

# **Product-Brand Store Web API**

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

in

**Computer Science and Engineering/Information  
Technology**

By

Shivansh Thakur (191215)

Under The Supervision of

Dr. Ruchi Verma

to



Department of Computer Science & Engineering and  
Information Technology

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

## Candidate's Declaration

I hereby declare that the work presented in this report entitled "**Product-Brand Store Web API**" 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 Feb 2023 to May 2023 under the supervision of **Dr. Ruchi Verma (Assistant Professor (SG) and Department of Computer Science & Engineering and Information Technology)**.

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



Shivansh Thakur, 191215

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



Dr. Ruchi Verma

Assistant Professor (SG)

Department of Computer Science & Engineering and Information Technology



Mithali R Shetty

Senior Lead Engineer

ZopSmart Technology

Dated: 09-05-23

# Plagiarism Certificate

JAYPEE UNIVERSITY OF INFORMATION TECHNOLOGY, WAKNAGHAT

## PLAGIARISM VERIFICATION REPORT

Date: 15 May 2023

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

Name: SHIVANSH THAKUR Department: CSE Enrolment No 191215

Contact No. 6284940515 E-mail. shivansh46470@gmail.com

Name of the Supervisor: Dr. RUCHI VERMA

Title of the Thesis/Dissertation/Project Report/Paper (In Capital letters): PRODUCT-BRAND STORE WEP API

### 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 = 59
- Total No. of Preliminary pages = 9
- Total No. of pages accommodate bibliography/references = 1

(Signature of Student)

### FOR DEPARTMENT USE

We have checked the thesis/report as per norms and found Similarity Index at 11 (%). 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)	
			Word Counts	
Report Generated on				
		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](mailto: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. Ruchi Verma, 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. Ruchi Verma, 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.

Shivansh Thakur,  
191215

# Table of Content

Certificate.....	i
Plagiarism Certificate.....	ii
Acknowledgement.....	iii
Table of Content.....	iv
List of Figures .....	vi
List of Tables .....	vii
Abstract .....	viii
<b>1. Introduction.....</b>	<b>1</b>
1.1 Company Profile .....	1
1.2 Introduction.....	1
1.3 Problem Statement .....	2
1.4 Objective .....	2
1.5 Methodology .....	2
1.5.1 Proposed Architecture.....	3
1.5.2 System Requirements.....	5
1.5.3 Database Used.....	5
1.6 Organization.....	6
<b>2. Literature Survey.....</b>	<b>8</b>
2.1 Technology and Literature Survey.....	8
2.1.1 Development Tools.....	8
2.1.2 Representational State Transfer (REST).....	10
2.1.3 Golang HTTP Package.....	12
2.1.4 Database Migration.....	13
2.2 Project Feasibility.....	13
2.2.1 Technological feasibility .....	13
2.2.2 Functional feasibility .....	13
2.2.3 Schedule feasibility.....	14
2.3 Application Plan .....	14
2.3.1. Justification and Project Building.....	14
2.3.2. Milestones and Deliverables .....	15
2.3.3. Roles and Responsibilities .....	15
2.3.4. Group Dependencies .....	16
2.3.5. Project Scheduling chart .....	17
<b>3. System Development and Implementation.....</b>	<b>18</b>
3.1 Study of Current System .....	18
3.2 Problems and Weaknesses of Current System .....	18

3.3 User Characteristics .....	18
3.4 Assumptions and Dependencies .....	19
3.5 System Design.....	23
3.6 Database Design.....	25
3.7 Implementation Environment:.....	26
3.8 Coding Example.....	27
<b>4.0 Performance Analysis .....</b>	<b>31</b>
4.1 Project Testing Strategy.....	31
4.2 Unit Testing Planning.....	31
4.3 Unit Tests.....	31
4.4 Test Cases .....	33
4.5 Swagger Documentation.....	37
<b>5.0 Conclusion and Discussion .....</b>	<b>40</b>
5.1 Conclusion .....	40
5.2 Discussion .....	40
5.3 Future Scope and Enhancements .....	40
5.4 Application Contributions .....	41
5.5 Limitations .....	41
<b>REFERENCE .....</b>	<b>42</b>
<b>APPENDICES .....</b>	<b>43</b>

## List of Figure

Figure 1.1 Go Language Symbol .....	3
Figure 1.2 MySQL Symbol.....	4
Figure 1.3 Git/GitHub Symbol.....	5
Figure 2.1 Iterative Waterfall Model .....	11
Figure 3.1 Implementation Diagram .....	16
Figure 3.2 Flow Chart of Login .....	19
Figure 3.3 Flowchart of create Diagram .....	20
Figure 3.4 Flowchart of GetByBrand .....	20

## List Of Table

Table 2.1 Roles and Responsibilities .....	12
Table 2.2 Application Schedule.....	13
Table 3.1 Product Data Dictionary.....	21
Table 3.2 Engine Data Dictionary.....	21
Table 3.3 Admin Data Dictionary.....	22
Table 4.1 Login Test Cases.....	36
Table 4.2 GetByID Object Test Cases.....	36
Table 4.3 Create Object Test Cases.....	38
Table 4.4 Update Test Cases.....	39



## **Abstract**

The Products-Brand Store Web API is an API which easily create/update/delete products as well as Admin Authentication for safety.

In everyday life there are many difficulties owners are facing. In these difficulties one of the difficulties is to manage online web Store/shop sites as well as in this hectic life schedule owners don't have time to do all the things manually.

In my Web API, I have created a backend of the Product-Brand Web Store site which has Admin –who can manage the site, who can manage the products. Now this Web API may work with any sort of UI without any prior knowledge of building/structuring of the API.

Apart from this Web API has UI friendly responses which can be easily utilised at the time of creation of the UI as well as it is also user friendly for users when used the API for testing in the postman. Also Web API is secure with basic sort of Authentication.

# Chapter 01: Introduction

## 1.1 Company Profile

Modern retail technology provider ZopSmart Technology offers all the resources needed to establish an online store. ZopSmart offers services like digital marketing, e-commerce, mobile commerce, and e-wallets.

ZopSmart help clients to define business processes and derive the optimum integration needs using industry best practices to increase the automation for data communication and for better decision making so that the business people can focus on business and technology does the data communication and enables better decision making.



### Services:

WebSite Development: End-to-End website development starting from UI/UX design to development of the Website or maintaining the website for clients.

Framework Development: Framework Development for making Internal/Client WebServices in Java/ Golang /NodeJS.

Mobile Development: Android Development, IOS Development

Company Website - <https://zopsmart.com>

## 1.2 Introduction:

This application is designed for store owners who have access to a database of data on online goods. Our goal is to make it simple for executives and administrators to automate their tasks with minimal effort. For instance, they can quickly produce quotes for any inquiries about online products.

### **1.3 Problem Statement:**

Nowadays it is easy to find online web stores on the internet, as online business has become very common disregarding the age limit and gender.

Using manual procedures may pave the way for a variety of obstacles when satisfying Customers and Employees of the respective online shopping. Valuable time and money of the Owners, Employees get wasted unnecessarily due to these manual dealings. These obstacles directly impact the dealership's profits, owners' interests, and both.

### **1.4 Objective:**

The following are the project's goals:

- Avoid the paper-based tasks at the shopping location, such as using diaries to record brand and product information.
- Eliminate data duplication by maintaining Product and Brand details across many devices (mobile, diary, etc.) and across multiple users (Owner, Employees).
- Eliminate the Owner's and Employees' wastage of time, resources, efforts, and money.
- Boost the productivity and efficiency of a web store's operations, services, and procedures including storing Product and Brand information.
- Boost employee and owner satisfaction.

### **1.5 Methodology:**

Some of the main components of this system are “Three Layered Architecture”, Postman-API testing, Swagger API documentation, Database Migrations etc. To start building the project, it is needed to set up the machine for the required tools for development.

Environment variables can be set in `/.bashrc` or `/.bash_profile`. These are hidden files, which is why they begin with a dot. The home directory (`~`) contains both. The system

environment contains several processes that are active, as well as Environment variables—variables that are set in the environment and have an impact on the processes that use them. These procedures will be impacted if the value of these variables changes.

OS having a command-line and graphical user interface that is based on Unix.

The default shell on Linux is BASH (Bourne Again Shell), a Unix shell.

Installing, upgrading, and maintaining packages are done via a package manager. APT: Advanced Packaging Tool (APT), SUDO: Superuser Do or Substitute User Do, and apt-get are the default package managers for Ubuntu.

### **1.5.1 Proposed Architecture:**

In Three Layered Architecture, layers are autonomous from one another and interact via interfaces. Basically, this aids in the modularization, readability, and maintainability of our application.

HTTP layer, Service layer, and Store layer are the three layers of this.

**1. HTTP layer:** verifies request body, header checks, and query/path parameters.

**2. Service layer,** which carries out business logic and interacts with the datastore layer.

**3. Store layer:** Executes queries at the database level.

1. Each layer uses an interface (specified methods with input parameters and output types) to communicate with its preceding and following layer.

2. Depending on the situation, each layer's interface, database, and server are mocked during testing.

### **Dependency Injection**

Dependency Injection is a way of creating code where the dependencies of a certain object or struct are provided at the moment the object is initialised.

We have the option to explicitly specify when to reuse an existing instance of a dependency and when to build new ones. The structs are no longer in charge of creating their dependencies, which loosens their bond with those dependencies.

### **Micro Services:**

A microservices-based approach to software development. It is an organisational and architectural framework for building compact, independent services that communicate with one another through well-defined APIs. Small, autonomous teams own and operate these services.

Microservices architectures make it easier to build and develop applications and speed up the time it takes to commercialise new features.

### **Flexibility in Scaling**

The demand for each microservice's underlying app feature can be scaled independently of the others.

This enables teams to maintain service uptime during moments of high demand, precisely size infrastructure, and accurately estimate the cost of a feature.

### **Easy to Deploy**

Microservices make it easier to test new ideas and roll them back if they don't work by enabling continuous integration and delivery. It is able to experiment more, revise code more quickly, and launch new features more quickly thanks to the low cost of failure.

### **Utilisable Code**

Software can be divided into distinct, well defined modules, which teams can then use for a variety of purposes. A service developed for one purpose may serve as the basis for another feature.

The project titled as “Product-Brand Store Web API” is a Web API. These API provide complete information about the product and its brand. The foundation of the online Store is a product database that houses all of the necessary product data.

It provides the find, create, update and delete functionality.

**key points about this application:**

- Giving users the option to input Product and Brand information.
- Giving users the option to edit Brand and Product information.
- Offering the option to display information about all Products and Brands.

