Web Application in Go using Three Layered Architecture

Major project report submitted in partial fulfillment of the requirement

for the degree of Bachelor of Technology

in

Computer Science and Engineering

By

Saksham Chaturvedi (191220)

UNDER THE SUPERVISION OF

Dr. Jagpreet Sidhu

Department of Computer Science & Engineering and Information Technology

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

DECLARATION

I hereby declare that the work presented in this report entitled "**Web Application in Go using Three Layered Architecture**" 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 15th Feb 2023 to 31st March 2023 under the supervision of **Dr. Jagpreet Sidhu, Assistant Professor (SG)** (Department of Computer Science & Engineering and Information Technology Jaypee University of Information Technology)

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

Saksham

Submitted by:
Saksham Chaturvedi (191220)
Computer Science & Engineering Department
Jaypee University of Information Technology

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

Supervised by:

[illegible]

Dr. Jagpreet Sidhu
Assistant Professor (SG)
Department of Computer Science & Engineering and Information Technology
Jaypee University of Information Technology

MShetty

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

JAYPEE UNIVERSITY OF INFORMATION TECHNOLOGY, WAKNAGHAT

PLAGIARISM VERIFICATION REPORT

Date: 12-05-2023

Type of Document (Tick): B.Tech Project Report

Name: SAKSHAM CHATURVEDI Department: CSE Enrolment No 191220

Contact No. 9582064701 E-mail. 191220@juitsolan.in Name of the Supervisor: Dr. Jagpreet Sidhu Title of the Thesis/Dissertation/Project Report/Paper (In Capital letters): Web Application in Go using Three Layered Architecture

UNDERTAKING

I undertake that I am aware of the plagiarism related norms/ regulations, if I am found guilty of any plagiarism and copyright violations in the above thesis/report even after award of degree, the University reserves the right 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 = 50
- Total No. of Preliminary pages = 6
- Total No. of pages accommodate bibliography/references = 2

Saksham

(Signature of Student)

FOR DEPARTMENT USE

We have checked the report as per norms and found the **Similarity Index** at **11**(%). Therefore, we are forwarding the complete report for final plagiarism check. The plagiarism verification report may be handed over to the candidate.

[illegible]

(Signature of Guide/Supervisor)

[illegible]

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

....

ACKNOWLEDGEMENT

Foremost, I would like to express my heartiest gratefulness to almighty God for his divine blessing that made it possible for me to complete the project successfully.

I am extremely grateful to my supervisor, **Dr. Jagpreet Sidhu**, Assistant Professor (SG), Department of CSE Jaypee University of Information Technology, Wakhnaghat, for his assistance to complete this assignment. My supervisor has extensive knowledge and a deep interest in the subject of Web development. His never-ending patience, intellectual direction, constant encouragement, constant and energetic supervision, constructive criticism, good suggestions, and reading many poor versions and fixing them at all stages made it possible to finish this job.

I would also like to express my gratitude to everyone who has directly or indirectly assisted me in making this project a success. In this unique scenario, I'd like to appreciate the different staff members, both teaching and non-teaching, who have developed their helpful assistance and facilitated my project. Finally, I must express my gratitude for my parents' unwavering support and patience.

Saksham

Submitted by:
Saksham Chaturvedi (191220)
Computer Science & Engineering Department
Jaypee University of Information Technology

TABLE OF CONTENTS

REFERENCES

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

Creating a web application is quite simple but the challenge comes when the code has to be tested, structured, cleaned and maintained and thus here we follow the Three Layered Architecture using Go language.

The three layers are handler, service and datastore which are all independent of each other. The handler layer receives the request body and then parses anything that is required from that request. It then calls the service layer where all the logic of the program is defined, ensures that the response is the required format and writes it to the response writer. This layer further communicates with the datastore layer. It takes whatever it needs from the handler layer and then calls the datastore layer. The datastore layer is where all the data is stored. It can be any data storage. The use case layer is the only layer that communicates with the datastore. That is how we test each layer independently making sure that no layer affects the other.

CHAPTER 1 : INTRODUCTION

1.1 Company

ZopSmart is a software solution technology company that provides you with all the tools to build your e-commerce business . ZopSmart has a suite of products that will help you build the perfect business you're aiming to open and run. It has different products such as Smart Store Eazy, Smart Payment Gateway, etc. Zopsmart is building next generation technology for the retail sector and their customers range from a small furniture shop to multinational retail chains and solutions include an e-commerce platform,Digital Marketing , m-Commerce, automated logistics systems, management platform, order management platform, and iOT devices. It also provides software solutions to some of the top-most firms and has its framework to work on.

1.2 Introduction

It is a basic web application that implements CRUD operations based on the three layered architecture. Programs at each layer have their own unit test. There is also an implementation of middleware that authenticates the http request before sending it to the server.

1.3 Objectives

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

1.4 Motivation

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

1.5 Libraries/Frameworks Used

GO - *A*n open source programming language developed by Google engineers aim to build simple, reliable, and efficient code for applications.

GO provides various packages that are used in this project such as:9

1. **net/http**: This package provides http client/server implementations.
2. **json**: Encoding and decoding of JSON is implemented by this package.
3. **errors**: Manipulation of errors is implemented by this function.
4. **database/sql**: This package provides SQL-like databases.

And for unit testing: gomock and sqlMock is used

MOVING FORWARD WITH GO

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

1.5 Technical Requirements

- **GOLand** is an IDE to write clean code .
- **Postman** API platform for building and using APIs.
- **Mysql** server provides a database management system with querying and connectivity capabilities

1.5.1 Hardware Configuration

Table 1 : Hardware Configuration

Processor	HP EliteBook
RAM	8 GB
Hard Disk	256 GB SSD
Monitor	13”
Mouse	Integrated
Keyboard	Integrated

1.5.2 Software Configuration

Table 2 : Software Configuration

Operating System	Ubuntu
Language	GO
Runtime environment	GO runtime
Package Manager	GO

CHAPTER 2 : LITERATURE SURVEY

GO Documentation

The GO programming language comes with extensive documentation that developers may use while creating programmes.

The GO documentation includes a detailed explanation of the language's syntax and usage, as well as numerous examples, best practices, and advice for developing efficient and secure code.

The official GO documentation contains a language tour- gotour as well as reference and package papers that define the GO standard library.

Github & Git

The official documentation Introduction to Git and GitHub gives an introduction of version control systems and how they function. ,

It focuses specifically on the Git version control system and how it combines with GitHub, a major web-based platform for code hosting and collaboration.

The manual describes how to utilize essential Git concepts including repositories, branches, commits, and merging in the context of software development.

It also explains how to get started with Git and GitHub, including how to create accounts, create repositories, and collaborate with other developers.

MySQL Documentation

MySQL is a prominent open-source relational database management system for effectively storing, retrieving, and managing data

It is built to manage massive volumes of data and several concurrent users, making it a popular choice for online applications that need a dependable database system.

Its SQL-based language supports a wide range of functions, including the ability to save and retrieve data, handle transactions, and run complicated queries.

GoMocks

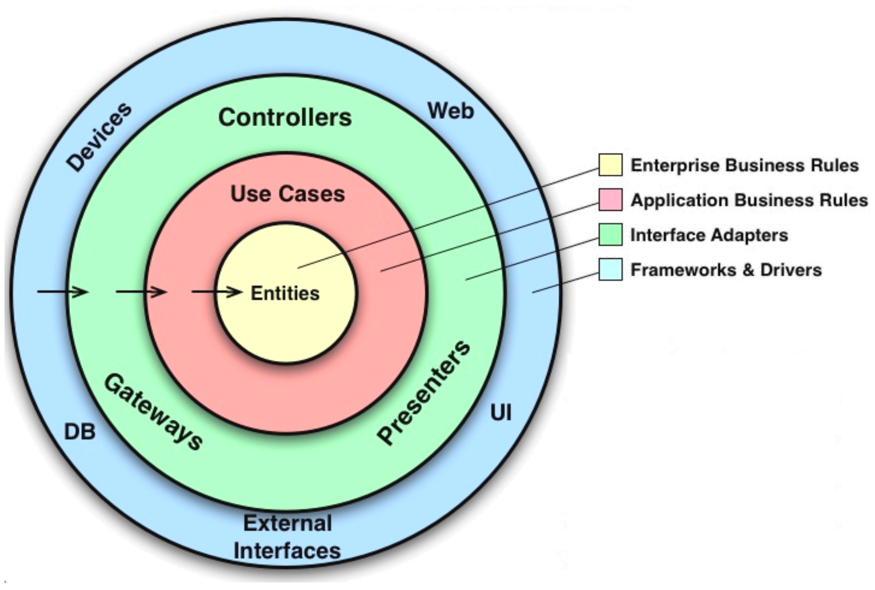
Gomock is a renowned mocking framework that was created explicitly for the Go programming language.

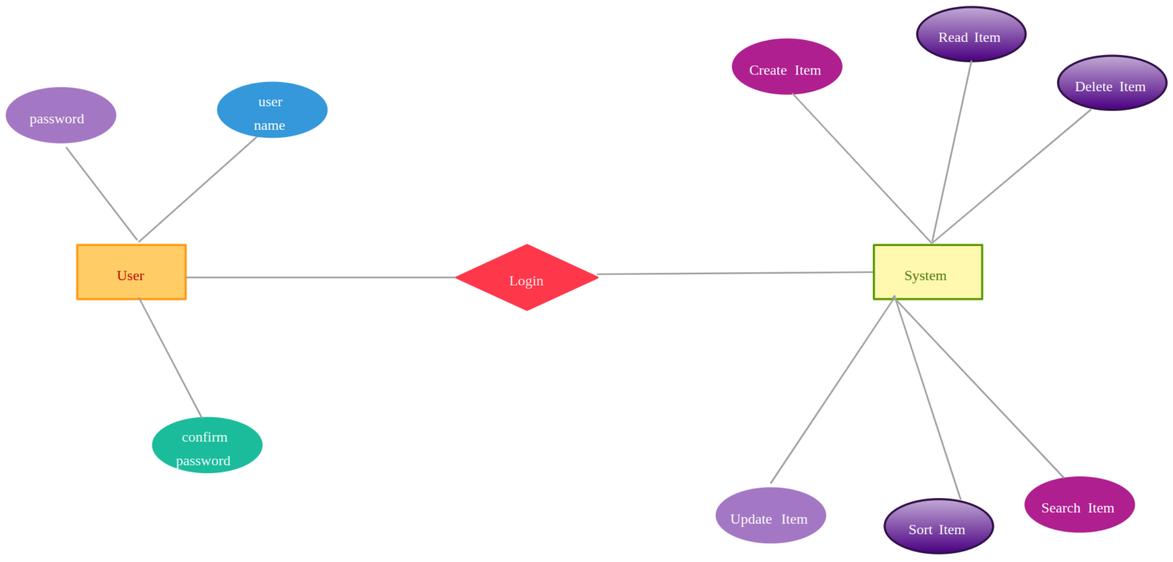
It enables developers to generate and design fake objects in order to test the behavior of their code without having an interaction with the actual implementation process.

Developers may use Gomock to construct effective unit tests that are segregated from external dependencies such as databases or web services.

This framework is well-known for integrating seamlessly with Go's built-in testing package, which makes it simple to use and set up.

