

Conversion of Microservice from PHP to Golang

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

in

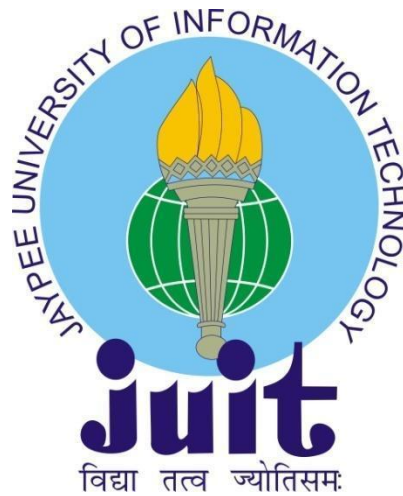
Computer Science and Engineering/Information Technology

By

Aryaman Sinha (191330)

Under the supervision of

Dr.Vipul Sharma and Dr.Nishant Jain



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 this project has been done by me under the supervision of **Dr.Nishant Jain, Department of Electronics and Communication and Dr.Vipul Sharma, Department of Computer Science & Engineering, Jaypee University of Information Technology**. I also declare that neither this project nor any part of this project has been submitted elsewhere for the award of any degree or diploma.

Aryaman Sinha 191330

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

Dr.Vipul Sharma

Dr.Nishant Jain

JAYPEE UNIVERSITY OF INFORMATION TECHNOLOGY, WAKNAGHAT

PLAGIARISM VERIFICATION REPORT

Date:

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

Name: _____ Department: _____ Enrolment No _____

Contact No. _____ E-mail. _____

Name of the Supervisor: _____

Title of the Thesis/Dissertation/Project Report/Paper (In Capital letters): _____

UNDERTAKING

I undertake that I am aware of the plagiarism related norms/ regulations, if I found guilty of any plagiarism and copyright violations in the above thesis/report even after award of degree, the University reserves the rights to withdraw/ revoke my degree/report. Kindly allow me to avail Plagiarism verification report for the document mentioned above.

Complete Thesis/Report Pages Detail:

- Total No. of Pages =
- Total No. of Preliminary pages =
- Total No. of pages accommodate bibliography/references =

(Signature of Student)

FOR DEPARTMENT USE

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

(Signature of Guide/Supervisor)

Signature of HOD

FOR LRC USE

The above document was scanned for plagiarism check. The outcome of the same is reported below:

Copy Received on	Excluded	Similarity Index (%)	Generated Plagiarism Report Details (Title, Abstract & Chapters)	
	<ul style="list-style-type: none">• All Preliminary Pages• Bibliography/Images/Quotes• 14 Words String		Word Counts	
Report Generated on		Submission ID	Character Counts	
			Total Pages Scanned	
			File Size	

Checked by
Name & Signature

Librarian

.....

Please send your complete thesis/report in (PDF) with Title Page, Abstract and Chapters in (Word File) through the supervisor at plagcheck.juit@gmail.com

ACKNOWLEDGEMENT

First and foremost, I want to give Almighty God my sincere gratitude and appreciation for His heavenly grace, which made it possible to finish the project work successfully.

Supervisor Dr.Vipul Sharma, Department of CSE, Jaypee University of Information Technology, and Supervisor Dr.Nishant Jain, Department of ECE, Jaypee University of Information Technology, both deserve my sincere gratitude and obligation of gratitude. To complete this research, my supervisor's deep knowledge and genuine interest in the topic of "Product Optimisation" were key factors. This project was made possible by their unending kindness, expert direction, ongoing encouragement, persistent and vigorous supervision, constructive criticism, insightful suggestions, and reviewing several subpar draughts and rectifying them at all levels.

I would like to sincerely thank my bosses for their wonderful assistance

Aryaman Sinha
(191330)

TABLE OF CONTENTS

CHAPTER 1: INTRODUCTION	Page no 7-9
CHAPTER 2 : LITERATURE SURVEY.....	Page no 10-34
CHAPTER 3 : IMPLEMENTATION.....	Page no 35-45
CHAPTER 4 : CONCLUSION.....	Page no 46

List of Figures

Sno.	Figure	Page No.
1	PHP	34
2	Main	35
3	Structs	36
4	Handler	37-39
5	Repository	40-42
6	Unit test handler layer	43-44
7	Docker Compose File	45

List of Tables

Sno.	Table	Page No.
1	Hardware Configuration	8-9
2	Software Configuration	9

ABSTRACT

A web application may be made rather easily, but testing, structuring, cleaning, and maintaining the code is a barrier which is highly required at a company where the code base is used by hundreds of developers.

Also, once other barrier that we need to address is upgradation of the code base with advancement is languages and multithreaded. Fo example, our current backend codebase comprises of PHP, but it is almost 10 times slower than Go language

To address the structuring issue, we use the Go language and work with the Three Layered Architecture. The three layers that are used are handlers, usecase, and repository. After receiving the request body, the handler layer parses it for any necessary information. The response is then sent to the usecase layer which is where we implement the business logic of the code. Incase we require any data, the data is collected from the repository layer which is where we request data from the database.

CHAPTER 1 : INTRODUCTION

Bobble AI is india's first and the biggest The Conversation Media Platform which enables seamless conversations. Because what friends say to one another is the most genuine and convincing kind of communication, it is a strong platform that enhances regular discussions.

1.2 Introduction

It is mostly based on CRUD but also includes complex business logics and uses the clean architecture. All the layers also have their own unit tests so that each step of the code can be tested.

1.3 Objectives

To convert, the existing codebase which is in PHP to Golang which is faster, and better performing

1.4 Motivation

To apply industrial practices and create an impenetrable, and seamless web application.

1.5 Libraries/Frameworks Used

Used libraries and frameworks A simple, dependable, and effective programming language called GO was created by Google developers to create code for apps. GO offers a number of the packages used in this project, including:

1. net/http: This package offers implementations for http client/server.
2. json: This package implements JSON encoding and decoding.
3. errors: This method implements the manipulation of errors.
4. database/sql: This package offers a database that resembles SQL.

And for unit testing: mockery library is used

Information Regarding PHP

All the backend framework such as implementing http request, sending response to server, writing program logic etc was written in PHP 1.5 Technical Requirements

- VSCode is an IDE to work on the code
- Postman is used for testing the api's built locally
- A database management system's querying and networking capabilities are provided by MySQL server.

1.5.1 Hardware Configuration

Table 1 : Hardware Configuration

Processor	Apple M1 chip, 8-core CPU
RAM	8 GB
Hard Disk	256 GB SSD
Monitor	13"
Mouse	
Keyboard	

Information Regarding Golang

The whole backend framework, including creating programme logic, delivering responses to servers, implementing http requests, etc., has been transformed to Go.

1.5 Requirements for Technology.

- An IDE for writing clean code is called GOLand.
- Postman, an API development and use platform.
- Mysql server offers connection and querying capabilities for database administration systems.

1.5.1 Hardware Configuration

Table 1 : Hardware Configuration

Processor	Apple M1 chip, 8-core CPU
RAM	8 GB
Hard Disk	256 GB SSD
Monitor	13''
Mouse	
Keyboard	

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

1) GO Documentation

R Griesemer, R Pike, and K Thompson developed the statically typed, open source, and compiled programming language known as Go when they were employed at Google.

2) MySQL Documentation

MySQL is an open-source relational database management system that enables us to store data, retrieve data, remove an item, and more. MySQL is short for My Structured Query Language.

3) Mockery

Mockery is a mocking framework that works nicely with the built-in testing package in the GO programming language.

4) Github and Git

An official platform that introduces you to the principles of Git, a version control system, and how GitHub integrates with it.

4.3 Study Material

LINUX

The study material discusses LINUX, which is a Unix-based operating system that supports both command line interface and graphical user interface. The default shell in Linux is BASH (Bourne Again Shell), and it has a package manager for installing, upgrading and cleaning packages.

The default package manager of Ubuntu is APT (Advanced Packaging Tool), and SUDO (superuser do or substitute user do) is used for running commands as a superuser. There are various processes and environment variables set in the system environment that affect the processes using them. We can set environment variables in hidden files called `~/.bashrc` or `/.bash_profile` present in the home directory().

The PATH specifies the locations to be searched to find a command. The study material also covers various LINUX commands such as ls, cd, touch, pwd, mkdir, rmdir/rm, mv, cat, chmod, and vi.

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

Packages GO

Every single Go programme is composed of packages. Every programme in the Go environment begins to run in the primary math/Rand package:- When running the rand package, the environment is deterministic.If we desire different results each time we use rand, it will return the same number.