**1.5.2 System Requirements:**

**Hardware Requirements :**

- Processor: I3
- Higher Ram: 1 GB / Higher
- Hard disk: 10 GB / Higher

**Software Requirements :**

- Technology: Go Language
- IDE : GoLand
- Designing Tool : Creatly, Lucid Chart
- Server Side Technologies: Go Lang
- Database Server: MySql
- Operating System: Linux, Mac

**1.5.3 Database Used**

MySQL is used as the database. The drivers and libraries we use to link programmes written in golang programming language to MySQL database servers are known as the MySQL Connectors and APIs. The database server and application server may be located on the same system or may communicate over a network.

By setting up several connections using the connection pooling functionality, MySQL enables users to manage data coming in from various sources at various rates and over time intervals, which helps improve system efficiency and lower overall resource

consumption. With connection pooling, new connections can be sent back to the pool automatically.

By using the Java Naming and Directory Interface (JNDI), you may access the application server configurations file and change the connection pool settings for your MySQL instance. Make sure to consider the available resources, such as RAM, CPUs, context switches, etc., when determining the size of your connection pool.

By using X DevAPI connections to compress data, MySQL enables users to reduce the amount of time data is transmitted over the network and is ingested. You must negotiate with the server and set the negotiation priority using the "compression algorithms" connection attribute in order to employ such compression algorithms.

By keeping an extensive log, MySQL keeps track of all database transactions like data transfers, updates, deletions, etc. By altering or setting the files, it enables users to configure and control the log maintenance.

## **1.6 Organization**

### **1.6.1. Company Profile:**

To start an online store, ZopSmart Technology provides all the resources required. Digital marketing, e-commerce, mobile commerce, and e-wallets are among the services provided by ZopSmart. To boost automation, ZopSmart assists clients in defining business processes and determining the ideal integration requirements.

### **1.6.2. Introduction:**

Our objective is to make it simple for administrators and executives to automate their activities quickly. For any enquiries concerning online products, for instance, they can promptly create estimates. This application is designed for store owners who have access to a database of data on online goods.

### **1.6.3. Problem Statement:**

As online business has grown quite prevalent, regardless of age or gender, it is now simple to locate online web retailers on the internet.

These manual transactions squander the Owners' and Employees' valuable time and money.

### **1.6.4. Objective:**

- Refrain from performing any paper-based duties while shopping, such as noting brand and product information in diaries.
- Preserve Product and Brand information across numerous devices (mobile, diary, etc.) and multiple users (Owner, Employees) to eliminate data duplication.
- Stop wasting time, resources, efforts, and money on the part of the owner and employees.

### **1.6.5. Methodology:**

The "Three Layered Architecture", Postman-API testing, Swagger API documentation, Database Migrations, etc. are some of the key elements of this system.

In a three-layered architecture, layers are separate from one another and communicate with one another through interfaces. In essence, this helps to make our application more modular, readable, and maintainable.



## Chapter 02: Literature Survey

### 2.1 Technology and Literature Survey:

#### 2.1.1 Development Tools:

GoLand Community Edition is an integrated development environment (IDE) used in computer programming specially for Go language. It is developed by Czech company JetBrains. GoLand Community version is a free, fully featured, and extensible IDE for individual developers, open source projects, education and academic research. GoLand is cross-platform with Windows, MacOS, Linux.

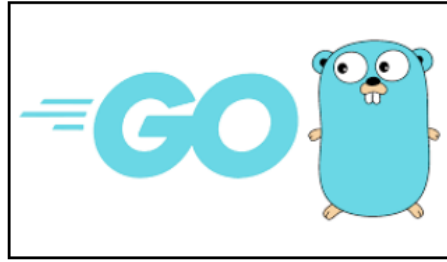
To assist developers in making changes to their code quickly and safely, GoLand also comes with a number of refactoring tools. For instance, the rename refactoring tool enables programmers to rename a symbol across their whole codebase, updating all references to that symbol at the same time.

Additionally, GoLand is fully integrated with GitHub, which makes it simple for developers to work together on projects and share code with their teams. It has functions like code review, version control, and the ability to initiate pull requests right from the IDE.

GoLand offers a number of debugging tools in addition to its code navigation and refactoring features to assist developers in finding and resolving bugs in their code. Developers may debug their code in real time thanks to the built-in debugger and interaction with the Delve debugger.

#### Language :- Go Language

Go is a general-purpose open source programming language that is sometimes known as Golang or Go. To construct stable and effective software, Google engineers created the language Go. Go is statically typed and explicit, most closely resembling C. Go is a compiled language. Go is utilised for many different applications, including DevOps, Write Rest Api, cloud and server side applications, and more.



**Figure 1.1 Go Language**

### **Database :- MySQL**

The core of Oracle's relational database management system (RDBMS), MySQL, is the structured query language (SQL).

A database is a planned collection of data. It might be anything from a simple grocery list to a photo gallery or a place to store the massive volumes of data in a corporate network. In particular, a relational database is a digital repository that gathers data and organises it according to the relational paradigm. In this paradigm, tables are made up of rows and columns, and relationships between data items all follow a strict logical structure. An RDBMS is a collection of software tools used to set up, run, and query such a database.

In addition to your other apps, web servers, and other software, MySQL Server can function smoothly on a desktop or laptop while requiring little to no maintenance. You can modify the settings to utilise all the RAM, CPU power, and I/O capacity if you dedicate a whole machine to MySQL. Additionally, MySQL can scale to networked clusters of machines.

Since its inception, MySQL Server has been used effectively in extremely demanding production environments, handling databases far faster than competing systems.



**Figure 1.2 MySQL**

## **Version Control :- Git/GitHub**

GitHub is a web-based platform for version control and collaboration for software engineers. Microsoft, GitHub's largest individual donor, bought the service for \$7.5 billion in 2018. Using a software as a service (SaaS) delivery model, GitHub was founded in 2008. Its foundation was Git, an open source code management system created by Linus Torvalds to hasten software development.

Git is a tool for storing project source code and tracking all code alterations. It enables developers to work on a project more successfully by offering methods for addressing possibly conflicting modifications from different developers.

GitHub's public repositories allow for the free modification, adaptation, and improvement of software by developers; nevertheless, the firm provides a number of paid plans for private repositories. Both public and private repositories hold all of a project's files together with each file's revision history. Repositories can have several collaborators and can be either public or private.



**Figure 1.3 Git/GitHub**

### **2.1.2 Representational State Transfer (REST)**

A REST API (also known as a RESTful API) is an application programming interface (API or web API) that complies with the restrictions of the REST architectural style. REST, also known as the representational state transfer protocol, was created by computer scientist Roy Fielding.

An API must meet the following requirements in order to be deemed RESTful:

1. An HTTP-based client-server architecture that consists of clients, servers, and resources.
2. Stateless client-server communication, where each request is independent and unconnected and no client data is stored between get requests.
3. Data that can be cached to speed up client-server communications.

### **The fundamentals of RESTful design**

**Client and server decoupling** In a REST API architecture, client and server programmes must be completely independent of one another. The client software should only be aware of the URI of the requested resource; it is unable to establish a connection to the server application in any other way. Except when it needs to send the client programme the required data over HTTP, a server application shouldn't modify the client software.

Each request must have all the information necessary to process it because REST APIs are stateless. In other words, server-side connectivity is not necessary for REST APIs. Server applications cannot save any information related to a client request.

Resources should be cacheable on both the client and server sides wherever possible. In addition, server answers must say if caching is possible for the requested resource. While enhancing client-side performance, the objective is to increase server-side scalability.

In a layered system architecture, REST API requests and responses move through various layers. Most of the time, client and server applications will speak to one another indirectly. There could be a variety of middlemen in the communication loop.

## **Status Codes for Responses**

1. OK, Success 200
2. Success+Created in 201
3. Request accepted, but not yet fulfilled (202)
4. There is no content (204)
5. Invalid Request and Syntax (400)
6. Not Found (404)
7. Method Not Allowed (405).
8. Internal Server Error: (500)

### **2.1.3 Golang HTTP Package**

A client and a server are provided by the http package. Handlers make up the server. The handler receives a request and then responds to it.

HTTP protocols, first

Create a Post-> New Data

Read: Retrieve data from Get.

Put -- update the info.

Delete: delete->delete data

ServingMux (Multiplexer)

An HTTP request multiplexer called ServeMux is in charge of matching URLs in requests to the right handlers and running them.

#### **2.1.4 Database Migration**

Application developers are in charge of creating, maintaining, and improving software; this may need you to modify or update the database structures. In a dynamic development environment, migration enables you to manage these changes quickly and consistently. The more you understand about database shaping, the better prepared you'll be to build a clear and efficient database for your application.

This capability is also offered by certain well-known frameworks like Django, Rails, and even some standalone libraries like Flyway and Liquibase.

As a version control system for your database, migrations let your team establish and share the specification of the database schema for the application.

A migration class has two methods: up and down. The schema modification you intend to make should be specified in the up method of your migration, and the down method should undo any changes performed by the up method. In other words, the database schema ought to remain constant if an up is followed by a down. For instance, if you create a table using the up approach, you should dump it using the down method.

The up method is utilised when moving a database forward in time, whereas the down approach is used when moving a database backward in time. Consequently, we have access to both older and more recent versions of our database.

### **2.2 Project Feasibility:**

#### **2.2.1 Technological Study**

It is easy to install in any system as needed because Golang was utilised in the project's design. It is easier to use, more effective, and easier for everyone to comprehend.

Massive amounts of data can be handled and stored procedures can be handled using MySQL.

#### **2.2.2 Functional Feasibility**

Anyone can use this application to review their own work, and it has a very broad user

base. These software engineer evaluation applications are becoming more prevalent day. And the application is divided into different modules so that every user has no privileges to see every module, they can see only their permissible module only.

