

GoLang - Three Layered Architecture

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

in

Computer Science and Engineering/Information Technology

By Rashi Singh (181228)

Under the supervision of

Dr. Himanshu Jindal to



Department of Computer Science & Engineering and Information
Technology

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

Candidate's Declaration

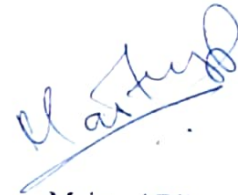
I hereby declare that the work presented in this report entitled "**GoLang - Three Layered Architecture**" in partial fulfillment of the requirements for the award of the degree of **Bachelor of Technology in Computer Science and Engineering/Information Technology** submitted in the department of Computer 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 February 2022 to July 2022 under the supervision of **Dr. Himanshu Jindal** (Assistant Professor (SG), Dept. CSE & IT).

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

Rashi Singh, 181228

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

Dr. Himanshu Jindal
Assistant Professor (SG)
Dept. CSE & IT
Technologies



Maitreyi Bilugu
Senior Lead Engineer
ZopSmart

ACKNOWLEDGEMENT

Firstly, I express my heartiest thanks and gratefulness to Almighty God for his divine blessing that makes it possible to complete the project work successfully.

I am really grateful and wish my profound indebtedness to Supervisor **Dr. Himanshu Jindal, Assistant Professor(SG)**, Department of CSE & IT, Jaypee University of Information Technology, Wakhnaghat. His endless patience, scholarly guidance, continual encouragement, constant and energetic supervision, constructive criticism, valuable advice, reading many inferior drafts, and correcting them at all stages have made it possible to complete this project.

I would like to express my heartiest gratitude to **Dr. Himanshu Jindal**, Department of CSE, for his kind help to finish my project and **ZopSmart Technologies** for providing me with the opportunity to work on this project.

I would also generously welcome each one of those individuals who have helped me straightforwardly or in a roundabout way in making this project a win. In this unique situation, I might want to thank the various staff individuals, both educating and non-instructing, which have developed their convenient help and facilitated my undertaking.

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

Rashi Singh (181228)

TABLE OF CONTENTS

1. Certificate
2. Acknowledgement
3. List of Abbreviations
4. List of Figures
5. List of Graphs
6. List of Tables
7. Abstract
8. Chapter-1 Introduction
 - 8.1. Introduction
 - 8.2. Problem Statement
 - 8.3. Objectives
 - 8.4. Methodology
 - 8.5. Organization
9. Literature Survey
10. System Development
11. Performance Analysis
12. Conclusions
 - 12.1. Conclusions
 - 12.2. Applications Contributions
13. References
14. Appendices

LIST OF ABBREVIATIONS

1. Go: GoLang
2. CSP: communicating sequential processes
3. SQL: Structured Query Language
4. DB: database
5. HTTP: hyper-text transfer protocol
6. VSCode: visual studio code
7. VCS: version control system
8. OS: operating system
9. CRUD: create, read, update, delete

LIST OF FIGURES

1. Fig.1 SQL/ Employee1 table
2. Fig.2 Connect Go to SQL
3. Fig.3 Employee Structure
4. Fig.4 Store Layer/ EmpByID function
5. Fig.5 Store Layer/ AddEmp, Employeeupdate, DelEmp functions
6. Fig.6 Service Layer/ GetByID, GetAll, Delete
7. Fig.7 Service Layer/ Create, Update
8. Fig.8 Handler Layer/ Get, Post function
9. Fig.9 Handler Layer/ Put, Delete functions
10. Fig.10 SQL/ Car table
11. Fig.11 SQL/ Engine table
12. Fig.12 Connect Go to SQL
13. Fig.13 Car Structure
14. Fig.14 Engine Structure
15. Fig.15 CRUD operations for store layer for the database CAR
16. Fig.16 CRUD operations for store layer for the database Engine
17. Fig.17 Service Layer functions
18. Fig.18 Middleware
19. Fig.19 Handler Layer Function - GetByID
20. Fig.20 Handler Layer Function - GetByBrand
21. Fig.21 Handler Layer Function - Create
22. Fig.22 Handler Layer Function - Update
23. Fig.23 Handler Layer Function - Delete
24. Fig.24 Testing Coverage
25. Fig.25 Integration Testing
26. Fig.26 Git repository

ABSTRACT

Over the past few decades we have seen a lot of programming languages come up. Each of these languages have had their own fallbacks and limitations. Therefore, GoLang was created by the Google developers in a search for a more friendly and easier language. GoLang is considered to be a robust system-level language that can be used for programming across big scale network servers and large distributed systems. It has worked as an alternative to C++ and Java. The syntax for GoLang is much more similar to C while the use of brackets, comma etc. is less which makes it similar to Python as well.