Seed:

Use " " with packages and ()-for clarity[Factored statement] with import.
Use capital letters with their package when exporting names, such as Pi(math.Pi).
To format everything, we may use the fmt: formatted i/o package.

Functions:func()

A function in GO may accept 0 or more arguments. After a variable name, a function's type is listed.

- func add(int a, int b) return int a+b
- Add(9, 2) to invoke a function: func main()
- Add(a, b int): ex- add(int a, int b)

Import: All the libraries and packages are imported into Go in alphabetical order.

Third-party packages are written after all the built-in packages.

File Watcher:

It is a tool that is used to import all packages, format the code by making it standard and tidy, and more.

To declare a variable or collection of variables, we use the var keyword. The variables may be declared at the package or function level: var i int, defaults to 0 and false for bool. One initializer can be used for each variable. For instance, var i, j int=4,2 var k, j=false, "no!". The short assignment statement:= can be used inside of a function in place of a var declaration with an implicit type, but outside of a function, we must use var, func, etc.

Using the syntax: func main() var m, n int=3,2 k,n:= true,"no!"

Global variables should never be used when generating variables.

The first values of variables declared without an explicit starting value are 0 (numerical), false (boolean), and "" (string).

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 fmt: formatted i/o package to format all this.

Functions:func()

In GO a function can take 0 or more agrs. The type of a function comes after variable name

```
func add(a int, b int) int{ return a+b}
```

```
To call a function- func main(){ add(9,2)}
```

```
ex- add(int a, int b): add(a, b int)
```

A function can return any number of results

```
ex- 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, makes 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 convenience, assignment statements can be used in place of implicitly typed var declarations outside of functions, we can't use :=; we have to use var, func, etc.

```
Ex:func main(){  
var m, n int=3,2  
k,n:= true,"no!"  
}
```

While creating variables we should always avoid global variables Variable declared without an explicit initial value are 0, numeric; false for boolean and "" for string

Type Conversion:

Type Converts say integer to float or vice-versa i.e converts 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`(character), `string`, `boolean`, or `numeric` values. They cannot be declared 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 with short statements meaning `if` can start with a short statement before execution but it should end with a semicolon. Any of the `else` blocks can access variables that are defined inside an `if` short statement.

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 i.e. no duplicate case or it gives us type mismatch error or compile time error.

POINTERS

Pointers holds the memory address of any value that is provided to it.

*K, a pointer to K value, *pointer value

To dereferencing/ indirecting, *k=28 ()

Pointer's zero value is nil.

declaration: var k* float, for example

& operator generates a pointer to its operand: k=&i → k=*p

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. Since array length is a property of its type, it cannot be changed. to declare and 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 array.It doesn't store any data, we just describe a section of an array.A

slice's underlying array and any other slices that have similar elements are affected when an element of a slice is changed..Eg: a:= names[1:2]; names-array

An array literal without the length is known as a slice literal.

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 array.

Make creates zeroed array and returns 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 underlying array.

RANGE

A Range is an iterative variant of a for loop that traverses a slice or map.

It returns an index and a duplicate of the value at that index for each iteration.

By assigning, we may also omit the index or value.

Syntax:

for j, _ := range power

for _, value := range power

If we only want index we can omit the second variable

for j:=range power

<< : times 2

>>: divide 2

For instance, 1 5 equals 32 or "1 times 2, 5 times". Additionally, 32 >> 5 is equivalent to "32 divided by 2, 5 times" or 1.

MAPS

A map links values to keys.
Value 0 map is zero. A keyless map lacks keys.
You can exclude the top-level type from the literal components if it is only a type name.

```
Ex: var m map[string]Vertex
func main() {
map = make(map[string]Vertex)
map["B Labs"] = Vertex{
40.68423, -74.39867,
} ; Or we can omit vertex
```

Mutating maps

To insert/update: `m[key]=elem`
To retrieve : `elem=m[key]`
To delete: `delete(m,key)` : to delete a key
To test if key is present: `elem, ok=m[key]` ; `ok=true` if key present else `false`

If we can't find key in the map, then `elem` is the zero value for the map's element type.
A key in any given map cannot be slice.

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

Like other values, functions are also values that may be transferred from one place to another.Both function parameters and return values can be made up of function values. It could be closure when it comes to Go functions.

A function value that makes references to variables outside of its body is known as a closure. The referenced variables are accessible by the function, and in this sense, the function is "bound" to the variables.

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.

Zero value of interface is nil.

Abstract type underlying which is our concrete type (struct, float, etc): can be thought of as a tuple

of a value and a concrete type: (value, type)

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

Type SWITCH

```
switch s := i.(type) {  
case T:  
// here s has type A  
case S:  
// here s has type B  
default:  
// no match; here s has the same type as i  
}
```

The declaration in a type switch has the same syntax as a type assertion `i.(T)`, but the specific type `T` is replaced with the keyword `type`.

STRINGERS

It is an ubiquitous interfaces defined by the `format(fmt)` package.

```
typebStringerbinterfaceb {  
String()bstring  
}
```

Readers

```
func (T) Read(b []byte) (n int, err error)
```

Type Image:

The actual struct that implements the Image interface is the RGBA type.

REST- Representation State Transfer

An application programming interface (API or web API) that complies with the constraints of the REST architectural style is referred to as a REST API (also known as a RESTful API). Computer scientist Roy Fielding developed the representational state transfer protocol, or REST.

Principles of RESTful Design

Decoupling of client and server - Client and server programmes must be totally independent of one another in a REST API architecture. Only the URI of the requested resource should be known by the client software; it cannot connect to the server application in any other way. A server application should not change the client software except to provide it with the necessary data over HTTP.

Due to the statelessness of REST APIs, each request must include all of the data required to process it. REST APIs, in other words, do not require any server-side connectivity. Any data relating to a client request cannot be saved by server programmes.

Cacheability - Resources should, wherever possible, 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. In the communication loop, there could be several different middlemen. REST APIs must be built in such a way that neither the client nor the server can access them.

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

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

1. HTTP protocols

Create : Post-> new data

Read : Get-> retrieve data

Update: Put-> update data

Delete: Delete-> delete data

2. ServeMux(Multiplexer)

- ServeMux is an HTTP request multiplexer.
- Responsible for matching URLs in request to an appropriate handler and executing it.
http.NewServerMux.[url handler using Handle and HandleFun methods]
Handle Method:
- It accepts a String and an http.Handler
- Func (mux *ServeMux) Handle(pattern string, handler Handler)
- http.Handler is an interface (second parameter in the Handle method) with the

ServeHTTP method

```
type Handler interface {  
    ServeHTTP(ResponseWriter, *Request)
```

```
}
```

```
Eg: func (h home) ServeHTTP(rw http.ResponseWriter, r *http.Request) {  
    rw.Write([]byte("Welcome to the Just Enough Go! blog series!"))  
}
```

```
mux := http.NewServeMux()  
mux.Handle("/", home{ })
```

- HandleFunc accepts the handler implementation in the form of a function (along with the path for which it is to be invoked).

```
func (mux *ServeMux) HandleFunc(pattern string, handler func(ResponseWriter, *Request))  
Server (HTTP server)  
server := http.Server{Addr: ":8080", Handler: mux} -> Addr is the address on which the  
server listens e.g. http://localhost:8080 and Handler is actually an http.Handler instance.
```

- If you just had a route or path which you wanted to handle, you can pass an instance of an http.Handler (e.g. home{ } in this case) and skip the ServeMux altogether.

- You can/should pass an instance of a ServeMux so that you can handle multiple routes/paths (e.g. /home, /items etc.) Internally, it works by dispatching or routing to the appropriate handler based on the path (URL) in http.Request.

```
func (mux *ServeMux) ServeHTTP(w ResponseWriter, r *Request)
```

- DefaultServeMux: We don't need to use an explicit ServeMux. The Handle and HandleFunc methods available in a ServeMux are also exposed as global functions in the net/http package.

- To start the HTTP server, you can use the http.ListenAndServe function, just as you would with a Server instance.