CHAPTER 3: SYSTEM DESIGN





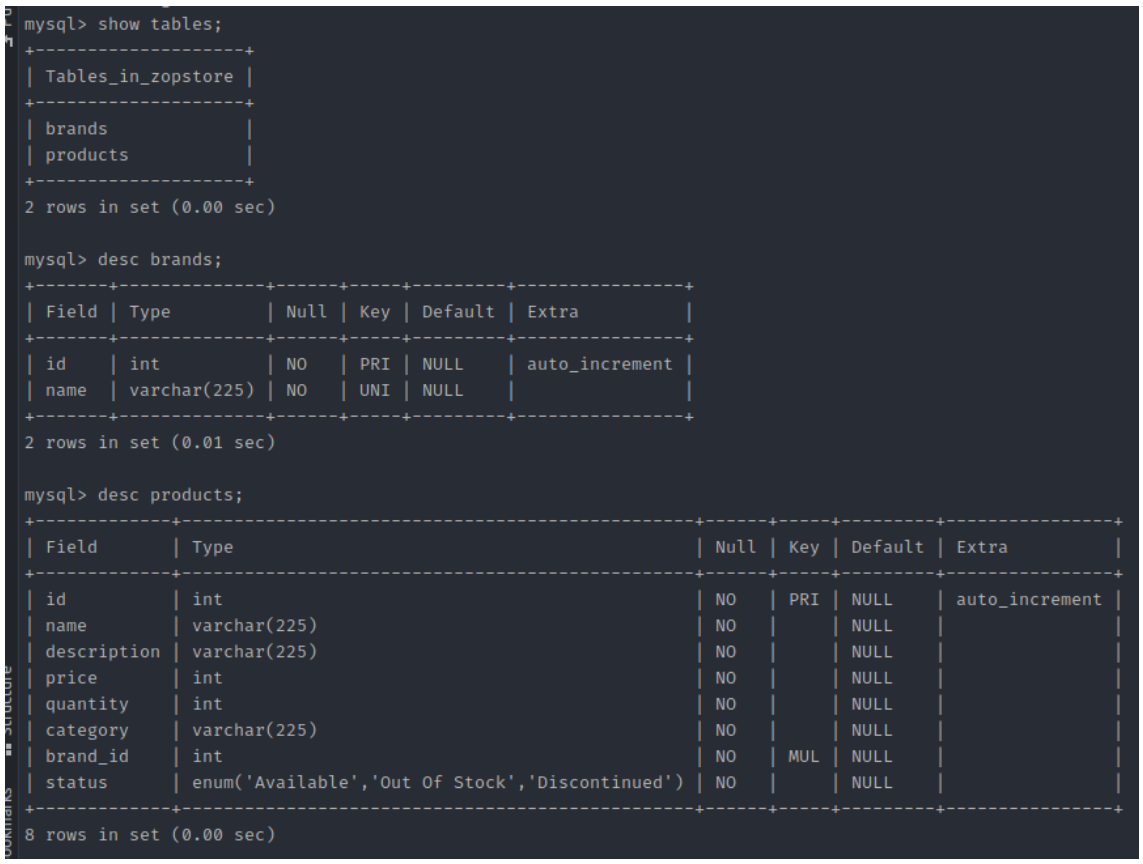
CHAPTER 4 : IMPLEMENTATION

4.1 Identification of features

The web application features:

- Creation of an entry with product and brand details
- Updation of the existing system
- Deletion of an existing entry
- Fetching details based on product/ brand ID
- Fetching details based on product/ brand name
- Fetching details based on product organization

4.2 SQL Schema



```
mysql> show tables;
+--------------------+
| Tables_in_zopstore |
+--------------------+
| brands             |
| products           |
+--------------------+
2 rows in set (0.00 sec)

mysql> desc brands;
+-------+--------------+------+-----+---------+----------------+
| Field | Type         | Null | Key | Default | Extra          |
+-------+--------------+------+-----+---------+----------------+
| id    | int          | NO   | PRI | NULL    | auto_increment |
| name  | varchar(225) | NO   | UNI | NULL    |                |
+-------+--------------+------+-----+---------+----------------+
2 rows in set (0.01 sec)

mysql> desc products;
+-------------+---------------------------------------------+------+-----+---------+----------------+
| Field       | Type                                        | Null | Key | Default | Extra          |
+-------------+---------------------------------------------+------+-----+---------+----------------+
| id          | int                                         | NO   | PRI | NULL    | auto_increment |
| name        | varchar(225)                                | NO   |     | NULL    |                |
| description | varchar(225)                                | NO   |     | NULL    |                |
| price       | int                                         | NO   |     | NULL    |                |
| quantity    | int                                         | NO   |     | NULL    |                |
| category    | varchar(225)                                | NO   |     | NULL    |                |
| brand_id    | int                                         | NO   | MUL | NULL    |                |
| status      | enum('Available','Out Of Stock','Discontinued') | NO   |     | NULL    |                |
+-------------+---------------------------------------------+------+-----+---------+----------------+
8 rows in set (0.00 sec)
```

4.3 Study Material

LINUX

Unix based OS with both command line interface and graphical user interface.

BASH(Bourne Again Shell) is a Unix shell and is the default shell in Linux.

Package manager is used for installing, upgrading and cleaning packages. Default package manager of Ubuntu is apt-get: *Advanced Packaging Tool* (APT), SUDO: **superuser do or substitute user do**

There are several processes running in the system environment, there are several variables set in the environment too called Environment variables and they affect the processes using them. If the value of these variables changed, processes using them will be affected.

We can set environment variables in ~/.bashrc or ~/.bash_profile. These are hidden files (hence start with a dot). Both are present in the home directory(~).

PATH: specifies the locations to be searched to find a command.

LINUX Commands:

1. ls: lists the files in a folder.
2. cd: change directory
3. touch: used to create an empty file.
4. pwd: print path to the folder we are currently working
5. mkdir: creates a folder or multiple folders To create nested folder -p mkdir folder/new
6. rmdir/rm: to delete an empty folder/ rm -rf folder(if another files/folder inside)
7. mv: move/ rename files cat: concatenate -> lists the contents of file
8. chmod: It sets the file permissions flags(define who can read, write to or execute the file) on a file or folder.
9. vi: visual editor

Go Workspace

It’s a hierarchy with two directories as it’s root:

1. src: containing goo source files,
2. bin: containing executable commands

GOPATH environment variable: It specifies the location of your workspace.
GOROOT is **a variable that defines where your Go SDK is located**.

GO Packages

Each and every go program is made of packages. All the program in go environment start running in the main package

- math/rand:- In package rand, environment is deterministic i.e. when run rand.In return same number, and if we want different results each time we use, rand.Seed
- With import use ()-for clarity[Factored statement] and " " with packages
- When exporting names use Capital letter with its package- ex: Pi(math.Pi)
- We can use the fmt: formatted i/o package to format all this.

Functions:

- In GO a function can take 0 or more agrs. The type of a function comes after the variable name.
  E.g. *func add(a int, b int) int{ return a+b}*

- To call a function-*func main(){ add(9,2)}*
  E.g. - *add(int a, int b): add(a, b int)*
- A function can return any number of results
  E.g.-*func swapString(str1, str 2string) (string, string)*
  { return str2,str1}
  func main(){ str1,str2:=swap(“A”,”B”) } [:= assignment operator]

Import:

- In go we import all the packages and libraries in alphabetic order.

- First all the inbuilt packages are written followed by third party packages.

File Watchers:

- It is a tool that is used to Imports all packages, formatting:
- Correct all the indentation, make code standard and clean, etc.

Variables

We use var to declare a variable or a list of variables. The variables can be at package or function level.

Declaration: var i int; default int=0, bool=false

- We can include an initializer one per var.
  Ex: *var i, j int=4,2*
  var k, j= false, "no!"

- Inside a function, we use := for a short assignment statement.

  we can't use :=; we have to use var, func,etc.

  Ex: *func main(){*
  var m, n int=3,2
  k,n:= true,"no!"
  }
- White creating variables we should always avoid global variables.

- The default value of variable integer is 0, for a bool: false and "" for string

Type Conversion

- Type Converts say integer to float or vice-versa i.e converse one var type to other.
  eg: *var j int=32, f:= float64(j)*

Type Interface

- If we have a declaration on rhs, the new var takes the same type.

Ex- *var i int*
j:= i
But when not specified, say v:=42; then the type depends on precision.

Constants

- We define constants using const.
- Constants can be char, str, bool and numeric val.
- We can’t declare a constant using :=
- Numeric const have high precision values

FOR

In go we have only for; no while or do while loop

1. for loop:
   for i:= 0; i<20; i++
2. for loop behaving like a while loop:
   for ; j<20; —> j>=1
   for i<10→i>=1
3. for loop behaving like do while loop
   for v=0;v<10; here v is local to for

IF

- In if statements of go, we do not need to use () but {} is required.
- If can start with a short statement before execution but it should end with a semicolon.
- Variables that are declared inside an if short statement are also available inside any of the else blocks.

SWITCH

- Switch of go works a little differently. Go runs only selected cases, not all cases that follows
- In go, break statements are not required after every case.
- Switch cases need not be constant.
- In switch values can be anything, not specifically an integer.
- In go, switch evaluates from top to bottom
- Switch without a condition is same as true- if,if-else
- We cannot have two cases with same condition ie. no duplicate case or it gives us type mismatch error or compile time error.

DEFER

- The Go programming language has a special feature called "defer" that delays the execution of a function until the enclosing function has finished and returned.
- The arguments supplied to a deferred function are evaluated immediately after the defer statement is executed, but the function call itself is not executed until the enclosing function has finished running and is about to return.
- This makes defer statements handy for tasks like shutting files, releasing resources, and other tasks that need to be completed after a function is done running.
- The most recently deferred function call will be executed first because Go employs a stack to keep track of deferred function calls. "Last in, first out" (LIFO) sequence is used in this situation.
- The "defer" keyword is used followed by the function that has to be deferred in a Go program to define a defer statement. This makes it simple to guarantee that specific actions are always carried out, despite a mistake or a panic.

POINTERS

- In Go, pointers are variables that store a value's memory address. They enable oblique access to memory-stored values.
- Use the * symbol and the type of the variable it will point to to declare a pointer. As an illustration, the statement var p *int declares a pointer with the name p that points to an integer value.
- Use the & operator after the value to obtain the value's memory address. For instance, x:= 10; p:= &x assigns the pointer p the memory location of x.
- Use the * operator and the pointer name to dereference a pointer. As an illustration, *p receives the value that p points to. Using *p = 20, a new value can also be assigned to the memory location pointed to by a pointer.

- A pointer's zero value is nil. A pointer that is nil indicates that it points to no memory location.
- Go creates a replica of the value you supply as an argument to a function. However, you can indirectly change the initial value when you give a pointer to a function.
- Pointers can be used to build intricate data structures such as linked lists, trees, and graphs.

STRUCTS

- Structs are collection of fields

```
ex: type V struct{
        X int
        Y int }

            Func main(){ print(V{1,2})

            } - basically prints 1,2
```

- Struct fields are accessed using v= V{11,12}.

To access an struct field we use a struct pointer.

```
var (

    v1 = Vtx{11,1 2} // has type Vertex

    v2 = Vtx{X: 11} // Y:0 is implicit

    p = &Vtx{11, 12} // has type *Vertex

)
```

ARRAYS

- Arrays are where we can store a collection of elements. Array length is part of its type i.e. cannot be resized.
- To declare an array:

```
Arr[] int
Arr=[] int {1,2}
```

SLICES

- In go we use slices as dynamically sized array declaration S [low:high]; incl. low

- Slices are like references to arrays. It doesn't store any data, we just describe a section of an array.
- if we change the elements of a slice, it also changes the corresponding elements of its underlying array and is reflected in other slices that share common elements. Eg: a:= names[1:2]; names-array.
- A slice literal is an array literal without the length.
- A slice structure, internally contains a pointer, length and a capacity field.

len(s): length of slice or we can say number of elements in slice.
cap(s): capacity of slice or we can say number of elements in underlying array

- Zero value of slice is nil. and thus length and capacity is nil i.e len=cap=0

- Slice can be made with build-in func make that is how we create dynamic sized arrays.

- Make creates a zeroed array and returns a slice that refers to the array.

  Ex: *a:=make([]int , len)*
  a:=make([]int, len, cap)
- Slice can contain any type including other slices. We can also append elements in a slice using append(slice, element1,element2…).

- If the capacity of the slice is less than the no. of elements to be appended, it automatically doubles the capacity. : cap(s)+1)*2: and creates a new slice, new memory is allocated and changes are not reflected on the underlying array.

RANGE

- A Range is a form of for loop that iterates over a slice/map.

- For each iteration it returns an index and copy of value at that index.

- We can also skip the index or value by assigning _.
- Syntax:
  for j, _ := range power
  for _, value := range power

  If we only want index we can omit the second variable.

MAP

- A map maps keys to values.
- Zero map has value nil. A nil map has no keys.
- If the top-level type is just a type name, you can omit it from the elements of the literal.
- Mutating maps
  - To insert/update: *m[key]=elem*
  - To retrieve: *elem=m[key]*
  - To delete: *delete(m,key): to delete a key*
- To check if key is there:
  - *elem, ok=m[key] ; ok=true* if the key is there otherwise false
- The zero value for the map's element type if we can't find the key in the map
- A key in any given map cannot be sliced.