GoLang provides fast paving compilation and execution, easier to read and debug code, easy versioning, consistent language, developing with multiple languages, easier maintenance, concurrency and multithreading.

Go can be defined to be a statically typed, compiled programming language. It was designed at Google by Robert Griesemer, Rob Pike and Ken Thompson. Go is considered to be syntactically similar to C, but also has many features including memory safety, garbage collection, structural typing, and CSP-style concurrency.

GoLang has a lot of applications in the real world. Some of the open source applications written in Go include-

- Caddy, an open source HTTP/2 web server with automatic HTTPS capability
- CockroachDB, an open source, survivable, strongly consistent, scale-out SQL database
- Docker, a set of tools for deploying Linux containers
- Kubernetes container management system

Some of the other companies and sites using Go include-

- Dropbox, who migrated some of their critical components from Python to Go
- Ethereum, The *go-ethereum* implementation of the Ethereum Virtual Machine blockchain for the *Ether* cryptocurrency
- Gitlab, a web-based DevOps lifecycle tool that provides a Git-repository, wiki, issue-tracking, continuous integration, deployment pipeline features
- Google, for many projects, notably including download server dl.google.com

In the course of my training/ internship, I have been working on the basics of GoLang which includes but is not limited to packages, variables, functions, flow control statements(for, if, else, switch, defer), structs, slices, and maps, methods, interfaces and concurrency. Also, I have worked on MySQL using docker and have connected the same to GoLang. I have made use of all the concepts learnt to create a project - “Car Dealership” which makes use of three layered architecture.

CHAPTER 1: INTRODUCTION

1.1 Introduction

GoLang is considered to be a robust system-level language that can be used for programming across big scale network servers and large distributed systems. It has worked as an alternative to C++ and Java. The syntax for GoLang is much more similar to C while the use of brackets, comma etc. is less which makes it similar to Python as well.

GoLang provides fast paving compilation and execution, easier to read and debug code, easy versioning, consistent language, developing with multiple languages, easier maintenance, concurrency and multithreading.

Go can be defined to be a statically typed, compiled programming language. It was designed at Google by Robert Griesemer, Rob Pike and Ken Thompson. Go is considered to be syntactically similar to C, but also has many features including memory safety, garbage collection, structural typing, and CSP-style concurrency.

1.2 Problem Statement

Over the past few decades we have seen a lot of programming languages come up. Each of these languages have had their own fallbacks and limitations. Therefore, GoLang was created by the Google developers in a search for a more friendly and easier language. A lot of notable companies have started to switch or at least implement some part of the code base/ architecture using GoLang because of its versatile nature providing a development friendly environment.

Therefore, it is necessary to understand and implement the basics of GoLang which includes but is not limited to packages, variables, functions, flow control statements(for, if, else, switch, defer), structs, slices, and maps, methods, interfaces and concurrency. Also, I have worked on MySQL using docker and have connected the same to GoLang.

1.3 Objectives

The main aim of this project is to understand and implement the basics of GoLang which includes but is not limited to packages, variables, functions, flow control statements(for, if, else, switch, defer), structs, slices, and maps, methods, interfaces and concurrency and use all the basic concepts along with MySQL, Git etc. to create a three layered architecture for the “Car Dealership” project.

1.4 Methodology

The training/ internship for GoLang went as follows

1. Basics of GoLang (reference: GoTour)
2. MySQL
3. Unit Testing
4. Git/ Github
5. Project

1.5 Organization

The rest of the paper is organized as follows: In chapter 2 we have presented the literature survey. Chapter 3 highlights the methodology and system development of the project. It represents various computational, and experimental concepts of the project. Also, we have focused on the software and hardware platforms needed for implementation. In chapter 4 we have presented the performance analysis of the project which specifies the coverage for the unit testing of the project. Chapter 5 presents the conclusions of the project and the observations seen in the results. It also provides the applications of the project and the future scope of the same.

CHAPTER 2: LITERATURE SURVEY

Abstract and Figures

When developing software today, we still use old tools and ideas. Maybe it is time to start from scratch and try tools and languages that are more in line with how we actually want to develop software. The Go Programming Language was created at Google by a rather famous trio: Rob Pike, Ken Thompson and Robert Griesemer. Before introducing Go, the company suffered from their development process not scaling well due to slow builds, uncontrolled dependencies, hard to read code, poor documentation and so on. Go is set out to provide a solution for these issues. The purpose of this master's thesis was to review the current state of the language. This is not only a study of the language itself but an investigation of the whole software development process using Go. The study was carried out from an embedded development perspective which includes an investigation of compilers and cross-compilation. We found that Go is exciting, fun to use and fulfills what is promised in many cases. However, we think the tools need some more time to mature. Keywords: Go, golang, language review, cross-compilation, developer tools, embedded

We have referred to this paper to understand the basics of GoLang.

