package main

import (

"net/http" //This provides HTTP client and server implementations

"database/sql" //This provides a generic interface around SQL (or SQL-like) databases

"fmt" // This implements formatted I/O with functions

/\*

Mysql dependency - use this command to install mysql dependency for go ---> go get -u github.com/go-sql-driver/mysql

\*/

\_ "github.com/go-sql-driver/mysql"

/\*

Use this command to implement a Library which uses a request router and dispatcher for matching incoming requests to their respective handler---> go get -u github.com/gorilla/mux

\*/

"github.com/gorilla/mux"

"io/ioutil"

"encoding/json"

/\*

Package handlers is a collection of handlers (aka "HTTP middleware")

for use with Go's net/http package (or any framework supporting http.Handler),including

use this command ---> go get github.com/gorilla/handlers

\*/

"github.com/gorilla/handlers"

)

//Customer Model

type Customer struct {

ID string `json:"id"`

Name string `json:"name"`

Address string `json:"address"`

}

//Connecting to mysql

func connect() \*sql.DB {

db, err := sql.Open("mysql", "root:1234@/Crud\_DB")

if err != nil {

fmt.Println("Error! Getting connection...")

}

return db

}

// for getting All Customer

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

w.Header().Set("Content-Type", "application/json")

var posts []Customer //-----> get Customer data in Array

db := connect() // ----> Connecting Mysql Database

result, err := db.Query("SELECT \* from customer") //----> Query for getting customers in database !

if err != nil { // ----> To see if statement has an Error

panic(err.Error())

}

defer result.Close() //-----> To close the statement

for result.Next() { --->Add to the for loop for get data

var post Customer

err := result.Scan(&post.ID, &post.Name, &post.Address)

if err != nil { ----> To see if the statement has an Error

panic(err.Error())

}

posts = append(posts, post)

}

fmt.Print(posts)

json.NewEncoder(w).Encode(posts)

}

// to specify the customer by customerID

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

w.Header().Set("Content-Type", "application/json")

params := mux.Vars(r) //-----> variable used for matching mux

db := connect() // ----> Connecting Mysql Database

result, err := db.Query("SELECT \* FROM customer WHERE id = ?", params["id"]) //----> Query for search customer from CustomerID

if err != nil { --> To see if the statement has an Error

panic(err.Error())

}

defer result.Close() //----->To close the statement

var post Customer

for result.Next() { //---->Add to the for loop for get data

err := result.Scan(&post.ID, &post.Name, &post.Address)

if err != nil { // ----> To see if Statement has an Error

panic(err.Error())

}

}

json.NewEncoder(w).Encode(post)

}

//To save Customer

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

db := connect() // ----> Connecting Mysql Database

stmt, err := db.Prepare("INSERT INTO customer VALUES(?,?,?)") //----> Query to insert Customers to the database !

if err != nil { // ----> To see if statement has an Error

panic(err.Error())

}

body, err := ioutil.ReadAll(r.Body)

if err != nil { // ----> To see if statement has an Error

panic(err.Error())

}

keyVal := make(map[string]string) // ----> for mapping with Required Feilds !

json.Unmarshal(body, &keyVal)

id := keyVal["id"]

name := keyVal["name"]

address := keyVal["address"]

fmt.Print(id + " " + name + " " + address) // -----> check for data print in console

\_, err = stmt.Exec(id, name, address)

if err != nil {

panic(err.Error())

}

respondwithJSON(w, http.StatusCreated,

map[string]string{"message": "created successfully"}) // ----> response massage

defer db.Close() //----> disconnect Connection !

}

// For Update Customer

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

params := mux.Vars(r) //-----> variable ued for matching mux

db := connect() // ----> connecting my sql database

stmt, err := db.Prepare("UPDATE customer SET name = ? , address = ? WHERE id = ?") //----> Query to Update a Customer in database !

if err != nil { //------> To see if statement is an Error !

panic(err.Error())

}

body, err := ioutil.ReadAll(r.Body)

if err != nil {

panic(err.Error())

}

keyVal := make(map[string]string) // ----->for mapping with Required Fields !

json.Unmarshal(body, &keyVal)

id2 := keyVal["id"]

newTitle := keyVal["name"]

address := keyVal["address"]

fmt.Print(id2 + " ")

\_, err = stmt.Exec(newTitle, address, params["id"]) //----> check for data print in console

if err != nil {

panic(err.Error())

}

respondwithJSON(w, http.StatusCreated, map[string]string{"message": "update successfully"}) // ----> response message

defer db.Close() //----> disconnect Connection !

}

// To delete a Customer

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

params := mux.Vars(r)

db := connect() // ----> connecting my sql database

stmt, err := db.Prepare("DELETE FROM customer WHERE id = ?") //----> Query for delete Customer in databse !

if err != nil {//------> To see if statusment is an Error !

panic(err.Error())

}

\_, err = stmt.Exec(params["id"]) //---->for mapping with Required ID !

if err != nil {

panic(err.Error())

}

respondwithJSON(w, http.StatusCreated, map[string]string{"message": "delete successfully"}) // ----->response message

defer db.Close() //----> disconnect Connection !

}

// this is the Main Method

func main() {

router := mux.NewRouter()

router.HandleFunc("/customer/{id}", updateCustomer).Methods("PUT")

router.HandleFunc("/customer", addCustomer).Methods("POST")

router.HandleFunc("/customer/{id}", deleteCustomer).Methods("DELETE")

router.HandleFunc("/customer/{id}", searchCustomer).Methods("GET")

router.HandleFunc("/customer", readAllCustomer).Methods("GET")

cors := handlers.AllowedMethods([]string{"\*", "PUT", "POST", "GET", "DELETE"})

http.ListenAndServe(":8000", handlers.CORS(cors)(router))

}

//Response with JSON

func respondwithJSON(w http.ResponseWriter, code int, payload interface{}) {

response, \_ := json.Marshal(payload)

fmt.Println(payload)

w.Header().Set("Content-Type", "application/json")

w.WriteHeader(code)

w.Write(response)

}

**Explanation**

In this example, we are implementing a basic Golang crud application by using Goland as a server side script and MySQL as a database.Here, we are implementing a Rest API in Golang that will perform add,view, update and delete operations.We are using gorilla/mux package to create a rest APIs in this application.