```
func ListenAndServe(addr string, handler Handler) error
```

- The handler parameter can be nil if you have used http.Handle and/or http.HandleFunc to specify the handler implementations for the respective routes.

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

HTTP Test Package

1. `HttpRequest`: The `httpptest.NewRequest` function provides a brand-new incoming server request that may be tested by feeding it to a `http.Handler`.

2. `HttpResponseWriter`: `w=httpptest.NewRecorder` type returns `httpptest.ResponseRecorder`. `ResponseRecorder` is an implementation of `http.ResponseWriter` that records its mutations for later inspection in tests. It can be used to pass into our server handler, record all the data that the handler will write to the response, and then return the data written after.

3. `w.Result()`: `func (rw *ResponseRecorder) Result` returns the response produced by the handler. At a minimum, the `StatusCode`, `Header`, `Body`, and optional `Trailer` fields in the returned `Response` will be filled up.

Layered architecture: Layers are autonomous from one another and interact via interfaces.

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 initialization 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.

Factory method

The issue of constructing product objects without identifying their concrete classes is resolved by this design approach. Instead of directly invoking the `new` operator, it offers a method that may be used to build objects.

1. Simple factory
2. Interface factories.

MICRO SERVICES

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 compact, small 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.

Micro services' advantages

Adaptable 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.

Simple Deployment

Continuous integration and delivery are made possible by microservices, making it simple to test new concepts and roll them back if they don't work. The cheap cost of failure makes it feasible for more experimentation, quicker code revisions, and speedier time-to-market for new features.

Reusable Code

By segmenting software into distinct, well-defined modules, teams may employ functions for a variety of reasons. A service developed for one purpose may serve as the basis for another feature. An application can self-bootstrap as a result of developers' ability to

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 Liquidbase 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, 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. It can also be used for a traditional (non-container) bare server in which applications are directly deployed.

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.

MATRICES: 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/metric endpoint where prometheus can scrape them

Frameworks we use for microservices

- Base Framework: [echo](#)
- Config Management: [vipser](#)
- Database Migration: [golang-migrate](#)
- Database ORM: [GORM](#)

Clean Architecture

The basic premise is to pan out your code into layers so each layer can be tested independently and the driver being a layer can be swapped out for another one without the inner layer being affected.

Branch Naming Convention

Branch Name	What is it used for
master	Deploying to production. Only reviewed and tested code should be merged in this
staging	Used for deployment to staging server where the code can be test. Can be deleted and remade if required (if 2 development branches have to be tested which contain conflicting code)
feature/*	Any feature which is being worked on. Should be hyphen-separated . Eg. feature/user-logs . In case of starting/overhauling a project, naming like feature/keyboard-theme-v2 is also accepted.
bugfix/*	Bug fixes for code
hotfix/*	Bug fixes for code which need to be deployed immediately

MR Best Practices

- Title should be short
- Description should highlighted all the changes done in bullet points
- Add a **WIP: Title Name** before your name if your branch is under development and is not ready for its final code review.

Ignored Files (Golang)

Certain files should not be uploaded to the git repo or the docker container. Examples include:

- .git/*
- <binary_name>
- config.yml
- go.sum (if you want to upload then follow the proper steps)
- .idea
- .vscode
- .DS_store

Naming Conventions

The following naming conventions should be followed for files, variables, function names, custom datatypes etc.

Files

- snake_case for filename
- continuouslowercase for folder name

Variables

- Variables should be in camelCase or PascalCase (if they need to be accessed by other packages)
- The variable name should give an idea of what kind of data it contains, for example

```
# Bad

n = []string{"Ajit", "Akansha", "Pal"}

for i, v := range n {
    // Some process here
}
```

```

# Better
names = []string{"Ajit", "Akansha", "Pal"}
for i, n := range names {
    // Some process here
}

# Best
customerNames = []string{"Ajit", "Akansha", "Pal"}
for index, customerName := range customerNames {
    // Some process here
}

```

- Structures should follow the same format

```

# Bad
type Cust struct {
    Name string
    Add  string
}

# Overkill
type Customer struct {
    CustomerName      string
    CustomerHomeAddress string
}

# Ideal
type Customer struct {
    Name      string
    HomeAddress string
}

```

```
}
```

- Any field names in plural like `Customers` or `Themes` is expected to contain an array of `Customer` or `Themes`. For example,

```
type Users struct {  
    GroupID int  
    Customers []Customer  
}
```

- While fetching a value using different params, use the `By` word in the name. For example, if we need to fetch a customer based on email or id, then we can use:

```
func (c *Customer) GetCustomerByID(id int) (customer Customer, error error){}
```

Here we avoided calling our function `GetByID` since when this function is called, it is not necessary that the calling object will be called `customer`. `customer.GetByID()` makes sense, `c.GetByID()` does not.

- `delivery`, `domain` and `repository` can have their own structs which dictate the request/response they will receive. However, in most cases there will only be 2. One will be used to get data from the repository and the other will be used to send data to our endpoint. Use `Customers` and `CustomersResponse` for them respectively.

Variable parsing

If there is some un-processed variable which is being fetched then the name similar names should not be used. Eg.

```
# Bad since keys and private keys can refer that they are the  
same data type or can be used in place of each other
```

```
keys = viper.GetString("ENCRYPTION_KEYS") # string  
privateKeys, err := loadEncryptionKeys(keys) # object
```

```
# Good
```

```
keyString = viper.GetString("ENCRYPTION_KEYS") # string
privateKeys, err := loadEncryptionKeys(keyString) # object
```

Application Structure:

/app

main.go

/config

database_config.go

application_config.go

/domain

/database

/migrations

/keyboardthemes

/delivery

/http

/helpers

keyboard_theme_helper.go

keyboard_theme_category_helper.go

keyboard_theme_handler.go

keyboard_theme_handler_test.go

/usecase

```
keyboard_theme_usecase.go
keyboard_theme_usecase_test.go
/repository
  /helpers
    keyboard_theme_repository_helpers.go
  /mysql
    keyboard_theme_mysql.go
    keyboard_theme_mysql_test.go
go.mod
config.yml.example
Makefile
.gitignore
.dockerignore
Dockerfile
.gitlab-ci.yml
Readme.md
```

Do not upload config.yml to the repo. It might contain sensitive data. Copy it, rename it as config.yml.example, change env values to sample values and then add to repo. Uploading go.sum is optional, however, if it is being done, then please ensure all the proper steps are taken

All files under helpers should ideally belong to the same package helpers. However if need arises there can be different packages if there are too many common functions. This should be avoided since it generally implies that the function name is not descriptive enough.

Configuration Management

Use Viper for configuration management.

In a traditional setup there would be a .env file along with another config file which would be used for application wide config params like keyboard_theme_default_results etc.

We combine this into one by using Viper. It can automatically pick up env variables by using viper.Environment() which ensures that we only need one configuration file.

Environment variables should be loaded in a structure, ensuring that we can move away from viper if required for the storage of the same. For example, the database host should not be accessed as viper.GetString("DATABASE_HOST") but should be accessed as config.DatabaseConfig.DATABASE_HOST or config.DatabaseConfig.Host

Loading Configuration file

Loading the configuration file should be carried out once. If there are multiple configuration structures (database, application, cache), they should not be loading the same file again. Load the file in an init() function or have a loader function which is called by main at the start.

Golang Error Handling

Standard Error Response

```
# Assume the following GET request is sent for an ID
corresponding to which a keyboard theme does not exist
```

```
# /v1/keyboardTheme/7
```

```
{
    "errorCode": "keyboardThemeNotFound",
    "errorDescription": "No keyboard theme was found with the
ID 7"
}
```

- `errorCode` has to be in camelCase and should contain a generic response about the error which no specifics pertaining to the request.
- `errorDescription` should not contain the value of `err` if the response is being sent to an external client. Always use a custom response.

Error Code Naming Convention

Name Type	When to use
<code>missingStickerID</code>	When a an input parameter is missing
<code>invalidStickerID</code>	When the input parameter does not match the format. Eg, expected an <code>int</code> but received a <code>string</code>
<code>stickerNotFound</code>	When a valid input parameter does not yield the expected result
<code>inactiveBufferedReader</code>	When you try to read from an inactive buffered reader

Error Handling

- In most cases, only the parent function should be responsible for handling the error.
This will avoid situations where the err code has to be checked

```
# Assume main is called by an API which has to return a response to a user
```

```
func main() {  
    sum, err := getSum()  
    if err != nil {  
        # Log the exact error to the logger so we can debug later  
        log.Error(err)  
    }  
}
```