CHAPTER 3: SYSTEM DEVELOPMENT

Computational

All the experiments were performed on -

Laptop: Dell Latitude E7470

OS: Ubuntu 0.02.3 LTS

Applications: VSCode

VCS: Git

Experimental

The training for GoLang went as follows

A. Basics of GoLang (reference: GoTour)

- a. Packages: act as groups containing programs having similar features grouped as a single unit; programs start running in the main package; packages need to be imported.
- b. Functions: can return any number of arguments; can be variadic
- c. Variables: var statement declares a list of variables; datatype must be specified as the last variable
- d. Basic data types: bool, string, int, float complex
- e. Zero values: are the values provided to variables declared without an explicit initial value
- f. Flow control statements (for, if-else, switch, defer)
- g. Pointers: zero value is <nil>
- h. Structs: can be declared as - type Vertex struct{}
- i. Arrays: can be declared as primes:=[6]int{2,3,5,7,11,13}
- j. Slices: var s[]int = primes[1:4]
- k. Maps: key-value pairs; zero value is <nil>;
- l. Methods: can be defined on types
- m. Interfaces: a set of method signatures
- n. Goroutine: lightweight threads managed by Go runtime; helps to achieve

concurrency

- o. Channel: can send and receive values to help establish communication among goroutines

B. MySQL

MySQL is an open-source RDBMS where RDBMS stands for relational database management system. Its name was derived from "My", the name of co-founder Michael Widenius's daughter, and "SQL", the abbreviation for Structured Query Language.

What does a relational database do? It organizes data into one or more data tables. In this table the data may be related to each other and these relations structure the data.

SQL is used to create, modify and extract data from the relational database, as well as control user access to the database.

MySQL also works with an operating system to implement a relational database in a computer's storage system, manages users, allows for network access and facilitates testing database integrity and creation of backups.

Movie Management

Implemented CRUD functions over a table Employee having fields as - id, name, email, role and connected it to Go.

```
mysql> use test;
Database changed
mysql> create table employee(id int, name varchar(20), email varchar(20), role varchar(20));
Query OK, 0 rows affected (0.02 sec)

mysql> INSERT INTO employee(id, name, email, role) VALUES(1, 'Rashi', 'rashi@zopsmart.com', 'Intern');
Query OK, 1 row affected (0.01 sec)

mysql> INSERT INTO employee(id, name, email, role) VALUES(3, 'Akanksha', 'ak@zopsmart.com', 'Intern');
Query OK, 1 row affected (0.01 sec)

mysql> select * from employee;
+----+-----+-----+-----+
| id | name  | email                | role  |
+----+-----+-----+-----+
|  1 | Rashi | rashi@zopsmart.com  | Intern |
|  3 | Akanksha | ak@zopsmart.com    | Intern |
+----+-----+-----+-----+
2 rows in set (0.00 sec)
```

Fig.1 SQL/ Employee1 table

```

package driver

import (
    "database/sql"
    "fmt"
)

func ConnectToSQL() *sql.DB {
    db, err := sql.Open("mysql", "root:password@tcp(127.0.0.1:3306)/test")
    if err != nil {
        panic(err.Error())
    } else {
        fmt.Println("Connected")
    }
    return db
}

```

Fig.2 Connect Go to SQL

The CRUD functions were first created for the store layer.

```

type Employee struct{
    Id int `json:"Id"`
    Name string `json:"Name"`
    Email string `json:"Email"`
    Role string `json:"Role`
}

```

Fig.3 Employee Structure

```

//readById
func EmpByID(Id int, db *sql.DB)(*Employee, error){
    var emp Employee
    log.Print(Id)

    if(Id<0){
        return nil, sql.ErrNoRows
    }

    rows, err:=db.Query("Select * FROM employee1 WHERE Id = ?", Id)
    if(err!=nil){
        log.Print(err)
        return &emp, err
    }

    defer rows.Close()

    for rows.Next(){
        log.Print("1")

        if err:=rows.Scan(&emp.Id, &emp.Name, &emp.Email, &emp.Role); err!=nil{
            return &emp, err
        }

    }

    log.Print(emp)

    return &emp, nil
}