PANIC: run-time error

Variadic Functions

- fmt.Println: It is an empty interface

- Variadic PARAMETER: …: can pass 0 or more values and can be of any type In an argument list, variadic is last variable

Function Values

- Functions can also be passed around as values.

- In this sense, the function is "bound" to the variables because it can access and assign to them.

METHODS

- A method is like a function with a special receiver argument.
- A method is similar to a function with an additional receiver parameter.
- On types, we may define methods.
- Methods for non-struct types can also be defined.
- Only receivers whose types are defined in the equal pkg as the method, including built-in types like int, can be declared with a method.
- A pointer receiver is useful for two reasons:
  - Allow for changes to the value that is pointed by the method's receiver.
  - Avoid replicating the value on each method call.
- Any method on a specific type should not have a mix of value and pointer receivers.

Receiver Arguments:

- Value receiver argument can only reference methods with value receiver whereas pointer receiver argument references methods with both value and pointer receiver: METHOD SETS
- We use value receivers when we don’t want changes to be reflected in the original value, while using slices, maps, etc
- We can use a pointer receiver when we want the changes to be reflected or when we want to access methods either way or when the struct is quite large to avoid duplicate copies.

INTERFACES

- Interface type is defined as method signature of a particular underlying base.

- A value of interface type can store any value. However, that value has to implement those methods.

  Syntax: type name interface{}

- Interfaces are implemented implicitly.. The zero value(interface) is equal to **nil**.

- The abstract type underlying an interface can be understood as (value, type)= a tuple consisting of a val & a concrete type.

- In the case of a pointer receiver: (&{Hello}, *main.T).

- The function will be called with a nil receiver. It throws a no null pointer exception if the concrete value of the interface itself is equal to nil. It is still possible for an interface value to include a nil concrete value.

- Neither a value nor a concrete type are stored in a nil interface value.

- There is no type inside the interface tuple to specify which concrete method to invoke. Therefore, calling a method on a nil interface results in a run-time error.

- The empty interface: interface is a type of interface that defines no methods.

- Any sort of value can be stored in an empty interface. Each type implements a minimum of 0 methods. Code that handles values of unknown types uses empty interfaces.

TYPE ASSERTION

- Access to the underlying concrete value of an interface value is provided through type assertions.

- T:=i.(T): This assertion says that the concrete type T is held by the interface value i and gives the variable t its underlying T value. This statement makes me panic if I don't have a T.

- If i has T, then t is the underlying value and ok is true. Otherwise none of these values are true. Otherwise, t is a null value of type T, ok is false, and no panic occurs.

Type SWITCH

- A *type switch* allows several type assertions in series.

- The only difference between a typeswitch and a standard switch statement is that types (rather than values) are supplied in the typeswitch case, and those types are then compared to the kinds of values contained by the specified interface value.

STRINGERS

- It is an ubiquitous interface defined by the format(fmt) package.

  type Stringer interface {
  String()bstring }

- A type that can describe itself as a substring.

Readers

- The io package defines the io.Reader interface, corresponding to the data stream's read end.

- Theano.Readerbinterfacebhas bRead method:
  func (T) Read(b []byte) (n int, err error)

- Read fills the specified byte slice with information and returns the no. of bytes filled as well as an error value. When the stream is finished, it produces an io.EOF error.

REST- REpresentational State Transfer

- A REST API allows users to interact with web services that follow RESTful architecture.
- An api that follows the REST architecture is also known as a RESTful API
- It adheres to the REST architectural style.
- REST = representational state transfer,
- It was created by Roy Fielding, a computer scientist.

Principles of RESTful Design

- **Decoupling of client and server** –
  - Client and server applications in a REST API design need to be totally independent of one another.
  - The only thing the client program should be aware of is the requested resource's URI.
  - No connection of any kind must be made to the server application.
  - Except to supply it with the required data via HTTP, a client application shouldn't be altered by a server application.
  - Each request must include all the information required to process it because REST APIs are stateless.
  - Server-side communication is not needed for REST APIs. Applications running on the server are not permitted to retain any information relating to client requests.

- **Cacheability** –
  - Whenever practical, resources should be cacheable on both the client and server sides. Server responses must also indicate if the requested resource can be cached.
  - The goal is to boost server-side scalability while improving client-side performance.

REST API calls and responses pass through multiple layers in a layered system architecture. In most circumstances, client and server programmes will communicate indirectly.

Response Status Codes

1. 200:OK, Success
2. 201: Success+Created
3. 202: Accepted, request received but not completed
4. 204: No content
5. 400: Bad Request, incorrect syntax
6. 404: Not found
7. 405: Method Not Allowed
8. 500: Internal Server Error

HTTP package

The http package provides a client and a server. The server is made of handlers. The handler takes a request and based on that it returns a response.

1. **Create** : Post-> new data
2. **Read** : Get-> retrieve data
3. **Update**: Put-> update data
4. **Delete**: Delete-> delete data

ServeMux(Multiplexer)

- ServeMux is an HTTP request multiplexer that executes requests by matching their URLs to the correct handlers.
- Use http.NewServeMux to create a new ServeMux and the Handle and HandleFunc methods to add a URL handler.
- A string and a http.Handler are accepted by the handle method. An interface with the ServeHTTP method is provided by the Handle method's second parameter.
- The handler implementation is sent to HandleFunc as a function along with the path for which it should be called.
- Server is an HTTP server that allows you to manage various routes and paths by passing an instance of a ServeMux.

- You can completely omit the ServeMux if you have a route or path that you want to manage by passing an instance of a http.Handler instead.
- The net/http package contains the Handle and HandleFunc methods as well as the DefaultServeMux.
- If you have used http.Handle and/or http.HandleFunc to define the handler implementations for the corresponding routes, the handler parameter for the http.ListenAndServe method, which starts the HTTP server, can be nil.

Functions as handlers

type HandlerFunc func(ResponseWriter, *Request)
HandlerFunc allows you to use ordinary functions as HTTP handlers. For example:

```
func welcome(rw http.ResponseWriter, req *http.Request) {
        rw.Write([]byte("Welcome to Just Enough Go"))
}
```

Or http.ListenAndServe(":8080", http.HandlerFunc(welcome))

Note: HandlerFunc(f) is a Handler that calls the function f

HTTPtest Package

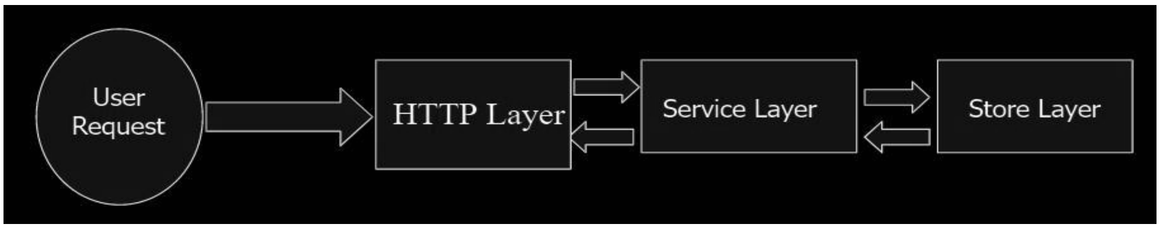
1. httpRequest: httptest.NewRequest- returns a new incoming server request, suitable for passing to an http.Handler for testing.

2. httpResponseWriter: w=httptest.NewRecorder a type that produces httptest.ResponseRecorder - ResponseRecorder implements http.ResponseWriter and records changes for subsequent analysis in tests. before being moved to our server

3. Handler stores the information it writes to the response in a database before returning the written information.

LAYERED ARCHITECTURE

Layers are independent of each other and communicate with each other through interface.



Our application's layers are intended to be independent of one another, and they communicate with one another via clearly defined interfaces. Our codebase may be made modular, readable, and maintainable using this strategy. The HTTP layer, the Service layer, and the Store layer are the three separate layers of the application.

The HTTP layer is in charge of checking headers, managing request body data, and validating query and path parameters. The business logic is put into practice by the Service layer, which also interacts with the Store layer to carry out any required data storage actions. Database-level queries must be implemented by the Store layer.

Through clearly defined interfaces that outline input parameters and output types, each layer communicates with the one underneath it and the one above it. This makes it simple to test each layer by simulating the server, database, or interface as necessary.

Basically this helps us make our application modular, readable and maintainable.

This has 3 layers - HTTP layer, Service layer, Store layer.

1. HTTP layer : Validates query/path parameters, request body, header checks.

2. Service layer : Implements business logic and communicates with datastore layer
3. Store layer : Implements database level queries.

Each layer communicates with its previous/next layer using an interface(methods with input parameters and output types are defined).

Testing of each layer is done by mocking its interface/DB/Server based on necessity.

Dependency Injection:

It is a style of writing code such that at the time the object is initialized the dependencies of a particular object/struct are provided.

We can explicitly choose when to create new instances of our dependencies and when to reuse the same instance. Our structs no longer have the responsibility for building their dependencies thus making our structs less tightly coupled to their dependencies.

Whenever an object or struct is initialized via dependency injection, its dependencies are supplied. This implies that we consciously decide when to employ previously created instances of our dependencies and when to build new ones. By doing this, we can simply change the dependency's implementation without having to change the code that depends on it.

This method also aids in simplifying and decoupling our code. Traditionally, dependencies are created and managed by objects in object-oriented programming. As a result, objects become tightly coupled, which makes it more difficult to test and manage them. We transfer this responsibility from the objects to a different component by using dependency injection. In this manner, objects are loosely connected to their dependents, which facilitates testing and maintenance.

Factory method

It is a design pattern that addresses the issue of constructing product objects without defining their specific classes. It defines a way for creating objects rather than invoking new operators directly.

1. **Simple factory**
2. **Interface factories.**

MICRO SERVICES

Microservices is a software development architectural and organizational strategy that consists of tiny, independent services that connect over well-defined APIs.

Small, autonomous services that interact through well defined APIs make up the architectural and organizational framework of microservices used in software development.

These services are owned and run by small, independent teams.

Applications may be expanded and developed more easily thanks to microservices architectures, which can speed up the time it takes to sell new features.

Benefits of micro services

- **Flexible Scaling**

  The demand for each microservice's underlying app feature may be expanded independently of the others. This helps teams to maintain service uptime during moments of high demand, precisely scale infrastructure, and accurately estimate the cost of a feature.

- **Easy Deployment**

  Microservices provide continuous integration and delivery. They make it simple to test new concepts and roll them back if they fail.

  The cost of failure is feasible for more experimentation, quicker code revisions, and speedier time-to-market for new features.

- **Reusable Code**

Software can be divided into distinct, well defined modules, which teams can then utilize for a variety of reasons. A service developed for one purpose may serve as the basis for another feature. Because developers may add new features without starting from scratch, an application can now self-bootstrap.

Database Migration

Developers are responsible for building, maintaining, and improving applications- this could need you to change or update the database structures.

Migration gives you the ability to handle these changes easily and consistently in an active development environment. The more you know about shaping your database, the better equipped you'll be to create an effective and concise database for your application.

Some popular frameworks such as Django, Rails and even some standalone libraries such as Flyway and Liquibase provide this feature too.

Migrations act as a version control system for your database, allowing your team to define and share the database schema definition for the application.

There are two methods in a migration class: up and down. The up method of your migration should specify the schema modification you want to do, and the down method should reverse the alterations made by the up method. In other words, if you conduct an up followed by a down, the database schema should remain identical. If you make a table in the up method, for example, you should dump it in the down method.

When migrating up the database – forward in time – the up method is used, whereas when migrating down the database – back in time – the down approach is used. Thus, we can go back and forth to older and newer versions of our database.

PROMETHEUS

Prometheus is an open-source monitoring and alerting tool created to monitor a highly dynamic container environment in real-time.

Additionally, it may be used for a conventional (non-container) bare server where programmes are delivered directly.

ARCHITECTURE

Prometheus Server: It does the actual monitoring work.

- Time Series Database: stores metrics data (*storage*)
- Data Retrieval Worker: Pulls data from metrics, applications, servers, etc. (*retrieval*)
- Accepts PromQL queries: consumed by external systems via the HTTP api.

Prometheus Monitors: It monitors single application, apache server, linux/windows server etc called *targets*.

Target: The target units such CPU status, memory/disk storage space, exceptions counts, request count/duration, etc are monitored.