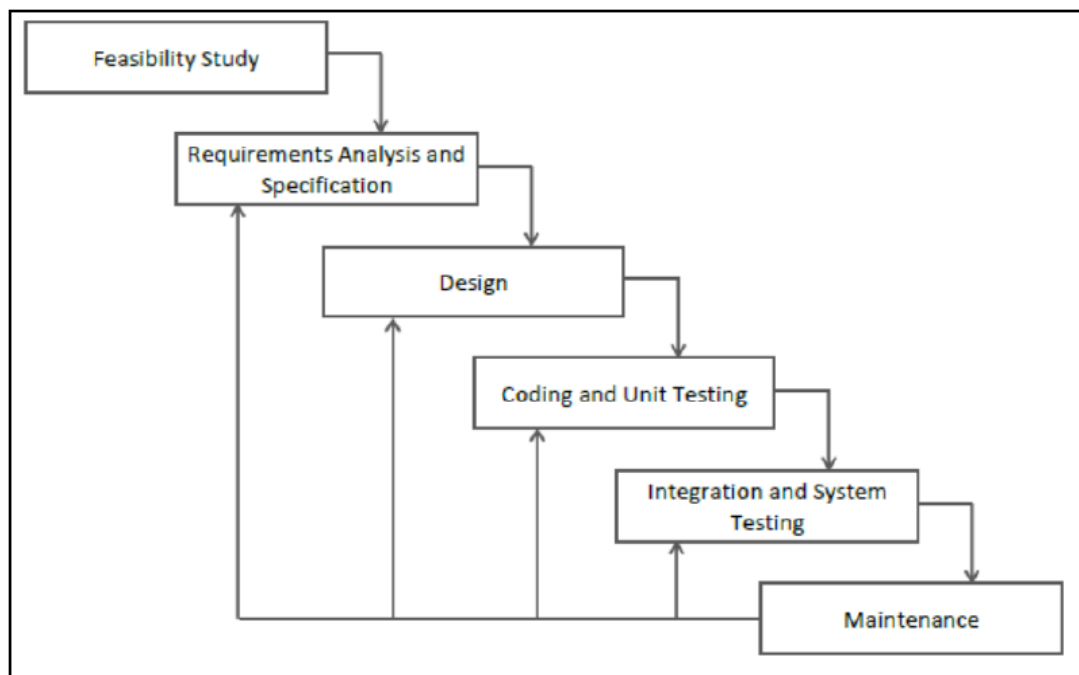
### 2.2.3 Schedule Feasibility

The project meets the requirement for time development feasibility since it is simple to run and can fulfil the essential requirements in the allotted time frame. Determining whether the deadlines were preferred or necessary was crucial.

## 2.3 Application Plan

### 2.3.1 Justification and Project Building

- Iterative Waterfall Model for Project Development
- This approach divides the cycle into the following phases:
  1. Project planning and study about requirements
  2. Analysis and formulation of requirements
  3. Design, Programming and Testing
  4. System testing and align with integration
  5. Maintaining the project



**Figure 2.1 Iterative Waterfall Model**

Iterative Waterfall Model for Project Development This approach divides the cycle into the following phases: Given that the primary requirements for the entire system were obtained at the outset, the Iterative Waterfall technique was chosen as the project's SDLC methodology. The iterative waterfall process offers the following benefits for projects where "Major requirements must be defined; however, some functionalities or requested enhancements may evolve with time":

### **2.3.2 Milestones and Deliverables**

- Feasibility analysis phase: 1 week
- Requirement analysis and Specification phase: 1 weeks
- Designing phase: Approximately 3 weeks
- Coding phase: Approximately 3 weeks
- All forms are bound with particular data from the database.
- Testing phase: Approximately 15 days
- All the needed modules were analysed and tested.
- All function work and navigate correctly with testing three times

### **2.3.3 Roles and Responsibilities**

As the project development was under an individual person, all the phases were divided into parts and each module was assigned to each person in the team. We need to complete a task within a specification defined and then finally integration of the whole work was done. I divided my work in many modules so I can design the whole application in a specific time.



<b>Name</b>	<b>System Analysis</b>	<b>Design</b>	<b>Backend Coding</b>	<b>Testing</b>	<b>Documentation</b>
Shivansh Thakur	✓	✓	✓	✓	✓

**Table 2.1 Roles and Responsibilities**

#### **2.3.4. Group Dependencies**

**The dependencies among the tasks include the following:**

- Analysis or System Requirement Study (SRS) is independent of all, yet will be started after completion of feasibility study and project planning.
- Designing of prototypes can be done simultaneously with system analysis.
- Development of the project is preceded by the designing of prototype and system analysis.
- Testing can be only done once the development of some major functionalities are completed and are ready to be tested.
- Documentation is independent of all the tasks and can be done as the other tasks proceed.
- Logical dependencies are components of a project that are essential to its success.
- Team-imposed practices result in preferential dependencies.
- Tasks with external dependencies depend on variables outside of the team's control.

### 2.2.5 Project Scheduling chart

No.	Task Name	Duration
1	System Requirement , Analysis and Project Planning	1 Week
2	System Design Layout	2 Week
3	Design	3 Week
4	Coding	3 Week
5	Testing	2 Week
6	Documentation	1 Week

**Table 2.2 Application Schedule**

## **Chapter-3 System Development and Implementation**

### **3.1 Study of Current System:**

Currently, the majority of online web store agencies run their management processes using a manual technique. Its owners lack a good system for managing details; dealership owners and their staff keep a diary to record information about all the brands and items, which frequently results in mistakes.

In the current system, if a consumer wants to buy a product or has a question about one, the product dealers and customers have very few, if any, dynamic means to display all the product-brand models, catalogues, quotes, etc. Even if many goods and their characteristics can be shown to consumers even when they are not currently accessible in the showroom, the current approach is extremely slow and not very appealing to them. When there is little or no information accessible about a specific brand, customers must make do with the product models that are currently on the market.

### **3.2 Problems and Weaknesses of Current System:**

- Most of the Online Web Store Agencies currently use a manual procedure to deal with its management processes.
- Although many products and their features can be shown to clients even when they are not currently available in the showroom, the current approach is extremely slow and not particularly appealing to them.

### **3.3 User Characteristics**

Mainly there is one system user who needs to access the system.

#### **System Administrator**

- Allow to adding Product and Brand Details
- Allow to Update all Product and Brand details
- Allow to View all Products and Brands the details

### **3.4 Assumptions and Dependencies:**

**Assumptions are described below :-**

- Users have some knowledge about the workflow of the system
- Server is running smoothly.
- Database updates are giving expected and accurate results.

**Dependencies are described as below :-**

- The system needs Internet/Wi-Fi Connection.

### **System Analysis**

#### **Requirement of new Application:**

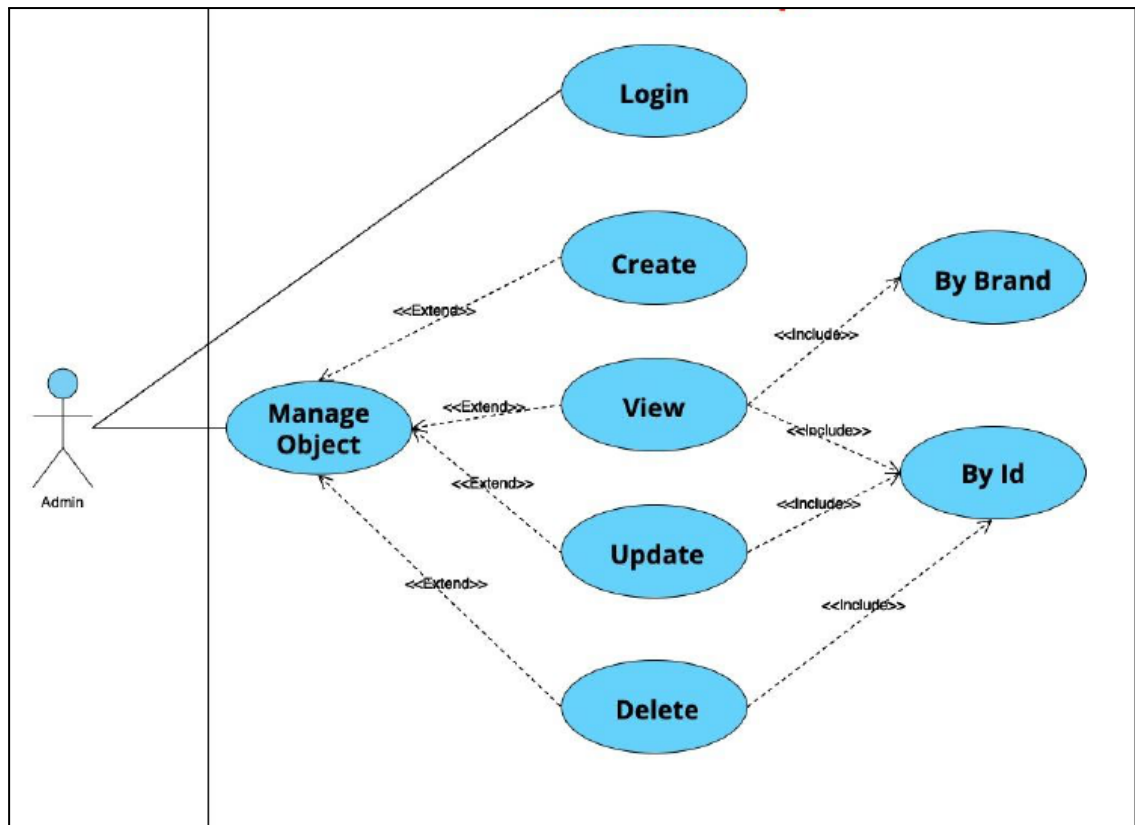
This new application requires a system that can run both the docker application, which can execute MySQL containers, and a golang server. To verify the functionality of the APIs, it should be possible to run Postman. The developers working on the project should have access to the Swaggers documentation in order to support diverse development environments. The development teams should also be given the necessary administrative access.

The needs listed in above are validated when requirement specifications are created. Users could request an illegal or unworkable solution, or experts might misinterpret the specifications. If not stopped in its tracks, this leads to a significant cost increase.

Thus, the requirements specification serves as a two-way insurance policy that ensures both the client and the software vendor comprehend the necessary functionality and scope of work.

### **Implementation Diagram**

This Figure is an implementation diagram created to illustrate the system of the Product-Brand Store. It illustrates the workflow of any online store.



**Figure 3.1 Implementation Diagram**

### System Requirements

#### Functional Requirements :

- **R1 : Create :** Adding new Product and their Respected Brand in database
  - Input :** Product name, Product Description, Product Price, Product Quantity, Product Category, Brand Name, Product Status.
  - Processing :** Insert data into database
  - Output :** data added in database
  
- **R2 : Delete :** Deleting item from database using id
  - Input :** Id
  - Processing :** Delete data from database
  - Output :** data Deleted from database

- **R3 : Update :** Updating item into database using id
  - Input :** Id
  - Processing :** update data into database
  - Output :** data updated into database
  
- **R4 : View :** View all the Product and their Brands
  - Processing :** Getting all the Product and their Brands from database
  - Output :** get all the products that is present database
  
- **R5 : View Particular Product Details :** View Particular Product with Brand
  - Input :** Name /Id
  - Processing :** Getting Product with their Brand according to input
  - Output :** get particular product with their brand details
  
- **R6 : Login :** For Login into system
  - Input :** Enter Username & Password.
  - Processing :** Validate credential
  - Output :** If all details is valid then login successfully
  
- **R7 : Registration :** for new registration
  - Input :** Enter Username, Email id , Password .
  - Processing :** Validate Details
  - Output :** If all details is valid than register successfully

## **Non-Functional Requirements :**

- **Security**

Before being saved to the database, sensitive data will be encrypted. Only administrators who have been authenticated are permitted access to the system's back end servers.