```

Fig.4 Store Layer/ EmpByID function

```

//insert
func AddEmp(emp Employee, db *sql.DB)(Employee, error){

    _, err:=db.Exec("INSERT INTO employee1 (Id, Name, Email, Role) VALUES (?,?,?,?)",emp.Id, emp.Name, emp.Email, emp.Role)
    if err!=nil{
        return emp, nil
    }
    return emp, nil
}

func Employeeupdate(emp *Employee, db *sql.DB) error {
    _, err := db.Exec("UPDATE employee1 SET Name = ?, Email=?, Role=? WHERE ID = ?", emp.Name, emp.Email, emp.Role, emp.Id)
    if err != nil {
        return errors.New("update failed")
    }
    return nil
}

//deleteById
func DelEmp(cond1 int, db *sql.DB)(error){

    if(cond1<0){
        return sql.ErrNoRows
    }
    rows, err:=db.Query("delete from employee1 Where Id = ?", cond1)
    if err!=nil{
        return err
    }

    defer rows.Close()

    return nil
}

```

Fig.5 Store Layer/ AddEmp, Employeeupdate, DelEmp functions


```

func (se *service) GetByID(id int) (*models.Movie, error) {

    if id <= 0 {
        return nil, errors.New("error invalid id")
    }
    movieObj, err := se.store.GetByID(id)
    if err != nil {
        return nil, err
    }
    return movieObj, nil
}

func (se *service) GetAll() ([]*models.Movie, error) {

    movieObj, err := se.store.GetAll()
    if err != nil {
        return nil, err
    }
    return movieObj, nil
}

func (se *service) Delete(id int) error {
    if id <= 0 {
        return errors.New("error invalid id")
    }
    _, err := se.store.GetByID(id)
    if err != nil {
        return err
    }
    err = se.store.Delete(id)
    if err != nil {
        return err
    }
    return nil
}

```

Fig 6. Service Layer/ GetByID, GetAll, Delete

```

func (se *service) Create(movieObj *models.Movie) (*models.Movie, error) {

    if movieObj.Name == "" {
        return nil, errors.New("error invalid name")
    }
    if movieObj.Plot == "" {
        return nil, errors.New("error invalid plot")
    }
    movieObjs, err := se.store.Create(movieObj)
    if err != nil {
        return nil, err
    }
    return movieObjs, nil
}

func (se *service) Update(movieObj *models.Movie) (*models.Movie, error) {
    if movieObj.Id < 0 {
        return nil, errors.New("error invalid id")
    }
    if movieObj.Name == "" {
        return nil, errors.New("error invalid name")
    }
    if movieObj.Plot == "" {
        return nil, errors.New("error invalid plot")
    }
    movieObjs, err := se.store.Update(movieObj)
    if err != nil {
        return nil, err
    }
    return movieObjs, nil
}

```

Fig 7. Service Layer/ Create, Update

```

//get
func EmpByID(w http.ResponseWriter, r *http.Request){

    db:=driver.ConnectToSQL()
    defer db.Close()

    params:=mux.Vars(r)
    id, _:=strconv.Atoi(params["id"])
    log.Print(id)
    use, err:=store.EmpByID(id, db)
    if(err!=nil){
        w.Write([]byte("Employee doesn't exist"))
    }else{
        res, _:=json.Marshal(use)
        w.Write(res)
    }
}

//post
func AddEmp(w http.ResponseWriter, r *http.Request) {

    db:=driver.ConnectToSQL()
    defer db.Close()

    var emp store.Employee
    emp.Id, _=strconv.Atoi(r.PostFormValue("Id"))
    emp.Name=r.PostFormValue("Name")
    emp.Email=r.PostFormValue("Email")
    emp.Role=r.PostFormValue("Role")

    _,err:=store.AddEmp(emp, db)
    if(err!=nil){
        w.Write([]byte("Failed to add new emp"))
    }else{
        w.Write([]byte("Successfully added"))
    }
}

```

Fig.8 Handler Layer/ Get, Post function

```

func Employeeupdate(w http.ResponseWriter, r *http.Request) {

    db:=driver.ConnectToSQL()
    defer db.Close()

    params := mux.Vars(r)
    id, _:=strconv.Atoi(params["id"])

    use, err:=store.EmpByID(id, db)
    use.Name=r.PostFormValue("Name")
    use.Email=r.PostFormValue("Email")
    use.Role=r.PostFormValue("Role")

    err=store.Employeeupdate(use, db)
    if(err!=nil){
        w.Write([]byte("Failed to update emp"))
    }else{
        w.Write([]byte("Successfully updated"))
    }
}

//delete
func DelEmp(w http.ResponseWriter, r *http.Request) {

    db:=driver.ConnectToSQL()
    defer db.Close()

    params := mux.Vars(r)
    id,_:=strconv.Atoi(params["id"])
    err:=store.DelEmp(id, db)
    if err != nil {
        w.Write([]byte("Failed to delete emp"))
    }else{
        w.Write([]byte("Successfully deleted"))
    }
}

