ZOPSTORE WEB APP BUILT USING THREE LAYERED ARCHITECTURE

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

in

Computer Science and Engineering

By

Rishabh Bharota (191237)

Under the supervision of

Dr. Diksha Hooda

to

Department of Computer Science & Engineering and Information Technology

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

Certificate

I hereby declare that the work presented in this report entitled **"Zopstore Web App"** in partial fulfillment of the requirements for the award of the degree of **Bachelor of Technology** in **Computer Science and Engineering/Information Technology** submitted in the department of Computer Science & Engineering and Information Technology**,** Jaypee University of Information Technology Waknaghat is an authentic record of my own work carried out over a period from January 2023 to May 2023 under the supervision of **Dr. Diksha Hooda** (Assistant Professor (SG), Department of CSE, Jaypee University of Information Technology, Waknaghat).

[illegible]

Rishabh Bharota
191237

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

Dr. Diksha Hooda
Assistant Professor (SG)
Computer Science & Engineering

[illegible]

Mithali R. Shetty
Senior Lead Engineer
Zopsmart Technology
Dated: 11-05-2023

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 Capitalletters): ______________________

____

____

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 my heartiest thanks and gratefulness to almighty God for His divine blessing in making us possible to complete the project work successfully.

I am really grateful and wish my profound indebtedness to Supervisor Dr. Diksha Hooda, Assistant Professor (SG), Department of CSE Jaypee University of Information Technology, Wakhnaghat. It is their sincerity that prompted me throughout the project to do hard work using industry-adopted technologies. Her endless patience, scholarly guidance, continual encouragement, constant and energetic supervision, constructive criticism, valuable advice, 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. Diksha Hooda, Assistant Professor (SG), 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 might want to thank the various staff individuals, both educating and non-instructing, which have developed their convenient help and facilitated my undertaking.

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

Rishabh Bharota

191237

Table of Contents

List of Abbreviations

Abbr.	**Full Form**
IoT	Internet of Things
CRUD	Create Read Update Delete
API	Application Programming Interface
SQL	Structured Query Language
OAS	OpenAPI Specification
HTTP	Hypertext Transfer Protocol
RDBMS	Relational Database Management System
REST	Representational State Transfer
JSON	JavaScript Object Notation
SOAP	Simple Object Access Protocol
XML	eXtensible Markup Language
RPC	Remote Procedure Call
FaaS	Function as a Service
DVCS	Distributed Version Control System
EDA	Event Driven Architecture
CI/CD	Continuous Integration/Continuous Deployment
AWS	Amazon Web Services
GCP	Google Cloud Platform
HBS	Handler Business Store
CDN	Content Delivery Networks

List of Figures

List of Tables

Abstract

A web application may be made rather easily, but testing, structuring, cleaning, and maintaining the code is a barrier. To address this, we use the Go language and adhere to the Three Layered Architecture.

The three layers—handler, service, and datastore—are separate from one another. After receiving the request body, the handler layer parses it for any necessary information. The response is then written to the response writer after the service layer, where the program's entire logic is defined, has been called. Additionally, this layer converses with the datastore layer. It invokes the datastore layer after taking what it requires from the handler layer. All of the data is kept in the datastore layer. Any data storage device can be used. The only layer that interacts with the datastore is the use case layer. This is how we test each layer separately to make sure they don't affect one another.

CHAPTER 1
INTRODUCTION

1.1 Company Profile

Zopsmart is a software solution firm that offers you all the resources you need to launch an online store. With the support of its product line, ZopSmart can help you create and manage the ideal company. It offers a variety of goods, including Smart Store Eazy and Smart Payment Gateway, among others. Customers of Zopsmart, which develops cutting-edge technology for the retail industry, range from independent furniture stores to large multinational chains. The company's solutions include an e-commerce platform, digital marketing, mobile commerce, automated logistics systems, management platforms, order management platforms, and internet of things (IoT) devices. It has its own framework to work on and offers software solutions to some of the best companies.

1.2 Introduction

It is a straightforward web application that uses the three-layered design to execute CRUD functions. Every layer of a programme has its own unit test. Additionally, there is a middleware implementation that verifies the http request before passing it to the server.

1.3 Problem Statement

To create testable, structured, clean and maintainable web applications by using industrial best
practices.

1.4 Objectives

An application or system can benefit from using a web API in an n-tier design in a number of ways. The following are some purposes or aims of using web APIs in an n-tier architecture:

- Modularity: Web APIs provide a modular approach to application development by separating the presentation layer from the business logic and data access layers. Greater flexibility, scalability, and maintainability of the application are made possible by this separation of concerns.
- Decoupling: By enabling various application layers to function independently of one another, Web APIs offer a decoupled approach to application development. As a result, layer dependencies are diminished, making maintenance, upgrades, and modifications simpler.
- Reusability: Web APIs can be utilized in a variety of applications and systems, increasing code reuse and cutting down on development costs and time.
- Scalability: Web APIs can be made to be more performant and better able to handle growing user demand and traffic. This is especially crucial for systems with heavy traffic or intricate data requirements.
- Security: Web APIs can be secured with various authentication and authorization procedures, enabling secure access to data and ensuring data privacy and integrity.
- Web APIs can offer compatibility between many platforms, languages, and systems, enabling the seamless integration of applications and systems.
- Standardization: Web APIs can follow established protocols and standards, fostering interoperability and simplifying system integration.

1.5 Motivation

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

1.6 Methodology

There are various phases involved in developing a web API using a 3-tier architecture, a MySQL database, Swagger documentation, Postman collection, and monitoring using Grafana and Prometheus. The steps in the procedure are as follows:

- Design the API: Define the API by specifying the HTTP methods, input and output formats, API endpoints, and authentication procedures. The OpenAPI Specification (OAS), a specification for creating APIs, is often used for this.
- Create the backend: Build the backend utilizing a web framework like Node.js or ASP.NET Core, create the backend application. Three layers will make up the backend application: a presentation layer, a business logic layer, and a data access layer. The business logic layer will handle the business logic, the data access layer will communicate with the MySQL database, and the presentation layer will display the API endpoints.
- Construct a MySQL database: Design the tables and relationships needed for the application, then create the MySQL database.
- Implement Swagger documentation: Use Swagger UI to automatically produce the API documentation from the OAS specification as you implement Swagger documentation. Each API endpoint's description, input criteria, and output format will be included in the documentation.
- Create Postman collection: Develop a Postman collection that can be used for testing and debugging the API. Each API endpoint, HTTP method, input parameter, and anticipated outcome should be included in the collection.
- Activate Grafana and Prometheus monitoring: Configure Grafana and Prometheus to track the usage, performance, and error rates of the API.

The metrics gathered by Prometheus can be shown on a dashboard using Grafana.

- Test the API: Use the Postman collection to test the API and make sure that all of the endpoints are functioning as they should.

In conclusion, there are a number of processes involved in developing a web API utilizing a 3-tier architecture and a MySQL database, as well as creating Swagger documentation, a Postman collection, and monitoring with Grafana and Prometheus. These actions include designing the API, building the backend, putting MySQL in place, putting Swagger documentation in place, making a Postman collection, putting Grafana and Prometheus monitoring in place, testing the API, and putting it in a live environment.

1.6.1 Software Requirements

To build the web API that should be extensible, robust we require a fast server side language, an API platform for testing the API, an API monitoring tool, a database and a tool for the API deployment. For this project we will be using:

Ubuntu

Based on the Debian distribution, the popular open-source Linux operating system Ubuntu was created. Here are some of Ubuntu's main attributes and features:

1. User-friendly: With a graphical user interface that is accessible to both novice and experienced users, Ubuntu is made to be simple to use and user-friendly.
2. Customizable: With a large selection of themes, icons, and other customization choices available, Ubuntu may be tailored and configured to meet individual preferences and needs.

3. Secure: Security features like firewalls, encryption, and secure booting are all integrated into Ubuntu, which is renowned for its robust security measures.

Fig. 1.1 Ubuntu Icon

Overall, Ubuntu is a flexible and configurable operating system that may be used for a variety of applications, including personal computers, servers, and cloud computing.

Golang(Server Side Language)

Google created the open-source programming language Go, sometimes referred to as Golang, in 2007. It was developed to fill the demand for an effective, simple-to-write programming language that can manage big software projects.

Fig. 1.2 Golang Icon

Golang's essential characteristics include:

1. Strongly typed: Golang is a strongly typed language, which necessitates the explicit declaration of variable types. As opposed to catching problems at runtime, this aids with compilation.
2. Quick performance: Golang is a compiled language that is speed-optimized, making it the best choice for creating high-performance applications.
3. Support for concurrency: Golang comes with built-in concurrency support, enabling programmers to create effective concurrent code without the use of additional libraries.
4. Garbage collection: The garbage collector in Golang makes it simpler for programmers to build memory-safe code by automatically managing memory allocation and deallocation.
5. Cross-platform support: Golang is perfect for creating cross-platform applications because it supports a wide range of platforms, including Windows, macOS, Linux, and numerous mobile operating systems.

Golang is used for a variety of purposes, such as:

1. Web development: Golang is well suited for creating web apps and APIs because it comes with built-in HTTP support.
2. Distributed systems: Golang is a great choice for creating distributed systems and microservices since it supports concurrency and low-level networking.
3. Network programming: Golang is a great option for creating network applications like servers and proxies since it supports networking.
4. System programming: Golang's low-level features make it ideal for creating system-level programmes like file systems, operating systems, and device drivers.

5. Data processing: Golang is a great option for developing data processing applications, such as big data and machine learning, because it supports concurrency and has effective memory management.

Golang is a strong programming language that can be used to create a variety of applications, from system programming to web development. It is becoming a more and more common option for contemporary software development because of its performance, concurrency support, and cross-platform features.

Postman

Developers frequently use Postman as an API development tool to create, test, and document APIs. Since its initial release in 2012, it has grown to be a popular tool for API development.

Among Postman's most important attributes are:

1. Postman enables API testing by letting developers send queries and receive answers. The GET, POST, PUT, DELETE, and other request types are supported.
2. Automated testing: Postman enables programmers to use scripts to automate API testing. This can reduce testing time and provide consistent API testing.
3. Collaboration: Postman enables developers to collaborate with team members and exchange APIs. It offers team management, documentation, and version control capabilities.
4. Mock servers: Postman enables programmers to build mock servers that mimic API responses. Without having to rely on outside services, this is helpful for testing APIs.
5. API documentation can be created and published using Postman by developers. This can aid with the usage of the API for other developers.

6. Collection runner: Postman makes it simpler to test and debug APIs by enabling developers to launch a group of queries with a single click.

All things considered, Postman is a strong tool for API development that saves developers time and raises the caliber of their APIs. It is a preferred option for developers all over the world because of its features for testing, collaboration, documentation, and automation.

Prometheus

In 2012, SoundCloud created Prometheus, an open-source monitoring and alerting platform. In addition to offering robust query and alerting features, it is made to gather and store time-series data. Prometheus has a number of important elements, including:

1. Data Collection: Prometheus is made to gather time-series data from a range of sources, such as HTTP endpoints, application metrics, and system metrics.
2. Data storing: Prometheus saves time-series data in a tailored database that is fast for writes and queries.
3. Querying: Users can run sophisticated queries and aggregations on time-series data using Prometheus' robust query language, PromQL.
4. Alerting: Prometheus offers a versatile and strong alerting system that enables users to create alerts based on unique criteria and thresholds.
5. Exporters: Prometheus includes a robust exporter ecosystem, which consists of plugins that let users gather metrics from a range of sources, including databases, web servers, and third-party services.
6. Visualization: Prometheus may be connected with well-liked visualization programmes like Grafana to deliver detailed time-series data visualizations.

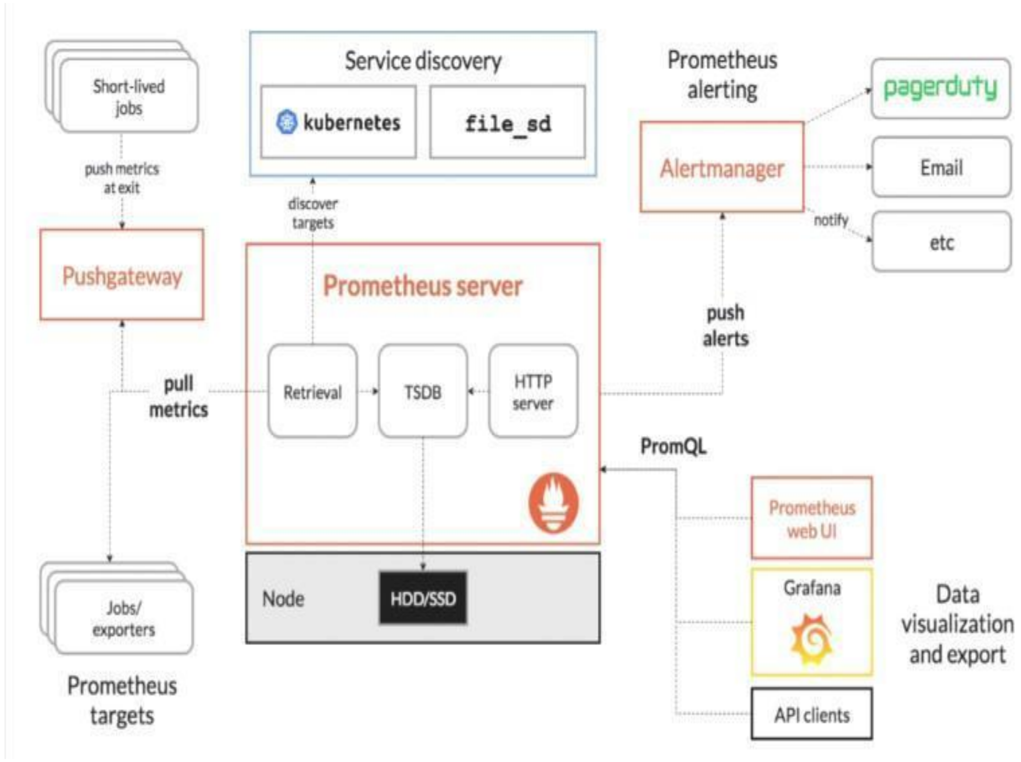


Fig. 1.3 Prometheus Architecture

Organizations of all sizes use Prometheus to keep an eye on their systems and programmes. It works particularly well in situations that are cloud native and may be used to keep an eye on containerized applications that are operating in Kubernetes clusters. Prometheus is a crucial tool for preserving the dependability and availability of contemporary systems thanks to its strong querying and alerting capabilities.

