|  |  |
| --- | --- |
| **Gerb-BMSTU_01** | **Министерство науки и высшего образования Российской Федерации**  **Федеральное государственное бюджетное образовательное учреждение**  **высшего образования**  **«Московский государственный технический университет**  **имени Н.Э. Баумана**  **(национальный исследовательский университет)»**  **(МГТУ им. Н.Э. Баумана)** |

ФАКУЛЬТЕТ **Информатика и системы управления**

КАФЕДРА **Компьютерные системы и сети (ИУ6)**

НАПРАВЛЕНИЕ ПОДГОТОВКИ **09.03.01 Информатика и вычислительная техника**

**Отчет**

**по лабораторной работе № 9**

**Название: Back-End разработка с использованием фреймворка Echo**

**Дисциплина: Языки интернет программирования**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Студент | ИУ6-33Б |  |  | Д.А. Лазутин |
|  | (Группа) |  | (Подпись, дата) | (И.О. Фамилия) |
|  |  |  |  |  |
| Преподаватель |  |  |  | В.Д. Шульман |
|  |  |  | (Подпись, дата) | (И.О. Фамилия) |

Москва, 2024

Цель работы — получение первичных навыков использования веб-фрейворков в BackEnd-разрабокте на Golang

Задание. Перекопировать код сервисов, полученный в ходе выполнения 8-й лабораторной работы, в соответствующие поддиректории в директории cmd Доработать сервисы таким образом, чтобы роутинг, обработка запросов, парсинг json, обработка ошибок и логирование осуществлялись на базе фреймворка Echo

count.go

package main

import (

"database/sql"

"fmt"

"log"

"net/http"

"strconv"

"github.com/labstack/echo/v4"

"github.com/labstack/echo/v4/middleware"

\_ "github.com/lib/pq"

)

const (

host = "localhost"

port = 5432

user = "postgres"

password = "postgres"

dbname = "sandbox"

)

type DatabaseProvider struct {

db \*sql.DB

}

type Counter struct {

ID int `json:"id"`

Value int `json:"value"`

}

func (dp \*DatabaseProvider) GetCounter() (\*Counter, error) {

query := "SELECT id, value FROM counter LIMIT 1"

row := dp.db.QueryRow(query)

var counter Counter

err := row.Scan(&counter.ID, &counter.Value)

if err == sql.ErrNoRows {

return nil, nil // Счетчик не найден

} else if err != nil {

return nil, err

}

return &counter, nil

}

func (dp \*DatabaseProvider) IncreaseCounter(value int) error {

query := "UPDATE counter SET value = value + $1 WHERE id = 1"

\_, err := dp.db.Exec(query, value)

return err

}

func (dp \*DatabaseProvider) initializeCounter() error {

var count Counter

query := "SELECT id FROM counter LIMIT 1"

err := dp.db.QueryRow(query).Scan(&count.ID)

if err == sql.ErrNoRows {

// Счетчик не найден, добавляем начальное значение

insertQuery := "INSERT INTO counter (value) VALUES ($1)"

\_, err := dp.db.Exec(insertQuery, 0)

if err != nil {

return err

}

}

return nil

}

func main() {

// Подключение к PostgreSQL

psqlInfo := fmt.Sprintf("host=%s port=%d user=%s password=%s dbname=%s sslmode=disable",

host, port, user, password, dbname)

db, err := sql.Open("postgres", psqlInfo)

if err != nil {

log.Fatal(err)

}

defer db.Close()

// Проверяем соединение

if err := db.Ping(); err != nil {

log.Fatal(err)

}

fmt.Println("Successfully connected to the database!")

// Инициализация провайдера БД

dbProvider := &DatabaseProvider{db: db}

// Инициализация счетчика, если он отсутствует

if err := dbProvider.initializeCounter(); err != nil {

log.Fatal(err)

}

// Инициализация Echo

e := echo.New()

// Middleware

e.Use(middleware.Logger())

e.Use(middleware.Recover())

// Роуты

e.GET("/count", func(c echo.Context) error {

counter, err := dbProvider.GetCounter()

if err != nil {

return echo.NewHTTPError(http.StatusInternalServerError, err.Error())

}

if counter == nil {

return echo.NewHTTPError(http.StatusNotFound, "Counter not found")

}

return c.JSON(http.StatusOK, counter)

})

e.POST("/count", func(c echo.Context) error {

countStr := c.FormValue("count")

if countStr == "" {

return echo.NewHTTPError(http.StatusBadRequest, "Parameter 'count' is required")

}

count, err := strconv.Atoi(countStr)

if err != nil {

return echo.NewHTTPError(http.StatusBadRequest, "Parameter 'count' must be a number")

}

if err := dbProvider.IncreaseCounter(count); err != nil {

return echo.NewHTTPError(http.StatusInternalServerError, err.Error())

}

return c.String(http.StatusOK, fmt.Sprintf("Counter increased by %d", count))

})

// Запуск сервера

e.Logger.Fatal(e.Start(":3333"))

}

query.go

package main

import (

"database/sql"

"fmt"

"log"

"net/http"

"github.com/labstack/echo/v4"

"github.com/labstack/echo/v4/middleware"

\_ "github.com/lib/pq"

)

const (

host = "localhost"

port = 5432

user = "postgres"

password = "postgres"

dbname = "sandbox"

