



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

О Т Ч Е Т

по лабораторной работе № 9

Название: Back-End разработка с использованием фреймворка Echo

Дисциплина: Языки интернет программирования

Студент

ИУ6-31Б
(Группа)

(Подпись, дата)

О.В. Белякова
(И.О. Фамилия)

Преподаватель

(Подпись, дата)

В.Д. Шульман
(И.О. Фамилия)

Москва, 2024

Цель работы – получения первичных навыков использования веб-фреймворков в бекэнд-разработке на golang.

Микросервис hello:

```
package main

import (
    "database/sql"
    "fmt"
    "log"
    "net/http"
    "github.com/labstack/echo/v4"
    _ "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 CountRequest struct {
    Msg string `json:"msg"`
}

// Методы для работы с базой данных
// SelectHello выбирает случайное сообщение из таблицы hello
func (dp *DatabaseProvider) SelectHello() (string, error) {
    var msg string
    row := dp.db.QueryRow("SELECT message FROM hello ORDER BY RANDOM() LIMIT 1")
    err := row.Scan(&msg)
    if err != nil {
        return "", err
    }
    return msg, nil
}

// InsertHello вставляет новое сообщение в таблицу hello
```

```

func (dp *DatabaseProvider) InsertHello(msg string) error {
_, err := dp.db.Exec("INSERT INTO hello (message) VALUES ($1)", msg)
if err != nil {
return err
}
return nil
}

// Обработчики HTTP-запросов
// GetHello - обрабатывает GET запрос для получения случайного сообщения
func (h *Handlers) GetHello(c echo.Context) error {
msg, err := h.dbProvider.SelectHello()
if err != nil {
c.Logger().Error("Ошибка при получении сообщения из базы данных:", err)
return c.JSON(http.StatusInternalServerError, map[string]string{"error": "Ошибка
при получении данных"})
}
return c.JSON(http.StatusOK, map[string]string{"message": msg})
}

// PostHello - обрабатывает POST запрос для добавления нового сообщения
func (h *Handlers) PostHello(c echo.Context) error {
var input CountRequest
if err := c.Bind(&input); err != nil {
c.Logger().Error("Ошибка парсинга JSON:", err)
return c.JSON(http.StatusBadRequest, map[string]string{"error": "Ошибка парсинга
JSON"})
}
err := h.dbProvider.InsertHello(input.Msg)
if err != nil {
c.Logger().Error("Ошибка при вставке сообщения в базу данных:", err)
return c.JSON(http.StatusInternalServerError, map[string]string{"error": "Ошибка
при добавлении сообщения"})
}
return c.JSON(http.StatusCreated, map[string]string{"message": input.Msg})
}

// Основная функция
func main() {
// Формируем строку подключения для PostgreSQL
psqlInfo := fmt.Sprintf("host=%s port=%d user=%s password=%s dbname=%s
sslmode=disable",
host, port, user, password, dbname)
// Создаем соединение с базой данных
db, err := sql.Open("postgres", psqlInfo)
if err != nil {
log.Fatal("Ошибка подключения к базе данных:", err)
}

```

```

defer db.Close()
// Проверка соединения с БД
if err := db.Ping(); err != nil {
log.Fatal("Ошибка пинга базы данных:", err)
}
// Создаем провайдер для работы с БД
dp := &DatabaseProvider{db: db}
// Создаем обработчики
h := &Handlers{dbProvider: dp}
// Создаем новый сервер Echo
e := echo.New()
// Регистрация маршрутов с обработчиками
e.GET("/get", h.GetHello)
e.POST("/post", h.PostHello)
// Запуск сервера
e.Logger.Fatal(e.Start(":8081"))
}