MySQL

MySQL is a relational database management system (RDBMS) that is open-source and was originally made available in 1995. As the backbone of numerous web applications, content management systems, and other software programmes, it is one of the most widely used databases in use today.

Some of MySQL's important characteristics include:

1. Relational database: MySQL is a relational database, which implies that relationships can be made between tables and that data is kept in tables with columns and rows.
2. Scalability: MySQL is made to be extremely scalable, enabling it to handle significant data loads and traffic volumes.
3. Cross-platform compatibility: MySQL is a flexible option for developers because it is available for a variety of operating systems, including Windows, Linux, and macOS.
4. High availability: MySQL offers capabilities like replication and clustering, which can increase database availability and decrease downtime.
5. Security: MySQL has several security features, such as support for encryption, encrypted connections, and user authentication.
6. Support for different programming languages: MySQL is a well-liked option for online applications because it can be accessed from several languages, including PHP, Java, and Python.

In conclusion, MySQL is a strong and adaptable database management system that is suitable for a variety of applications. It is a well-liked option among developers worldwide due to its scalability, cross-platform compatibility, high availability, security features, and support for numerous programming languages.

Docker

Developers can put their apps and dependencies into containers using the open-source Docker framework. It is simple to deploy apps across many environments, including development, testing, and production, thanks to containers' portability and minimal weight.

Among Docker's most important attributes are:

1. Containerization: Using Docker, developers may put their applications and dependencies inside of portable, self-contained environments called containers.
2. Portability: Docker containers are portable and lightweight, enabling the deployment of programmes across many settings.
3. Consistency: Docker offers a straightforward and reliable method to package and deploy apps, ensuring that they function identically in various contexts.
4. Scalability: By running several containers on the same host or across multiple hosts, Docker makes it simple to scale applications up or down.
5. Security: A number of security measures are offered by Docker, including support for encryption, secure connections, and user authentication.

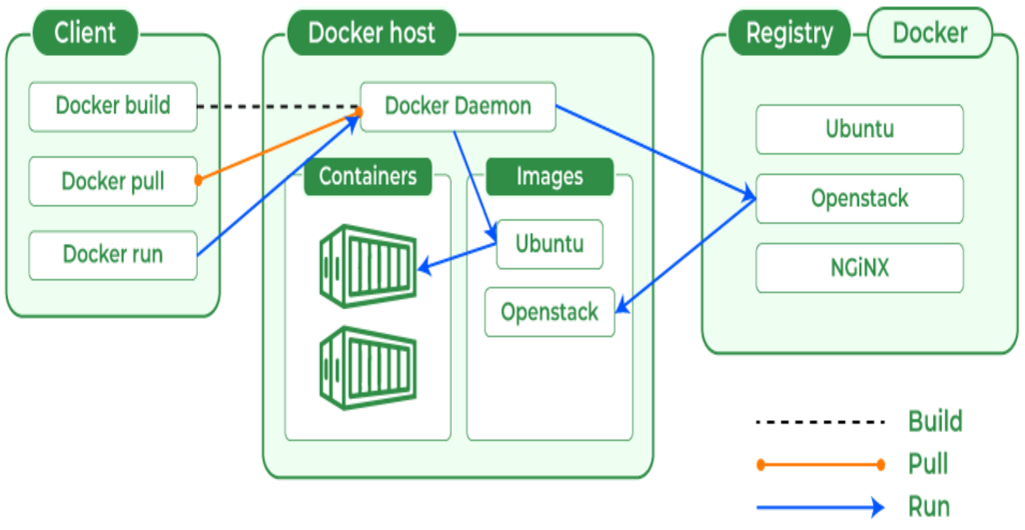


Fig. 1.4 Docker Architecture

Overall, Docker is a strong tool that is frequently utilized in the deployment and development of contemporary software. It is a well-liked option among developers worldwide because of its containerization, portability, consistency, scalability, and security capabilities.

Kubernetes

An open-source platform called Kubernetes, commonly referred to as K8s, automates the deployment, scaling, and management of containerized applications. The Cloud Native Computing Foundation (CNCF) now maintains it after Google initially built it.

By offering features like these, Kubernetes offers a mechanism to manage and coordinate containerized workloads at scale.

1. Automatic scaling: Depending on the workload, Kubernetes can automatically increase or decrease the number of container replicas.
2. Self-healing: Kubernetes can automatically identify and swap out broken containers, maintaining the application's high availability.
3. Service discovery and load balancing: Kubernetes gives containers a mechanism to find and connect to one another and can distribute traffic among a number of containers.
4. Rolling updates and rollbacks: Kubernetes has the ability to do rolling updates and rollbacks of containerized applications, guaranteeing zero downtime and lowering the risk of failure.
5. Configuration management: Kubernetes offers a method for controlling how application configurations and secrets are updated and stored securely.

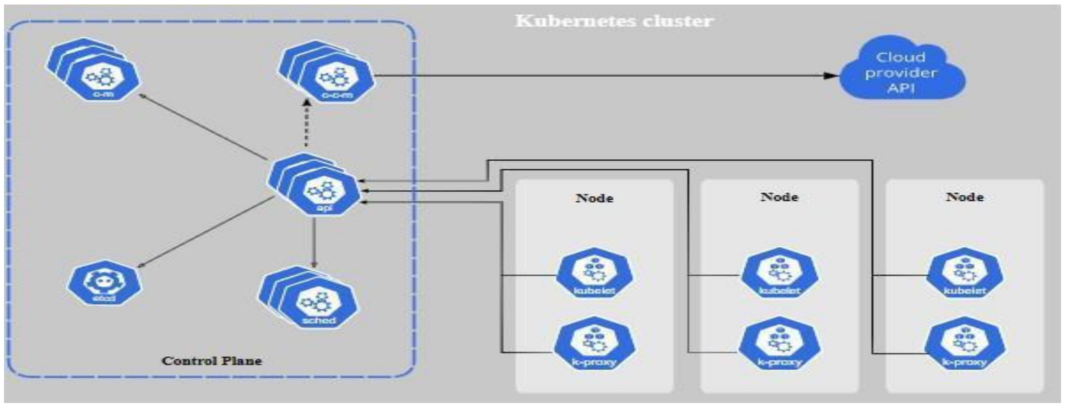


Fig. 1.5 Kubernetes Architecture

Overall, Kubernetes is a potent platform that is frequently employed in the deployment and development of contemporary applications. It is a well-liked option for administering containerized applications at scale due to its automation, scaling, self-healing, service discovery, load balancing, rolling updates, rollbacks, and configuration management features.

1.6.2 Proposed Approach

I am using the handler/business/store (HBS) architecture also referred to as the three-layered architecture. It is a variant of the three-layered architecture pattern that divides an application into three distinct layers or tiers, each of which has a particular duty.

Here is a quick breakdown of each layer:

1. The handler layer: Also known as the presentation layer, is in charge of processing user input and output. It collects user requests and then typically uses a user interface to give answers to the user. Such elements as web pages, forms, and UI controls are part of this layer.

2. Business layer (application layer): The business layer is in charge of carrying out the application's business logic. In order to retrieve or alter data, it coordinates with the store layer and handles user requests that are sent from the handler layer. Business objects, application services, and process components are included in this layer.

3. The store layer: Also known as the data layer, is in charge of storing and retrieving data from the underlying data storage system, such as a database or file system. It offers a uniform interface through which the business layer can access and modify data. Data access items, data transfer objects, and database connectors are included in this layer.

The best method to employ when creating an API (Application Programming Interface) relies on the needs of the application, the programming language being used, and the resources at hand. Some of the most popular methods for creating an API are listed below:

1. REST API: Based on the HTTP protocol, REST (Representational State Transfer) is a popular architectural paradigm for creating APIs. When manipulating resources, REST APIs use HTTP methods (GET, POST, PUT, DELETE, etc.) and return data in JSON (JavaScript Object Notation) format.

2. SOAP API: SOAP (Simple Object Access Protocol) is an internet protocol for transferring structured data. The message format is described by XML (eXtensible Markup Language) in SOAP APIs, and the service interface is described by WSDL (Web Services Description Language). Since they are dependable and enable cutting-edge features like transactions and security, SOAP APIs are frequently used in enterprise applications.

3. GraphQL API: Developed by Facebook, GraphQL is a query language for APIs. Clients can describe the precise data they require in their queries using GraphQL APIs, which employ a single endpoint to receive requests. Because of its adaptability, effectiveness, and capacity to minimize both over- and under-fetching of data, GraphQL APIs are growing in popularity.

4. Remote Procedure Call (RPC) API: RPC is a protocol for invoking functions or processes on distant servers. RPC APIs employ a client-server architecture in which the client requests something from the server, which then responds. Distributed systems frequently employ RPC APIs due to their effectiveness and support for remote procedure calls.

In this project I am following the REST architecture style because it is easy to use, scalable, and flexible. REST will be best suited for a small scale application.

The method chosen for developing a microservice relies on the needs of the application, the programming language being used, and the resources available. The following are some of the most popular methods for developing a microservice:

1. RESTful microservice: Based on the HTTP protocol, REST (Representational State Transfer) is a popular architectural paradigm for creating microservices. In order to modify resources, RESTful microservices use HTTP methods (GET, POST, PUT, DELETE, etc.) and return data in JSON (JavaScript Object Notation) format. Because of their ease of use, scalability, and flexibility, RESTful microservices are popular.

2. Messaging-based microservices: To communicate with one another, messaging-based microservices employ message brokers. Each service has the ability to post messages to particular subjects and subscribe to certain topics. Due to their fault tolerance and decoupling features, messaging-based microservices are widely employed.

3. Function-as-a-Service (FaaS) microservices use serverless infrastructure provided by cloud service providers to operate small, stateless processes. Each function runs for a brief amount of time and is activated by a particular event, such as an HTTP request. FaaS microservices are popular because of their scalability, efficiency, and simplicity of implementation.

4. A sidecar proxy is used by Service Mesh microservices to control communication between services. The sidecar proxy provides capabilities like traffic management, service discovery, and security while intercepting all service-to-service communication. Because of their advantages for visibility, robustness, and scalability, Service Mesh microservices are frequently employed.

As this project is a small scale project then it will be best to use RESTful microservices, as I will not be extending it into a big scale application and using these other methods will just add additional overhead.

1.6.3 Database

I am using MySQL as my database as it is easily scalable, secure and a relational database. As the tables in my database will be related to one another and MySQL handles these cases very well.

1.7 Organization of Report:

Chapter 1: This chapter gives a brief introduction to the Zopstore Web App, the objective of the system, and its motivation. In this chapter, we also discuss the problem statement of our project around which our project aim revolves. We also discuss the methodology or solution that has been used to create the API.

Chapter 2: This chapter contains literature surveys that provide a summary of different blogs and documentation which helped me with how the code will be structured and how to make it extensible and robust.

Chapter 3: In this chapter, my main aim was to explain the step-by-step method to build the project.

Chapter 4: In this chapter, I will provide the test result for different layers i.e Store layer, Business Layer and Handler Layer.

Chapter 5: This chapter contains a summary and conclusion of the report and resultant API that we got from the final project.

CHAPTER 2
LITERATURE SURVEY

2.1 Git and Github

GitHub and Git have evolved into indispensable tools for software development teams all over the world. Developers can follow changes to their code over time using the distributed version control system (DVCS) known as Git. A platform for Git repositories is provided by GitHub, a web-based hosting service, allowing developers to collaborate on projects, exchange code, and support open-source initiatives.

Git is a DVCS that enables programmers to set up local repositories on their computers, edit the code there, and then merge their modifications back into the main repository. Git's distributed architecture enables programmers to independently work on various areas of the codebase and merge their changes when they are complete. Git enables developers to work offline as well, making it a great option for those who don't always have dependable internet access.

Git's capability to track changes to the code over time is one of its most important advantages. Git keeps track of every modification made to the code, including who made it, when it was modified, and what was changed. With the help of this functionality, developers can undo changes, monitor the progress of the codebase, and find the root of any defects or other problems. By offering a cloud-based infrastructure for hosting, managing, and working together on Git repositories, GitHub elevates Git to a new level. Pull requests, code reviews, issue tracking, project management tools, and connections with other development tools are just a few of the many capabilities that GitHub offers to make developers' jobs easier.