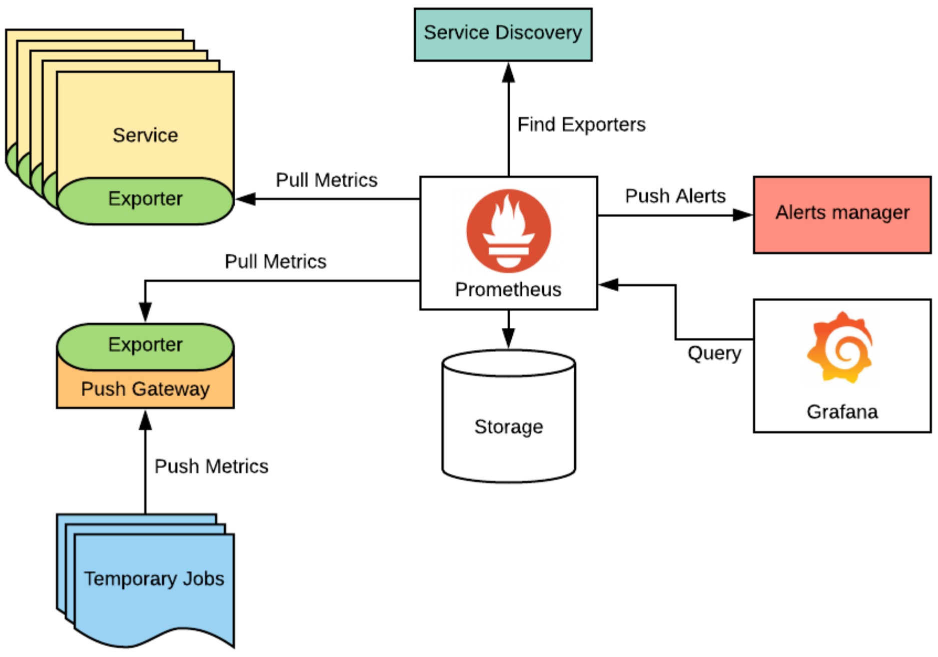
Matrics: Unit that is monitored for a specific target. It is defined as human-readable, text based and have two entities:

- Help: Description of what metrics is.
- Type: 4-types-
    1. Counter
    2. Summary
    3. Gauge
    4. Histogram

Exporter: Some servers already expose prometheus endpoints therefore don't need extra service to gather metrics but many services need another component called *exporter*.

It’s a script or service that fetches metrics from target and converts it to correct format that prometheus understands and exposes it to its own/ matric endpoint where Prometheus can scrape them.

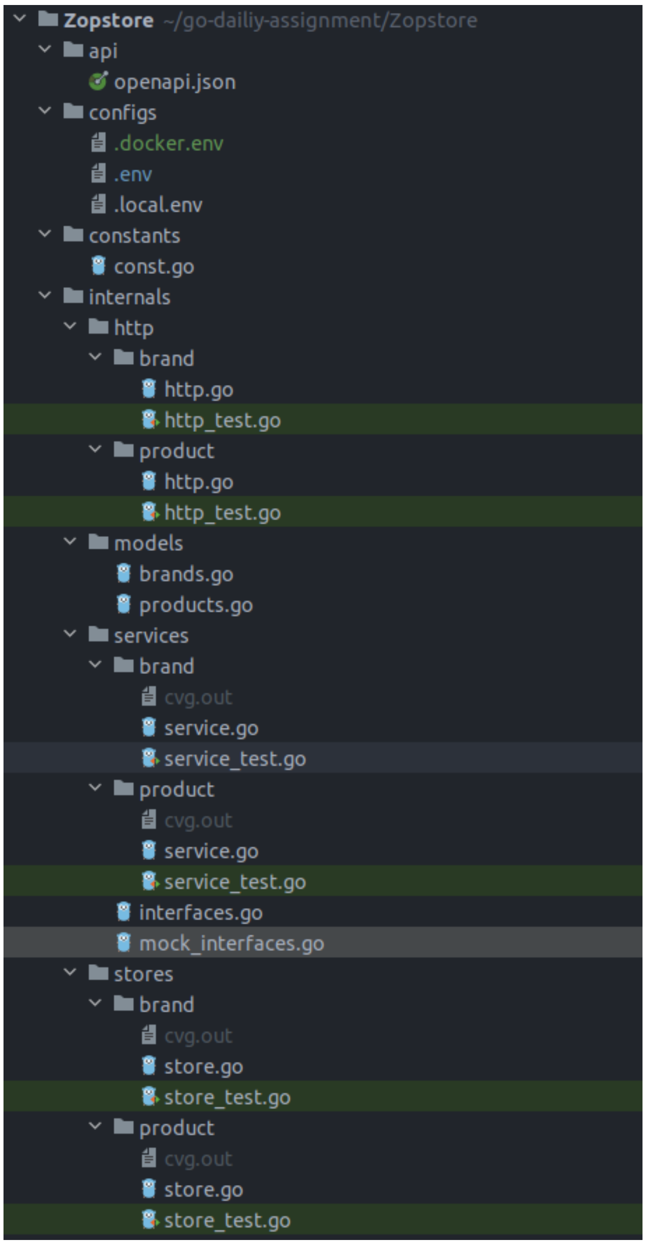
Workflow:



PROJECT LAYOUT

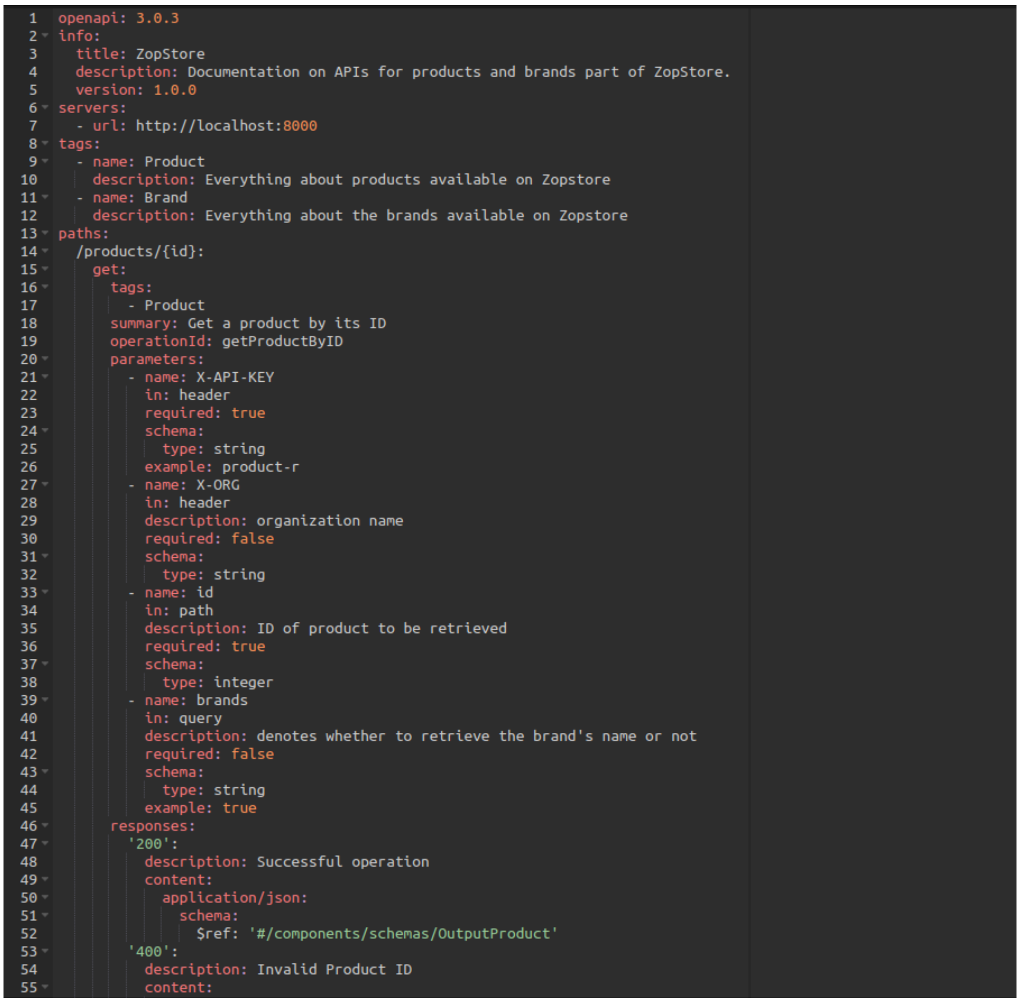
It’s a hierarchy of multiple directories:

1. **api**: containing the swagger documentation,
2. **configs**: containing DB credentials for local testing
3. **constants**: containing constant variables used in the project
4. **internals**: containing the 3-layered architecture for brands and products entities
5. **middlewares**: containing the middleware source files

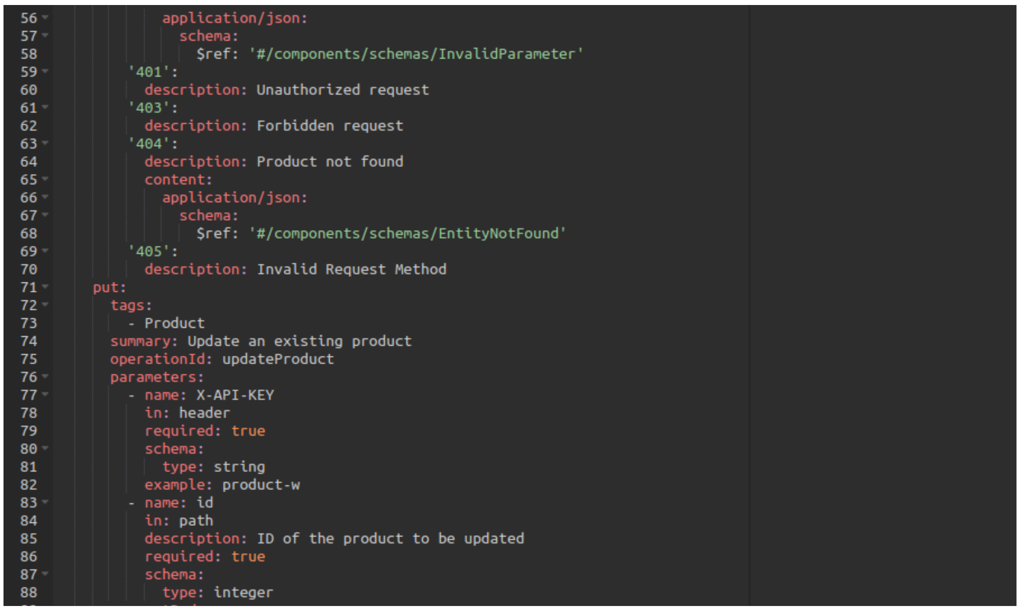


```
Zopstore ~/go-dailiy-assignment/Zopstore
  api
    openapi.json
  configs
    .docker.env
    .env
    .local.env
  constants
    const.go
  internals
    http
      brand
        http.go
        http_test.go
      product
        http.go
        http_test.go
    models
      brands.go
      products.go
    services
      brand
        cvg.out
        service.go
        service_test.go
      product
        cvg.out
        service.go
        service_test.go
      interfaces.go
      mock_interfaces.go
    stores
      brand
        cvg.out
        store.go
        store_test.go
      product
        cvg.out
        store.go
        store_test.go
```

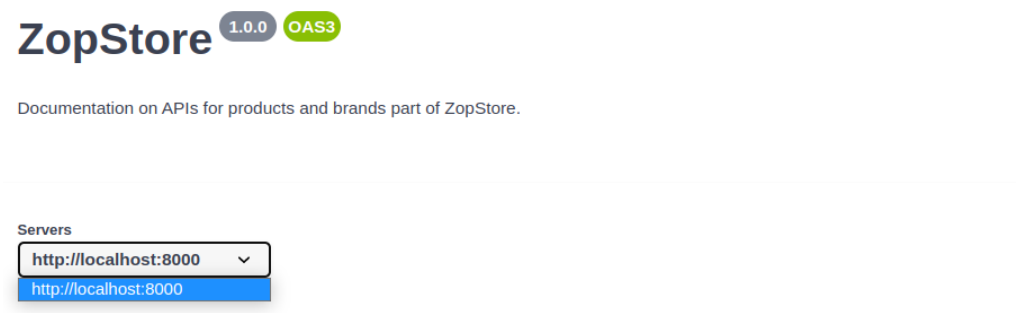
SWAGGER DOCUMENTATION



```yaml
openapi: 3.0.3
info:
  title: ZopStore
  description: Documentation on APIs for products and brands part of ZopStore.
  version: 1.0.0
servers:
  - url: http://localhost:8000
tags:
  - name: Product
    description: Everything about products available on Zopstore
  - name: Brand
    description: Everything about the brands available on Zopstore
paths:
  /products/{id}:
    get:
      tags:
        - Product
      summary: Get a product by its ID
      operationId: getProductByID
      parameters:
        - name: X-API-KEY
          in: header
          required: true
          schema:
            type: string
          example: product-r
        - name: X-ORG
          in: header
          description: organization name
          required: false
          schema:
            type: string
        - name: id
          in: path
          description: ID of product to be retrieved
          required: true
          schema:
            type: integer
        - name: brands
          in: query
          description: denotes whether to retrieve the brand's name or not
          required: false
          schema:
            type: string
          example: true
      responses:
        '200':
          description: Successful operation
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/OutputProduct'
        '400':
          description: Invalid Product ID
          content:
```