```

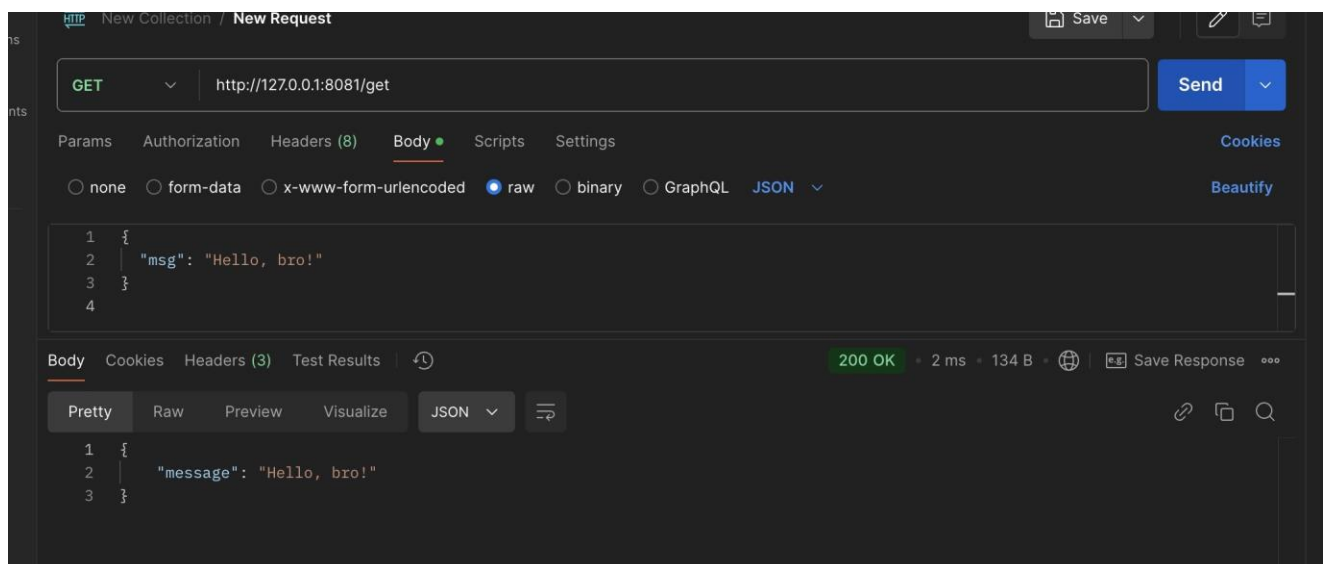


Рисунок 1 – микросервис hello.

Микросервис Count:

```

package main
import (
"database/sql"
"fmt"
"log"
"net/http"
"github.com/labstack/echo/v4"
_ "github.com/lib/pq"
)

```

```

const (
host = "localhost"
port = 5432
user = "postgres"
password = "postgres"
dbname = "count"
)
type DatabaseProvider struct {
db *sql.DB
}
type CountRequest struct {
Count int `json:"count"`
}
type Handlers struct {
dbProvider *DatabaseProvider
}
// Методы для работы с базой данных
// SelectCount выбирает текущее значение счетчика из базы данных
func (dp *DatabaseProvider) SelectCount() (int, error) {
var count int
row := dp.db.QueryRow("SELECT count FROM counters WHERE id = 1")
err := row.Scan(&count)
if err != nil {
if err == sql.ErrNoRows {
// Если записи нет, создаем начальный счетчик
_, err := dp.db.Exec("INSERT INTO counters (count) VALUES (0)")
if err != nil {
return 0, err
}
count = 0
} else {
return 0, err
}
}
return count, nil
}
// UpdateCount обновляет значение счетчика в базе данных
func (dp *DatabaseProvider) UpdateCount(increment int) error {
_, err := dp.db.Exec("UPDATE counters SET count = count + $1 WHERE id = 1",
increment)
if err != nil {
return err
}
return nil
}

```

```

}

// Обработчики HTTP-запросов
// GetCount обрабатывает GET запрос для получения текущего значения счетчика
func (h *Handlers) GetCount(c echo.Context) error {
    count, err := h.dbProvider.SelectCount()
    if err != nil {
        return c.JSON(http.StatusInternalServerError, map[string]string{"error":
            err.Error()})
    }
    return c.JSON(http.StatusOK, map[string]int{"count": count})
}

// PostCount обрабатывает POST запрос для увеличения счетчика
func (h *Handlers) PostCount(c echo.Context) error {
    var input CountRequest
    if err := c.Bind(&input); err != nil {
        return c.JSON(http.StatusBadRequest, map[string]string{"error": "Ошибка парсинга
            JSON"})
    }
    if input.Count <= 0 {
        return c.JSON(http.StatusBadRequest, map[string]string{"error": "Значение count
            должно быть положительным числом"})
    }
    err := h.dbProvider.UpdateCount(input.Count)
    if err != nil {
        return c.JSON(http.StatusInternalServerError, map[string]string{"error":
            err.Error()})
    }
    return c.JSON(http.StatusOK, map[string]string{"message": fmt.Sprintf("Счетчик
        увеличен на %d", input.Count)})
}

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)
    db, err := sql.Open("postgres", psqlInfo)
    if err != nil {
        log.Fatal("Ошибка подключения к базе данных:", err)
    }
    defer db.Close()
    // Проверка соединения
    if err := db.Ping(); err != nil {
        log.Fatal("Ошибка пинга базы данных:", err)
    }
    // Создаем провайдер для работы с БД