```

Fig.9 Handler Layer/ Put, Delete functions

Car Dealership

The Car-Dealership project consists of two tables in the database for car and engine. Below are their descriptions in MySQL.

```
[mysql> describe CAR;
+-----+-----+-----+-----+-----+-----+
| Field      | Type          | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| ID         | varchar(36)   | NO   | PRI | NULL    |       |
| Name       | varchar(25)   | YES  |     | NULL    |       |
| Year       | int           | YES  |     | NULL    |       |
| Brand      | varchar(25)   | YES  |     | NULL    |       |
| FuelType   | varchar(25)   | YES  |     | NULL    |       |
| Engine     | varchar(36)   | YES  |     | NULL    |       |
+-----+-----+-----+-----+-----+-----+
6 rows in set (0.00 sec)
```

Fig.10 SQL/ Car table

```
[mysql> describe ENGINE;
+-----+-----+-----+-----+-----+-----+
| Field          | Type          | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| ID             | varchar(36)   | NO   | PRI | NULL    |       |
| Displacement   | int           | YES  |     | NULL    |       |
| NoCYLINDERS    | int           | YES  |     | NULL    |       |
| ENGRANGE       | int           | YES  |     | NULL    |       |
+-----+-----+-----+-----+-----+-----+
4 rows in set (0.01 sec)
```

Fig. 11 SQL/ Engine table

```
db.go x
1 package main
2
3 import (
4     "database/sql"
5     "fmt"
6
7     _ "github.com/go-sql-driver/mysql"
8 )
9
10 func dbConnect() (db *sql.DB) {
11     db, err := sql.Open("mysql", "root:password@tcp(0.0.0.0:3306)/car_dealership")
12
13     if err != nil { err } else {
14         fmt.Println(a...: "Database Connected Successfully")
15     }
16 }
17
18
19 return
20 }
```

Fig.12 Connect Go to SQL

The CRUD functions were first created for the store layer.

```
type Car struct {
    ID        uuid.UUID `json:"ID"`
    Name      string    `json:"Name"`
    Year      int       `json:"Year"`
    Brand     BrandType `json:"Brand"`
    FuelType  FuelType  `json:"FuelType"`
    Engine    Engine    `json:"Engine"`
}
```

Fig.13 Car Structure

```

package models

import "github.com/google/uuid"

type Engine struct {
    ID          uuid.UUID `json:"EngineID"`
    Displacement int       `json:"Displacement"`
    NoCYLINDERS int       `json:"NoCYLINDERS"`
    ENGRANGE    int       `json:"ENGRANGE"`
}

```

Fig.14 Engine Structure

```

// CreateCar for inserting the car's data into db
func (c storeCar) CreateCar(ctx context.Context, car *models.Car) error {
    id := uuid.New()
    _, err := c.db.ExecContext(ctx, query: "INSERT INTO CAR VALUES (?, ?, ?, ?, ?, ?)", id.String(), car.Name, car.Year,
        car.Brand, car.FuelType, car.Engine.ID.String())
    car.ID = id

    if err != nil : errors.New("error in inserting row") ↗

    return nil
}

// Update for updating the car's data into db
func (c storeCar) Update(ctx context.Context, car *models.Car) error {
    _, err := c.db.ExecContext(ctx, query: "UPDATE CAR SET NAME = ?, YEAR = ?, BRAND = ?, FUELTYPE = ? WHERE ID = ?",
        car.Name, car.Year, car.Brand, car.FuelType, car.ID.String())
    if err != nil : errors.New("error in updating") ↗

    return nil
}

// Delete for deleting the car in the db
func (c storeCar) Delete(ctx context.Context, id uuid.UUID) error {
    _, err := c.db.ExecContext(ctx, query: "DELETE FROM CAR WHERE ID = ?", id.String())
    if err != nil : errors.New("error in deleting") ↗

    return nil
}

// GetByID for retrieving the car's data from db by carId
func (c storeCar) GetByID(ctx context.Context, id uuid.UUID) (models.Car, error) {
    rows := c.db.QueryRowContext(ctx, query: "SELECT * FROM CAR WHERE ID = ?", id.String())

    var car models.Car
    err := rows.Scan(&car.ID, &car.Name, &car.Year, &car.Brand, &car.FuelType, &car.Engine.ID)

    if err != nil : car, errors.New("invalid id") ↗

    return car, nil
}