```
        # Return a generic error to the user
        makeResponse("unableToGetSum", 400)
    }
    makeResponse(sum, 200)
}
```

Application startup errors

Since these errors have to be returned to an internal source, i.e. during deployment, application details can be passed here

Env variable missing

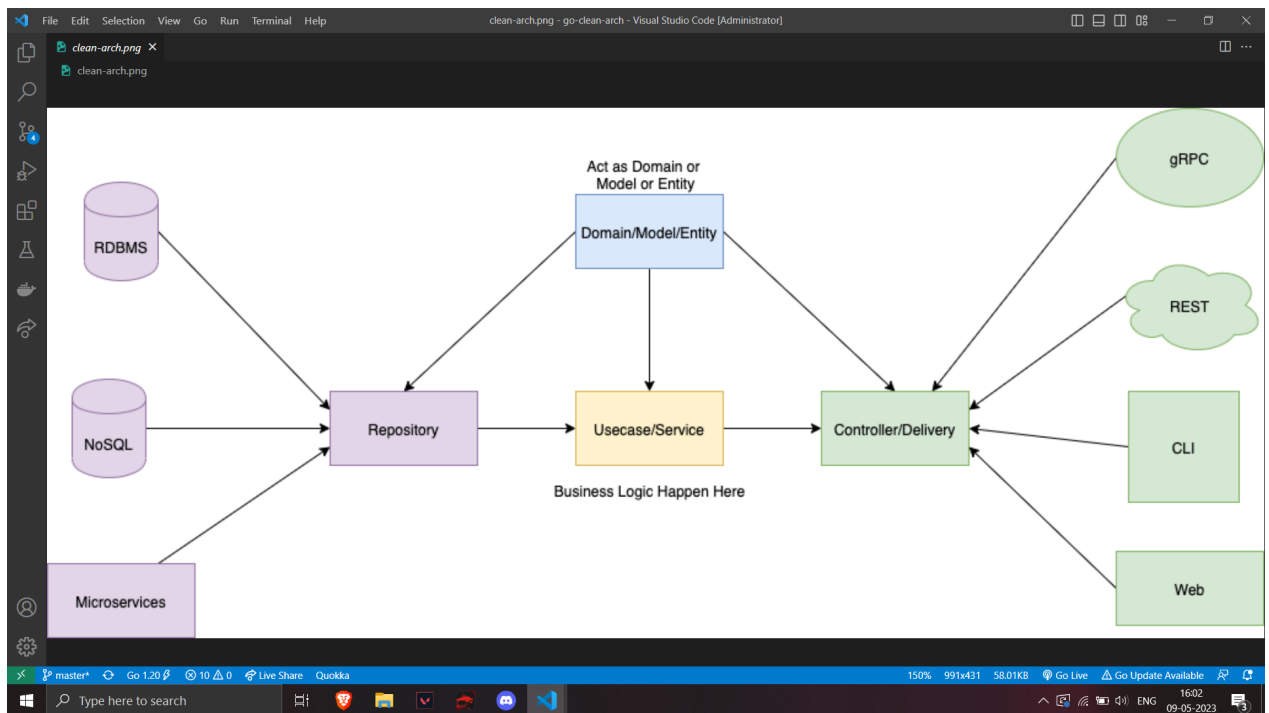
While returning an error for an env variable, ensure that the env variable name is added in the error message so it can be easily debugged. Eg. If `DATABASE_HOST` was not set, then the error message should be `DATABASE_HOST was not set in the environment`.

A centralised logger should be used so as to prevent using multiple loggers. While using echo framework, the default logger requires the a `echo.Echo` type variable (which is initialised at the start during app startup).

To prevent passing a pointer to this variable to whichever package that requires logging, simply create a new package called `logger` which will accept a `*echo.Echo` during app startup and will have wrapper functions which will use the default echo logger. This way all other functions which need to call echo logger will now call this package instead.

This way, we can switch loggers at one place without having to change the code elsewhere.

Clean Architecture



PHP

Hypertext Pre-processor, or PHP. It may be incorporated in HTML and is mostly used as a general-purpose scripting language to create dynamic online content. Using PHP

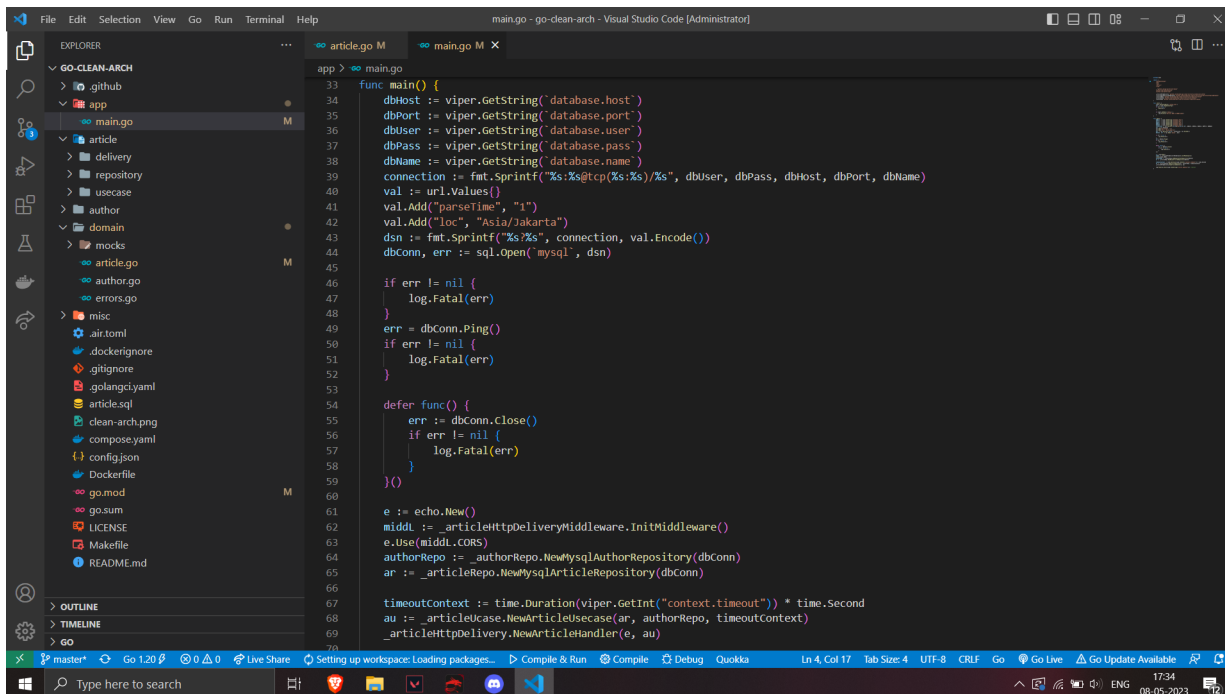
be utilised as a replacement for the JSP/Java System, ASP.NET/C#/VB.NET, and Macromedia ColdFusion. PHP is simple to use and has many similarities to Perl and other structured programming languages. PHP is a more complex language than merely coding.

It is a complete programming language that can create Graphical User Interface Applications as well as be used from a command line. Numerous popular operating systems, including Linux and Windows, are compatible with PHP, which also supports a wide range of database systems, including MySQL. The fact that PHP is dynamically typed is one characteristic that contributes to its popularity.

Variables may hold any kind of object and do not need to be defined. PHP arrays may store several sorts of objects, including other arrays. Numerous open-source libraries are included into PHP, along with modules for FTP and database server access.

CHAPTER 3: IMPLEMENTATION

Main

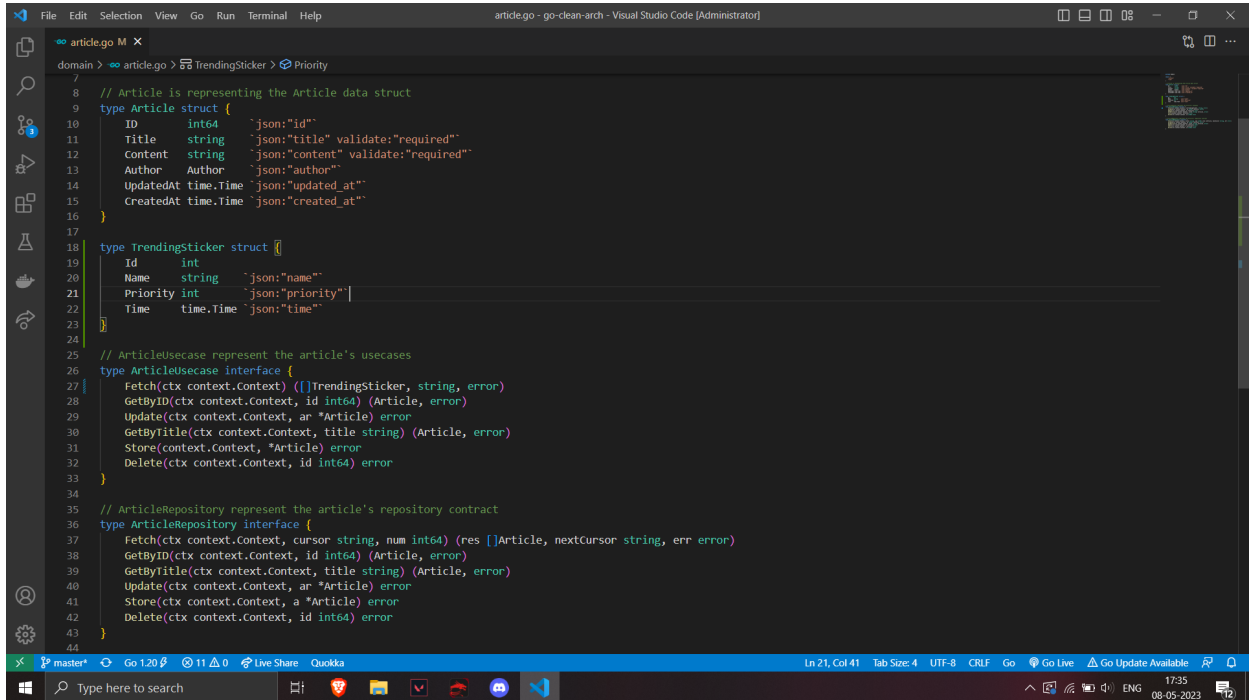