```

```


dp := &DatabaseProvider{db: db}
// Создаем обработчики
h := &Handlers{dbProvider: dp}
// Создаем новый сервер Echo
e := echo.New()
// Регистрация обработчиков
e.GET("/count/get", h.GetCount)
e.POST("/count/post", h.PostCount)
// Запуск сервера на порту 8081
e.Logger.Fatal(e.Start(":8081"))
}

```

```

(base) kristinadoronina@MacBook-Pro-Kristina count % go run main.go

```

 **v4.12.0**
 High performance, minimalist Go web framework
<https://echo.labstack.com>

-----0/-----
 0\
 => http server started on [::]:8081
 █

Строка 120, столбец 20 Размер интервала таблицы: 4 UTF-8 LF { } G

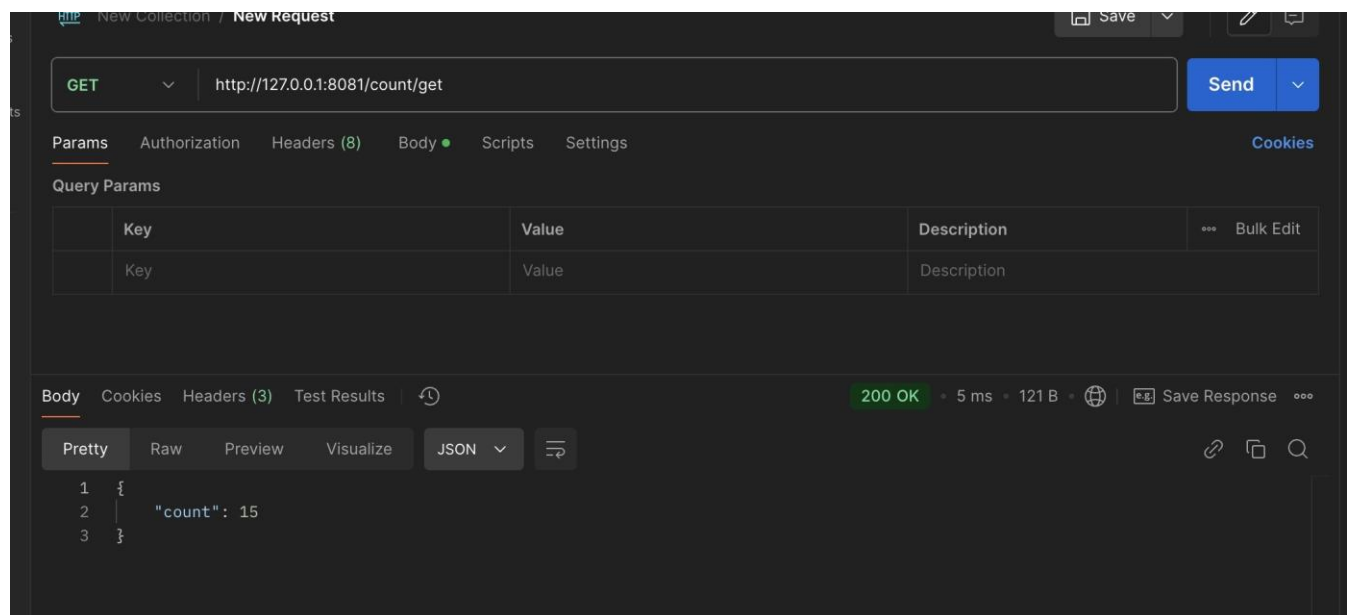


Рисунок 2 – микросервис count.

