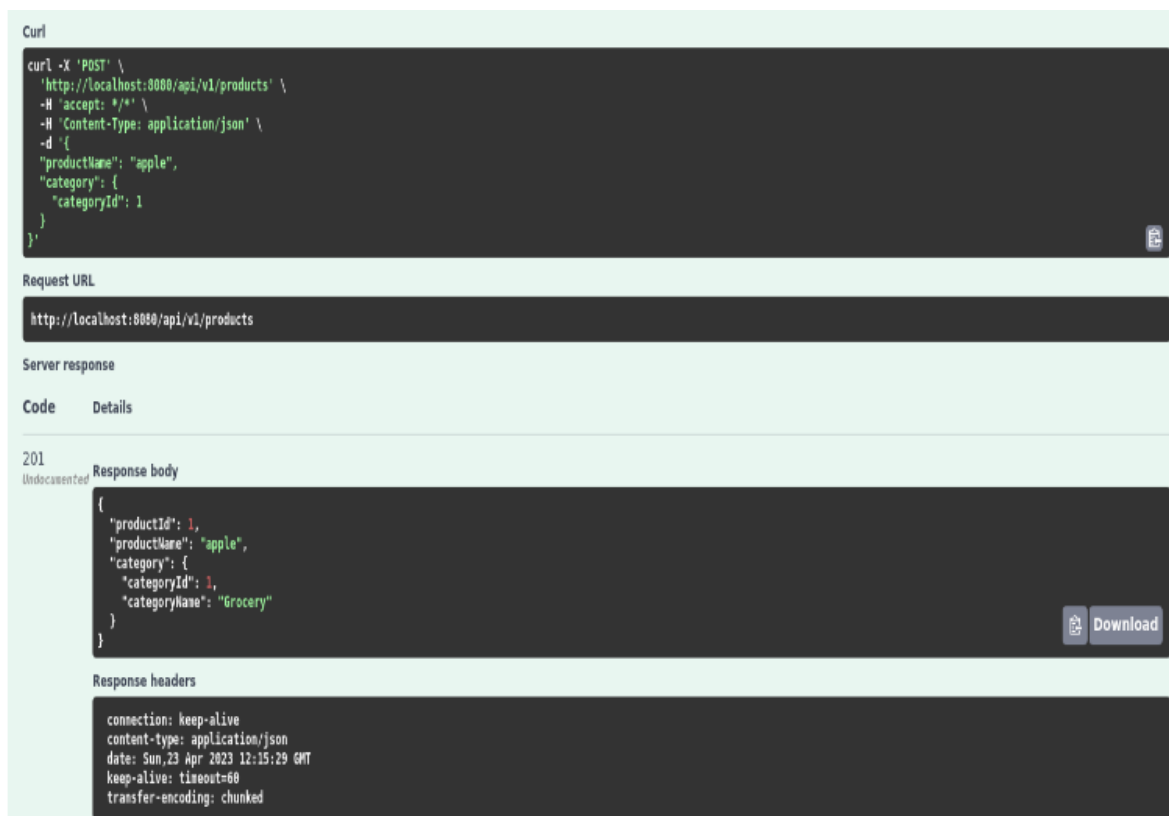


The screenshot shows a REST client interface for a POST request to the endpoint `/api/v1/products`. The interface includes a "Parameters" section with "No parameters" listed, and a "Request body" section where the content type is set to `application/json`. The request body contains the following JSON:

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

At the bottom of the interface, there are "Execute" and "Clear" buttons.

Figure 19. input to create a product



The screenshot displays the output of a REST client interface. It includes a "Curl" section with the following command:

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

The "Request URL" section shows `http://localhost:8080/api/v1/products`. The "Server response" section shows a `201` status code and a "Response body" containing the following JSON:

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

The "Response headers" section shows the following headers:

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

Figure 20. Output of product

Edge Case -

If the product name is empty -

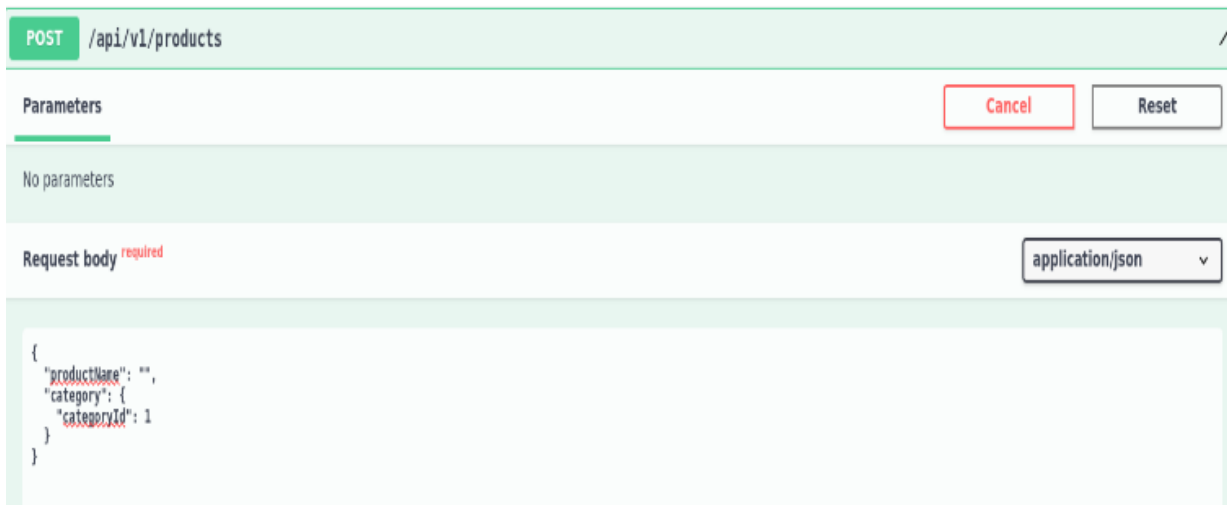


Figure 21. Edge case if productName is empty



Figure 22. Output of edge case

Get products by product ID -

The screenshot shows an API client interface for a PATCH request to the endpoint `/api/v1/products/{productId}`. The parameters section includes:

Name	Description
productid * required integer(\$int32) (path)	1
productName string (query)	cherry
categoryid integer(\$int32) (query)	1

Below the parameters are buttons for "Execute" and "Clear".

The Responses section shows the following details:

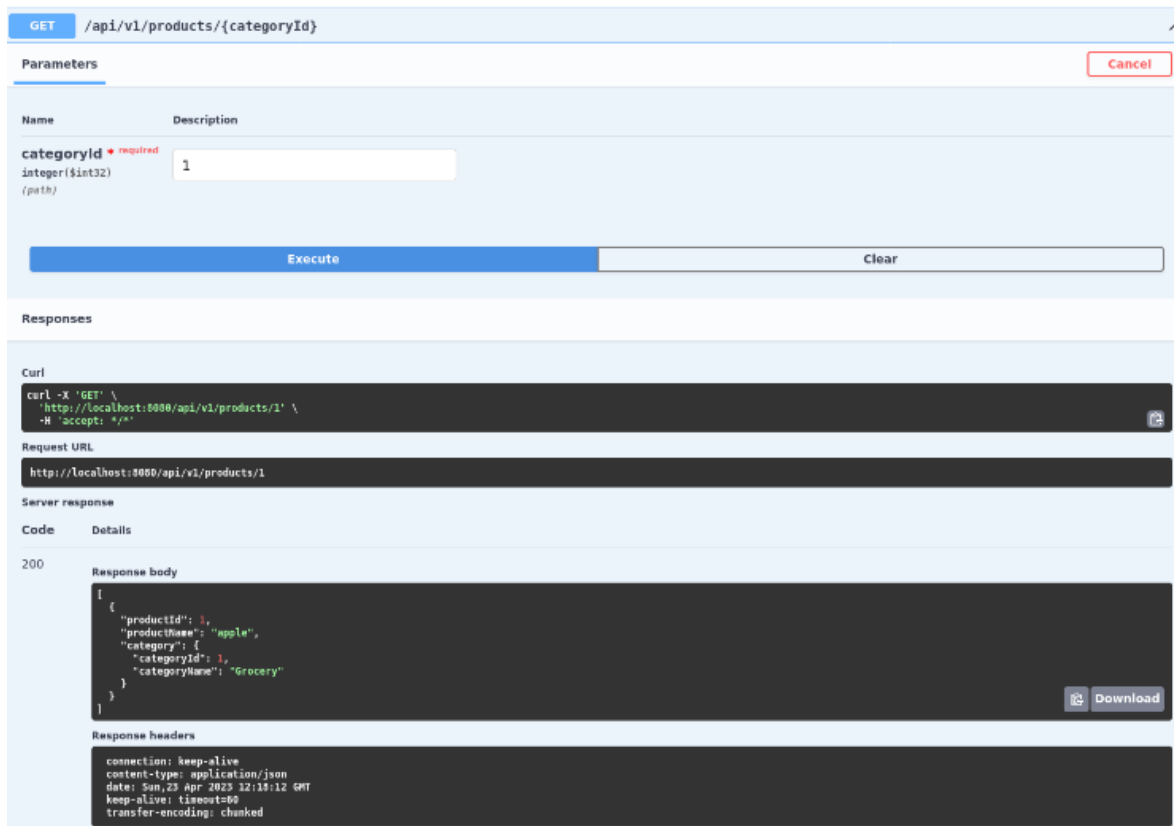
- Curl:**

```
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	<pre>Response body { "productId": 1, "productName": "cherry", "category": { "categoryId": 1, "categoryName": "Grocery" } }</pre>

Figure 23. Get product by product id output

Get product by category id



The screenshot shows a REST client interface for a GET request to the endpoint `/api/v1/products/{categoryId}`. The `categoryid` parameter is set to `1`. The response status is `200`, and the response body is a JSON object:

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

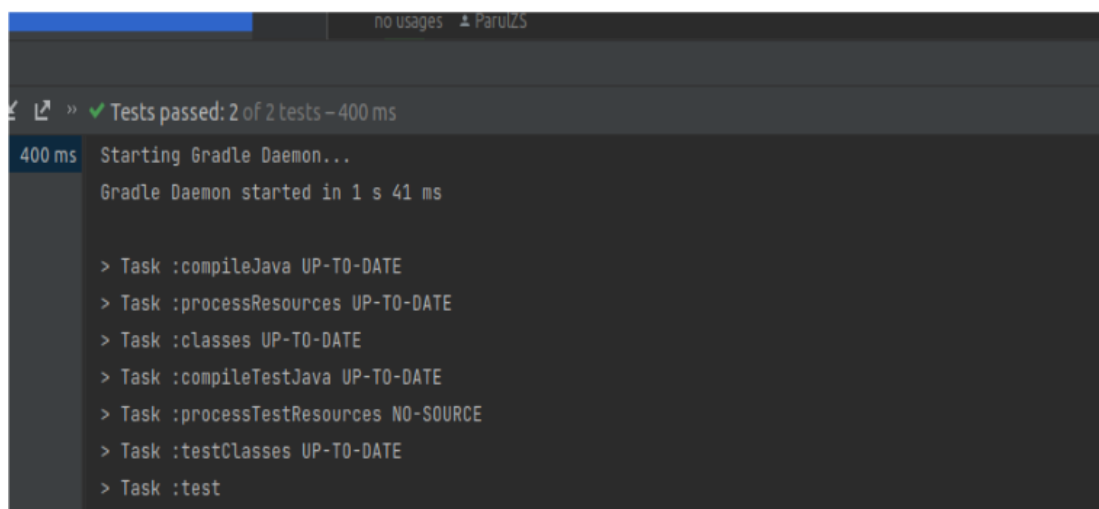
The response headers are:

```
connection: keep-alive
content-type: application/json
date: Sun, 23 Apr 2023 12:14:12 GMT
keep-alive: timeout=60
transfer-encoding: chunked
```

Figure 24. Get product by category ID

4.5 Unit testing using Mockito -

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
```

Figure 25. Category Controller test cases

```

✓ 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
21:20:09.838 [Test worker] DEBUG org.springframework.boot.test.autoconfigure.web.servlet.MockMvcTestContextBootstrapper - Neither @ContextConfiguration nor
@ContextHierarchy found for test class [ProductControllerTest]: using SpringBootTestContextLoader
21:20:09.851 [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}.
21:20:09.853 [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.
21:20:09.163 [Test worker] DEBUG org.springframework.boot.test.autoconfigure.web.servlet.MockMvcTestContextBootstrapper - Using ContextCustomizers for test class
[ProductControllerTest]: [DisableAutoConfigurationContextCustomizer, DisableObservabilityContextCustomizer, TypeExcludeFiltersContextCustomizer,
PropertyMappingContextCustomizer, Customizer, ReportsContextCustomizer, ExcludeFilterContextCustomizer, DuplicateJsonObjectContextCustomizer, MockitoContextCustomizer]
21:20:09.351 [Test worker] DEBUG org.springframework.context.annotation.ClassPathScanningCandidateComponentProvider - Identified candidate component class: file
[//home/parul/ideaProjects/zs-java-assignments-parul/build/classes/java/main/com/zopsmart/assignment11/Main.class]
21:20:09.353 [Test worker] INFO org.springframework.boot.test.context.SpringBootTestContextBootstrapper - Found @SpringBootTestConfiguration com.zopsmart.assignment11.Main
for test class com.zopsmart.assignment11.controller.ProductControllerTest
21:20:09.396 [Test worker] DEBUG org.springframework.boot.test.autoconfigure.web.servlet.MockMvcTestContextBootstrapper - Using TestExecutionListeners for test class
[ProductControllerTest]: [ServletTestExecutionListener, DirtiesContextBeforeModesTestExecutionListener, ApplicationEventsTestExecutionListener,
MockitoTestExecutionListener, DependencyInjectionTestExecutionListener, DirtiesContextTestExecutionListener, TransactionalTestExecutionListener,
SqlScriptsTestExecutionListener, EventPublishingTestExecutionListener, RestDocsTestExecutionListener, MockRestServiceServerResetTestExecutionListener,