```
33 func main() {
34     dbHost := viper.GetString("database.host")
35     dbPort := viper.GetString("database.port")
36     dbUser := viper.GetString("database.user")
37     dbPass := viper.GetString("database.pass")
38     dbName := viper.GetString("database.name")
39     connection := fmt.Sprintf("%s:%s@tcp(%s:%s)/%s", dbUser, dbPass, dbHost, dbPort, dbName)
40     val := url.Values{}
41     val.Add("parseTime", "1")
42     val.Add("loc", "Asia/Jakarta")
43     dsn := fmt.Sprintf("%s%s", connection, val.Encode())
44     dbConn, err := sql.Open("mysql", dsn)
45
46     if err != nil {
47         log.Fatal(err)
48     }
49     err = dbConn.Ping()
50     if err != nil {
51         log.Fatal(err)
52     }
53
54     defer func() {
55         err := dbConn.Close()
56         if err != nil {
57             log.Fatal(err)
58         }
59     }()
60
61     e := echo.New()
62     middle := _articleHttpDeliveryMiddleware.InitMiddleware()
63     e.Use(middle.CORS)
64     authorRepo := _authorRepo.NewMySQLAuthorRepository(dbConn)
65     ar := _articleRepo.NewMySQLArticleRepository(dbConn)
66
67     timeoutContext := time.Duration(viper.GetInt("context.timeout")) * time.Second
68     au := _articleUcase.NewArticleUcase(ar, authorRepo, timeoutContext)
69     _articleHttpDelivery.NewArticleHandler(e, au)
70 }
```

Main is the file which is saved under the app file folder and is the entypoint for the application to start. It consists of all the main details like

- DB Connection
- URL endpoint routing
- Handler Functions

Structs



```
7
8 // Article is representing the Article data struct
9 type Article struct {
10     ID         int64      `json:"id" validate:"required"`
11     Title      string     `json:"title" validate:"required"`
12     Content    string     `json:"content" validate:"required"`
13     Author     Author     `json:"author"`
14     UpdatedAt  time.Time `json:"updated_at"`
15     CreatedAt  time.Time `json:"created_at"`
16 }
17
18 type TrendingSticker struct {
19     Id         int
20     Name      string    `json:"name"`
21     Priority   int       `json:"priority"`
22     Time      time.Time `json:"time"`
23 }
24
25 // ArticleUsecase represent the article's usecases
26 type ArticleUsecase interface {
27     Fetch(ctx context.Context) ([]TrendingSticker, string, error)
28     GetByID(ctx context.Context, id int64) (Article, error)
29     Update(ctx context.Context, ar *Article) error
30     GetByTitle(ctx context.Context, title string) (Article, error)
31     Store(context.Context, *Article) error
32     Delete(ctx context.Context, id int64) error
33 }
34
35 // ArticleRepository represent the article's repository contract
36 type ArticleRepository interface {
37     Fetch(ctx context.Context, cursor string, num int64) (res []Article, nextCursor string, err error)
38     GetByID(ctx context.Context, id int64) (Article, error)
39     GetByTitle(ctx context.Context, title string) (Article, error)
40     Update(ctx context.Context, ar *Article) error
41     Store(ctx context.Context, a *Article) error
42     Delete(ctx context.Context, id int64) error
43 }
44
```

Structs basically save all the structure that is to be used in the application. It is denoted by domains.

It also contains function description of all the layers:

- Handler layer
- Usecase layer
- Repository layer

A good practice here is to use table name which is when the structs are defined for the repository layer, the table name can be defined so that when the query is called from the repository layer, the table name does not need to be explicitly mentioned.

The function descriptions are of the format interface and are exported to all the layer's individual files.

Handler

```
File Edit Selection View Go Run Terminal Help article_handler.go - go-clean-arch - Visual Studio Code [Administrator]
article_handler.go M X
article > delivery > http > article_handler.go > (*ArticleHandler).FetchArticle
24 // HTTPMETHODS will fill all the methods of the ArticleHandler
25 func NewArticleHandler(e *echo.Echo, us domain.ArticleUsecase) {
26     handler := &ArticleHandler{
27         AUSeCase: us,
28     }
29     e.GET("/articles", handler.FetchArticle)
30     e.POST("/articles", handler.Store)
31     e.GET("/articles/:id", handler.GetByID)
32     e.DELETE("/articles/:id", handler.Delete)
33 }
34
35 // FetchArticle will fetch the article based on given params
36 func (a *ArticleHandler) FetchArticle(c echo.Context) error {
37     ctx := c.Request().Context()
38
39     listAr, nextCursor, err := a.AUSeCase.Fetch(ctx)
40     if err != nil {
41         return c.JSON(getStatusCode(err), ResponseError{Message: err.Error()})
42     }
43     c.Response().Header().Set("X-Cursor", nextCursor)
44     return c.JSON(http.StatusOK, listAr)
45 }
46
47
48 // GetByID will get article by given id
49 func (a *ArticleHandler) GetByID(c echo.Context) error {
50     idP, err := strconv.Atoi(c.Param("id"))
51     if err != nil {
52         return c.JSON(http.StatusNotFound, domain.ErrNotFound.Error())
53     }
54     id := int64(idP)
55     ctx := c.Request().Context()
56
57     art, err := a.AUSeCase.GetByID(ctx, id)
58     if err != nil {
59         return c.JSON(getStatusCode(err), ResponseError{Message: err.Error()})
60     }
61 }
X master Go 1.20.0 10 0 Live Share Quokka Ln 36, Col 62 Tab Size: 4 UTF-8 CRLF Go Go Live Go Update Available 17:36 08-05-2023
Type here to search
```

```
File Edit Selection View Go Run Terminal Help article_handler.go - go-clean-arch - Visual Studio Code [Administrator]
article_handler.go M X
article > delivery > http > article_handler.go > (*ArticleHandler).FetchArticle
65
66 func isRequestValid(m *domain.Article) (bool, error) {
67     validate := validator.New()
68     err := validate.Struct(m)
69     if err != nil {
70         return false, err
71     }
72     return true, nil
73 }
74
75 // Store will store the article by given request body
76 func (a *ArticleHandler) Store(c echo.Context) (err error) {
77     var article domain.Article
78     err = c.Bind(&article)
79     if err != nil {
80         return c.JSON(http.StatusUnprocessableEntity, err.Error())
81     }
82
83     var ok bool
84     if ok, err = isRequestValid(&article); !ok {
85         return c.JSON(http.StatusBadRequest, err.Error())
86     }
87
88     ctx := c.Request().Context()
89     err = a.AUSeCase.Store(ctx, &article)
90     if err != nil {
91         return c.JSON(getStatusCode(err), ResponseError{Message: err.Error()})
92     }
93
94     return c.JSON(http.StatusCreated, article)
95 }
96
97 // Delete will delete article by given param
98 func (a *ArticleHandler) Delete(c echo.Context) error {
99     idP, err := strconv.Atoi(c.Param("id"))
100     if err != nil {
101         return c.JSON(http.StatusNotFound, domain.ErrNotFound.Error())
102     }
X master Go 1.20.0 10 0 Live Share Quokka Ln 36, Col 62 Tab Size: 4 UTF-8 CRLF Go Go Live Go Update Available 17:37 08-05-2023
Type here to search
```