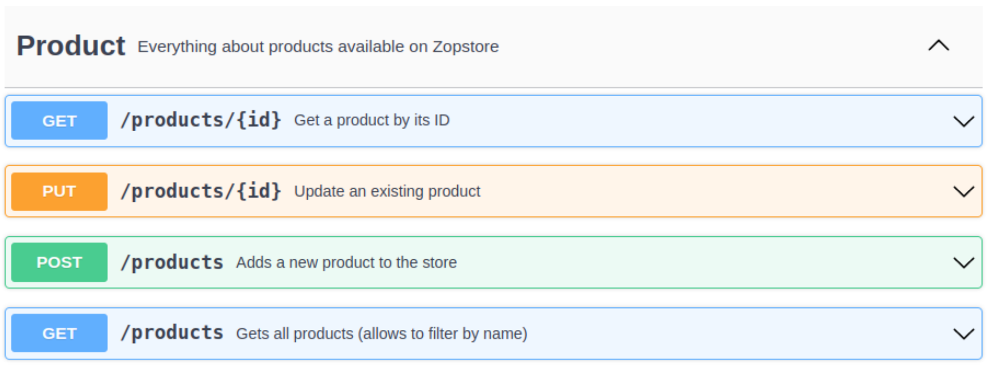
```yaml
            application/json:
              schema:
                $ref: '#/components/schemas/InvalidParameter'
        '401':
          description: Unauthorized request
        '403':
          description: Forbidden request
        '404':
          description: Product not found
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/EntityNotFound'
        '405':
          description: Invalid Request Method
    put:
      tags:
        - Product
      summary: Update an existing product
      operationId: updateProduct
      parameters:
        - name: X-API-KEY
          in: header
          required: true
          schema:
            type: string
          example: product-w
        - name: id
          in: path
          description: ID of the product to be updated
          required: true
          schema:
            type: integer
```



ZopStore 1.0.0 OAS3

Documentation on APIs for products and brands part of ZopStore.

Servers
http://localhost:8000
http://localhost:8000



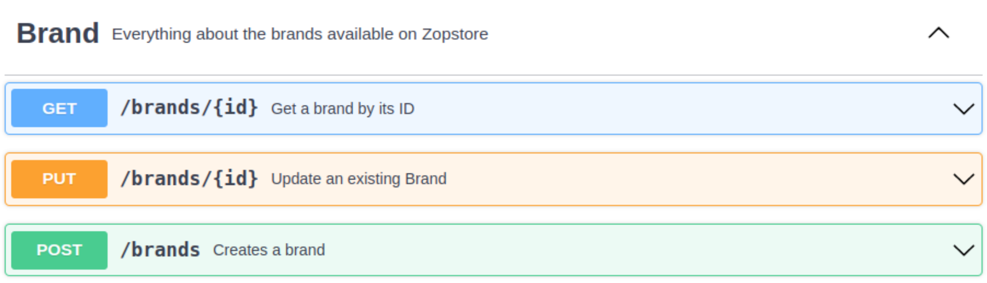
Product Everything about products available on Zopstore

GET `/products/{id}` Get a product by its ID

PUT `/products/{id}` Update an existing product

POST `/products` Adds a new product to the store

GET `/products` Gets all products (allows to filter by name)



Brand Everything about the brands available on Zopstore

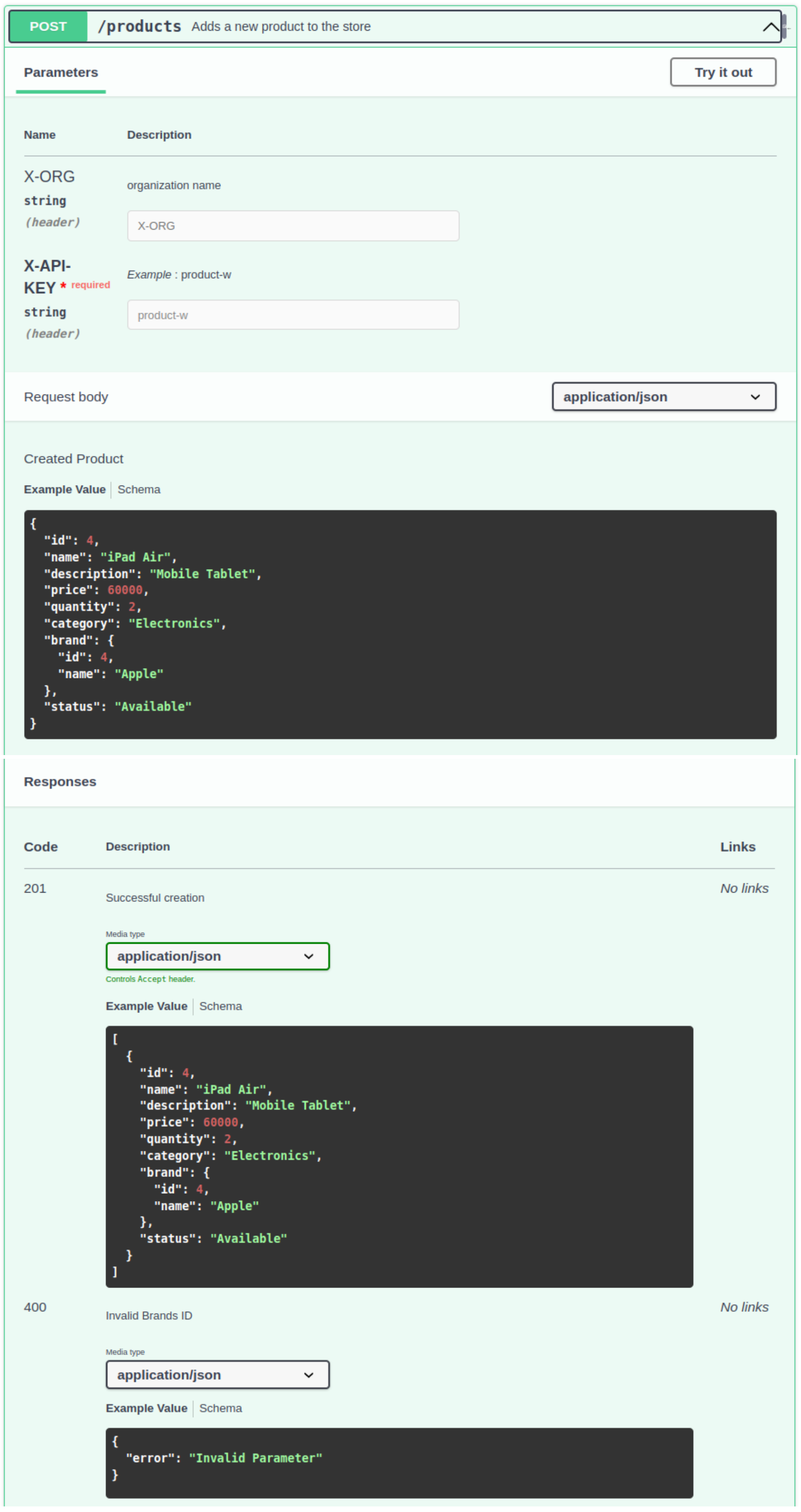
GET `/brands/{id}` Get a brand by its ID

PUT `/brands/{id}` Update an existing Brand

POST `/brands` Creates a brand

PRODUCTs

Create: Here we create a new entry by passing the requested json body followed by unmarshalling it and then passing it to the service layer to check if all the data passed is valid and according to the parameters defined.



POST **/products** Adds a new product to the store

Parameters Try it out

Name	Description
X-ORG string (header)	organization name X-ORG
X-API-KEY * required string (header)	*Example* : product-w product-w

Request body application/json

Created Product

Example Value | Schema

```
{
  "id": 4,
  "name": "iPad Air",
  "description": "Mobile Tablet",
  "price": 60000,
  "quantity": 2,
  "category": "Electronics",
  "brand": {
    "id": 4,
    "name": "Apple"
  },
  "status": "Available"
}
```

Responses

Code	Description	Links
201	Successful creation	*No links*

Media type application/json
Controls Accept header.

Example Value | Schema

```
[
  {
    "id": 4,
    "name": "iPad Air",
    "description": "Mobile Tablet",
    "price": 60000,
    "quantity": 2,
    "category": "Electronics",
    "brand": {
      "id": 4,
      "name": "Apple"
    },
    "status": "Available"
  }
]
```

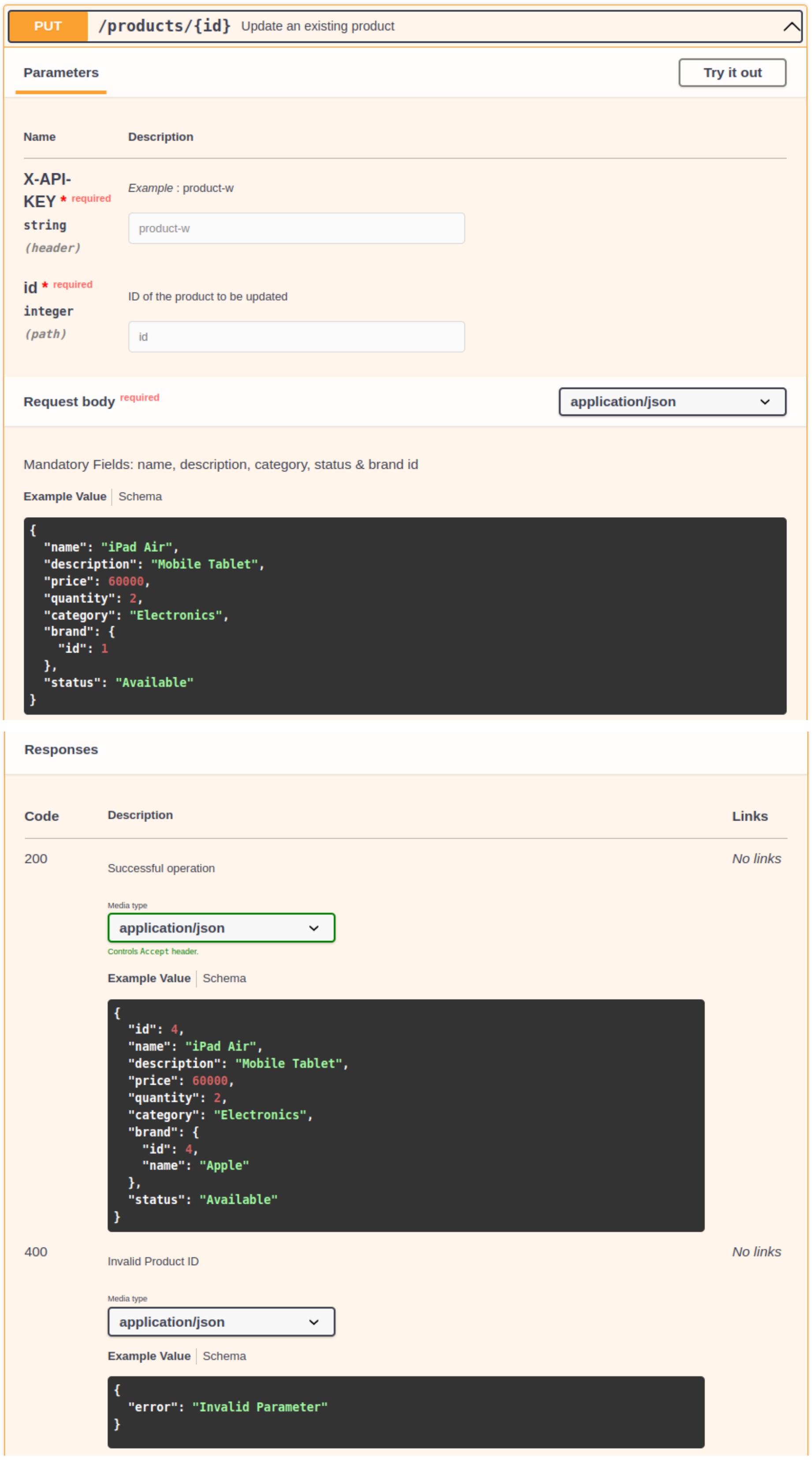
Code	Description	Links
400	Invalid Brands ID	*No links*

Media type application/json

Example Value | Schema

```
{
  "error": "Invalid Parameter"
}
```

Update: Here we update an existing entry by passing the requested json body followed by unmarshalling it and then passing it to the service layer to check if all the data passed is valid and according to the parameters defined.