Developers can automate their software activities using GitHub Actions, an automation tool offered by GitHub. It enables the development of unique workflows that can be sparked by various occasions, such as code modifications or pull requests, and carry out a variety of operations, such executing tests, constructing packages, or deploying software. Workflows can be modified to meet particular project requirements and are specified using YAML files. It is simple to interface with well-known tools and services thanks to GitHub Actions, which also offers a marketplace where developers can discover and use pre-built actions. In general, GitHub Actions speeds up the development process by automating repetitive operations, cutting down on errors, and fostering better teamwork.

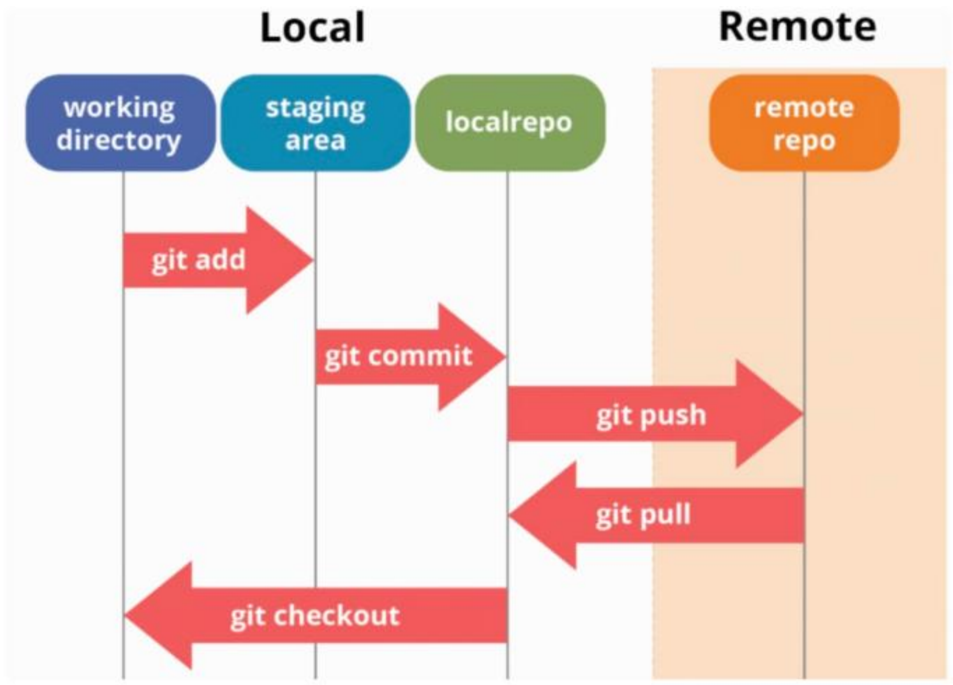


Fig. 2.1 Git Workflow

The social coding tools on GitHub are among its most important advantages. Developers can collaborate on open-source projects, share their code with the world, and support other people's efforts. Developers can promote their work on GitHub and establish their reputations among the developer community.Powerful developers can suggest modifications to a codebase and submit them for evaluation by other team members using GitHub's pull request tool.

Members of the team can review the suggested modifications, make suggestions for enhancements, and confirm that the changes don't introduce any bugs or problems thanks to this approach. A robust issue tracking system is also available on GitHub, enabling developers to keep track of bugs, feature requests, and other problems that come up during the development process. The issue tracking system enables team members to delegate responsibilities to one another, monitor development, and guarantee that issues are treated quickly and effectively.

The robust project management tools offered by GitHub give developers access to a number of capabilities for organizing tasks, deadlines, and milestones. Team members can maintain focus and make sure the project is moving forward according to schedule thanks to the project management tools. Another key advantage of GitHub is the way it integrates with other development tools. Popular development platforms like JIRA, Trello, and Slack are all easily integrated with GitHub, enabling developers to communicate and work together more efficiently.

Git and GitHub have been widely adopted by developers and organizations around the world as a result of their popularity. As their main version control and collaboration technologies, Git and GitHub have been adopted by several large organizations, including Microsoft, Google, and IBM. Additionally, GitHub hosts a large number of open-source projects, including well-known libraries and frameworks like React, Angular, and Node.js.

Finally, Git and GitHub have revolutionized the software development process by allowing programmers to communicate more successfully, work more productively, and produce higher-quality code. Developers can follow changes to the code over time thanks to Git's distributed version control mechanism, and GitHub offers a robust infrastructure for hosting, managing, and working together on Git repositories. Git and GitHub are vital tools for developers and businesses of all sizes because they together offer a strong platform for version control, collaboration, and project management in the software development industry.

2.2 REST

2.2.1 Introduction

Roy Fielding first developed the architectural design known as REST, or Representational State Transfer, in his doctoral thesis from 2000. A set of rules called REST is applied when creating and designing web services. Different systems can communicate with one another via the internet using RESTful web services because they are lightweight, quick, and scalable. The de facto industry standard for creating and implementing web APIs is REST.

2.2.2 Architecture of REST

Client-server architecture based on a set of restrictions or principles is known as REST. These guidelines, sometimes known as the "REST constraints," consist of the following:

1. Client-server architecture: In this setup, the client and server are kept apart and communicate with one another over a standardized interface.
2. Stateless: RESTful web services are stateless, meaning that between requests, the server does not retain any client context. The server does not save any session data or client state, and each request from the client is handled as a separate request.
3. Cacheable: RESTful web services can be cached, allowing clients to speed up performance by caching server answers.
4. Standard HTTP methods like GET, POST, PUT, DELETE, and PATCH constitute the basis of the unified interface that RESTful web services offer. It is simple for various systems to connect with one another because of this uniform interface.

5. Layered system: RESTful web services are constructed using a layered architecture, which implies that each layer has a distinct function and only communicates with other levels that are immediately above it.

2.2.3 Advantages of REST

Several advantages of RESTful web services include:

1. Scalability: Scalable RESTful web services are capable of handling high throughputs of requests and responses.
2. Versatility: RESTful web services are versatile and may be utilised with a variety of operating systems, devices, and programming languages.
3. Ease of use: RESTful web services are straightforward and simple to comprehend and use, making them the perfect choice for those who are new to web development.
4. Reliability: RESTful web services provide a high degree of availability and dependability.
5. Performance: RESTful web services can deliver high levels of performance since they are small, quick, and lightweight.
6. Reusability: RESTful web services can be employed in a variety of systems and applications since they are reusable.

2.2.4 REST Implementation

RESTful web services are implemented using conventional HTTP methods including GET, POST, PUT, DELETE, and PATCH. These techniques are used to carry out various actions on resources that the web service has made available. Each resource is identified by a special code known as a URI (Uniform Resource Identifier).

The procedures for putting into practise RESTful web services are as follows:

1. List the resources that the web service needs to expose.
2. Give each resource a special URI.

3. Specify the HTTP techniques that will be used to manipulate the resources.
4. Use your preferred programming language to put the techniques into practice.
5. Use a RESTful client, such as Postman, to test the web service.

2.3 Event Driven Architecture

The architectural design pattern known as event-driven architecture (EDA) makes it easier for various software components to communicate with one another by using events. When something interesting occurs in the system, like a new order being placed, a user logging in, or a file being uploaded, events are generated in EDA. Other components then take in these events and use them to change their own states or to start new actions.

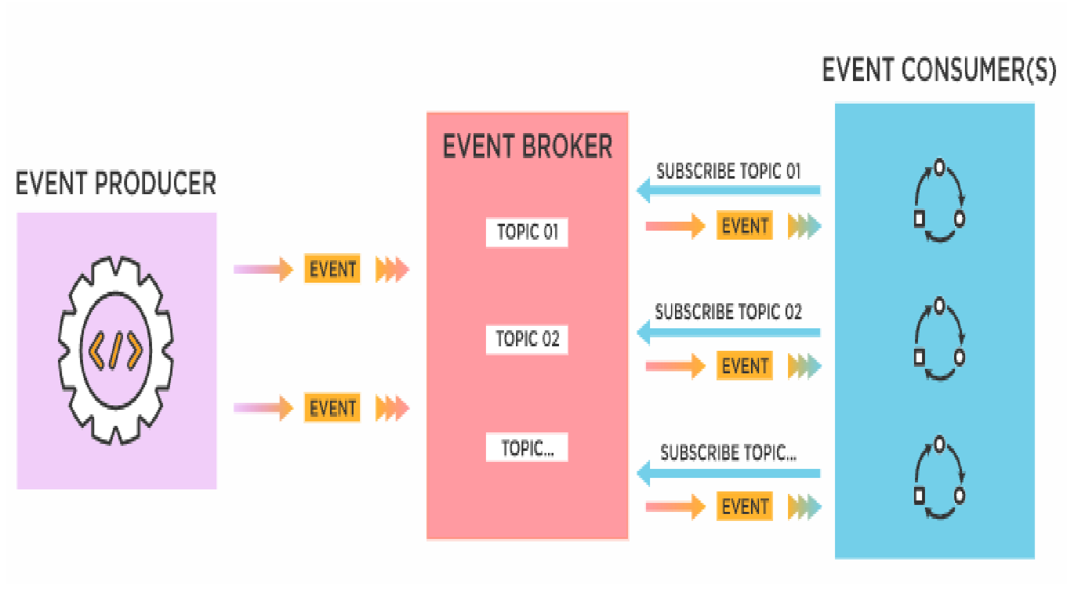


Fig. 2.2 Event Driven Architecture

Due to its capability to manage intricate and dynamic systems including several services and data sources, EDA has experienced a substantial increase in popularity in recent years.

2.3.1 Important Event Driven Architecture Concepts:

1. Sources and Consumers of the Event:Event sources and event consumers are the two basic categories of components in an event-driven architecture. Any element that produces events, like an application or a sensor, is referred to as an event source. Any component that monitors events and responds to them, such as updating a database or sending a notice, is referred to as an event consumer.
2. Event Channels: The lines of communication used to convey events from event sources to event consumers are known as event channels. Message queues, event logs, and event hubs are just a few examples of the various event channel kinds. The selection of an event channel is based on both the system needs and the characteristics of the events.
3. Event Sourcing: An application's state is stored as a series of events using the design pattern known as "event sourcing." Each event, such as a new order being made or a payment being processed, reflects a change in the application's state. It is possible to rebuild the state of the application at any time by saving the state as events. This is very helpful for event auditing, debugging, and replay.
4. CQRS: A design pattern called CQRS (Command Query Responsibility Segregation) divides an application's read and write actions into independent components. CQRS is frequently used in event-driven architectures to distinguish between event producers and event consumers. This lessens the dependency between components and gives the programme more flexibility while scaling.
5. Microservices driven by events: One common way to build event-driven architecture is through event-driven microservices. Each microservice in this method is in charge of handling a certain kind of event. When an event is formed, it is sent to the relevant microservice, which receives it, modifies its state, and, if necessary, generates new events. Because each

microservice can be separately deployed and scaled, this enables an architecture that is both extremely scalable and adaptable.

2.3.2 Event Driven Architecture Advantages

1. Scalability: Because it is decoupled, event-driven architecture has a great capacity for scaling. The system can handle high numbers of events without being overloaded, and components can be added or deleted without having an impact on the rest of the system. Because of this, it is the perfect architecture for systems with heavy traffic or erratic workloads.
2. Adaptability: High levels of flexibility are offered by event-driven architecture since new or removed components can be added or removed without affecting other system components. Because of this, it is simple to change the system over time to new requirements.
3. Fault Tolerance:Event-driven architecture is fault-tolerant because individual components can keep working even while others fail. The system can continue to function even if a component fails since the events it was in charge of can be forwarded to another component or tried again.
4. Loose Coupling: Because events are the only means through which components can communicate, event-driven design encourages loose coupling between them. This lessens the dependence between components and facilitates system development and maintenance.
5. Processing in Real Time: Due to the fact that events are broadcast as soon as they are generated, event-driven architecture enables real-time processing of events. For applications that need real-time updates, like financial trading or gaming, this enables almost immediate processing and reaction times.

There are various ways in which event-driven architecture is different from other architectural patterns, including 3-tier architecture or microservices architecture. The display layer, the application layer, and the data layer are the three layers that make up a three-tier architecture. The presentation layer manages user interaction, the application layer manages business logic, and the data layer manages data storage and retrieval. Each layer is in charge of a certain task.

The system is divided into a number of autonomous services in a microservices design, each of which is in charge of a certain task. Through the use of APIs, these services talk to one another, giving the system a great degree of adaptability.

The generation, detection, and response to events are the primary concerns of event-driven architecture, in contrast to 3 tier architecture and microservices design. The focus is on event processing and response, while it may integrate features of a 3-tier design or a microservices architecture.

2.4 Monitoring

To guarantee a web endpoint's availability, performance, and dependability, monitoring is crucial. It entails keeping track of and measuring a variety of parameters, including availability, error rates, and response times. The use of monitoring enables the early detection of possible issues, allowing for the quick mitigation or prevention of them. The significance of monitoring a web endpoint or service and some recommended practices for doing so will be covered in this article.