```
File Edit Selection View Go Run Terminal Help article_handler.go - go-clean-arch - Visual Studio Code [Administrator]
article_handler.go M article_handler.go M X
article > delivery > http > article_handler.go > (*ArticleHandler).FetchArticle
96
97 // Delete will delete article by given param
98 func (a *ArticleHandler) Delete(c echo.Context) error {
99     idp, err := strconv.Atoi(c.Param("id"))
100     if err != nil {
101         return c.JSON(http.StatusNotFound, domain.ErrNotFound.Error())
102     }
103     id := int64(idp)
104     ctx := c.Request().Context()
105
106     err = a.Usecase.Delete(ctx, id)
107     if err != nil {
108         return c.JSON(http.StatusOK, ResponseError{Message: err.Error()})
109     }
110     return c.NoContent(http.StatusNoContent)
111 }
112
113
114 func getStatusCode(err error) int {
115     if err == nil {
116         return http.StatusOK
117     }
118 }
119
120 logrus.Error(err)
121 switch err {
122 case domain.ErrInternalServerError:
123     return http.StatusInternalServerError
124 case domain.ErrNotFound:
125     return http.StatusNotFound
126 case domain.ErrConflict:
127     return http.StatusConflict
128 default:
129     return http.StatusInternalServerError
130 }
131 }
132 }
```

A handler layer has the main functions which the url end point points to and is used call the usecase layer which in return calls the repository layer.

Our handler layer also has the ability to check all the query parameters, formdata etc with the help of helpers' function

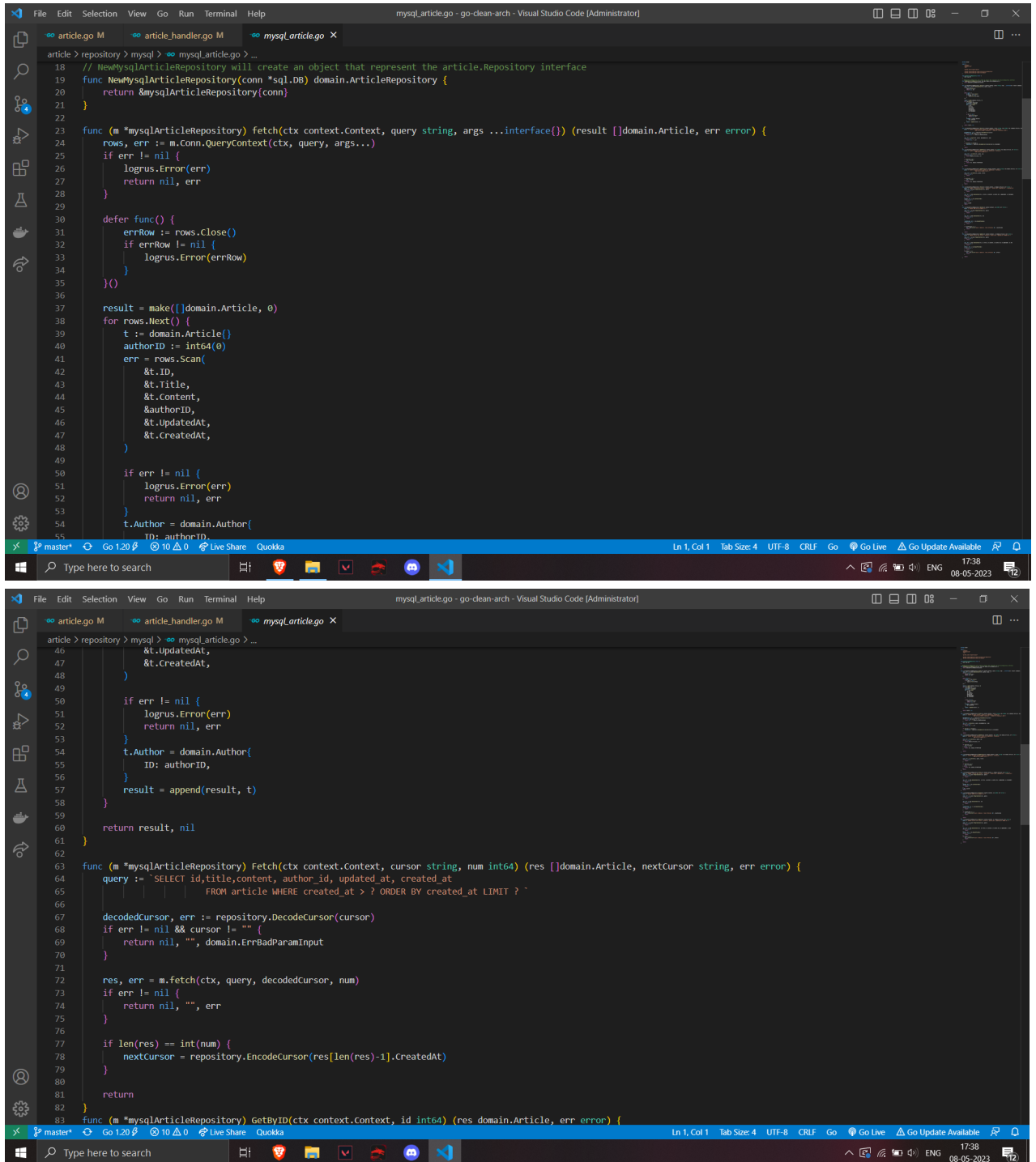
Usecase

```
File Edit Selection View Go Run Terminal Help article_ucase.go - go-clean-arch - Visual Studio Code [Administrator]
article_ucase.go M article_handler.go M article_ucase.go 1 X
article > usecase > article_ucase.go > ...
84 func (a *articleUsecase) Fetch(c context.Context, cursor string, num int64) (res []domain.Article, nextCursor string, err error) {
85     if num == 0 {
86         num = 10
87     }
88
89     ctx, cancel := context.WithTimeout(c, a.contextTimeout)
90     defer cancel()
91
92     res, nextCursor, err = a.articleRepo.Fetch(ctx, cursor, num)
93     if err != nil {
94         return nil, "", err
95     }
96
97     res, err = a.fillAuthorDetails(ctx, res)
98     if err != nil {
99         nextCursor = ""
100     }
101     return
102 }
103
104 func (a *articleUsecase) GetByID(c context.Context, id int64) (res domain.Article, err error) {
105     ctx, cancel := context.WithTimeout(c, a.contextTimeout)
106     defer cancel()
107
108     res, err = a.articleRepo.GetByID(ctx, id)
109     if err != nil {
110         return
111     }
112
113     resAuthor, err := a.authorRepo.GetByID(ctx, res.Author.ID)
114     if err != nil {
115         return domain.Article{}, err
116     }
117     res.Author = resAuthor
118     return
119 }
120
```

```
File Edit Selection View Go Run Terminal Help article_ucase.go - go-clean-arch - Visual Studio Code [Administrator]
article_ucase.go M article_handler.go M article_ucase.go 1 X
article > usecase > article_ucase.go > ...
120
121 func (a *articleUsecase) Update(c context.Context, ar *domain.Article) (err error) {
122     ctx, cancel := context.WithTimeout(c, a.contextTimeout)
123     defer cancel()
124
125     ar.UpdatedAt = time.Now()
126     return a.articleRepo.Update(ctx, ar)
127 }
128
129 func (a *articleUsecase) GetByTitle(c context.Context, title string) (res domain.Article, err error) {
130     ctx, cancel := context.WithTimeout(c, a.contextTimeout)
131     defer cancel()
132     res, err = a.articleRepo.GetByTitle(ctx, title)
133     if err != nil {
134         return
135     }
136
137     resAuthor, err := a.authorRepo.GetByID(ctx, res.Author.ID)
138     if err != nil {
139         return domain.Article{}, err
140     }
141     res.Author = resAuthor
142     return
143 }
144
145
146 func (a *articleUsecase) Store(c context.Context, m *domain.Article) (err error) {
147     ctx, cancel := context.WithTimeout(c, a.contextTimeout)
148     defer cancel()
149     existedArticle, _ := a.GetByTitle(ctx, m.Title)
150     if existedArticle != (domain.Article{}) {
151         return domain.ErrConflict
152     }
153
154     err = a.articleRepo.Store(ctx, m)
155     return
156 }
157
```

A usecase layer is the one which handles all the business logic

Repository



The image displays two screenshots of a Visual Studio Code editor window, showing Go code for a repository interface and its implementation. The editor is titled "mysql_article.go - go-dean-arch - Visual Studio Code [Administrator]".

The first screenshot shows the `mysql_article.go` file with the following code:

```
18 // NewMySQLArticleRepository will create an object that represent the article.Repository interface
19 func NewMySQLArticleRepository(conn *sql.DB) domain.ArticleRepository {
20     return &mysqlArticleRepository{conn}
21 }
22
23 func (m *mysqlArticleRepository) fetch(ctx context.Context, query string, args ...interface{}) (result []domain.Article, err error) {
24     rows, err := m.Conn.QueryContext(ctx, query, args...)
25     if err != nil {
26         logrus.Error(err)
27         return nil, err
28     }
29
30     defer func() {
31         errRow := rows.Close()
32         if errRow != nil {
33             logrus.Error(errRow)
34         }
35     }()
36
37     result = make([]domain.Article, 0)
38     for rows.Next() {
39         t := domain.Article{}
40         authorID := int64(0)
41         err = rows.Scan(
42             &t.ID,
43             &t.Title,
44             &t.Content,
45             &authorID,
46             &t.UpdatedAt,
47             &t.CreatedAt,
48         )
49
50         if err != nil {
51             logrus.Error(err)
52             return nil, err
53         }
54         t.Author = domain.Author{
55             ID: authorID,
```

```
118 func (m *mysqlArticleRepository) Store(ctx context.Context, a *domain.Article) (err error) {
119     query := "INSERT article SET title=?, content=?, author_id=?, updated_at=?, created_at=?"
120     stmt, err := m.Conn.PrepareContext(ctx, query)
121     if err != nil {
122         return
123     }
124
125     res, err := stmt.ExecContext(ctx, a.Title, a.Content, a.Author.ID, a.UpdatedAt, a.CreatedAt)
126     if err != nil {
127         return
128     }
129     lastID, err := res.LastInsertID()
130     if err != nil {
131         return
132     }
133     a.ID = lastID
134     return
135 }
136
137 func (m *mysqlArticleRepository) Delete(ctx context.Context, id int64) (err error) {
138     query := "DELETE FROM article WHERE id = ?"
139
140     stmt, err := m.Conn.PrepareContext(ctx, query)
141     if err != nil {
142         return
143     }
144
145     res, err := stmt.ExecContext(ctx, id)
146     if err != nil {
147         return
148     }
149
150     rowsAffected, err := res.RowsAffected()
151     if err != nil {
152         return
153     }
154
155     if rowsAffected != 1 {
156         err = fmt.Errorf("weird Behavior. Total Affected: %d", rowsAffected)
157         return
158     }
159     return
160 }
161
162 func (m *mysqlArticleRepository) Update(ctx context.Context, ar *domain.Article) (err error) {
163     query := "UPDATE article set title=?, content=?, author_id=?, updated_at=? WHERE ID = ?"
164
165     stmt, err := m.Conn.PrepareContext(ctx, query)
166     if err != nil {
167         return
168     }
169
170     res, err := stmt.ExecContext(ctx, ar.Title, ar.Content, ar.Author.ID, ar.UpdatedAt, ar.ID)
171     if err != nil {
172         return
173     }
174     affect, err := res.RowsAffected()
175     if err != nil {
176         return
177     }
178     if affect != 1 {
179         err = fmt.Errorf("weird Behavior. Total Affected: %d", affect)
180         return
181     }
182     return
183 }
184 }
185
```

Repository layer is the one which handles fetching data from the database.

Unit Tests Repository Layer

```

mysqarticle_test.go - go-clean-arch - Visual Studio Code [Administrator]
article.go M  article_handler.go M  mysqlarticle_test.go X

article > repository > mysql > mysqlarticle_test.go > ...
16 func TestFetch(t *testing.T) {
17     db, mock, err := sqlmock.New()
18     if err != nil {
19         t.Fatalf("an error '%s' was not expected when opening a stub database connection", err)
20     }
21
22     mockArticles := []domain.Article{
23         {
24             ID: 1, Title: "title 1", content: "content 1",
25             Author: domain.Author{ID: 1}, UpdatedAt: time.Now(), CreatedAt: time.Now(),
26         },
27         {
28             ID: 2, Title: "title 2", content: "content 2",
29             Author: domain.Author{ID: 1}, UpdatedAt: time.Now(), CreatedAt: time.Now(),
30         },
31     }
32
33     rows := sqlmock.NewRows([]string{"id", "title", "content", "author_id", "updated_at", "created_at"}).
34         AddRow(mockArticles[0].ID, mockArticles[0].Title, mockArticles[0].Content,
35             mockArticles[0].Author.ID, mockArticles[0].UpdatedAt, mockArticles[0].CreatedAt).
36         AddRow(mockArticles[1].ID, mockArticles[1].Title, mockArticles[1].Content,
37             mockArticles[1].Author.ID, mockArticles[1].UpdatedAt, mockArticles[1].CreatedAt)
38
39     query := "SELECT id,title,content, author_id, updated_at, created_at FROM article WHERE created_at > \\? ORDER BY created_at LIMIT \\?"
40
41     mock.ExpectQuery(query).WillReturnRows(rows)
42     a := articleMySQLRepo.NewMySQLArticleRepository(db)
43     cursor := repository.EncodeCursor(mockArticles[1].CreatedAt)
44     num := int64(2)
45     list, nextCursor, err := a.Fetch(context.TODO(), cursor, num)
46     assert.NotEmpty(t, nextCursor)
47     assert.NoError(t, err)
48     assert.Len(t, list, 2)
49 }
50
run test | debug test
51 func TestGetByID(t *testing.T) {
52     db, mock, err := sqlmock.New()

```

Unit Test Usecase layer

```

article_ucase_test.go - go-clean-arch - Visual Studio Code [Administrator]
article.go M  article_handler.go M  article_ucase_test.go 2 X

article > usecase > article_ucase_test.go > ...
run test | debug test
17 func TestFetch(t *testing.T) {
18     mockArticleRepo := new(mocks.ArticleRepository)
19     mockArticle := domain.Article{
20         Title: "hello",
21         Content: "Content",
22     }
23
24     mockListArticle := make([]domain.Article, 0)
25     mockListArticle = append(mockListArticle, mockArticle)
26
27     run test | debug test
28     t.Run("success", func(t *testing.T) {
29         mockArticleRepo.On("fetch", mock.Anything, mock.AnythingOfType("string"),
30             mock.AnythingOfType("int64")).Return(mockListArticle, "next-cursor", nil).Once()
31         mockAuthor := domain.Author{
32             ID: 1,
33             Name: "Iman Tumorang",
34         }
35         mockAuthorRepo := new(mocks.AuthorRepository)
36         mockAuthorRepo.On("getByID", mock.Anything, mock.AnythingOfType("int64")).Return(mockAuthor, nil)
37         u := usecase.NewArticleUsecase(mockArticleRepo, mockAuthorRepo, time.Second*2)
38         num := int64(1)
39         cursor := "12"
40         list, nextCursor, err := u.Fetch(context.TODO(), cursor, num)
41         cursorExpected := "next-cursor"
42         assert.Equal(t, cursorExpected, nextCursor)
43         assert.NotEmpty(t, mockcursor)
44         assert.NoError(t, err)
45         assert.Len(t, list, len(mockListArticle))
46
47         mockArticleRepo.AssertExpectations(t)
48         mockAuthorRepo.AssertExpectations(t)
49     })
50
51     run test | debug test
52     t.Run("error-failed", func(t *testing.T) {
53         mockArticleRepo.On("fetch", mock.Anything, mock.AnythingOfType("string"),

```

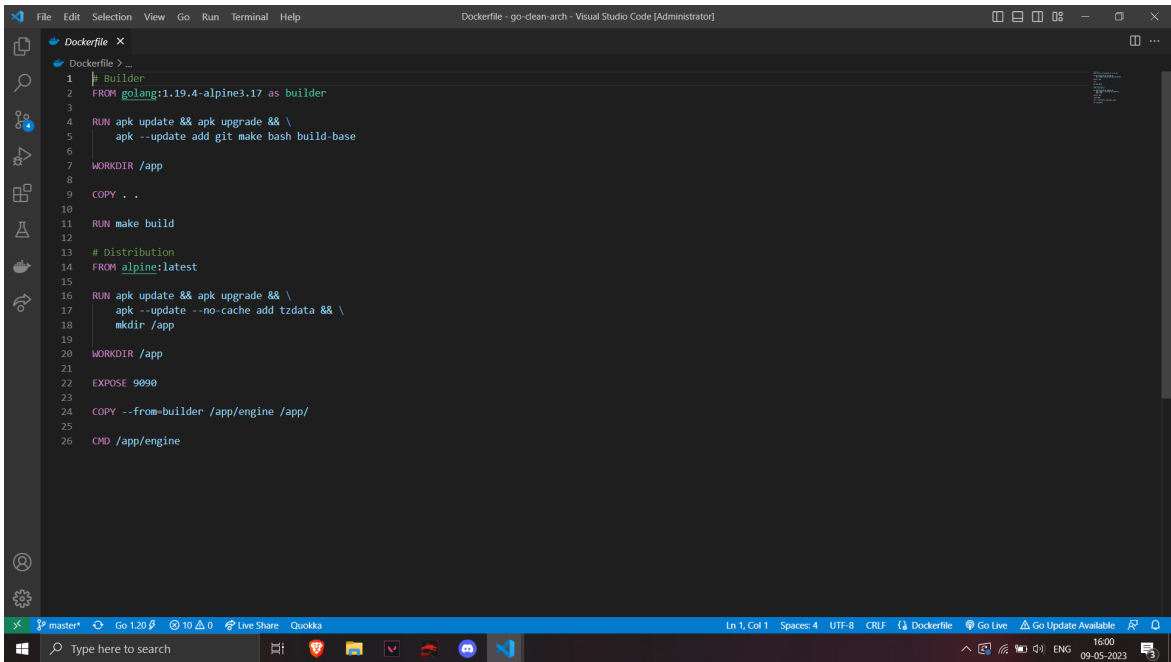
Unit test handler layer