)

type DatabaseProvider struct {

db \*sql.DB

}

type User struct {

ID int `json:"id"`

Name string `json:"name"`

}

// Методы работы с базой данных

func (dp \*DatabaseProvider) GetUser(name string) (\*User, error) {

query := "SELECT id, name FROM users WHERE name = $1"

row := dp.db.QueryRow(query, name)

var user User

err := row.Scan(&user.ID, &user.Name)

if err == sql.ErrNoRows {

return nil, nil // Пользователь не найден

} else if err != nil {

return nil, err

}

return &user, nil

}

func (dp \*DatabaseProvider) AddUser(name string) error {

query := "INSERT INTO users (name) VALUES ($1)"

\_, err := dp.db.Exec(query, name)

return err

}

func main() {

// Формирование строки подключения для PostgreSQL

psqlInfo := fmt.Sprintf("host=%s port=%d user=%s password=%s dbname=%s sslmode=disable",

host, port, user, password, dbname)

// Подключение к PostgreSQL

db, err := sql.Open("postgres", psqlInfo)

if err != nil {

log.Fatal(err)

}

defer db.Close()

// Проверка соединения

if err := db.Ping(); err != nil {

log.Fatal(err)

}

fmt.Println("Successfully connected to the database!")

// Инициализация провайдера БД

dbProvider := &DatabaseProvider{db: db}

// Инициализация Echo

e := echo.New()

// Middleware

e.Use(middleware.Logger())

e.Use(middleware.Recover())

// Роуты

e.GET("/api/user", func(c echo.Context) error {

name := c.QueryParam("name")

if name == "" {

return echo.NewHTTPError(http.StatusBadRequest, "Parameter 'name' is required")

}

user, err := dbProvider.GetUser(name)

if err != nil {

return echo.NewHTTPError(http.StatusInternalServerError, err.Error())

}

if user == nil {

return echo.NewHTTPError(http.StatusNotFound, "User not found")

}

return c.JSON(http.StatusOK, user)

})

e.POST("/api/user", func(c echo.Context) error {

var user User

if err := c.Bind(&user); err != nil {

return echo.NewHTTPError(http.StatusBadRequest, "Invalid JSON format")

}

if err := dbProvider.AddUser(user.Name); err != nil {

return echo.NewHTTPError(http.StatusInternalServerError, err.Error())

}

return c.String(http.StatusCreated, fmt.Sprintf("User %s added successfully", user.Name))

})

// Запуск сервера

e.Logger.Fatal(e.Start(":9000"))

}

hello.go

package main

import (

"database/sql"

"fmt"

"log"

"net/http"

"github.com/labstack/echo/v4"

"github.com/labstack/echo/v4/middleware"

\_ "github.com/lib/pq"

)

const (

host = "localhost"

port = 5432

user = "postgres"

password = "postgres"

dbname = "sandbox"

)

type Handlers struct {

dbProvider DatabaseProvider

}

type DatabaseProvider struct {

db \*sql.DB

}

type Message struct {

Msg string `json:"msg"`

}

// Обработчики HTTP-запросов

func (h \*Handlers) GetHello(c echo.Context) error {

msg, err := h.dbProvider.SelectHello()

if err != nil {

return echo.NewHTTPError(http.StatusInternalServerError, err.Error())

}

return c.JSON(http.StatusOK, map[string]string{"message": msg})

}

func (h \*Handlers) PostHello(c echo.Context) error {

input := new(Message)

if err := c.Bind(input); err != nil {

return echo.NewHTTPError(http.StatusBadRequest, err.Error())

}

if err := h.dbProvider.InsertHello(input.Msg); err != nil {

return echo.NewHTTPError(http.StatusInternalServerError, err.Error())

}

return c.NoContent(http.StatusCreated)

}

// Методы для работы с базой данных

func (dp \*DatabaseProvider) SelectHello() (string, error) {

var msg string

row := dp.db.QueryRow("SELECT message FROM hello ORDER BY RANDOM() LIMIT 1")

if err := row.Scan(&msg); err != nil {

return "", err

}

return msg, nil

}

func (dp \*DatabaseProvider) InsertHello(msg string) error {

\_, err := dp.db.Exec("INSERT INTO hello (message) VALUES ($1)", msg)

return err

}

func main() {

// Формирование строки подключения для PostgreSQL

psqlInfo := fmt.Sprintf("host=%s port=%d user=%s password=%s dbname=%s sslmode=disable",

host, port, user, password, dbname)

// Создание соединения с сервером PostgreSQL

db, err := sql.Open("postgres", psqlInfo)

if err != nil {

log.Fatal(err)

}

defer db.Close()

// Создаем провайдер для БД

dp := DatabaseProvider{db: db}

// Создаем экземпляр структуры с обработчиками

h := Handlers{dbProvider: dp}

// Инициализация Echo

e := echo.New()

// Middleware

e.Use(middleware.Logger())

e.Use(middleware.Recover())

// Роутинг

e.GET("/get", h.GetHello)

e.POST("/post", h.PostHello)

// Запуск сервера

e.Logger.Fatal(e.Start(":8081"))

}

Результаты тестирования приводить не имеет смысла, они совпадают с прошлой лабораторной работы.

Вывод. В ходе выполнения лабораторной работы были изучены методы работы с Echo.