```

Figure 26. Product controller test case

Test for service 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
22:28:09.461 [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"'.

```

Figure 27. Category service test case

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
22:28:59.891 [Test worker] ERROR com.zopsmart.assignment11.controller.ProductController - No Such Category Exists
22:28:59.908 [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"'.

```

Figure 28. Product service test case

Chapter 5

Conclusion

5.1 Conclusion

Finally, with Java, a three-layered architecture is a well-established way to develop scalable and maintainable software systems. The three levels - display, business logic, and data access - establish a clear separation of concerns, allowing for faster software development, testing, and maintenance. When developing a three-layered architecture, be sure that each layer has a clear and distinct duty and that they communicate with one other via well-defined interfaces. Furthermore, the usage of design patterns such as MVC or MVP might help to improve the architecture's scalability and maintainability. Overall, a three-layered architecture in Java is a strong and effective method to designing software applications, and it can help to produce high-quality software systems with careful planning and implementation.

In a nutshell, we have successfully created a highly functioning REST API that is both dependable and scalable using Spring Boot. The API's three-layer architecture makes it simple to maintain and adjust the codebase as needed. We have successfully stored and efficiently retrieved massive volumes of data by using a PostgreSQL database. The versatility of the API is one of its key characteristics; it can be applied to a variety of applications, such as e-commerce platforms and inventory management systems. It is straightforward to construct specialized interfaces for a range of applications since it is possible to quickly obtain all commodities, all categories, and all products under a certain category. Another useful feature that makes it easier to find and fix problems in complex applications is the logging capabilities of the API.

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 API's capability has been enhanced by the implementation of a product creation and update API. Developers can now programmatically create and update goods, saving time and effort. To sum up, our Spring Boot-based REST API is a flexible, scalable, and reliable tool for creating a wide range of applications. Its three-layer design, use of a PostgreSQL database, test cases, logging functionality, Request and Response Headers, and product creation and update capabilities make it an excellent choice for developers. We are confident that this API will help developers create high-quality applications that meet their business needs.

5.2 Future Scope

1. Cloud Development -

Spring Boot apps will most likely continue to expand to accommodate cloud-native development with technologies like Kubernetes, Docker, and serverless computing as the use of cloud computing technology grows.

2. Microservice Architecture -

The use of microservices in application design is growing in popularity, and Spring Boot applications will probably continue to develop with features like service discovery and fault tolerance to better accommodate microservices architecture.

3. Reactive Programming -

Software developers are increasingly using reactive programming techniques, and are anticipated to offer stronger support for these frameworks, which include Reactor.

4. Devops - Spring Boot applications will probably develop to offer better support for DevOps practices and Continuous Integration and Continuous Deployment (CI/CD) pipelines as the demand for quicker software delivery grows.

5. AI and ML -

In software development, machine learning and AI integration are becoming more and more common. By integrating with frameworks like TensorFlow and Keras, Spring Boot apps may offer improved support for these technologies.

6. Improved Security -

Software development is always concerned with security, and Spring Boot apps will probably keep developing to offer improved assistance for secure coding procedures and connection with security tools and libraries.

5.3 Limitations

The application implements only the backend part but front end can be done for the same to make the application more attractive and user friendly.

References

1. *Java documentation - get started* (2023) *Oracle Help Center*. Available at:
<https://docs.oracle.com/en/java/>
2. Paraschiv, W. by: E. (2022) *Getting started with Mockito @Mock, @Spy, @Captor and @injectmocks, Baeldung*. Available at:
<https://www.baeldung.com/mockito-annotations>
3. *Overview* (2023) *Docker Documentation*. Available at:
<https://docs.docker.com/get-started/>
4. *Microservices*. Available at: <https://spring.io/microservices> (Accessed: 12 May 2023).
5. *PostgreSQL*. Available at: <https://www.postgresql.org/about/> (Accessed: 12 May 2023).