- **Responsive**

The API needs to respond. Therefore, the portion that is entirely portable on any system utilising any web browser ought to be able to utilise all system characteristics, including those of any current or future hardware platform. On PCs, laptops, and other devices, the system should function.

- **Maintainability**

The system can be readily maintained by the developers, who will be given full access and access privileges.

- **Accessibility**

The authorised developer has access to the system at any time and from any location.

- **Usability**

This system is very important in providing Developer satisfaction as they want depending on the requirement and capability.

- **Error Handling**

System should restrict entering the wrong data. System should be able to prompt various error messages to the user if they provide wrong inputs to enter the required data.

### 3.5 System Design

#### 3.5.1 Sequence Diagrams:

When a user accesses a login page, this flowchart begins. After entering their username and password, the user clicks the login button. The user login information is sent as a message from the login page to the database for verification. After the username and password have been verified by the database, two potential routes are revealed (alternative fragment). A notification indicating that the user login has been acknowledged is sent by the database if the login information is accurate. The user receives a notice from the system informing them that their login information was incorrect. The password field on the system is cleared. After that, the user can reenter their password.

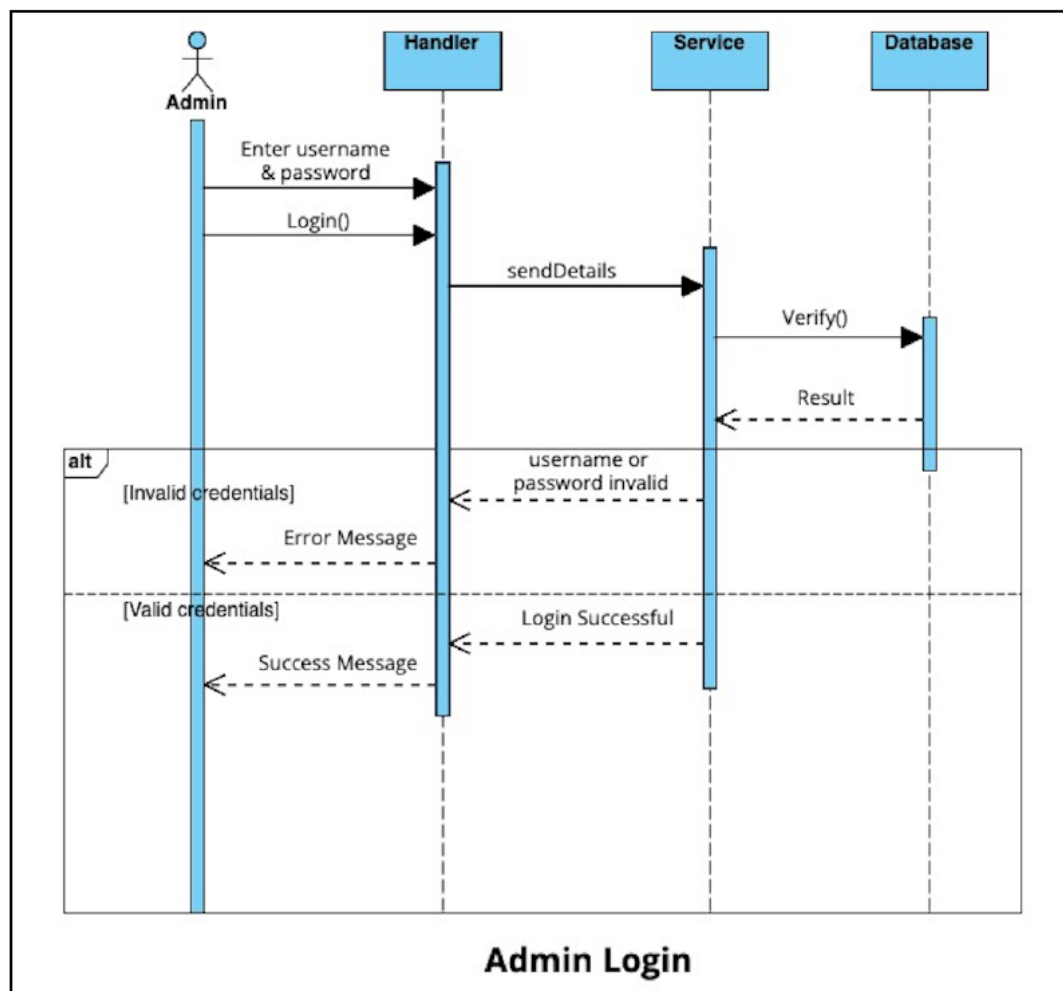


Figure 3.2 Flowchart of Login



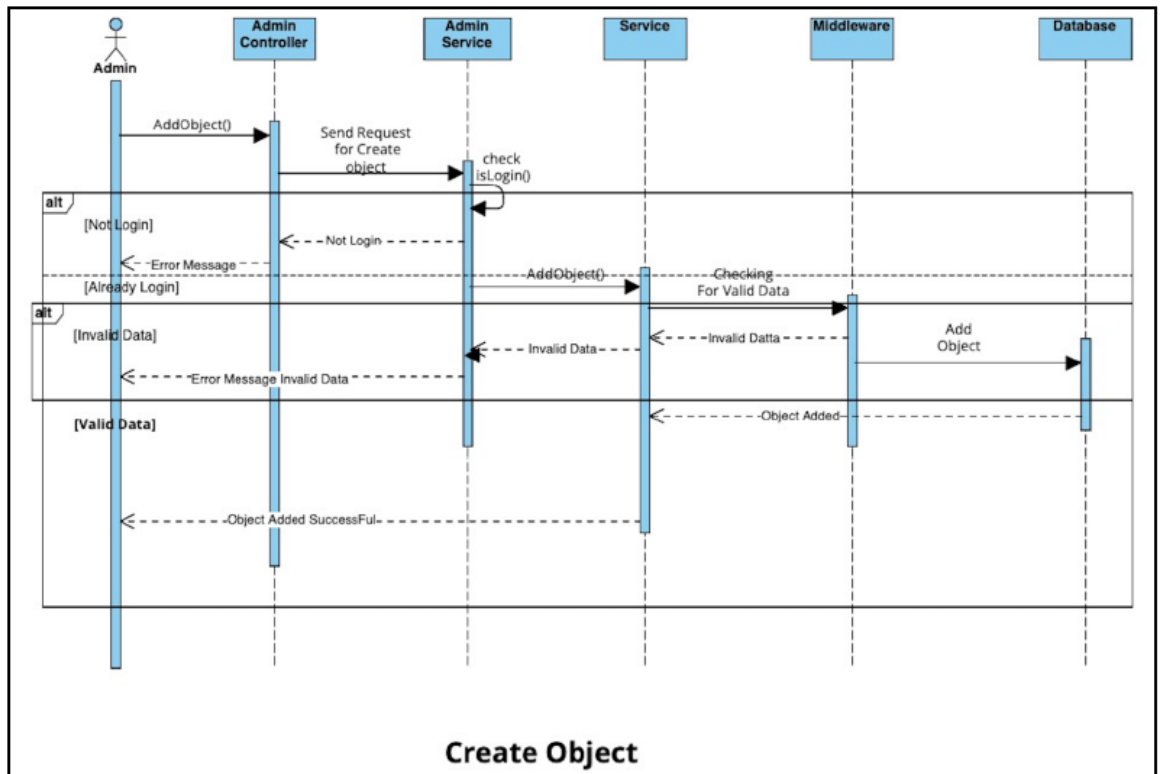


Figure 3.3 Flowchart of Create Object

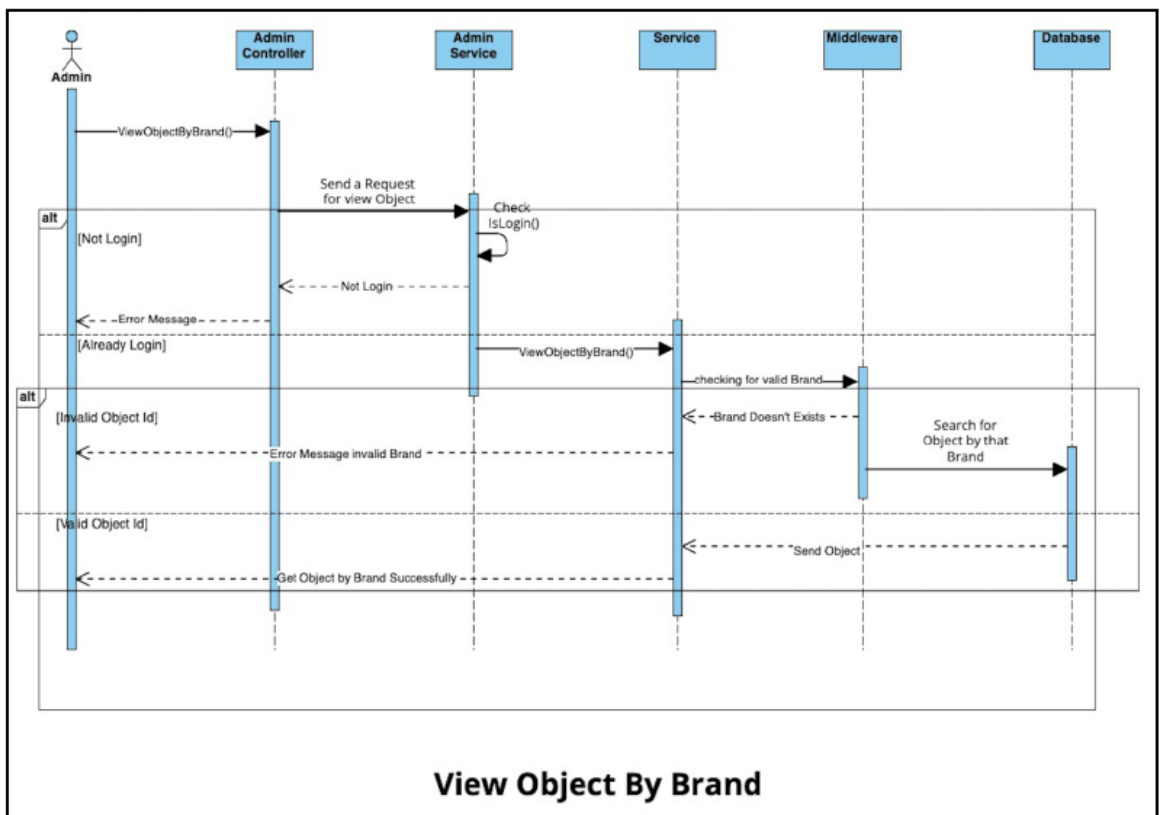


Figure 3.4 Flowchart of GetByBrand

### 3.6 DATABASE DESIGN

#### 3.6.1 Data Dictionary

Table 3.1 Product Data Dictionary

<b>Field name</b>	<b>Type</b>	<b>Size</b>	<b>NOT NULL?</b>	<b>Primary Key ?</b>
Product_id	integer	-	yes	yes
Name	varchar	35	yes	-
Description	varchar	250	-	-
Price	integer	-	-	-
Quantity	integer	-	-	-
Category	varchar	30	-	-
Brand Name	varchar	20	yes	-
Status	enum	-	yes	-