PUT `/products/{id}` Update an existing product

Parameters

Try it out

Name	Description
X-API-KEY * required string *(header)*	*Example* : product-w product-w
id * required integer *(path)*	ID of the product to be updated id

Request body required — application/json

Mandatory Fields: name, description, category, status & brand id

Example Value | Schema

```
{
  "name": "iPad Air",
  "description": "Mobile Tablet",
  "price": 60000,
  "quantity": 2,
  "category": "Electronics",
  "brand": {
    "id": 1
  },
  "status": "Available"
}
```

Responses

Code	Description	Links
200	Successful operation	*No links*
400	Invalid Product ID	*No links*

200 — Media type: application/json (Controls `Accept` header.)

Example Value | Schema

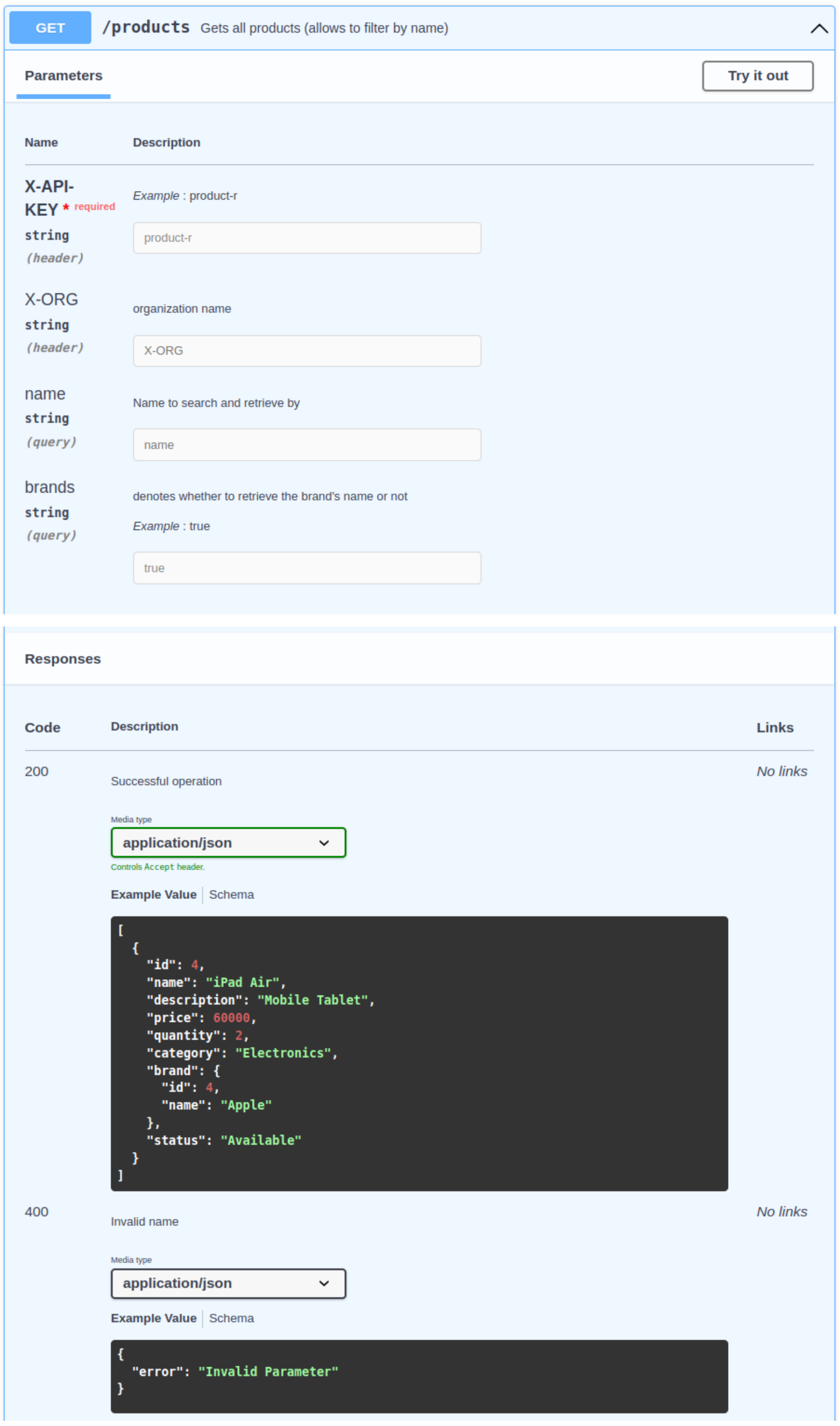
```
{
  "id": 4,
  "name": "iPad Air",
  "description": "Mobile Tablet",
  "price": 60000,
  "quantity": 2,
  "category": "Electronics",
  "brand": {
    "id": 4,
    "name": "Apple"
  },
  "status": "Available"
}
```

400 — Media type: application/json

Example Value | Schema

```
{
  "error": "Invalid Parameter"
}
```

Get All: Here we fetch details of all the products, inside which we can configure to fetch all products by brand name or simply show details of all the products. The parameters are then passed to the service layer to check if all the data passed is valid and according to the parameters defined.


GET /products Gets all products (allows to filter by name)

Parameters | Try it out

Name	Description
X-API-KEY * required string (header)	*Example* : product-r product-r
X-ORG string (header)	organization name X-ORG
name string (query)	Name to search and retrieve by name
brands string (query)	denotes whether to retrieve the brand's name or not *Example* : true true

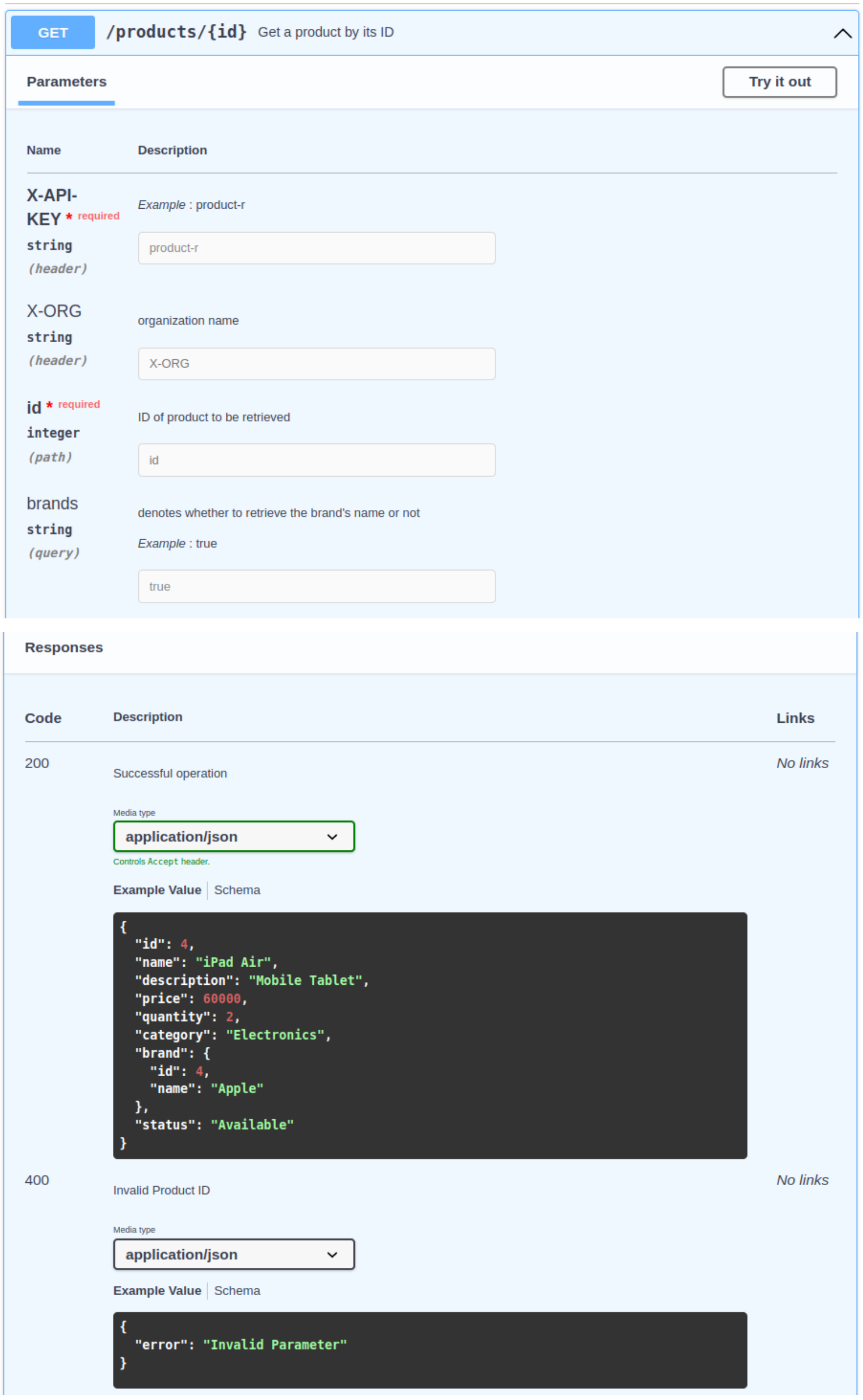
Responses

Code	Description	Links
200	Successful operation Media type: application/json Controls Accept header. Example Value | Schema	*No links*
400	Invalid name Media type: application/json Example Value | Schema	*No links*

```
[
  {
    "id": 4,
    "name": "iPad Air",
    "description": "Mobile Tablet",
    "price": 60000,
    "quantity": 2,
    "category": "Electronics",
    "brand": {
      "id": 4,
      "name": "Apple"
    },
    "status": "Available"
  }
]
```

```
{
  "error": "Invalid Parameter"
}
```

Get By Id: Here we fetch details of a product by product Id and then pass it to the service layer to check if all the data passed is valid and according to the parameters defined.