2.4.1 Monitoring a Web Endpoint or Service Is Important

Making sure a web endpoint or service is available and performing well is the main goal of monitoring it. The user experience, business operations, and income can all be negatively impacted by unavailability or performance difficulties on a web endpoint or service, which can be anything from a website, API, or database server. Early issue detection and prompt response are made possible via monitoring, which is essential for guaranteeing a fluid user experience and minimizing the impact on the business. Monitoring can assist with capacity planning, security, and compliance in addition to availability and performance. Organizations can expand their infrastructure appropriately by tracking indicators like resource utilization, traffic patterns, and security events. Additionally, companies may make sure that they are abiding by compliance standards and identify any security holes or vulnerabilities before they cause serious problems.

2.4.2 Monitoring a Web Endpoint or Service: Best Practises

There are various best practices that organizations can adhere to in order to monitor a web endpoint or service successfully. These consist of:

1. Establish Monitoring Objectives: Establishing monitoring objectives is the first step in keeping track of a web endpoint or service. This entails figuring out the measurements that are essential to the organization's objectives as well as the frequency and alarm threshold. To be effective, objectives must be SMART (specific, measurable, achievable, relevant, and time-bound).
2. Employ a Variety of Monitoring Tools: No one monitoring tool is capable of offering total visibility into a web endpoint or service. Application performance monitoring (APM), network monitoring, and log analysis technologies should all be used by organizations.These tools can offer several viewpoints and allow for thorough service monitoring.
3. Monitor from Various Locations: Monitoring from various locations is essential for ensuring that the service is accessible and operating at its best for users in various geographic locations. Businesses should employ monitoring tools that let them mimic user queries from various locations and gauge the availability and response time.
4. Configure Alerts and Notifications: When a problem is found, alerts and notifications should be configured to notify the proper parties. To enable prompt response, these warnings must be timely, pertinent, and actionable.

In order to make sure that alerts are functioning properly, they should also be examined frequently.

5. Examine and Act on the Data: Regular analysis of monitoring data is necessary to spot trends, patterns, and potential problems. To address any issues proactively, organizations should have a mechanism in place for analyzing the data and taking appropriate action. This entails figuring out what caused the problem in the first place and taking action to stop it from happening again.
6. Continuously Improve Monitoring: Monitoring is a process that never ends, so organizations should constantly work to make their monitoring procedures more efficient. This entails taking into account stakeholder comments, selecting fresh metrics to track, and assessing brand-new technologies and methods for monitoring.

2.4.3 Conclusion

For a web endpoint or service to be reliable, fast, and available, monitoring is essential. It lets businesses identify possible problems before they arise and act quickly to prevent or minimize them, delivering a flawless user experience and lessening the impact on the bottom line. Organizations can make sure that their web endpoints and services are operating at peak efficiency and fulfilling their objectives by adhering to best practices, such as defining monitoring objectives, using multiple monitoring tools, monitoring from various locations, setting up alerts and notifications, analyzing and acting on the data, and continuously improving monitoring.

2.5 Testing

To make sure that the programme satisfies the requirements and performs as intended, testing is a critical component of software development. A software architectural pattern called HBS (Handler-Business-Store) divides an application into three discrete layers: the handler layer, the business layer, and the store layer.

Each layer has unique duties, and testing each layer is crucial to ensuring the software's quality. Incoming client requests are handled by the handler layer, which is also in charge of validating input data and converting requests into business actions. The application's main functionality is implemented by the business layer, which also handles requests from the handler layer and interacts with the store layer to retrieve or persist data. Data storage, retrieval, and manipulation fall under the purview of the store layer. To store or get data, it interfaces with the business layer.

To assure the quality of the software in the HBS architecture, it is essential to test each layer separately. Let's go over the testing for each layer in more depth. The handler layer manages incoming requests and verifies input data. Validating the input data, the request handling logic, and the expected result are all part of testing this layer. The handler layer's most popular testing method is unit testing. The main goal of unit tests is to test each isolated function or method in a piece of code. Mocking frameworks can be used to segregate the test code and imitate external dependencies. It is possible to construct unit tests for the handler layer to evaluate the logic for handling requests, input verification, and output formatting.

For the handler layer, integration testing is also a critical testing method. Integration testing guarantees proper communication between the handler layer and external dependencies like databases or APIs. To verify the request processing logic with real data from external dependencies, integration tests for the handler layer can be built. The business layer is in charge of carrying out the application's main logic. Verifying that the business logic is sound and the outcome matches expectations includes testing this layer. The most popular testing method for the business layer is unit testing. It is possible to write unit tests to test distinct methods or functions. Mocking frameworks can be used to segregate the test code and imitate external dependencies. The business layer's basic logic, data processing, and data validation may all be tested using unit tests.

Another crucial testing method for the business layer is integration testing. Integration testing guarantees proper communication between the business layer and external dependencies like databases or APIs. It is possible to develop integration tests for the business layer to test the central logic using real data from external dependencies.

Store layer testing, Data storage, retrieval, and manipulation are the responsibilities of the store layer. Making sure the logic for data storage, retrieval, and manipulation is sound requires testing this layer. The most popular testing method for the storage layer is unit testing. It is possible to write unit tests to test distinct methods or functions. Mocking frameworks can be used to segregate the test code and imitate external dependencies. To test the logic for data storage, retrieval, and modification, unit tests for the store layer can be made. The store layer's testing methodology must also include integration testing. Integrity checks ensure that the store layer works properly with external dependencies like databases or APIs. To test data storage, retrieval, and manipulation using actual data from external dependencies, integration tests for the store layer can be written.

The HBS architecture offers a distinct separation of concerns, which makes testing each layer individually more doable. Testing is a crucial component of software development. The two most frequently utilized testing methods in the HBS architecture are unit testing and integration testing. Additionally, by employing automated testing tools, developers may speed up the testing procedure, lowering the risk of human mistake and enabling them to identify and address problems rapidly.

2.6 Continuous Integration / Continuous Development

A software development strategy called CI/CD, which stands for Continuous Integration/Continuous release or Continuous Deployment, tries to speed up the release of new features and updates to an application. Continuously and iteratively, the process entails automated testing, building, and deploying code updates to production settings. With this strategy, development teams may produce software more frequently and to a higher standard, which ultimately speeds up time to market and improves customer satisfaction.

The pipeline of the CI/CD is composed of a stage called Continuous Integration (CI). Every time a developer sends new code to a common repository, the process of automatically constructing, testing, and confirming the changes is referred to as continuous integration. In order to stop faults in the codebase from developing into larger problems later on, CI aims to find and repair them as soon as feasible. Development teams can quickly find and resolve errors, enhance code quality, and guarantee that the application is always in a release-ready state by continuously integrating code changes.

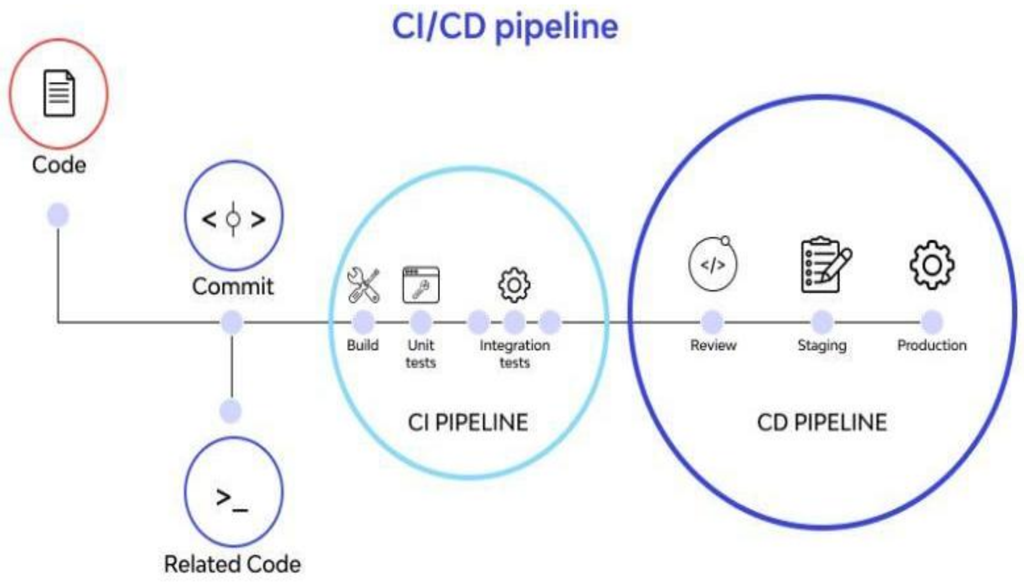


Fig. 2.3 CI/CD Pipeline

The second stage of the CI/CD process is Continuous Delivery (CD). It entails automating the deployment of the application to a testing and verification environment before it is made available to production. Making the deployment of new features and updates to production as seamless and dependable as feasible is CD's main objective. Development teams can lower the possibility of human mistake and make sure that new modifications are always properly and consistently deployed by automating the deployment process.

The last step in the CI/CD process is continuous deployment (CD). When new modifications are automatically sent to production after passing all relevant tests and inspections in the pipeline, it is referred to as this process. Using this strategy, development teams can quickly deploy new features and upgrades to the application, sometimes many times each day. The automated testing and deployment methods, as well as the codebase's quality, must be highly trusted for continuous deployment to be successful.

There are various advantages of CI/CD. First of all, it enables development teams to roll out new functions and updates more often, reducing the time it takes for an application to reach the market and responding to shifting consumer expectations. Secondly, by identifying and correcting mistakes early in the development process, it helps to raise the overall quality of the programme. Thirdly, by automating the testing and deployment processes, it lowers the possibility of human error and makes sure that fresh updates are constantly and dependably rolled out. Fourthly, it makes it possible for developers to obtain feedback on their code modifications more quickly and to make changes in real-time.

A combination of tools, procedures, and best practices are needed to implement CI/CD. Jenkins, Travis CI, CircleCI, GitLab, and Azure are some of the most popular CI/CD tools. The CI/CD pipeline's different stages, such as building, testing, and deploying code changes, can be automated using these tools.

Several crucial steps must be taken in order to implement CI/CD. To make the codebase dependable, testable, and manageable, development teams must first define a set of coding standards and best practices. Second, they must develop an automated testing method that examines both functional and non-functional requirements, as well as all facets of the application. Thirdly, a continuous integration process that automates the creation and testing of code changes on a shared repository needs to be put into place. The deployment of fresh modifications to staging environments must be automated, thus they must set up a continuous delivery pipeline. Finally, they must make sure the pipeline is secure, scalable, and reliable.

Finally, CI/CD is a crucial part of contemporary software development. It offers a platform for automatically carrying out the various steps of the development process, such as creating, testing, and deploying code modifications.

2.7 Database Migration

The process of moving data from one database to another, generally across successive iterations of the same database or between totally unrelated database management systems, is referred to as database migration. It is a crucial step in the software development process because it lets developers make changes to their database schemas and to their apps over time without impairing the user experience.

It is crucial to have a successful database migration procedure in place in the quick-paced software development environment of today, where new features and bug patches are regularly deployed. Tools for database migration are useful in this situation. By automating the data migration process from one database to another, these technologies cut down on the time and effort needed to complete the conversion. Database migration can be done using a variety of tools, each of which has its own special features and advantages. The most well-liked database migration tools are DbUp, Flyway, and Liquibase.

An open-source database migration programme called Flyway offers a straightforward and adaptable method of managing database migrations. Database migrations are handled by Flyway using a method akin to a version control system, where each migration is given a version number. Flyway tracks the current version of the database and automatically applies the appropriate migrations to bring the database up to date. Developers can design new migrations using SQL scripts or Java-based migrations.

Another free database migration tool is Liquibase, which enables programmers to control database schema changes with a single XML or YAML configuration file. Numerous databases are supported by Liquibase, which also offers a number of capabilities, such as the ability to generate change logs, rollback changes, and test migrations before implementing them.

A straightforward and dependable method of managing database migrations is offered by the.NET-based database migration programme DbUp. DbUp offers a number of capabilities, including the ability to apply changes in a transactional manner and to perform database backups before migrations are performed. It employs plain SQL scripts to express the database schema changes. There are a few standard practices that developers should adhere to when doing database migration, regardless of the tool they choose. Before beginning the transfer procedure, a backup of the current database must be made. By doing this, developers are guaranteed to have a backup plan in case the migration process encounters a problem.

Before implementing the migration procedure on the production database, developers should extensively test it. This entails applying the migration in a staging or development environment and testing that the application functions as intended afterward. Thirdly, developers should use a version control system like Git to keep track of the modifications they make to database structure. The database schema can be changed in the past thanks to this, and modifications can be undone if necessary.

In conclusion, database migration is a critical step in the creation of software, and developers should have a solid procedure in place. Database migrations can be performed more quickly and with less work by using tools like Flyway, Liquibase, or DbUp, which help automate the process. A seamless and successful migration procedure can be ensured by adhering to best practices, which include making backups, doing comprehensive testing, and keeping track of changes.

CHAPTER 3
SYSTEM DEVELOPMENT

3.1 Web Project Development

A web project's development goes through several stages and procedures. Each stage, from preparation to deployment, needs to be carefully thought out and carried out. The following are the crucial processes required in creating a web project in this article.