Table 3.2 Brand Data Dictionary

<b>Field name</b>	<b>Type</b>	<b>Size</b>	<b>NOT NULL ?</b>	<b>Primary Key ?</b>
Brand_id	integer	-	yes	yes
Brand Name	varchar	-	no	-

Table 3.3 Admin Data Dictionary

Field name	Type	Size	NOT NULL ?	Primary Key ?
Admin_id	Int		yes	yes
Name	Varchar	25	yes	-
Email	Varchar	25	yes	-
Password	Varchar	16	yes	-

### 3.7 Implementation Environment:

#### For Implementation we have used:

- Golang as Programming Language
- MySql as Database

#### Modules Specification

- Admin Module

#### Coding Standards

While writing our code we took the utmost care to follow the basic coding standards while writing a Golang code like,

- Limited use of global variables.
- Following proper naming conventions of local variables, global variables, constants and functions.
- Doing proper indentation.
- Proper error handling.
- Adding comments for better understanding.

### 3.9 Project Structure :

#### Three layer architecture

The handler/presentation tier, or user interface, the storage/application tier, where data is processed, and the data tier, where the application's associated data is stored and managed, are the three logical and physical computing tiers that make up the well-known three-tier architecture.

The main advantage of a three-tier architecture is that each tier may be built concurrently by a different development team and can be updated or scaled as necessary without affecting the other tiers because each tier runs on its own infrastructure. The application starts from the main.go file.

**main.go:** Source file includes all the endpoints and routers.

We require an entry point, or the point at which the programme execution starts, in order to run an application in Go as an executable programme. The main() function in the package main serves as such an entry point.

The Go compiler is informed by the package's "main" file that the package should be built as an executable programme rather than a shared library. We will configure numerous HTTP servers using http.Server in our main.go file. Your handler functions will also be updated so they may access the context.

To start the server we have to type “go run main.go” in the terminal and it gets started on port 8080. The server that listens for requests arriving from HTTP clients and one or more request servers make up a Go HTTP server.

Now, the server is ready to receive any request on the endpoints that it defines.

The request first goes to the Handler layer.

## **HANDLER LAYER:**

**http.go** : Here we process the request, unmarshal the body, check some basic validations and send it to the service layer.

**Create:** In this case, we create a new entry by sending the desired json body to the service layer, unmarshalling it, and then checking that all the data sent is accurate and adheres to the parameters specified.

**Update:** In this case, we update an already-existing record by passing the requested json body, unmarshalling it, and then delivering it to the service layer to be validated for validity and adherence to the specified parameters.

**Get By Id (Read):** In this case, we retrieve product information based on the product's id before passing it on to the service layer, which will verify that all the data is accurate and within the parameters that have been set.

**GetAll (Index):** To get all the products created, a user can make a request to get all the products for a given query-param like name. This endpoint takes the request and responds with a complete body including all the product-brand details.

## **SERVICE LAYER:**

**service.go:** We want to ensure that all of the business logic is sound before storing the data in the database, so we follow the same procedure at this layer. We guarantee that all fields are validated in accordance with the established rules.

**Create:** We verify that every newly added data is accurate and complies with established guidelines. We pass it to the datastore layer to store the data in the database as soon as we determine that there are no errors.

**Update:** We verify that all of the updated data is accurate and complies with the established rules. We pass it to the datastore layer to store the data in the database as soon as we determine that there are no errors.

**GetById and GetAll:**

We verify that the data obtained by Id is accurate and in accordance with the established regulations. We pass it to the datastore layer to retrieve the database's data once we determine there are no errors.

In GetAll, we verify the data and pass a query-param value to the store layer, based on this value we filter the response body.

**DATA STORE LAYER**

**store.go:** In this layer, we create the database query to save the data and check for any database-based issues.

The data is kept in the datastore. Any data storage device can be used. The only layer that interacts with the datastore is the use case layer. In this manner, each layer can be checked independently from the others.

**Create:** In this layer, a new entry is made by running a SQL query, and the data is subsequently saved in a database.

**Update:** In this layer, we conduct a SQL query to update an existing entry before storing the results in a database.

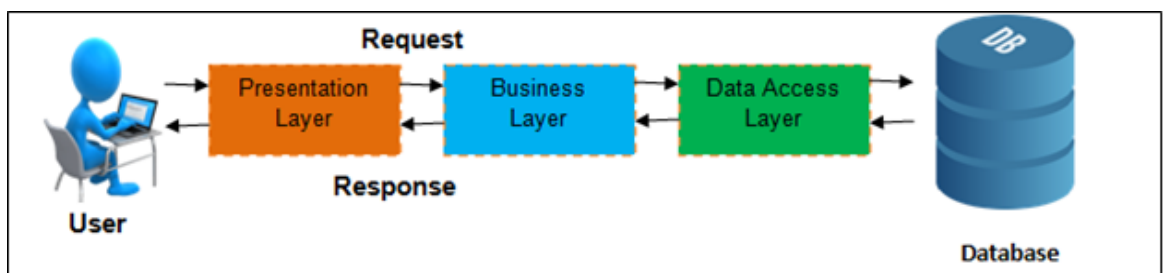
**Get By Id and Get All :**

Using a SQL query, we retrieve an entry in this tier before storing the information in the database.

In Get All by Name, we retrieve product name-specific information using a SQL query before storing the results in a database.

## MIDDLEWARE:

The phrase "middleware," which has been used in the context of software engineering since the late 1960s, can refer to a variety of contemporary software components. Application runtimes, enterprise application integration, and numerous cloud services are examples of middleware. Middleware frequently handles data management, application services, communications, authentication, and application programming interface (API) management.



As the name suggests, three-tier architecture is a hierarchical software architecture that has three different, independent tiers or layers. In a three-tier architecture, each tier has a specific task to complete and is made up of the following tiers: presentation, business, and data access, in that order. The architecture's primary function is to make it possible for software programmes to effectively and swiftly respond to user inputs or requests. A streamlined illustration of three-tier architecture can be seen below.

## **Chapter-4: Performance Analysis**

### **4.1 Project Testing Strategy**

Black box testing will be the testing approach employed in the project. Black box testing involves applying the application's anticipated inputs while just looking at the results.

### **4.2 Unit Testing Planning**

For the ensuing phases, the development process iterates this testing sub-process several times.

- Unit testing.
- Check the liners

After the coding of a unit of code (module or programme) is complete, that unit is tested. Integration testing examines how well the many programmes that comprise a system fit together, interact with one another as intended, and have proper interfaces. System testing makes certain that the system complies with its specific design requirements. Users test a system during acceptance testing to see whether it accurately implements the software requirements specification.

To make sure that each component is accurate and that the assembly or combination of components is correct, testing is done in such a hierarchical manner. Simply testing the entire system in the end would probably reveal component faults that would be very expensive to find and rectify. To find and correct mistakes, we have carried out both unit testing and system testing. Below is a quick summary of each.

### **4.3 Unit Tests**

After development is complete, the goal of unit testing is to test a single unit of code (a programme or group of programmes) using the unit test specifications. It is crucial to



subject them to quality and verification assessments because testing will depend on how accurate and comprehensive the test specifications are.

## Linter Check

Performed a linter check using command “golangci-lint” run which makes sure that the program is properly formatted and follows standard code guidelines such as no go cognitive complexity etc. There were no linter errors found in this project.

## Unit Tests Requirements

- Determining if Code goes covers the reports that have verified the existence of and adherence to coding standards are available as part of the testing process.
- Examining unit test requirements
- Check that the programme specifications and the unit test specifications are compatible.
- Check to see if all conditions and null data conditions are present.

## Program testing Cover profile

```
Terminal: Local + - v
raramuri@raramuri-HP-EliteBook-840-G7-Notebook-PC:~/github/new-assignment/go-daily-assignment/Zopstore$ go test ./... -coverprofile=c.out
?    github.com/Zopsmart-Training/go-daily-assignment/tree/fe/zopstore/shivansh/Zopstore    [no test files]
?    github.com/Zopsmart-Training/go-daily-assignment/tree/fe/zopstore/shivansh/Zopstore/constants [no test files]
ok   github.com/Zopsmart-Training/go-daily-assignment/tree/fe/zopstore/shivansh/Zopstore/internal/http/brand 0.897s coverage: 100.0% of statements
ok   github.com/Zopsmart-Training/go-daily-assignment/tree/fe/zopstore/shivansh/Zopstore/internal/http/product 1.359s coverage: 100.0% of statements
?    github.com/Zopsmart-Training/go-daily-assignment/tree/fe/zopstore/shivansh/Zopstore/internal/models [no test files]
?    github.com/Zopsmart-Training/go-daily-assignment/tree/fe/zopstore/shivansh/Zopstore/internal/services [no test files]
ok   github.com/Zopsmart-Training/go-daily-assignment/tree/fe/zopstore/shivansh/Zopstore/internal/services/brand 0.279s coverage: 100.0% of statements
ok   github.com/Zopsmart-Training/go-daily-assignment/tree/fe/zopstore/shivansh/Zopstore/internal/services/product 0.422s coverage: 100.0% of statements
?    github.com/Zopsmart-Training/go-daily-assignment/tree/fe/zopstore/shivansh/Zopstore/internal/stores [no test files]
ok   github.com/Zopsmart-Training/go-daily-assignment/tree/fe/zopstore/shivansh/Zopstore/internal/stores/brand 0.277s coverage: 100.0% of statements
ok   github.com/Zopsmart-Training/go-daily-assignment/tree/fe/zopstore/shivansh/Zopstore/internal/stores/product 0.421s coverage: 100.0% of statements
ok   github.com/Zopsmart-Training/go-daily-assignment/tree/fe/zopstore/shivansh/Zopstore/middleware 0.014s coverage: 100.0% of statements
raramuri@raramuri-HP-EliteBook-840-G7-Notebook-PC:~/github/new-assignment/go-daily-assignment/Zopstore$
```

Covering every path in the programme during testing (using white-box testing) is the best technique to ensure that you have addressed every aspect of control flow. This indicates that all branches are exercised for a case statement, all branches are exercised for a "if" statement, the loop is used once, several times, or ignored for a while statement, and all parts of complex logical expressions are exercised. This refers to Path Testing .