Микросервис Query:

```

package main

import (
    "database/sql"
    "fmt"
    "log"

```

```

"net/http"
"github.com/labstack/echo/v4"
_ "github.com/lib/pq"
)

const (
host = "localhost"
port = 5432
user = "postgres"
password = "postgres"
dbname = "query"
)

type Handlers struct {
dbProvider *DatabaseProvider
}

type DatabaseProvider struct {
db *sql.DB
}

// Метод для получения приветствия по имени
func (h *Handlers) GetGreeting(c echo.Context) error {
name := c.QueryParam("name")
if name == "" {
return c.JSON(http.StatusBadRequest, map[string]string{"error": "Нет параметра 'name'"})
}

greeting, err := h.dbProvider.SelectGreeting(name)
if err != nil {
c.Logger().Error("Ошибка при получении приветствия: ", err)
return c.JSON(http.StatusInternalServerError, map[string]string{"error": "Ошибка при получении приветствия"})
}

return c.String(http.StatusOK, greeting)
}

// Методы для работы с базой данных
// SelectGreeting получает приветствие из базы данных
func (dp *DatabaseProvider) SelectGreeting(name string) (string, error) {
var greeting string
row := dp.db.QueryRow("SELECT greeting FROM greetings WHERE name = $1", name)
err := row.Scan(&greeting)
if err != nil {
if err == sql.ErrNoRows {
// Если записи нет, создаем новое приветствие для данного имени
_, err := dp.db.Exec("INSERT INTO greetings (name, greeting) VALUES ($1, $2)",
name, fmt.Sprintf("Hello, %s!", name))
if err != nil {

```



```

return "", err
}
greeting = fmt.Sprintf("Hello, %s!", name)
} else {
return "", err
}
}
return greeting, nil
}

func main() {
// Формируем строку подключения к базе данных
psqlInfo := fmt.Sprintf("host=%s port=%d user=%s password=%s dbname=%s
sslmode=disable",
host, port, user, password, dbname)
// Открываем соединение с базой данных
db, err := sql.Open("postgres", psqlInfo)
if err != nil {
log.Fatal("Ошибка подключения к базе данных:", err)
}
defer db.Close()
// Проверяем соединение с базой данных
if err := db.Ping(); err != nil {
log.Fatal("Ошибка пинга базы данных:", err)
}
// Создаем провайдер для работы с базой данных
dp := &DatabaseProvider{db: db}
// Создаем обработчики
h := &Handlers{dbProvider: dp}
// Создаем новый экземпляр Echo
e := echo.New()
// Регистрируем маршруты
e.GET("/api/user", h.GetGreeting)
// Запуск сервера на порту 8081
e.Logger.Fatal(e.Start(":8081"))
}

```

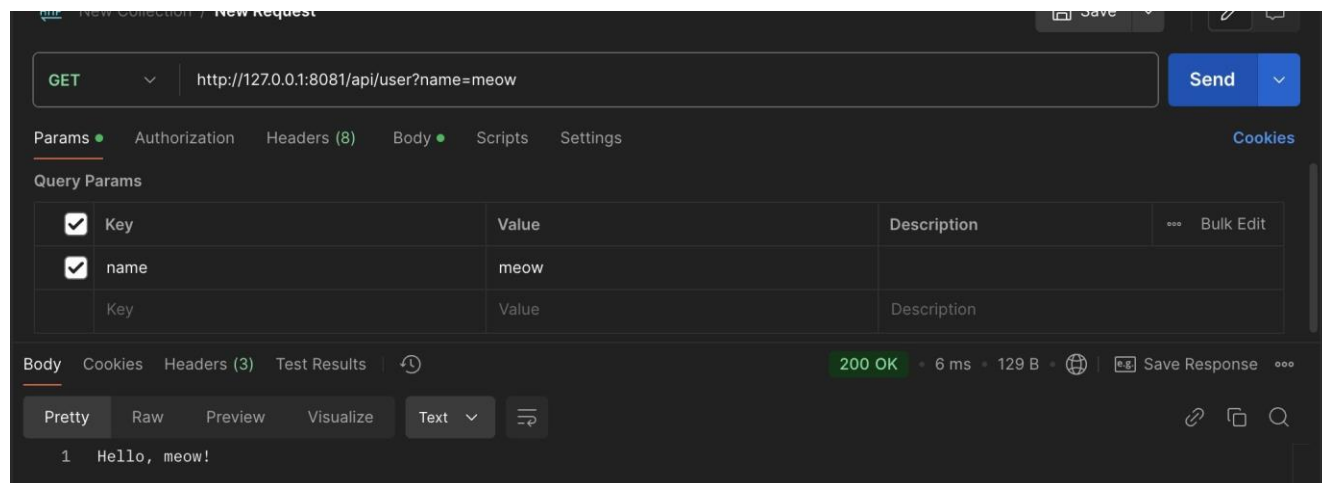


Рисунок 3 – микросервис query.

Заключение – интегрировали echo и модифицировали программы из 8 лабы.