GET /products/{id} Get a product by its ID

Parameters

Try it out

Name	Description
X-API-KEY * required string *(header)*	*Example* : product-r product-r
X-ORG string *(header)*	organization name X-ORG
id * required integer *(path)*	ID of product to be retrieved id
brands string *(query)*	denotes whether to retrieve the brand's name or not *Example* : true true

Responses

Code	Description	Links
200	Successful operation Media type: application/json Controls Accept header. Example Value | Schema	*No links*
400	Invalid Product ID Media type: application/json Example Value | Schema	*No links*

200 example value:

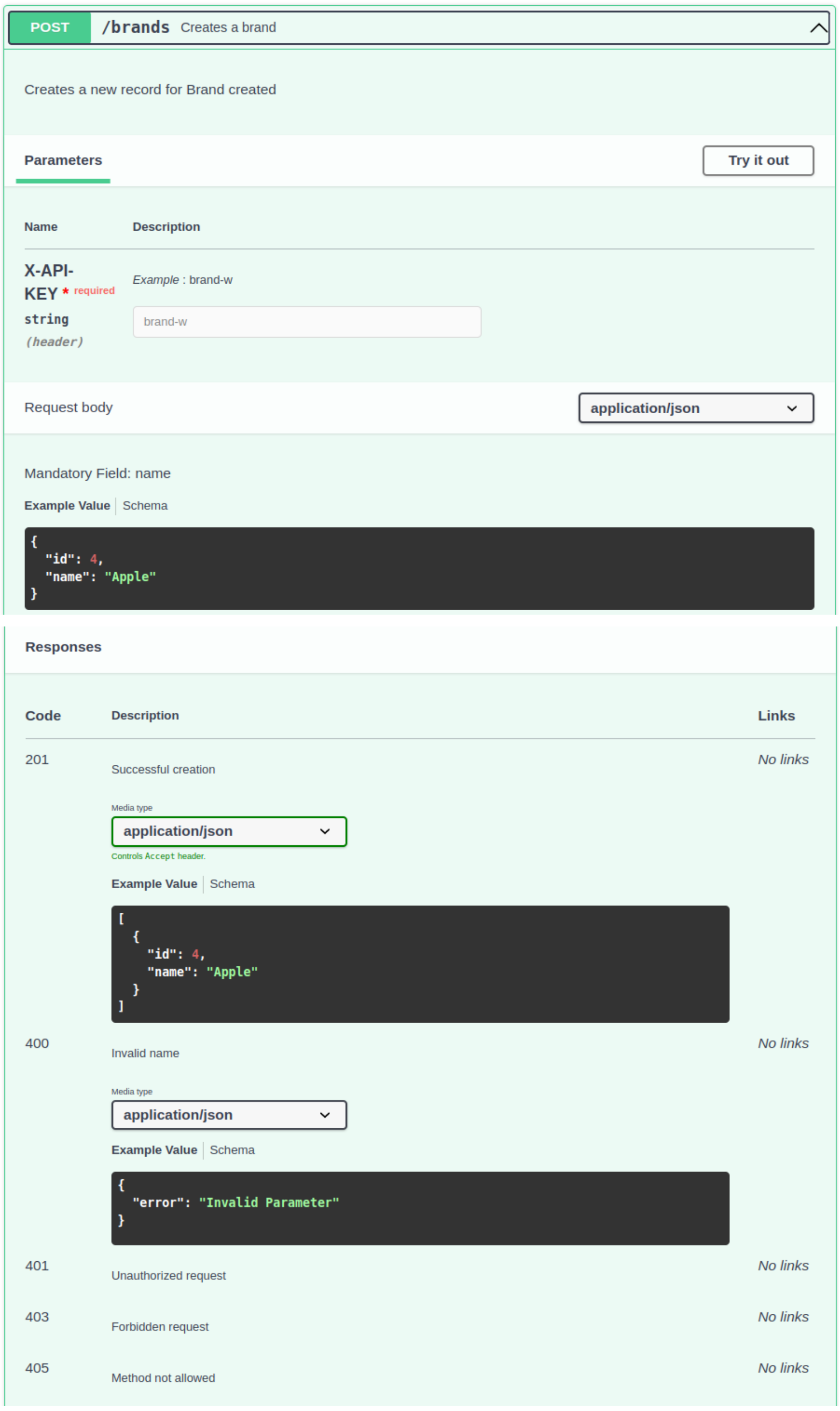
```
{
  "id": 4,
  "name": "iPad Air",
  "description": "Mobile Tablet",
  "price": 60000,
  "quantity": 2,
  "category": "Electronics",
  "brand": {
    "id": 4,
    "name": "Apple"
  },
  "status": "Available"
}
```

400 example value:

```
{
  "error": "Invalid Parameter"
}
```

BRANDs

Create: Here we create a new entry by passing the requested json body followed by unmarshalling it and then passing it to the service layer to check if all the data passed is valid and according to the parameters defined.



POST /brands Creates a brand

Creates a new record for Brand created

Parameters

Try it out

Name	Description
X-API-KEY * required string *(header)*	*Example* : brand-w brand-w

Request body: application/json

Mandatory Field: name

Example Value | Schema

```
{
  "id": 4,
  "name": "Apple"
}
```

Responses

Code	Description	Links
201	Successful creation Media type: application/json Controls Accept header. Example Value | Schema	*No links*

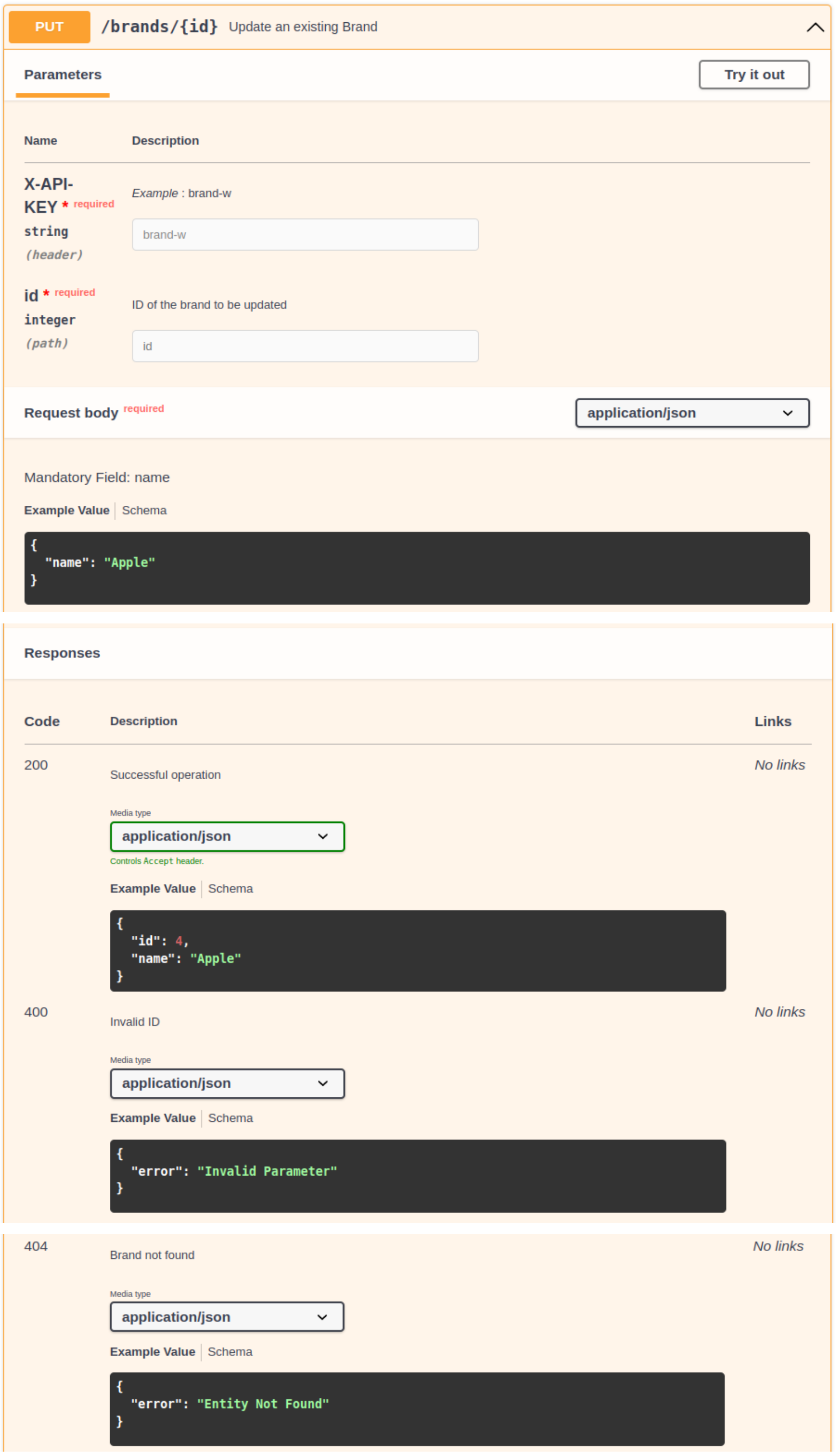
```
[
  {
    "id": 4,
    "name": "Apple"
  }
]
```

Code	Description	Links
400	Invalid name Media type: application/json Example Value | Schema	*No links*

```
{
  "error": "Invalid Parameter"
}
```

Code	Description	Links
401	Unauthorized request	*No links*
403	Forbidden request	*No links*
405	Method not allowed	*No links*

Update: Here we update an existing entry by passing the requested json body followed by unmarshalling it and then passing it to the service layer to check if all the data passed is valid and according to the parameters defined.



PUT /brands/{id} Update an existing Brand

Parameters

Try it out

Name	Description
X-API-KEY * required string *(header)*	*Example* : brand-w brand-w
id * required integer *(path)*	ID of the brand to be updated id

Request body required

application/json

Mandatory Field: name

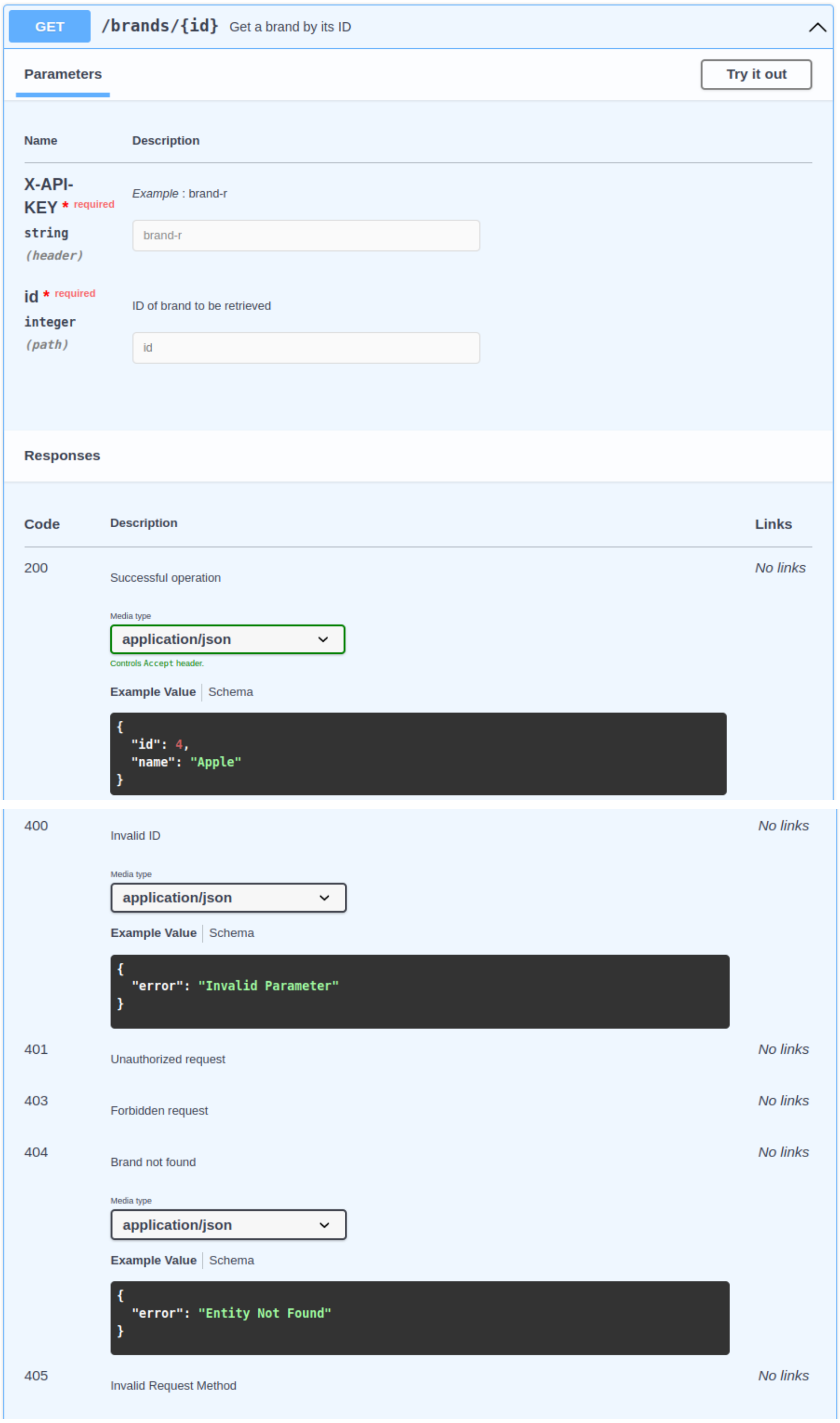
Example Value | Schema

```
{
  "name": "Apple"
}
```

Responses

Code	Description	Links
200	Successful operation Media type: application/json Controls Accept header. Example Value | Schema `{ "id": 4, "name": "Apple" }`	*No links*
400	Invalid ID Media type: application/json Example Value | Schema `{ "error": "Invalid Parameter" }`	*No links*
404	Brand not found Media type: application/json Example Value | Schema `{ "error": "Entity Not Found" }`	*No links*

Get By Id: Here we fetch details of a product by product Id and then pass it to the service layer to check if all the data passed is valid and according to the parameters defined.