```

```

// GetByBrand for retrieving the cars from db according to specific brand
func (c storeCar) GetByBrand(ctx context.Context, brand string) ([]models.Car, error) {
    rows, err := c.db.QueryContext(ctx, query: "SELECT * FROM CAR WHERE BRAND = ?", brand)
    if err != nil : []models.Car{}, errors.New("error in fetching brand") ↗

    var cars []models.Car

    for rows.Next() {
        var car models.Car
        err = rows.Scan(&car.ID, &car.Name, &car.Year, &car.Brand, &car.FuelType, &car.Engine.ID)

        if err != nil : cars, errors.New("error in reading row") ↗

        cars = append(cars, car)
    }

    err = rows.Err()
    if err != nil : []models.Car{}, errors.New("rows error") ↗

    defer rows.Close()

    return cars, nil
}

```

Fig.15 CRUD operations for store layer for the database CAR


```

// GetEngine for retrieving engine's details from db
func (e storeEngine) GetEngine(ctx context.Context, id uuid.UUID) (models.Engine, error) {
    rows := e.db.QueryRowContext(ctx, query: "SELECT * FROM ENGINE WHERE ID = ?", id.String())
    eng := models.Engine{}
    err := rows.Scan(&eng.ID, &eng.Displacement, &eng.NoCYLINDERS, &eng.ENGRANGE)

    if err != nil : eng, errors.New("invalid ID") ↗

    return eng, nil
}

// CreateEngine for inserting engine's details into db
func (e storeEngine) CreateEngine(ctx context.Context, eng *models.Engine) (uuid.UUID, error) {
    id := uuid.New()
    _, err := e.db.ExecContext(ctx, query: "INSERT INTO ENGINE VALUES (?, ?, ?, ?)", id.String(), eng.Displacement,
        eng.NoCYLINDERS, eng.ENGRANGE)
    eng.ID = id

    if err != nil : uuid.Nil, errors.New("query error") ↗

    return id, nil
}

// UpdateEngine for updating engine's details into db
func (e storeEngine) UpdateEngine(ctx context.Context, eng *models.Engine) error {
    _, err := e.db.ExecContext(ctx, query: "UPDATE ENGINE SET DISPLACEMENT = ?, NoCYLINDERS = ?, ENGRANGE = ? WHERE ID = ?",
        eng.Displacement, eng.NoCYLINDERS, eng.ENGRANGE, eng.ID.String())

    if err != nil : errors.New("query error") ↗

    return nil
}

// DeleteEngine for deleting engine from db by specific engine id
func (e storeEngine) DeleteEngine(ctx context.Context, id uuid.UUID) error {
    _, err := e.db.ExecContext(ctx, query: "DELETE FROM ENGINE WHERE ID = ?", id.String())

    if err != nil : errors.New("query error") ↗

    return nil
}

```

Fig.16 CRUD operations for store layer for the database Engine

To add an extra layer of security, after the implementation of store layer functions we implemented business logic.

```

// isValidYear check for valid year
func isValidYear(y int) bool {
    if y < 1980 || y > time.Now().Year() {
        return false
    }

    return true
}

// isValidElectricEngine check for valid Electric engine
func isValidElectricEngine(eng *models.Engine) bool {
    if eng.ENGRANGE == 0 || eng.Displacement != 0 || eng.NoCYLINDERS != 0 {
        return false
    }

    return true
}

// isValidPetrolEngine check for valid Petrol Engine
func isValidPetrolEngine(eng *models.Engine) bool {
    if eng.ENGRANGE != 0 || eng.NoCYLINDERS == 0 || eng.Displacement == 0 {
        return false
    }

    return true
}

// isValidEngine check for valid Engine
func isValidEngine(car *models.Car) bool {
    if car.FuelType == models.PETROL || car.FuelType == models.DIESEL {
        return isValidPetrolEngine(&car.Engine)
    }

    return isValidElectricEngine(&car.Engine)
}

```

Fig.17 Service Layer functions

We also implemented middleware to provide authorization.

```
// AuthenticationMiddleware check whether the request is authorized or not
func AuthenticationMiddleware(handler http.Handler) http.Handler {
    return http.HandlerFunc(func(w http.ResponseWriter, req *http.Request) {
        val := req.Header.Get(key: "Authorization")
        if val != "123" {
            http.Error(w, error: "Unauthorized Access", http.StatusUnauthorized)
            return
        }
        handler.ServeHTTP(w, req)
    })
}
```