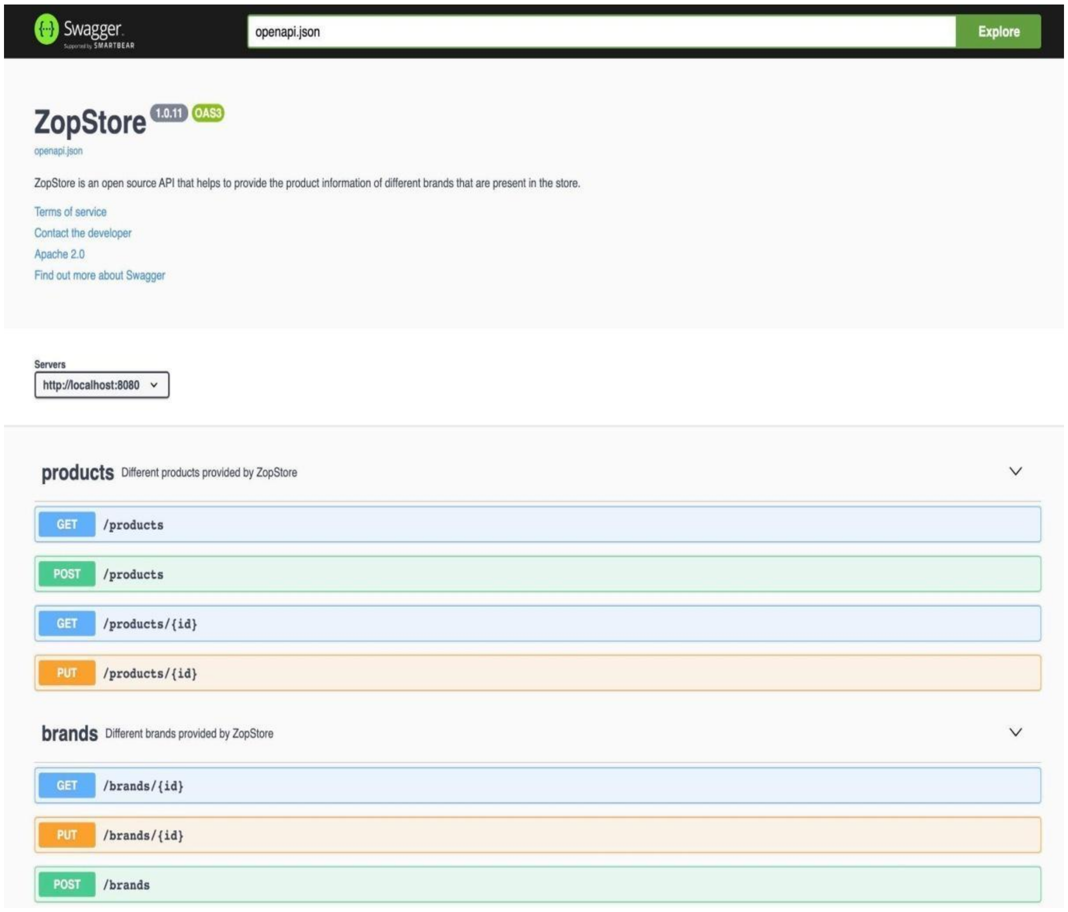
1. Planning: Making a plan is the first step in any web project. The project's goals, objectives, and target audience must all be specified. Defining the project's scope, schedule, and budget is equally crucial.
2. Research: After the project's scope has been established, research must be carried out to comprehend the market, rival products, and user requirements. Surveying, user testing, and competitive analysis are all part of this.
3. Design: The next step is to develop a wireframe and prototype for the web project based on the study results. Layout, user interface, and user experience design are all included.
4. Development: After the design is accepted, the web project's actual development gets under way. The authoring of code, the integration of APIs, and the building of databases are all part of the process.
5. Testing: Testing is a crucial step in the creation of a website. This entails checking the website for mistakes, bugs, and security flaws. To make sure the website is fully functional and meets user needs, the testing procedure should be exhaustive.
6. Deployment: After the website has been completely tested, the server should host it. In order to do this, the server must be configured, the website must be launched, and the website's files must be moved to the server.

7. Maintenance: To keep the website operating smoothly after it has been released, maintenance is necessary. The software of the website has to be updated, errors are fixed, and new features are added.

Every one of these procedures is essential for creating a web project. It is crucial to remember that the process is not linear and that some of the steps may overlap. For instance, during the development phase, design modifications could take place, and testing might identify issues with the design that need to be fixed. As a result, creating a web project is a challenging process that calls for careful planning, investigation, design, programming, testing, deployment, and maintenance. Following a planned approach and maintaining close communication with stakeholders are crucial for the project's success.

3.2 Designing the API

RESTful APIs can be designed and documented by developers using the potent tool Swagger. It offers developers a user-friendly interface for creating, editing, and testing their APIs. Using Swagger, developers can quickly generate an interactive documentation of their API that other developers can access and understand. This aids in enhancing user and developer collaboration and communication.



ZopStore 1.0.11 OAS3

openapi.json

ZopStore is an open source API that helps to provide the product information of different brands that are present in the store.

Terms of service
Contact the developer
Apache 2.0
Find out more about Swagger

Servers
http://localhost:8080

products Different products provided by ZopStore

```
GET /products
POST /products
GET /products/{id}
PUT /products/{id}
```

brands Different brands provided by ZopStore

```
GET /brands/{id}
PUT /brands/{id}
POST /brands
```

Fig. 3.1 Swagger - API Documentation

The steps for creating an API with Swagger are as follows:

1. Define the API: The API must be defined next. Endpoints, request and response parameters, as well as data types, must all be specified. This procedure is made simpler by the simple-to-use editor offered by Swagger.
2. Document the API: The API must first be defined before it can be documented. Developers and users can better comprehend an API's capabilities with the help of documentation. In order to describe the API, including endpoints, request parameters, response types, and other pertinent details, Swagger offers a user-friendly interface.
3. Test the API: After documenting the API, testing it is the next step. Developers may test the functionality, request and response parameters,

and error handling of their APIs using Swagger's integrated testing environment.

4. Export the API: The API must be exported when it has been designed and tested. Developers can export their API using Swagger in a number of different formats, including JSON, YAML, or HTML.

In conclusion, creating and testing RESTful APIs can be done quickly and easily by designing an API using Swagger. The user-friendly interface offered by Swagger makes it simple for developers to define, document, and test their APIs. With Swagger, developers can quickly produce interactive documentation that is simple for other developers to access and understand, which enhances cooperation and communication between developers and users.

3.3 Creating a Web Server Using Golang

A web server is a computer programme that receives and processes requests from client devices in order to deliver the desired content. It serves as the hub for communication between web browsers and web pages, making it the foundation of the World Wide Web. The web server responds with the requested material, such as a web page, image, or video, when a user types the address of a website into their web browser.

An operating system, web server software, database server, and programming language are only a few of the parts that make up a web server. The operating system controls the computer's resources, and the web server software handles incoming requests and delivers the desired material. The programming language is used to develop dynamic content, while the database server is used to store and retrieve data.

Golang (commonly known as Go) is a popular programming language used for creating web servers. Golang is a Google-developed open-source programming language that is renowned for its effectiveness, speed, and simplicity. It is a popular option for developing high-performance web servers since it is simple to learn and has built-in concurrency features.

There are a number of important factors to take into account when developing a web server in Golang. The API endpoints, or URLs that the web server will react to, must first be defined. The structure of the application and the kinds of requests it may process will be determined by these endpoints. The database schema and the methods for storing and retrieving the data should also be taken into account. Relational databases like MySQL or PostgreSQL are simple to work with thanks to Golang's built-in support for SQL databases. The use of other database types, such as NoSQL databases like MongoDB, is also an option.

The use of middleware is another factor to take into account while developing a Golang web server. Software known as middleware, which resides in between the web server and the application, adds extra features like rate restriction, logging, and authentication. Golang supports a wide variety of middleware packages from outside vendors, so adding this capability to your web server is simple.

The built-in concurrency characteristics of Golang are one of its main benefits when developing web servers. Concurrency, or the capacity to carry out many actions concurrently, is a special feature of the Go programming language known as goroutines. Goroutines are thin threads that eliminate the overhead associated with conventional threads and enable the execution of numerous tasks simultaneously. Golang contains built-in support for channels, which are used for inter-goroutine communication in addition to goroutines. Goroutines may communicate and synchronize with one another via channels, which makes it simple to create highly concurrent and effective web servers. There are numerous deployment options available for Golang web servers. Use of Docker, which makes it simple to package and deploy Golang apps in a containerized environment, is one well-liked choice. Docker containers can be readily deployed to a range of contexts, including cloud-based services like Amazon Web Services (AWS) or Google Cloud Platform (GCP). They are small, portable, and lightweight.

Utilizing Kubernetes, an open-source container orchestration platform that automates the deployment, scaling, and management of containerized applications, is an additional deployment option. Kubernetes is an effective technology for managing Golang web servers at scale because it offers cutting-edge features like automatic scaling, rolling upgrades, and service discovery.

Golang is a strong and effective programming language that is suitable for creating web servers, to sum up. It is a popular option for web server development due to its integrated concurrency features, support for SQL databases, and large library of third-party packages. Considerations like API endpoints, database schema, and middleware should be taken into account when creating a Golang web server. When deploying, solutions like Docker and Kubernetes offer effective tools for managing and scaling Golang web servers.

3.4 Database Design

Table 3.1 Product table

Field name	**Type**	**Size**	**NOT NULL**	**Primary Key**
product_id	integer	-	yes	yes
name	varchar	255	yes	-
description	varchar	255	-	-

price	integer	-	-	-
quantity	integer	-	-	-
category	varchar	255	-	-
brand_id	varchar	255	yes	-
status	enum	-	yes	-

Table 3.2 Brand table

Field name	Type	Size	NOT NULL	Primary Key
brand_id	integer	-	yes	yes
name	varchar	-	no	-

Table 3.3 Admin Table

Field name	Type	Size	NOT NULL	Primary Key
id	int		yes	yes
name	varchar	255	yes	-
email	varchar	255	yes	-
password	varchar	16	yes	-

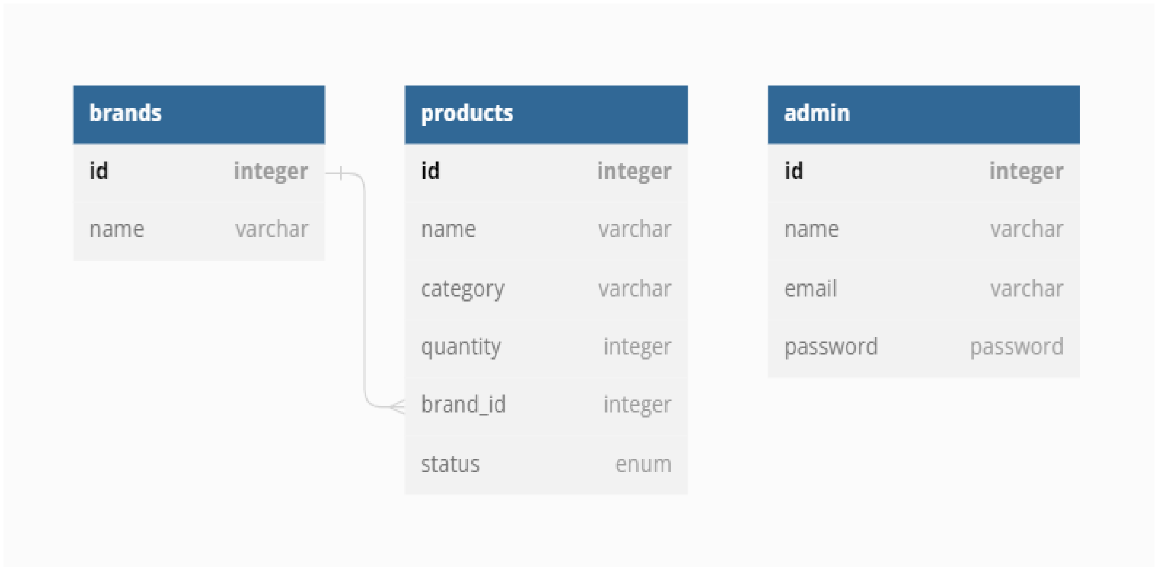


Fig. 3.2 Database Structure

3.5 Testing the Server

A Golang web server must be tested to make sure all of its modules and functionalities are operating as intended. Unit testing, integration testing, and end-to-end testing are some of the methods used to test a Golang web server.

Unit testing isolates specific functions and modules from the rest of the codebase to test them individually. This guarantees that every function performs as intended and generates the right result for a particular input. The testing package included with the standard library is used for unit testing in Golang. The testing package offers a number of tools for creating and executing tests.

Integration testing examines the interactions between various server components to make sure they function properly together. The httptest package in Golang can be used for integration testing. A method for testing HTTP handlers and responses is provided by the httptest package. In end-to-end testing, every component of the web server is tested in a real-world setting. This kind of testing makes certain that the server performs as anticipated from the user's point of view. Tools like Selenium, Puppeteer, or Cypress can be used for end-to-end testing.