If the full Boolean expression is tested in control structures evaluated to both true and false, Branch Testing reports that information.

The coverage extends to switch statement cases, exception handlers, and interrupt handlers. Branch testing is included in path testing since it takes into account all potential combinations of individual branch circumstances. A more straightforward method is statement testing, which checks to see if each programme statement has been run at least once.

#### 4.4 Test Cases

Table 4.1 Test Cases For Login

<b>Test Object Descriptions</b>	<b>Test Conditions (Input)</b>	<b>Expected Result</b>	<b>Status Code</b>	<b>Actual Result</b>	<b>Pass/ Fail</b>
<b>Login</b>	Email is not correct	Validation error	404	As Expected	Pass
	Password is not correct	Validation error	404	As Expected	Pass

Table 4.4 Test Cases for GetById Object

<b>Test Object Descriptions</b>	<b>Test Conditions (Input)</b>	<b>Expected Result</b>	<b>Status Code</b>	<b>Actual Result</b>	<b>Pass/ Fail</b>
<b>Get By Id</b>	Valid Id	View Object for particular Id	200	As Expected	Pass

	Invalid Id	Error Message Invalid Id	404	As Expected	Pass
	Empty Id	Error Message Please Enter Id	400	As Expected	Pass

The screenshot shows a REST client interface for a project named 'zopstore'. The selected endpoint is 'product-api / GetByID (x-api-key)' with a GET method. The request URL is 'http://localhost:8080/products/1?brand=true'. The response is a 200 OK status with a 61 ms response time and 564 B of data. The response body is displayed in JSON format:

```

1  {
2    "data": {
3      "id": 1,
4      "name": "macbook",
5      "description": "Laptop",
6      "price": 45000,
7      "quantity": 1,
8      "category": "electronics",
9      "brand": {
10       "id": 1,
11       "name": "Apple"
12     },
13     "status": "Available"
14   }
15 }

```

**Success Case (200 code) - Get request for product**

Table 4.2 Test Cases for Create Object

Test Object Descriptions	Test Conditions (Input)	Expected Result	Status Code	Actual Result	Pass/Fail
Delete	Valid Id	Object Created Successfully	201	As Expected	Pass
	Invalid Brand	Error Message Invalid Brand	404	As Expected	Pass
	Empty Id	Error Message Please Enter Id	400	As Expected	Pass

The screenshot shows a REST client interface for a 'zopstore / product-api / Create (x-api-key / x-org)' endpoint. The request is a POST to 'http://localhost:8080/products'. The request body is a JSON object:

```

1 {
2   "name": "macbook pro",
3   "description": "Laptop",
4   "price": 45000,
5   "quantity": 1,
6   "category": "electronics",
7   "brand": {
8     "id": 1,

```

The response status is 201 Created, with a response time of 20 ms and a body size of 573 B. The response body is a JSON object:

```

1 {
2   "data": {
3     "id": 4,
4     "name": "macbook pro",
5     "description": "Laptop",
6     "price": 45000,
7     "quantity": 1

```

**Success Case (201 code) - Post request for product**

Table 4.3 Test Cases for Update Object

Test Object Descriptions	Test Conditions (Input)	Expected Result	Status Code	Actual Result	Pass/Fail
Update Object	Name or Brand or Description or Price or Quantity or Statuts	Object Updated Successful-ly	200	As Expected	Pass
	If Id == 1002	Invalid Id	400	As Expected	Pass

The screenshot shows a REST client interface for a project named 'zopstore'. The selected endpoint is 'product-api / Update (x-api-key)' with a PUT method and URL 'http://localhost:8080/products/1'. The request body is in JSON format, containing product details: id: 1, name: 'mac', description: 'Laptop', price: 45000, quantity: 1, category: 'electronics', and brand: { id: 1, name: 'Apple'}. The response is a 200 OK status with a response time of 20 ms and a body size of 560 B. The response body is displayed in JSON format, showing the updated product data.

```

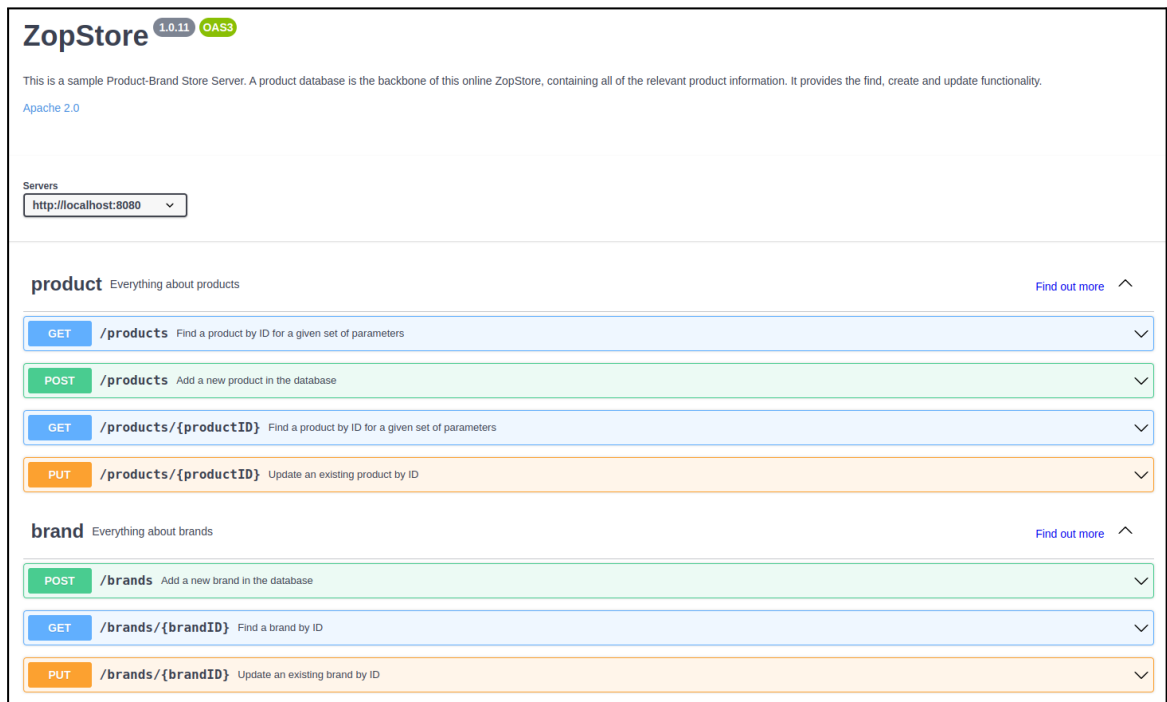
1  {
2    "data": {
3      "id": 1,
4      "name": "mac",
5      "description": "Laptop",
6      "price": 45000,
7      "quantity": 1,
8      "category": "electronics",
9      "brand": {
10     "id": 1,
11     "name": "Apple"
12   },
13     "status": "Available"
14   }
15 }

```

**Success Case (200 code) - Update request for product**

## 4.5 Swagger documentation:

Developers can write interactive, machine- and human-readable API documentation using the Swagger framework. Information like supported operations, parameters and outputs, authorisation needs, accessible endpoints, and required licences are frequently included in API specifications.



The screenshot displays the Swagger UI for the ZopStore API. At the top, the title 'ZopStore' is shown with version '1.0.11' and 'OAS3' tags. Below the title, a brief description states: 'This is a sample Product-Brand Store Server. A product database is the backbone of this online ZopStore, containing all of the relevant product information. It provides the find, create and update functionality.' The Apache 2.0 license is also mentioned. A 'Servers' dropdown menu is set to 'http://localhost:8080'. The main content is organized into two sections: 'product' and 'brand'. The 'product' section includes four endpoints: GET /products (Find a product by ID for a given set of parameters), POST /products (Add a new product in the database), GET /products/{productID} (Find a product by ID for a given set of parameters), and PUT /products/{productID} (Update an existing product by ID). The 'brand' section includes three endpoints: POST /brands (Add a new brand in the database), GET /brands/{brandID} (Find a brand by ID), and PUT /brands/{brandID} (Update an existing brand by ID). Each endpoint is color-coded: blue for GET, green for POST, and orange for PUT. 'Find out more' links are present at the end of each section.

### Swagger documentation for the Product-Brand API

The core of all goodness in Swagger is the ability of APIs to describe their own structures. Why is it so fantastic? We can, however, automatically create stunning and interactive API documentation by reading the structure of your API.

Its sophisticated auto-completion feature helps us create code more quickly. It is simple to set up and allows developers to quickly build server stubs for the API. Developers may see in real-time how the API design is progressing, including how a third-party developer might interact with the API, by receiving fast responses from the stubs.

For instance, the **Products GET endpoint** accepts some query-parameters and filters the returned products based on the parameters. This GET endpoint additionally addresses all improbable scenarios for both invalid and legitimate queries. Other endpoint documentations mention similar validations.

**product** Everything about products [Find out more](#) ^

**GET** **/products** Find a product by ID for a given set of parameters

Retrieves the product information from the database for given query param and valid headers.