Fig.18 Middleware

After the above implementation the CRUD functions were executed using the handler layer i.e. gorilla.mux.

```
// GetByID return car's details in the response
func (c handler) GetByID(w http.ResponseWriter, req *http.Request) {
    w.Header().Set(key: "Content-Type", value: "application/json")

    ctx := req.Context()
    Vars := mux.Vars(req)
    carID := Vars["id"]

    id, err := uuid.Parse(carID)
    if err != nil {
        log.Println(err)
        w.WriteHeader(http.StatusBadRequest)

        return
    }

    res, err := c.service.GetByID(ctx, id)
    if err != nil {
        w.WriteHeader(http.StatusBadRequest)
        _, _ = w.Write([]byte("bad request"))

        return
    }

    data, err := json.Marshal(res)
    if err != nil {
        _, _ = w.Write([]byte("Marshal error"))

        return
    }

    w.WriteHeader(http.StatusAccepted)
    _, _ = w.Write(data)
}
```

Fig.19 Handler Layer Function - GetByID

```

// GetByBrand return the all the Car's data of specific brand in the response
func (c handler) GetByBrand(w http.ResponseWriter, req *http.Request) {
    ctx := req.Context()
    brand := req.URL.Query().Get( key: "brand")
    engine := req.URL.Query().Get( key: "engine")

    res, err := c.service.GetByBrand(ctx, brand, engine)
    if err != nil {
        w.WriteHeader(http.StatusBadRequest)
        _, _ = w.Write([]byte("bad requests"))

        return
    }

    data, err := json.Marshal(res)

    if err != nil {
        w.WriteHeader(http.StatusBadRequest)
        _, _ = w.Write([]byte("marshal error"))

        return
    }

    w.WriteHeader(http.StatusAccepted)
    _, _ = w.Write(data)
}

```

Fig.20 Handler Layer Function - GetByBrand

```

// Create unmarshal the body of the request and pass it to the service layer
func (c handler) Create(w http.ResponseWriter, req *http.Request) {
    ctx := req.Context()

    body, err := io.ReadAll(req.Body)

    if err != nil {
        w.WriteHeader(http.StatusBadRequest)
        fmt.Println(err)
    }

    var car models.Car

    err = json.Unmarshal(body, &car)
    if err != nil {
        w.WriteHeader(http.StatusBadRequest)
        _, _ = w.Write([]byte("Invalid Body"))

        return
    }

    err = c.service.Create(ctx, &car)
    if err != nil {
        w.WriteHeader(http.StatusBadRequest)
        _, _ = w.Write([]byte(err.Error()))

        return
    }

    w.WriteHeader(http.StatusCreated)

    data := fmt.Sprintf("Car ID #{car.ID}")
    _, _ = w.Write([]byte(data))
}

```

Fig.21 Handler Layer Function - Create

```

// Update unmarshal the body of the request and pass it to the service layer
func (c handler) Update(w http.ResponseWriter, req *http.Request) {
    ctx := req.Context()
    Vars := mux.Vars(req)
    carID := Vars["id"]

    body, err := io.ReadAll(req.Body)
    if err != nil {
        w.WriteHeader(http.StatusBadRequest)
        _, _ = w.Write([]byte("bad request"))

        return
    }

    var msg models.Car

    err = json.Unmarshal(body, &msg)
    if err != nil {
        w.WriteHeader(http.StatusBadRequest)
        _, _ = w.Write([]byte("Unmarshal error"))

        return
    }

    msg.ID, err = uuid.Parse(carID)
    if err != nil {
        log.Println(err)
        w.WriteHeader(http.StatusBadRequest)

        return
    }

    err = c.service.Update(ctx, &msg)
    if err != nil {
        w.WriteHeader(http.StatusBadRequest)
        _, _ = w.Write([]byte("bad request"))

        return
    }

    w.WriteHeader(http.StatusAccepted)
    _, _ = w.Write([]byte("updated successfully"))
}

```

Fig.22 Handler Layer Function - Update

```

// Delete extracts the id from path and pass it to service layer
func (c handler) Delete(w http.ResponseWriter, req *http.Request) {
    ctx := req.Context()
    Vars := mux.Vars(req)
    carID := Vars["id"]

    id, err := uuid.Parse(carID)
    if err != nil {
        w.WriteHeader(http.StatusBadRequest)

        return
    }

    err = c.service.Delete(ctx, id)
    if err != nil {
        w.WriteHeader(http.StatusBadRequest)
        _, _ = w.Write([]byte("bad requests"))

        return
    }

    w.WriteHeader(http.StatusAccepted)
    _, _ = w.Write([]byte("car deleted"))
}