It is crucial to make sure the tests are automated, repeatable, and cover the most portion of the codebase possible while testing a Golang web server. As a result, the server is stable and dependable and faults are found earlier in the development process.The server must undergo stress testing and performance testing in addition to the aforementioned testing techniques. Stress testing entails putting the server under a lot of traffic while observing how it responds. When performing performance testing, the server's response times and resource usage are examined under various circumstances.

In conclusion, testing a Golang web server combines stress testing, performance testing, end-to-end testing, integration testing, and unit testing. To guarantee the stability and dependability of the server, it is essential to make sure that the tests are automated, repeatable, and cover the largest portion of the codebase.

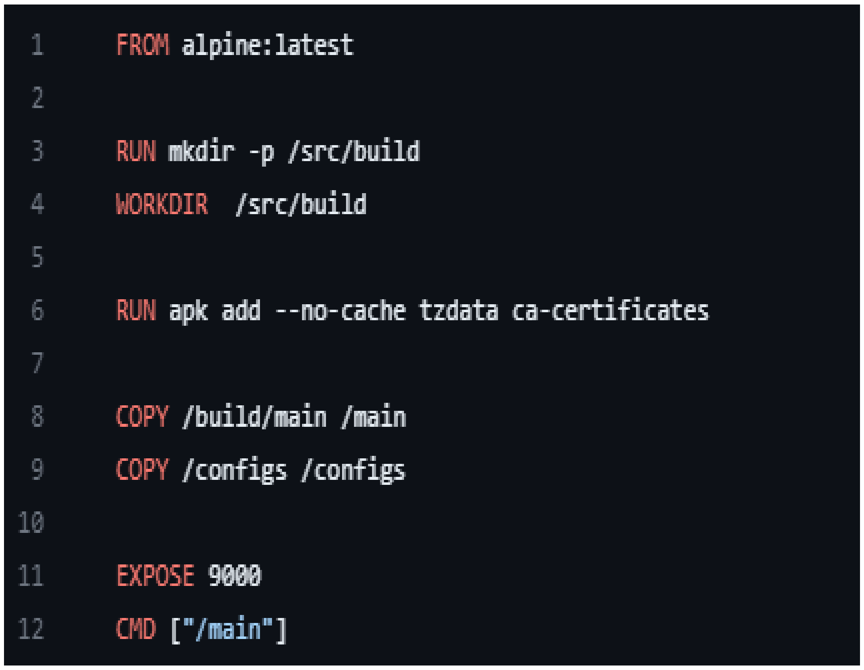
3.6 Deploying the Web Application using Kubernetes and Docker

Tools like Docker and Kubernetes are now necessary for delivering web apps. Developers can package apps and their dependencies into small, portable containers using the containerization technology Docker. Contrarily, Kubernetes is an open-source platform that streamlines the administration, scaling, and deployment of containerized applications.

3.6.1 Dockerize the application

Containerizing a web application is the first step in utilizing Docker and Kubernetes to deploy it. A lightweight, portable container that can operate on any platform that supports Docker is created by containerizing the programme and all of its dependencies.

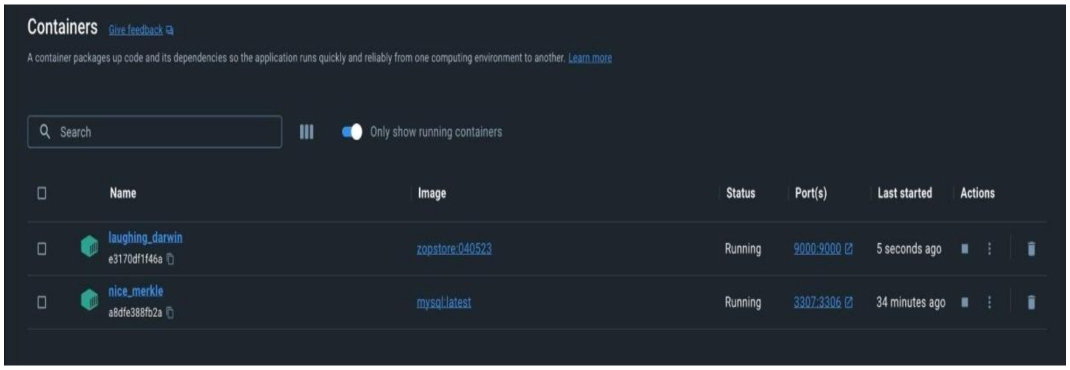
Create a Dockerfile in the project's root directory to containerize the application. A script called the Dockerfile instructs Docker on how to create the container image.



```
FROM alpine:latest

RUN mkdir -p /src/build
WORKDIR  /src/build

RUN apk add --no-cache tzdata ca-certificates

COPY /build/main /main
COPY /configs /configs

EXPOSE 9000
CMD ["/main"]
```

Fig. 3.3 Dockerfile

In this Dockerfile, an official main.go runtime is used as the parent image. The working directory is set to /src/build, the configs and the artifact are copied there, the dependencies are installed, the remaining application code is copied there, the environment variable PORT is set to 9000, the application's port is exposed, and the application is started using the build file.


Containers Give feedback

A container packages up code and its dependencies so the application runs quickly and reliably from one computing environment to another. Learn more

Search

Only show running containers

	Name	Image	Status	Port(s)	Last started	Actions
☐	laughing_darwin e3170df1f46a	zopstore:040523	Running	9000:9000	5 seconds ago	
☐	nice_merkle a8dfe388fb2a	mysql:latest	Running	3307:3306	34 minutes ago	

Fig. 3.4 Docker Containers

This command instructs Docker to use the Dockerfile located in the current directory to create an image with the name my-web-app.

3.6.2 Push the image to a registry

Pushing the container image to a container registry is the next stage in the deployment of a web application using Docker and Kubernetes. Container images are kept and accessible by servers and other developers in a container registry. Docker Hub is a well-liked container registry that enables developers to store and distribute container images without charge. Use the next command to push the container image to Docker Hub:

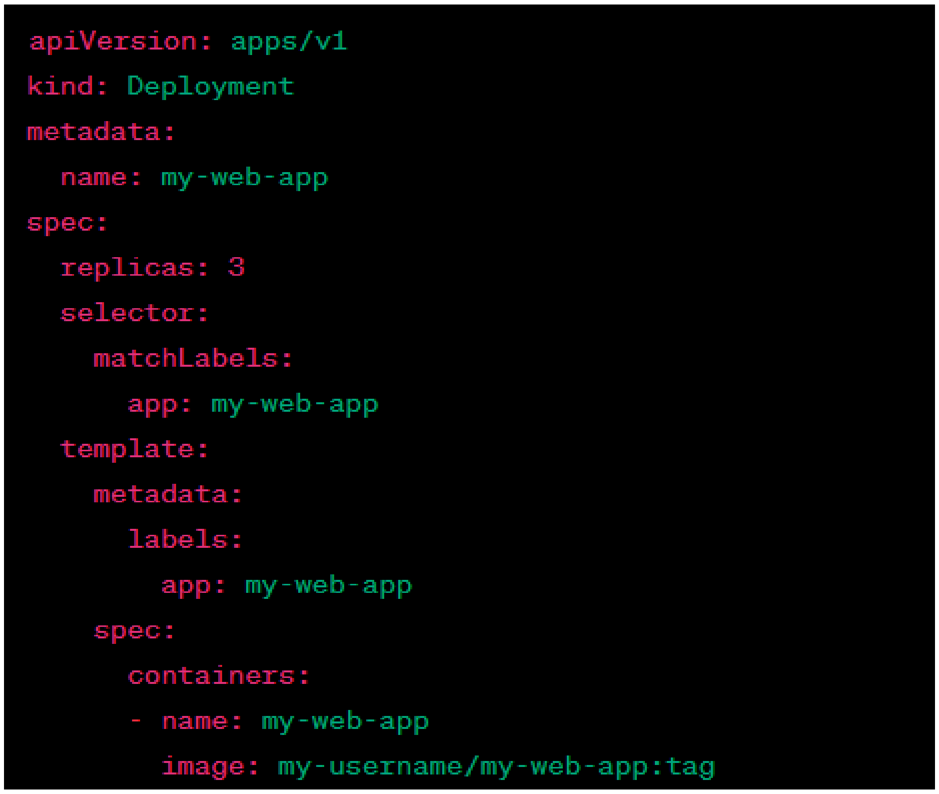
```
docker push my-username/my-web-app:tag
```

Fig. 3.5 Command to push image to the registry

The my-username parameter specifies your Docker Hub username, and the command instructs Docker to submit the my-web-app image to Docker Hub with the tag tag.

3.6.3 Deploy the application to Kubernetes

Delivering the containerized application to Kubernetes is the last step in delivering a web application using Docker and Kubernetes.



```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: my-web-app
spec:
  replicas: 3
  selector:
    matchLabels:
      app: my-web-app
  template:
    metadata:
      labels:
        app: my-web-app
    spec:
      containers:
      - name: my-web-app
        image: my-username/my-web-app:tag
```

Fig. 3.6 Deployment File

Make a YAML-formatted Kubernetes deployment file first. The container image, the number of replicas, and any additional configuration options are all specified in the deployment file along with the desired state of the application.

CHAPTER 4
PERFORMANCE ANALYSIS

4.1 Project Testing Strategy

A well-defined testing strategy must be in place to assure the quality of the software being built, as testing is a critical component of software development. In the Handler Business Store (HBS) design, testing is essential to verifying that the various system levels are performing as expected and that the system as a whole is operating as intended.

The handler layer, the business layer, and the store layer are the three levels that make up the HBS architecture. Each layer in the system is in charge of particular duties, and testing is required to make sure each layer is operating as planned.

4.1.1 Unit Testing

Individual units of code can be tested using a testing technique called unit testing. Each layer of the HBS architecture is subjected to unit testing. Before the code is included into the larger system, unit testing aids in finding any bugs.

Handler Layer: The client's requests and responses are handled by the handler layer. The various API endpoints should be tested as part of the handler layer's unit tests to make sure they are returning the right response for each request. Unit tests for the handler layer should verify error handling, authentication, and authorization in addition to the endpoints. When something goes wrong, it is crucial to make sure that the API produces the proper error messages, is secure, and only permits authorized people to access it.

Business Layer: Implementing business logic and handling data processing fall under the purview of the business layer. The many functions that perform the business logic should be the primary focus of unit tests for the business layer. Unit tests for the business layer should test data validity and error handling in addition to the functions. It is crucial to verify that the functions correctly handle errors and that the data being processed is accurate.

Store Layer: Data storage and retrieval from the database are the responsibilities of the store layer. In order to ensure that the data is being stored and retrieved appropriately, unit tests for the store layer should concentrate on testing the various database operations. Unit tests for the store layer should evaluate data validity and error handling in addition to the database operations. It is crucial to verify that the data being stored is accurate and that database operations handle errors effectively.

4.1.2 Integration Testing

A testing method called integration testing is used to examine how various pieces of code communicate with one another. Integration testing is carried out in the HBS architecture to make sure that the various system layers interact properly.

Business Layer and Handler Layer: The main goal of integration testing between the handler layer and the business layer should be to examine the interactions between the two layers. It is crucial to confirm that the business logic is being implemented appropriately and that the data being communicated across the levels is valid.

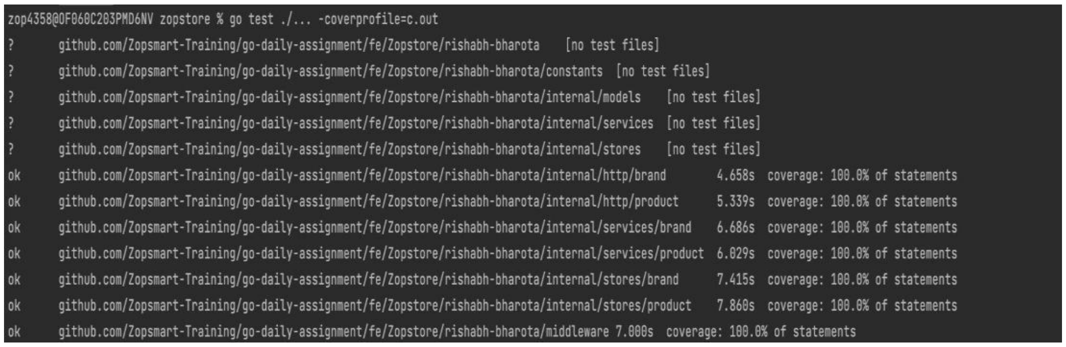
Store Layer and Business Layer: The main goal of business-store integration testing should be to examine the interactions between the two layers. It is crucial to confirm that the data being transferred between the layers is accurate and that the data is entering and leaving the database correctly.

4.1.3 End to End Testing

A testing method used to test the entire system is end-to-end testing. End-to-end testing is carried out in the HBS architecture to ensure that the entire system is operating as intended. The API endpoints should undergo end-to-end testing to confirm that they are returning the appropriate response for each request. The complete data flow through the system, from the client request to the data being saved in the database, should be tested as part of end-to-end testing.