```
21     "github.com/bxcodec/go-clean-arch/domain/mocks"
22   )
23
24   run test | debug test
25   func TestFetch(t *testing.T) {
26     var mockArticle domain.Article
27     err := faker.FakeData(&mockArticle)
28     assert.NoError(t, err)
29     mockCase := new(mocks.ArticleUsecase)
30     mockListArticle := make([]domain.Article, 0)
31     mockListArticle = append(mockListArticle, mockArticle)
32     num := 1
33     cursor := "2"
34     mockCase.On("Fetch", mock.Anything, cursor, int64(num)).Return(mockListArticle, "10", nil)
35
36     e := echo.New()
37     req, err := http.NewRequestWithContext(context.TODO(), echo.GET, "/article?num=1&cursor="+cursor, strings.NewReader(""))
38     assert.NoError(t, err)
39
40     rec := httptest.NewRecorder()
41     c := e.NewContext(req, rec)
42     handler := articleHttp.ArticleHandler{
43       AUsecase: mockCase,
44     }
45     err = handler.FetchArticle(c)
46     require.NoError(t, err)
47
48     responseCursor := rec.Header().Get("X-Cursor")
49     assert.Equal(t, "10", responseCursor)
50     assert.Equal(t, http.StatusOK, rec.Code)
51     mockCase.AssertExpectations(t)
52   }
53
54   run test | debug test
55   func TestFetchError(t *testing.T) {
56     mockCase := new(mocks.ArticleUsecase)
57     num := 1
58     cursor := "2"
```

Middlewares

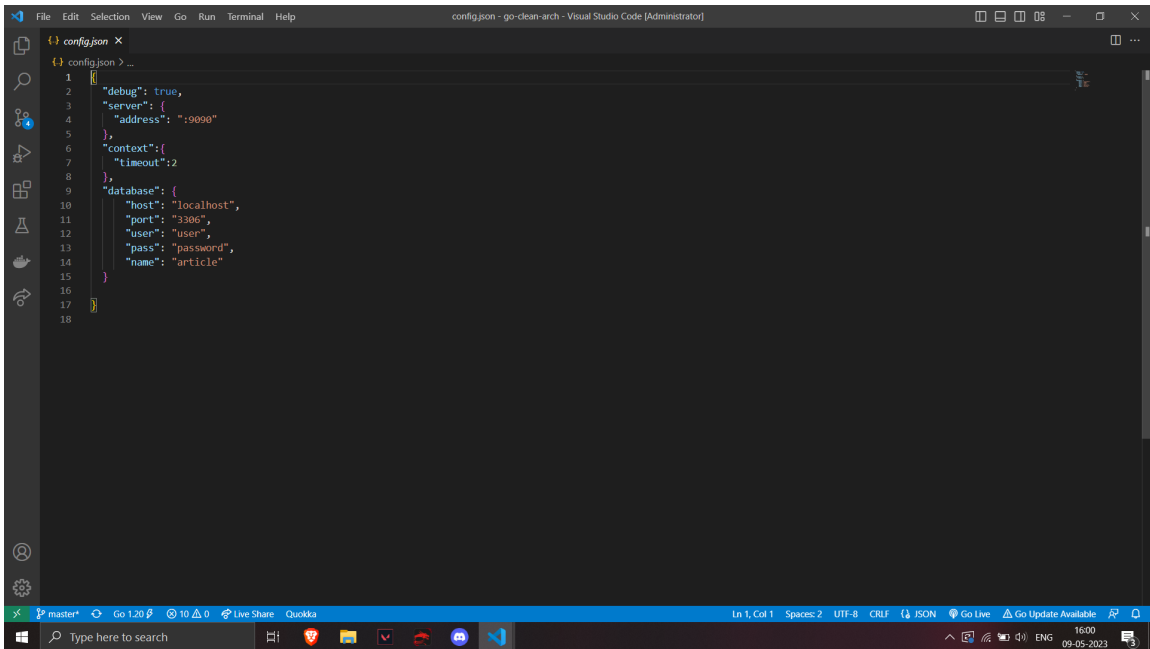
```
1 package middleware
2
3 import "github.com/labstack/echo"
4
5 // GoMiddleware represent the data-struct for middleware
6 type GoMiddleware struct {
7   // another stuff , may be needed by middleware
8 }
9
10 // CORS will handle the CORS middleware
11 func (m *GoMiddleware) CORS(next echo.HandlerFunc) echo.HandlerFunc {
12   return func(c echo.Context) error {
13     c.Response().Header().Set("Access-Control-Allow-Origin", "*")
14     return next(c)
15   }
16 }
17
18 // InitMiddleware initialize the middleware
19 func InitMiddleware() *GoMiddleware {
20   return &GoMiddleware{}
21 }
22 }
```

Dockerfile



```
1 Builder
2 FROM golang:1.19.4-alpine3.17 as builder
3
4 RUN apk update && apk upgrade && \
5     apk --update add git make bash build-base
6
7 WORKDIR /app
8
9 COPY . .
10
11 RUN make build
12
13 # Distribution
14 FROM alpine:latest
15
16 RUN apk update && apk upgrade && \
17     apk --update --no-cache add tzdata && \
18     mkdir /app
19
20 WORKDIR /app
21
22 EXPOSE 9090
23
24 COPY --from-builder /app/engine /app/
25
26 CMD /app/engine
```

Configuration File



```
1 {
2   "debug": true,
3   "server": {
4     "address": ":9090"
5   },
6   "context": {
7     "timeout": 2
8   },
9   "database": {
10    "host": "localhost",
11    "port": "3306",
12    "user": "user",
13    "pass": "password",
14    "name": "article"
15  }
16 }
17
18 }
```

Docker Compose File

```
File Edit Selection View Go Run Terminal Help compose.yaml - go-clean-arch - Visual Studio Code [Administrator]
compose.yaml
1 version: "3.7"
2 services:
3   web:
4     image: go-clean-arch
5     container_name: article_management_api
6     ports:
7       - 9090:9090
8     depends_on:
9       mysql:
10        condition: service_healthy
11     volumes:
12       - ./config.json:/app/config.json
13
14   mysql:
15     image: mysql:5.7
16     container_name: go_clean_arch_mysql
17     command: mysqld --user=root
18     volumes:
19       - ./article.sql:/docker-entrypoint-initdb.d/init.sql
20     ports:
21       - 3306:3306
22     environment:
23       - MYSQL_DATABASE=article
24       - MYSQL_USER=user
25       - MYSQL_PASSWORD=password
26       - MYSQL_ROOT_PASSWORD=root
27     healthcheck:
28       test: ["CMD", "mysqladmin", "ping", "-h", "localhost"]
29       timeout: 5s
30       retries: 10
31
```

CHAPTER 4: 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 convert the code from php to golang 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. The Kubernetes 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.

5.3 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://echo.labstack.com/guide/>
- [3] <https://gorm.io/docs/>

ORIGINALITY REPORT

9%

SIMILARITY INDEX

9%

INTERNET SOURCES

3%

PUBLICATIONS

%

STUDENT PAPERS

PRIMARY SOURCES

1	www.fedrianto.com Internet Source	2%
2	github.com Internet Source	1%
3	www.coursehero.com Internet Source	1%
4	ir.juit.ac.in:8080 Internet Source	1%
5	golang.cafe Internet Source	1%
6	Shiju Varghese. "Go Recipes", Springer Science and Business Media LLC, 2016 Publication	<1%
7	it.3hdeng.me Internet Source	<1%
8	867380699.github.io Internet Source	<1%
9	code84.com Internet Source	<1%