**Parameters** [Try it out](#)

Name	Description
<b>x-api-key</b> * required string (header)	Authorization header key is product-r <input type="text" value="product-r"/>
<b>X-ORG</b> string (header)	Authorization header key is organization name <input type="text" value="zs"/>
<b>brand</b> string (query)	If brand set to true, it will get the products with brand-name, else without brand-name. Default value : available <input type="text" value="true"/>
<b>name</b> string (query)	If product name is given, it will filter products by product name, else without product-name. Default value : available <input type="text" value="macbook"/>

### Example: Product GET endpoint

**Responses**

Code	Description	Links
200	successful operation	No links

Media type:

Example Value | Schema

```
{
  "id": 1,
  "name": "macbook",
  "description": "Laptop",
  "price": 45000,
  "quantity": 1,
  "category": "electronics",
  "brand": {
    "id": 1,
    "name": "Apple"
  },
  "status": "Available"
}
```

### Response with 200 status code for valid requests

401	Unauthorized. If header "x-api-key" is not set to "product-r".	No links
	Media type <input type="text" value="text/plain; charset=utf-8"/>	
	Example Value   Schema	
	<b>Error: Unauthorized</b>	
403	Permission not granted.	No links
	Media type <input type="text" value="text/plain; charset=utf-8"/>	
	Example Value   Schema	
	<b>Error: Forbidden</b>	
404	Entity not found.	No links
	Media type <input type="text" value="application/json"/>	
	Example Value   Schema	
	<pre>[   {     "code": "Entity Not Found",     "reason": "No 'products' found for Id: '900'",     "datetime": {       "value": "IST",       "timezone": "2023-03-26T16:42:35.000Z"     }   } ]</pre>	

## Error Responses for invalid requests



## **Chapter-5 CONCLUSION**

### **5.1 Conclusion**

I learned so much while working on this project, which was amazing. I experienced all of the project development phases while working on this project, which really opened my eyes to the world of software engineering. I got a sense of the developer industry through the pleasure of working and the thrill of meeting different obstacles. I learned how professional software is created as a result of this project.

In this project, we created a dependable, simple, affordable, and practical system to manage product and brand details. Owners may therefore easily and effectively manage the details of their online business. For the owner, it saves a tonne of time and money.

### **5.2 Discussion**

We started the project by understanding the basics of Golang, Git/Github, Postman, MySQL then understanding the workflow flow of software development life cycle and after that moved to implementation. I took time to learn all these things. Implementation of this WEB API took a major portion of time. And then I tested the whole WEB API. Once that was complete, I had confidence that the project would reach a successful conclusion.

### **5.3 Future Scope and Enhancements**

- Efficiency increase
- Authentication technique can be enhance
- New field to be include in API Data Dictionary
- Structure of API can be enhance
- Adding more numbers of Filters for searching the data

## **5.4 Application Contributions**

Several real-world and open source applications that GoLang has contributed below.

1. The container management system and a suite of tools for deploying Linux containers called Docker
2. Dropbox switched several of its crucial Python components to Go.
3. Ethereum, a blockchain for the cryptocurrency Ether that uses the go-ethereum version of the Ethereum Virtual Machine.
4. Gitlab, a web-based platform for the DevOps lifecycle that offers a Git repository, a wiki, functionality for recording issues, continuous integration, deployment pipelines, etc.

## **5.5 Limitations**

- Limited numbers of Filters
- Limited Numbers of data
- Authentication in API is not much secure
- The application implements only the backend part

## REFERENCES

1. [https://docs.microsoft.com/en-us/azure/architecture/best-practices/api design](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](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/>

## APPENDICES

### Application Code Examples:

By entering "go run main.go" in the terminal, the server is launched on port 8080. A Go HTTP server is made up of a server that watches for requests coming from HTTP clients and one or more request servers.

**main.go:** Source file includes all the endpoints and routers.

**.env file:** Environment variables provide details on the working environment of the process (production, development, build pipeline, etc.). Passwords, API credentials, and other private information that shouldn't be written directly in code are stored as environment variables in Node. Any settings or configuration information that may vary between environments must be configured using environment variables.

A screenshot of a code editor window with two tabs: ".env" and ".local.env". The ".env" tab is active and shows four lines of code. Line 1: APP\_NAME = product-api. Line 2: APP\_VERSION = dev. Line 3: HTTP\_PORT = 8080. Line 4: LOG\_LEVEL = INFO. The fourth line is highlighted in yellow. The editor has a light gray background and a dark gray border.

```
.env x .local.env x
1 APP_NAME = product-api
2 APP_VERSION = dev
3 HTTP_PORT = 8080
4 LOG_LEVEL = INFO
```

### **.env config file**

After setting up the .env file we can set the .local.env file for our local environment variables.

```
.env x .local.env x
1 DB_NAME = zopstore
2 DB_HOST = localhost
3 DB_PORT = 3306
4 DB_USER = root
5 DB_PASSWORD = shivansh
6 DB_DIALECT = mysql
```

### **.local.env config file**

After setting up all the configs now we are ready to run the server. To run the server use “go run main.go” in the terminal.

Now the server is ready to receive any request on the endpoints that it defines. The request first goes to the Handler layer.

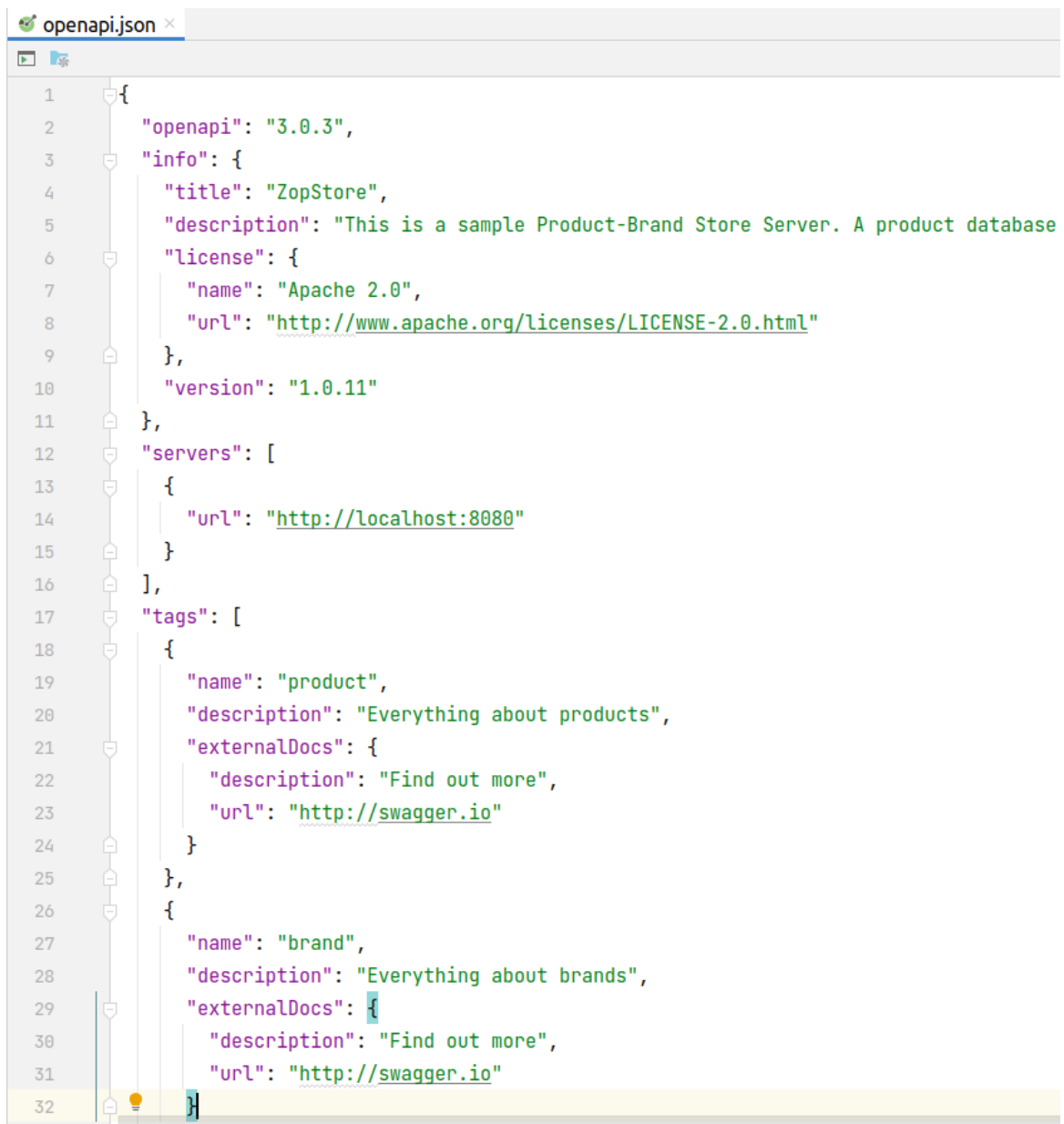
### **HANDLER LAYER:**

Here we process the request, unmarshal the body, check some basic validations and send it to the service layer.

By forwarding calls to the following layer, known as Business Logic, the Handler layer handles incoming requests and interactions. In order to meet the requirements already existent in the presentation layer itself, it could also activate other systems or applications. Since it is the only layer responsible for consumer engagement, it is interesting to note that the presentation layer of the current application communicates with other applications through their presentation layers.

## Openapi.json:

A standard for a machine-readable interface definition language for describing, generating, consuming, and visualising online services is the OpenAPI standard, formerly known as the Swagger Specification.