4.2 Unit Testing Results

Test Coverage of 100%:

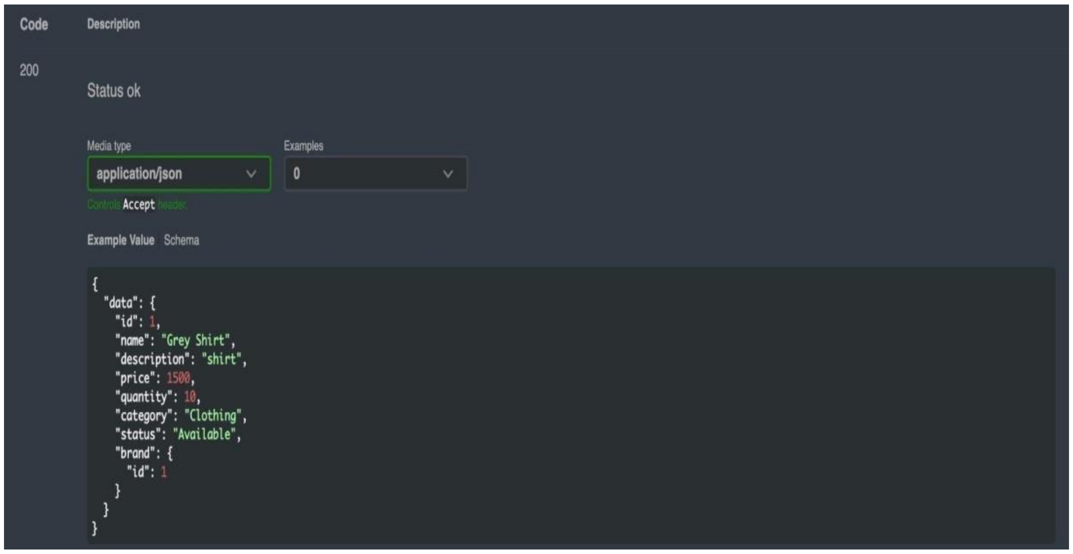


```
zop4358@0F060C203PMD6NV zopstore % go test ./... -coverprofile=c.out
?       github.com/Zopsmart-Training/go-daily-assignment/fe/Zopstore/rishabh-bharota    [no test files]
?       github.com/Zopsmart-Training/go-daily-assignment/fe/Zopstore/rishabh-bharota/constants  [no test files]
?       github.com/Zopsmart-Training/go-daily-assignment/fe/Zopstore/rishabh-bharota/internal/models    [no test files]
?       github.com/Zopsmart-Training/go-daily-assignment/fe/Zopstore/rishabh-bharota/internal/services  [no test files]
?       github.com/Zopsmart-Training/go-daily-assignment/fe/Zopstore/rishabh-bharota/internal/stores    [no test files]
ok      github.com/Zopsmart-Training/go-daily-assignment/fe/Zopstore/rishabh-bharota/internal/http/brand        4.658s  coverage: 100.0% of statements
ok      github.com/Zopsmart-Training/go-daily-assignment/fe/Zopstore/rishabh-bharota/internal/http/product      5.339s  coverage: 100.0% of statements
ok      github.com/Zopsmart-Training/go-daily-assignment/fe/Zopstore/rishabh-bharota/internal/services/brand    6.686s  coverage: 100.0% of statements
ok      github.com/Zopsmart-Training/go-daily-assignment/fe/Zopstore/rishabh-bharota/internal/services/product  6.029s  coverage: 100.0% of statements
ok      github.com/Zopsmart-Training/go-daily-assignment/fe/Zopstore/rishabh-bharota/internal/stores/brand      7.415s  coverage: 100.0% of statements
ok      github.com/Zopsmart-Training/go-daily-assignment/fe/Zopstore/rishabh-bharota/internal/stores/product    7.860s  coverage: 100.0% of statements
ok      github.com/Zopsmart-Training/go-daily-assignment/fe/Zopstore/rishabh-bharota/middleware 7.000s  coverage: 100.0% of statements
```

Fig. 4.1 Testing Coverage

4.3 End to End Testing Results

Responses for a GET call to the products endpoint



Code Description

200

Status ok

Media type: application/json

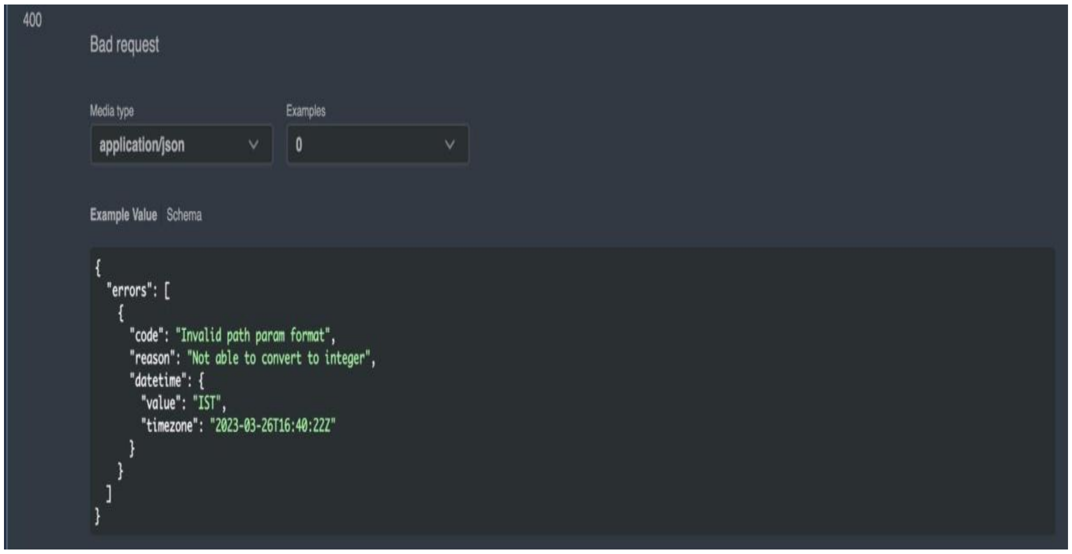
Examples: 0

Controls Accept header.

Example Value Schema

```
{
  "data": {
    "id": 1,
    "name": "Grey Shirt",
    "description": "shirt",
    "price": 1500,
    "quantity": 10,
    "category": "Clothing",
    "status": "Available",
    "brand": {
      "id": 1
    }
  }
}
```

Fig. 4.2 Status code 200 response



400

Bad request

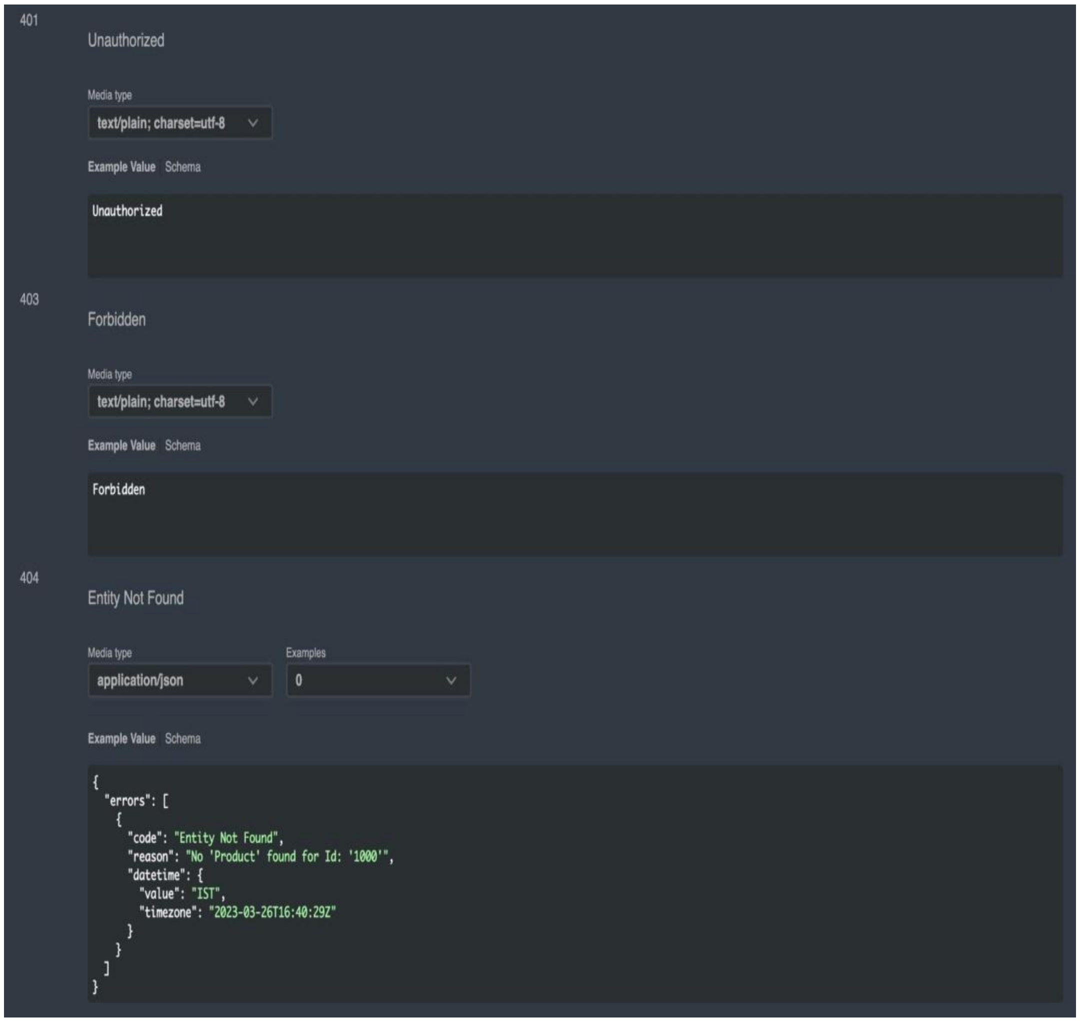
Media type: application/json

Examples: 0

Example Value Schema

```
{
  "errors": [
    {
      "code": "Invalid path param format",
      "reason": "Not able to convert to integer",
      "datetime": {
        "value": "IST",
        "timezone": "2023-03-26T16:40:22Z"
      }
    }
  ]
}
```

Fig. 4.3 Status code 400 response


401

Unauthorized

Media type: text/plain; charset=utf-8

Example Value | Schema

```
Unauthorized
```

403

Forbidden

Media type: text/plain; charset=utf-8

Example Value | Schema

```
Forbidden
```

404

Entity Not Found

Media type: application/json

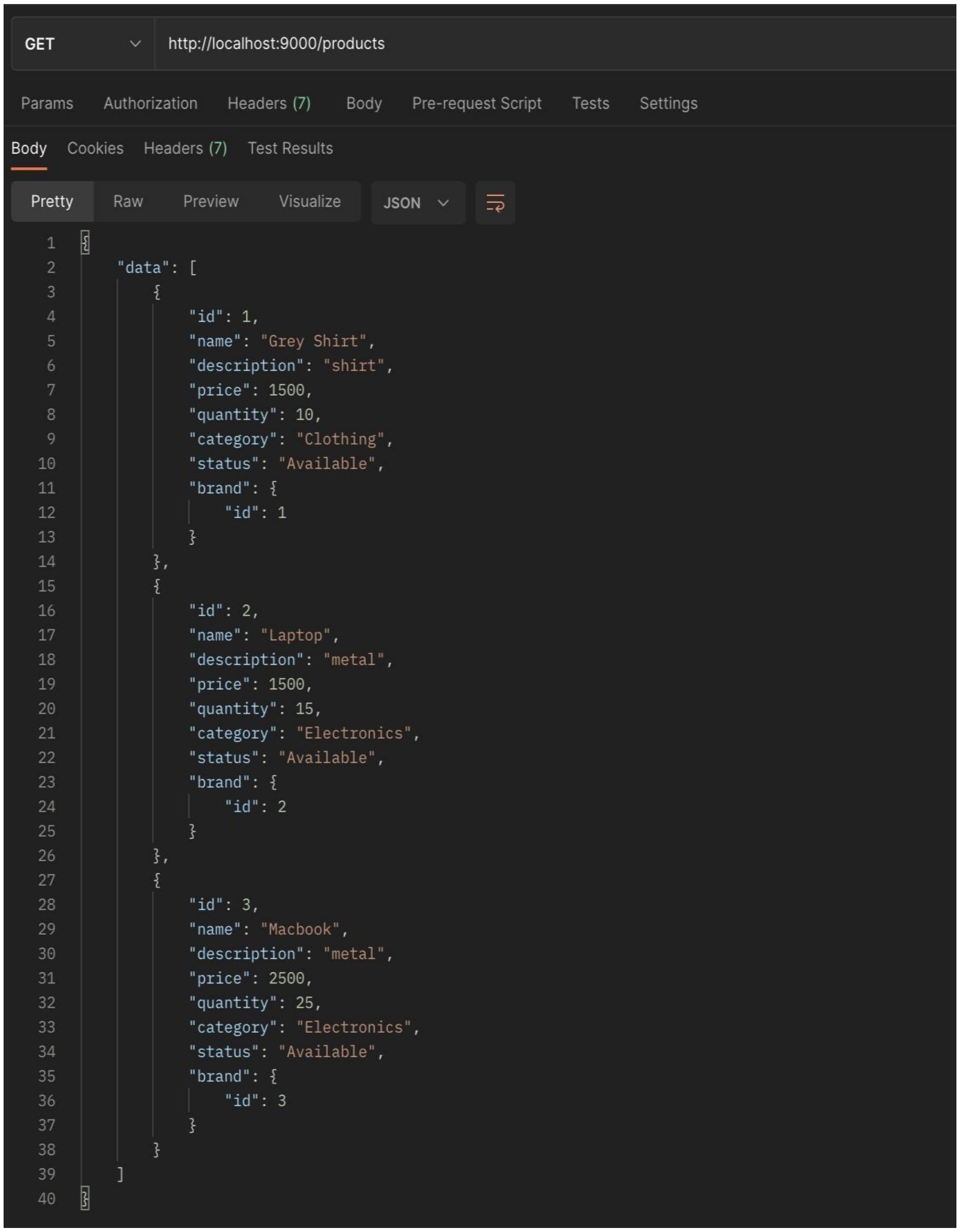
Examples: 0

Example Value | Schema

```
{
  "errors": [
    {
      "code": "Entity Not Found",
      "reason": "No 'Product' found for Id: '1000'",
      "datetime": {
        "value": "IST",
        "timezone": "2023-03-26T16:40:29Z"
      }
    }
  ]
}
```