```

Fig.23 Handler Layer Function - Delete

CHAPTER 4: PERFORMANCE ANALYSIS

Movie Management

```
raramuri@Raramuri:~/repos/go/src/golangprog/rest_api-mysql/CRUD/store$ go tool cover -func coverage.txt
golangprog/rest_api-mysql/CRUD/store/store_crud.go:25: ExpByID      92.9%
golangprog/rest_api-mysql/CRUD/store/store_crud.go:56: AddExp      100.0%
golangprog/rest_api-mysql/CRUD/store/store_crud.go:65: Employeeupdate 100.0%
golangprog/rest_api-mysql/CRUD/store/store_crud.go:74: DelExp      100.0%
total:                               (statements) 96.6%
```

Fig.24 Testing Coverage

Car Dealership

A. Integration Testing: It is done to test different units and modules as a combined entity. Performed Integration Testing over the CRUD functions across all the three layers and was able to achieve a coverage of .

```
✓ Tests passed: 1 of 1 test - 1 sec 211 ms
<3 go setup calls>
/Users/zoo5245/go/go1.17/bin/go tool test2json -t /private/var/folders/hb/x4rt9x7s5ldgfb1881sstspc0000gp/T/GoLand/___Test_Main_in_github_com_zosmart_carDealership3Layer.test -test.
=== RUN   Test_Main
Database Connected Successfully
--- PASS: Test_Main (0.05s)
PASS
Process finished with the exit code 0
```

Fig.25 Integration Testing

B. Unit Testing: It is done to test smallest testable units individually. Coverage obtained across the three layers are as follows -
Store Layer: 100%
Service Layer: 100%
Handler Layer: 92.4%
Middleware: 100%

B. Git/ Github:

Git is a version control system and is defined as a software for tracking changes in any set of files, usually used for coordinating work among programmers collaboratively developing source code during software development.

Its goals include speed, data integrity, and support for distributed, non-linear workflows.

Learned the basics of Git and pushed the CRUD functions file and testing file to github.

The screenshot shows a GitHub repository interface. At the top, there are navigation elements: a dropdown menu for the 'main' branch, a link to '1 branch', and a link to '0 tags'. To the right are buttons for 'Go to file', 'Add file', and 'Code'. Below this is a commit history table with the following entries:

Commit Hash	Author	Time
c90f6a5	now	2 commits
carDealership3Layer	Initial commit	now
README.md	Initial commit	4 minutes ago

Below the table, the content of the 'README.md' file is displayed. It features a heading 'Major-Project' and a paragraph: 'This repo contains the Car-Dealership project'.

Fig.26 Git repository

CHAPTER 5: CONCLUSION

5.1 Conclusions

The main aim of the training was to be able to understand and implement the concepts of GoLang, MySQL, Unit Testing, Integration Testing, implementation of middleware, being able to create a layered architecture and Git/ Github which has been achieved in the past three weeks.

5.2 Applications Contributions

GoLang has a lot of applications in the real world. Some of the open source applications written in Go include-

- Caddy, an open source HTTP/2 web server with automatic HTTPS capability
- CockroachDB, an open source, survivable, strongly consistent, scale-out SQL database
- Docker, a set of tools for deploying Linux containers
- Kubernetes container management system

Some of the other companies and sites using Go include-

- Dropbox, who migrated some of their critical components from Python to Go
- Ethereum, The *go-ethereum* implementation of the Ethereum Virtual Machine blockchain for the *Ether* cryptocurrency
- Gitlab, a web-based DevOps lifecycle tool that provides a Git-repository, wiki, issue-tracking, continuous integration, deployment pipeline features
- Google, for many projects, notably including download server dl.google.com

REFERENCES

1. Westrup, E., & Pettersson, F. (2014). Using the Go Programming Language in Practice. *Department of Computer Science, Faculty of Engineering LTH.*
2. <https://go.dev/tour/welcome/1>
3. <https://en.wikipedia.org/wiki/MySQL>

ORIGINALITY REPORT

20%

SIMILARITY INDEX

19%

INTERNET SOURCES

4%

PUBLICATIONS

15%

STUDENT PAPERS

PRIMARY SOURCES

1	en.wikipedia.org Internet Source	9%
2	wikimili.com Internet Source	4%
3	Submitted to VIT University Student Paper	4%
4	Submitted to University of Greenwich Student Paper	1%
5	pastebin.com Internet Source	1%
6	www.ncbi.nlm.nih.gov Internet Source	1%

Exclude quotes On

Exclude bibliography On

Exclude matches < 14 words