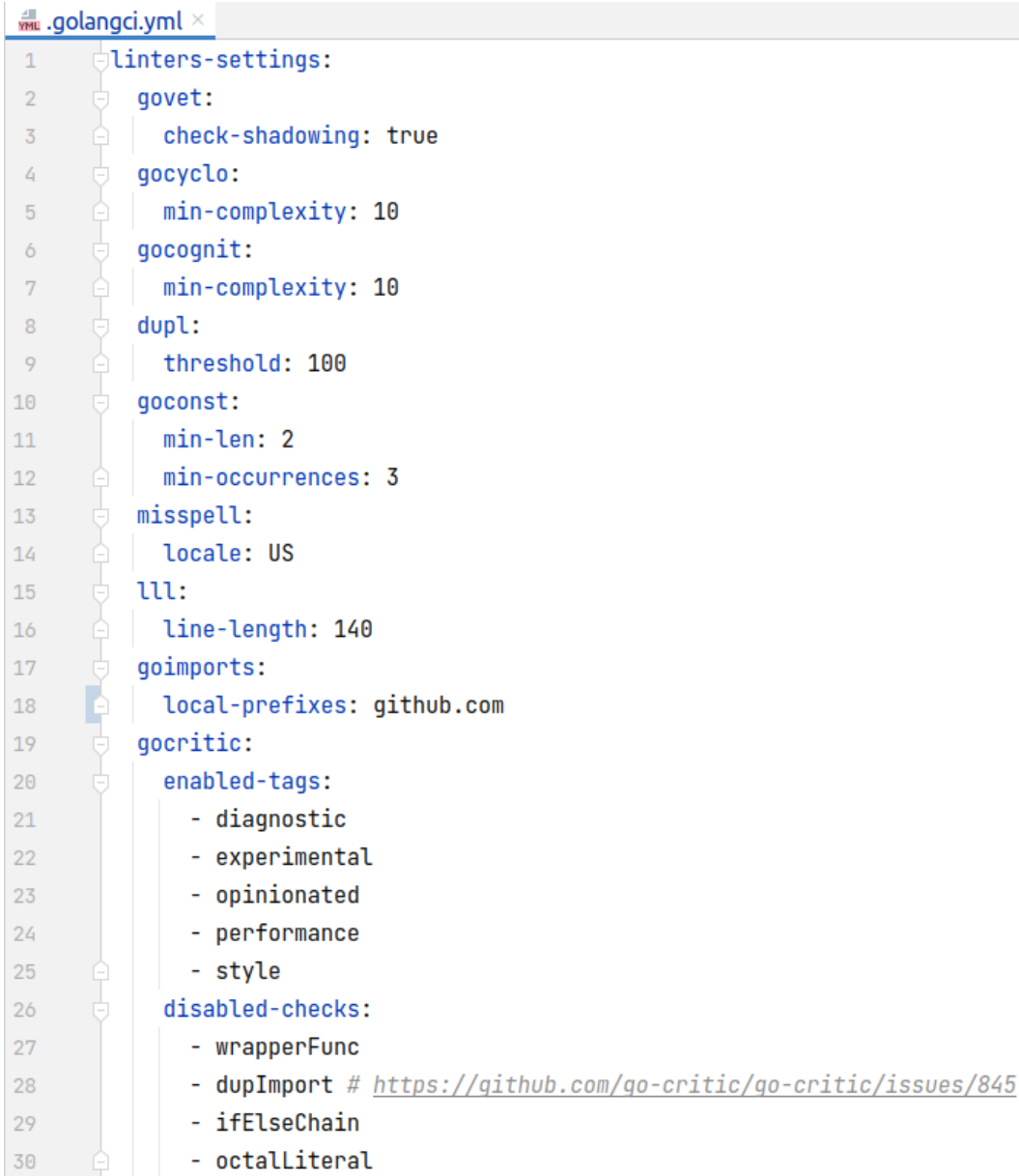


```
1  {
2    "openapi": "3.0.3",
3    "info": {
4      "title": "ZopStore",
5      "description": "This is a sample Product-Brand Store Server. A product database",
6      "license": {
7        "name": "Apache 2.0",
8        "url": "http://www.apache.org/licenses/LICENSE-2.0.html"
9      },
10     "version": "1.0.11"
11   },
12   "servers": [
13     {
14       "url": "http://localhost:8080"
15     }
16   ],
17   "tags": [
18     {
19       "name": "product",
20       "description": "Everything about products",
21       "externalDocs": {
22         "description": "Find out more",
23         "url": "http://swagger.io"
24       }
25     },
26     {
27       "name": "brand",
28       "description": "Everything about brands",
29       "externalDocs": {
30         "description": "Find out more",
31         "url": "http://swagger.io"
32       }
33     }
34   ]
35 }
```

Openapi.json file

## **.golangci.yml:**

GolangCI-Lint looks for configuration files in every directory from the first analysed path's directory all the way up to the root. GolangCI-Lint will look in your home directory for a configuration file if none have been found. Run `golangci-lint` with the `-v` option to examine which configuration file is being utilised and where it was sourced from. The file's configuration parameters are the same as command-line options. Only within the configuration file (not the command-line) can certain linters' settings be changed.

A screenshot of a code editor showing the contents of a .golangci.yml file. The editor has a tab at the top labeled ".golangci.yml" with a close button. The code is displayed in a light blue font on a white background. The file contains a YAML configuration for GolangCI-Lint, with line numbers 1 through 30 on the left margin. The configuration includes sections for "linters-settings", "govet", "gocyclo", "gocognit", "dupl", "goconst", "misspell", "lll", "goimports", "gocritic", and "disabled-checks". The "gocritic" section lists several tags: diagnostic, experimental, opinionated, performance, and style. The "disabled-checks" section lists wrapperFunc, dupImport (with a link to a GitHub issue), ifElseChain, and octalLiteral.

```
1  linters-settings:
2    govet:
3      check-shadowing: true
4    gocyclo:
5      min-complexity: 10
6    gocognit:
7      min-complexity: 10
8    dupl:
9      threshold: 100
10   goconst:
11     min-len: 2
12     min-occurrences: 3
13   misspell:
14     locale: US
15   lll:
16     line-length: 140
17   goimports:
18     local-prefixes: github.com
19   gocritic:
20     enabled-tags:
21       - diagnostic
22       - experimental
23       - opinionated
24       - performance
25       - style
26     disabled-checks:
27       - wrapperFunc
28       - dupImport # https://github.com/go-critic/go-critic/issues/845
29       - ifElseChain
30       - octalLiteral
```

## **.golangci.yml file**

## SERVICE LAYER:

This layer guarantees that all fields are validated in accordance with the established rules.

### CheckMissingFields():

```
service.go x
82 // CheckMissingFields adds missing fields into slice of strings and return tha
83 func CheckMissingFields(product *models.Product) []string { 3 usages  ± shivansh-zs
84     missingData := make([]string, 0)
85
86     if product.Name == "" {
87         missingData = append(missingData, elems...: "Name")
88     }
89
90     if product.Price == 0 {
91         missingData = append(missingData, elems...: "Price")
92     }
93
94     if product.Quantity == 0 {
95         missingData = append(missingData, elems...: "Quantity")
96     }
97
98     if product.Category == "" {
99         missingData = append(missingData, elems...: "Category")
100    }
101
102    if product.Brand.ID == 0 {
103        missingData = append(missingData, elems...: "Brand ID")
104    }
105
106    if product.Status == "" {
107        missingData = append(missingData, elems...: "Status")
108    }
109
110    return missingData
111 }
112
```

### CheckMissingFields function in Service/Business Layer

## DATA STORE LAYER

The data is kept in the datastore. Any data storage device can be used. The only layer that interacts with the datastore is the use case layer.



## MIDDLEWARE:

Due to its extension from the conventional 2-tier architecture, the 3-tier architecture is regarded as the second generation of client-server architecture. The third tier of the three-tier architecture between the client and database servers adds an application server as middleware.

## KeyMiddleware() Method:

```
middleware.go x
1 package middleware
2
3 import (
4     "context"
5     "net/http"
6     "strings"
7
8     "github.com/Zopsmart-Training/go-daily-assignment/tree/fe/zopstore/shivansh/Zopstore/constants"
9 )
10
11 type ValidEndpoints struct { 7 usages  shivansh-zs
12     Method []string
13     Endpoint []string
14 }
15
16 // KeyMiddleware Process the request using valid methods and endpoints for authentication.
17 func KeyMiddleware(handler http.Handler) http.Handler { 3 usages  shivansh-zs *
18     tokenMethods := make(map[string]ValidEndpoints)
19
20     tokenMethods["product-r"] = ValidEndpoints{Method: []string{"GET"},
21         Endpoint: []string{"products", "brands"}}
22     tokenMethods["product-w"] = ValidEndpoints{Method: []string{"POST", "PUT"},
23         Endpoint: []string{"products", "brands"}}
24     tokenMethods["brand-r"] = ValidEndpoints{Method: []string{"GET"},
25         Endpoint: []string{"brands"}}
26     tokenMethods["brand-w"] = ValidEndpoints{Method: []string{"POST", "PUT"},
27         Endpoint: []string{"brands"}}
28
29     return http.HandlerFunc(func(w http.ResponseWriter, r *http.Request) {
30         reqToken := r.Header.Get("x-api-key")
31
32         _, ok := tokenMethods[reqToken]
33         if !ok {
34             w.WriteHeader(http.StatusUnauthorized)
35             return
36         }
37
38         reqMethod := r.Method
39         p := strings.Split(r.URL.String(), "/")[1]
40         reqEndpoint := strings.Split(p, "?")[0]
41     })

```

## ValidateMethod() method:

```
middleware.go x
34     w.WriteHeader(http.StatusUnauthorized)
35     return
36 }
37
38     reqMethod := r.Method
39     p := strings.Split(r.URL.String(), sep: "/")[1]
40     reqEndpoint := strings.Split(p, sep: "?")[0]
41
42     var methodFound = ValidateMethod(reqMethod, reqToken, tokenMethods)
43     var endpointFound = ValidateEndpoint(reqEndpoint, reqToken, tokenMethods)
44
45     if !methodFound || !endpointFound {
46         w.WriteHeader(http.StatusForbidden)
47         return
48     }
49
50     handler.ServeHTTP(w, r)
51 })
52 }
53
54 // ValidateMethod returns a flag, it tells whether the request method is valid or not
55 func ValidateMethod(reqMethod, reqToken string, tokenMethods map[string]ValidEndpoints) bool {
56     validMethod := tokenMethods[reqToken].Method
57
58     var flag = false
59
60     for _, mtd := range validMethod {
61         if reqMethod == mtd {
62             flag = true
63             break
64         }
65     }
66
67     return flag
68 }
69 }
```

## ValidateEndpoint() and OrgMiddleware() methods:

```
middleware.go x
70
71 // ValidateEndpoint returns a flag, it tells whether the request endpoint is valid
72 func ValidateEndpoint(reqEndpoint, reqToken string, 2 usages ± shivansh-zs*
73     tokenMethods map[string]ValidEndpoints) bool {
74     validEnd := tokenMethods[reqToken].Endpoint
75
76     var flag = false
77
78     for _, edp := range validEnd {
79         if reqEndpoint == edp {
80             flag = true
81             break
82         }
83     }
84
85     return flag
86 }
87
88 // OrgMiddleware Process the request using valid methods & endpoints, and adds orga
89 func OrgMiddleware(handler http.Handler) http.Handler { 3 usages ± shivansh-zs
90     return http.HandlerFunc(func(w http.ResponseWriter, r *http.Request) {
91         org := r.Header.Get(key: "x-org")
92
93         switch r.Method {
94         case http.MethodGet:
95             url := r.URL
96             q := url.Query()
97             q.Add(key: "organization", org)
98             url.RawQuery = q.Encode()
99         case http.MethodPost, http.MethodPut:
100             ctx := context.WithValue(r.Context(), constants.CtxValue, org)
101             r = r.WithContext(ctx)
102         default:
103             w.WriteHeader(http.StatusMethodNotAllowed)
104             return
105         }
106
107         handler.ServeHTTP(w, r)
108     })
109 }
```