GET `/brands/{id}` Get a brand by its ID

Parameters | Try it out

Name	Description
X-API-KEY * required string (header)	*Example* : brand-r brand-r
id * required integer (path)	ID of brand to be retrieved id

Responses

Code	Description	Links
200	Successful operation Media type: application/json Controls Accept header. Example Value | Schema	*No links*
400	Invalid ID Media type: application/json Example Value | Schema	*No links*
401	Unauthorized request	*No links*
403	Forbidden request	*No links*
404	Brand not found Media type: application/json Example Value | Schema	*No links*
405	Invalid Request Method	*No links*

200 example value:

```
{
  "id": 4,
  "name": "Apple"
}
```

400 example value:

```
{
  "error": "Invalid Parameter"
}
```

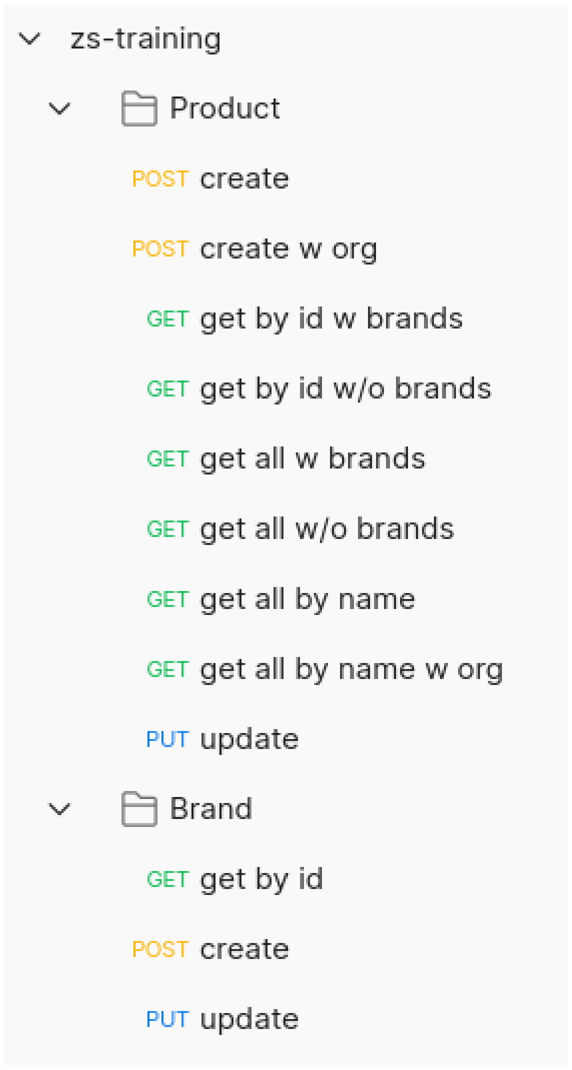
404 example value:

```
{
  "error": "Entity Not Found"
}
```

POSTMAN COLLECTION

PostMan is an API platform for creating and utilizing APIs and to help you design better APIs faster, Postman improves collaboration and simplifies every stage of the API lifecycle.

The below figure is a collection of all the endpoints available for the **Products** and **Brands** entities present in the Zopstore.



```
zs-training
    Product
        POST create
        POST create w org
        GET get by id w brands
        GET get by id w/o brands
        GET get all w brands
        GET get all w/o brands
        GET get all by name
        GET get all by name w org
        PUT update
    Brand
        GET get by id
        POST create
        PUT update
```

CHAPTER 4 : Performance Analysis

Unit Testing:

When building ethical Go programmes, unit testing is a crucial component. The go test command executes tests, and the testing package offers the tools we need to create unit tests.

This code is in package main for demonstration purposes, but it could be in any package. Usually, testing code is housed in the same package as the code it is testing.

Using a table-driven approach, where test inputs and anticipated outputs are specified in a table and a single loop runs through them to execute the test logic, is idiomatic since writing tests may be repetitive.

Benchmarking:

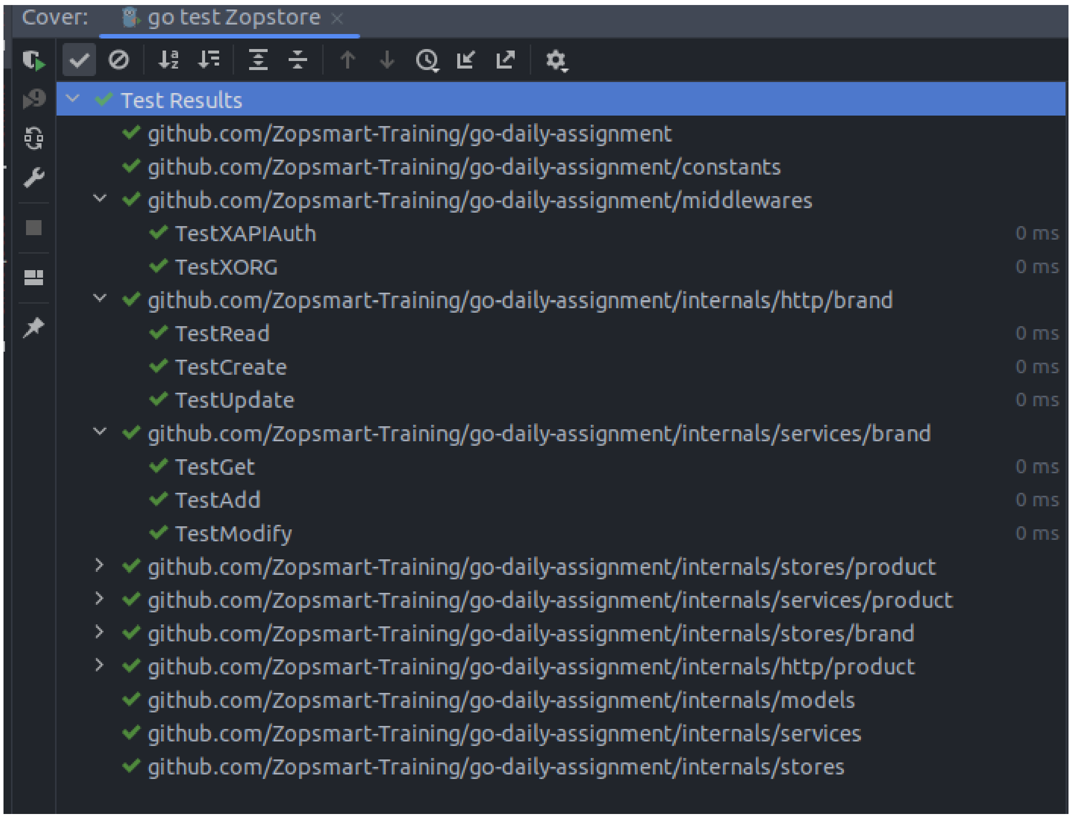
A benchmark is a special kind of function that runs a code segment several times and measures each output's performance against a benchmark to determine the code's overall degree of performance.

Golang comes with built-in benchmarking tools in the testing package and the go tool, allowing you to create relevant benchmarks without the need to install any dependencies.

Creating a benchmark is as easy as creating a function with the name Benchmark and the necessary signature, because the Go tooling integrates testing and benchmarks. The benchmark only gets type testing. B. It tells us how many times we should do the procedure. By default, the function will run for a full second if a number of iterations are given. This is done to show statistical significance.

1. Unit Test Coverage

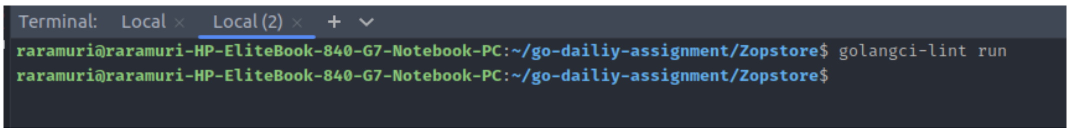
Performed unit test coverage and found all **28** tests ran successfully i.e., **PASS** with a total coverage of **100%**



```
Cover:  go test Zopstore
✓ Test Results
  ✓ github.com/Zopsmart-Training/go-daily-assignment
  ✓ github.com/Zopsmart-Training/go-daily-assignment/constants
  ✓ github.com/Zopsmart-Training/go-daily-assignment/middlewares
    ✓ TestXAPIAuth                                                          0 ms
    ✓ TestXORG                                                              0 ms
  ✓ github.com/Zopsmart-Training/go-daily-assignment/internals/http/brand
    ✓ TestRead                                                              0 ms
    ✓ TestCreate                                                            0 ms
    ✓ TestUpdate                                                            0 ms
  ✓ github.com/Zopsmart-Training/go-daily-assignment/internals/services/brand
    ✓ TestGet                                                               0 ms
    ✓ TestAdd                                                               0 ms
    ✓ TestModify                                                            0 ms
  ✓ github.com/Zopsmart-Training/go-daily-assignment/internals/stores/product
  ✓ github.com/Zopsmart-Training/go-daily-assignment/internals/services/product
  ✓ github.com/Zopsmart-Training/go-daily-assignment/internals/stores/brand
  ✓ github.com/Zopsmart-Training/go-daily-assignment/internals/http/product
  ✓ github.com/Zopsmart-Training/go-daily-assignment/internals/models
  ✓ github.com/Zopsmart-Training/go-daily-assignment/internals/services
  ✓ github.com/Zopsmart-Training/go-daily-assignment/internals/stores
```

2. 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 gocognit complexity or funlen to be 0 etc. There were **no** linter errors found in this project.



```
Terminal:  Local  Local (2)  +
raramuri@raramuri-HP-EliteBook-840-G7-Notebook-PC:~/go-dailiy-assignment/Zopstore$ golangci-lint run
raramuri@raramuri-HP-EliteBook-840-G7-Notebook-PC:~/go-dailiy-assignment/Zopstore$
```

CHAPTER 5 : CONCLUSION

5.1 Results Achieved

The main aim of the training was to be able to understand and implement the concepts of **GoLang, MySQL, Unit Testing,** being able to create a web application successfully performing basic CRUD operations and can be tested using postman using the **three layered architecture.**

5.2 Applications Contributions

GoLang have been part of a variety of real world/ open source applications, some of the which are listed below:

1. Docker, a collection of tools for deploying Linux containers, the Kubernetes container management system, and Dropbox, which converted some of their important components from Python to Go Ethereum, are all examples of companies that have migrated from Python to Go Ethereum.
2. The go-ethereum Ethereum Virtual Machine implementation, blockchain for the Ether cryptocurrency
3. Gitlab is a web-based DevOps lifecycle solution that includes a Git repository, a wiki, issue tracking, continuous integration, and a deployment pipeline, among other capabilities.

5.3 Limitations

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

5.4 Future Work / Scope

1. Front-end for application
2. Make the program more extensive

REFERENCES

1. https://go.dev/doc/
2. https://github.com/golang/mock
3. https://github.com/DATA-DOG/go-sqlmock
4. https://github.com/gorilla/mux
5. https://dev.mysql.com/doc/
6. https://www.linux.org/
7. https://docs.docker.com/
8. https://kubernetes.io/docs/home/
9. https://ngdocs.harness.io/
10. https://prometheus.io/docs/introduction/overview/
11. https://www.tutorialspoint.com/go/index.htm
12. https://www.techtarget.com/searchitoperations/definition/Go-programming-language
13. https://twitter.com/golang?lang=en
14. https://www.freecodecamp.org/news/what-is-go-programming-language/
15. https://github.com/golang/go
16. https://www.geeksforgeeks.org/go-programming-language-introduction/
17. https://go.dev/
18. https://gobyexample.com/
19. https://go.dev/doc/tutorial/getting-started
20. https://golangbyexample.com/
21. https://quii.gitbook.io/learn-go-with-tests/
22. https://www.digitalocean.com/community/tutorials/how-to-write-unit-tests-in-go-using-go-test-and-the-testing-package
23. https://www.postman.com/