Fig. 4.4 Status code 401, 403 and 404 responses

Testing the API with Postman (GET call to the products endpoint)

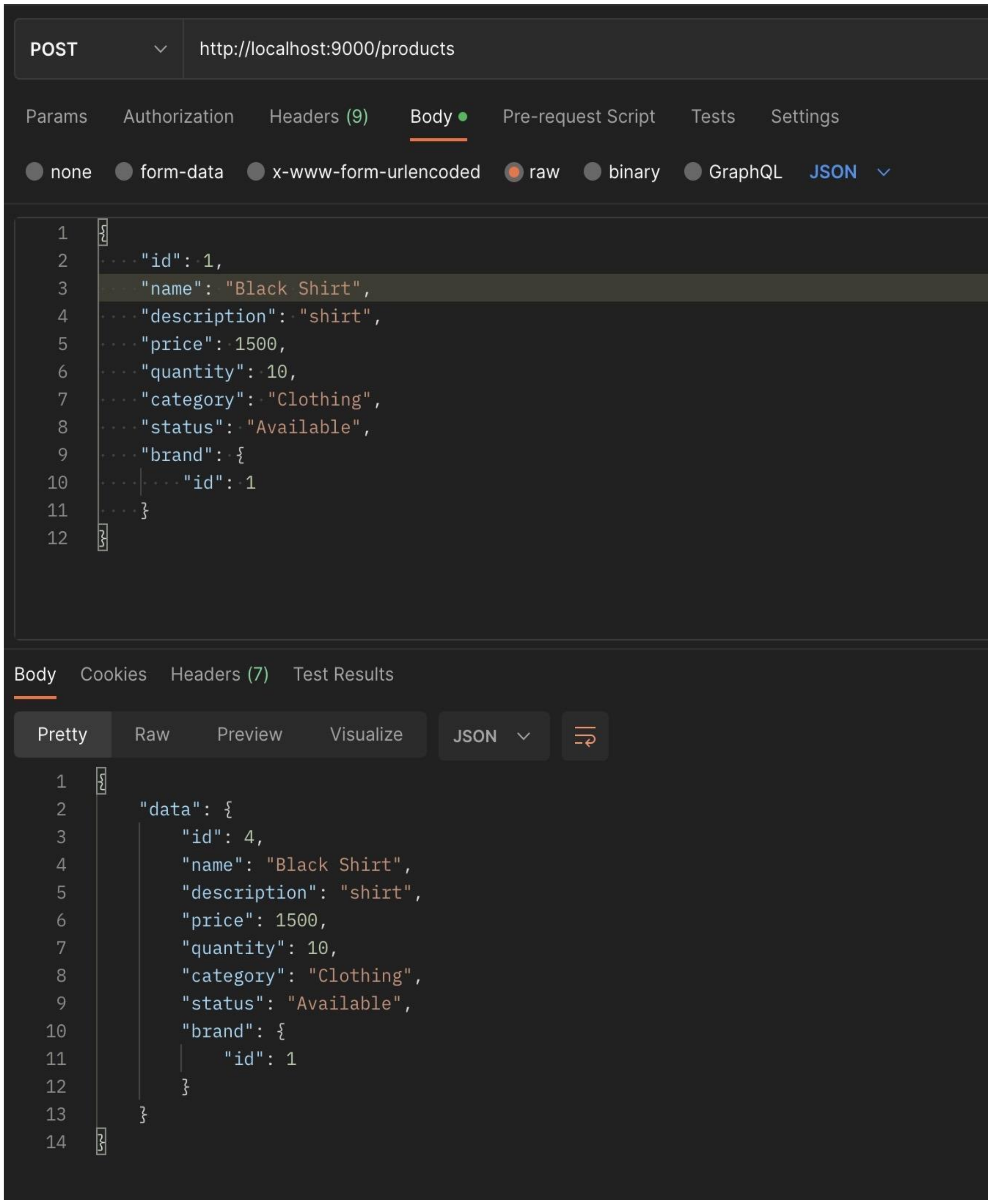


```
http://localhost:9000/products
```

```
{
    "data": [
        {
            "id": 1,
            "name": "Grey Shirt",
            "description": "shirt",
            "price": 1500,
            "quantity": 10,
            "category": "Clothing",
            "status": "Available",
            "brand": {
                "id": 1
            }
        },
        {
            "id": 2,
            "name": "Laptop",
            "description": "metal",
            "price": 1500,
            "quantity": 15,
            "category": "Electronics",
            "status": "Available",
            "brand": {
                "id": 2
            }
        },
        {
            "id": 3,
            "name": "Macbook",
            "description": "metal",
            "price": 2500,
            "quantity": 25,
            "category": "Electronics",
            "status": "Available",
            "brand": {
                "id": 3
            }
        }
    ]
}
```

Fig. 4.5 Postman responses for ‘get all products’

POST call to the products endpoint



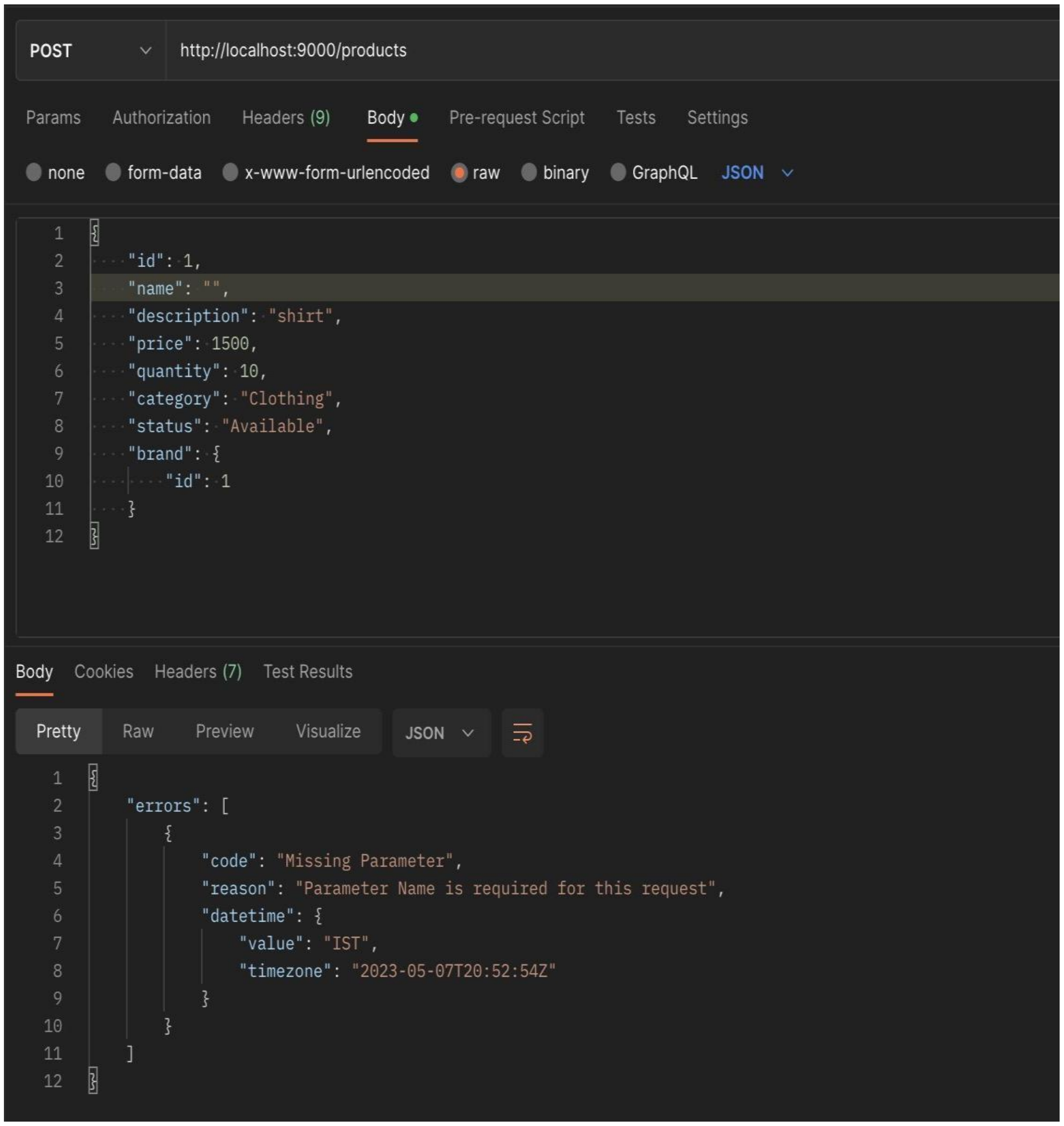
http://localhost:9000/products

```
{
    "id": 1,
    "name": "Black Shirt",
    "description": "shirt",
    "price": 1500,
    "quantity": 10,
    "category": "Clothing",
    "status": "Available",
    "brand": {
        "id": 1
    }
}
```

```
{
    "data": {
        "id": 4,
        "name": "Black Shirt",
        "description": "shirt",
        "price": 1500,
        "quantity": 10,
        "category": "Clothing",
        "status": "Available",
        "brand": {
            "id": 1
        }
    }
}
```

Fig. 4.6 Postman response for adding a product

Bad Request: Wrong request body format



http://localhost:9000/products

```
{
    "id": 1,
    "name": "",
    "description": "shirt",
    "price": 1500,
    "quantity": 10,
    "category": "Clothing",
    "status": "Available",
    "brand": {
        "id": 1
    }
}
```

```
{
    "errors": [
        {
            "code": "Missing Parameter",
            "reason": "Parameter Name is required for this request",
            "datetime": {
                "value": "IST",
                "timezone": "2023-05-07T20:52:54Z"
            }
        }
    ]
}
```

Fig. 4.7 Postman error response

4.4 Checking Linters

Tools called golang linters examine Go code and give input on its quality, potential flaws, and stylistic problems. They support developers in raising the caliber of their code and guarantee that coding standards and best practices are followed. Golang linter tools can be used to enforce coding standards, find performance problems, and check for common programming errors. These tools operate by scanning the source and notifying the user of any flaws they discover. They can also be connected to Continuous Integration (CI) platforms to deliver feedback on commits and pull requests.

Several well-known Golang linters are:

1. Golangci-lint is a quick and effective linter that can look for problems like unnecessary code, empty block statements, and other things.
2. GoLint - This is a less complex linter that looks for formatting problems, unneeded imports, and other typical coding errors.
3. GoVet is a tool that the Go compiler includes that scans code for potentially dangerous structures including shadowed variables, unreachable code, and more.

The use of linters is crucial for enhancing the quality of your Golang code. Regular use of them will enable you to identify and address possible problems early in the development cycle, producing codebases that are more resilient, effective, and manageable.

CHAPTER 5
CONCLUSIONS

5.1 Conclusion

Working on this project was incredible since I learned so much. It gave me the opportunity to participate at each stage of the project's development, which was a truly eye-opening experience for me. The thrill of working and the satisfaction of overcoming various challenges gave me a sense of the development industry. This project taught me how to make professional software. In this project, we developed a system to handle product and brand information that is dependable, easy to use, economical, and useful.This makes it possible for owners to handle their internet business's details quickly and efficiently. It provides significant time and financial savings to the owner.

5.2 Future Scope

Future scope for my project:

1. Designing an effective user interface and user experience: Effective UI and UX can increase users' overall engagement with a web application. This can be done by making the online application's design, layout, color scheme, typography, and usability better.
2. Increasing Performance: Quicker response and loading times can enhance user experience and lower bounce rates. Optimisations on the client and server sides as well as the use of content delivery networks (CDNs) can all boost performance.
3. Including New Features: The functionality and user engagement of the web application can be enhanced by including new features. Through user feedback, industry trends, and competitor analysis, these features can be found.

REFERENCES

1. https://docs.microsoft.com/en-us/azure/architecture/best-practices/api design
2. https://github.com/DATA-DOG/go-sqlmock
3. https://github.com/golang/mock
4. https://go.dev/doc/tutorial/
5. https://medium.com/swlh/developing-a-web-application-in-go-using-the layered-architecture-8fc13209c808
6. https://github.com/gorilla/mux
7. https://dev.mysql.com/doc/
8. https://www.linux.org/
9. https://docs.docker.com/
10. https://kubernetes.io/docs/home/
11. https://ngdocs.harness.io/
12. https://prometheus.io/docs/introduction/overview/