



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 Компьютерные системы и сети

**О Т Ч Е Т**

**по лабораторной работе № 9**

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

**Дисциплина:** Языки интернет-программирования

Студент	<u>ИУ6-32Б</u> (Группа)	<u>24.11.2024</u> (Подпись, дата)	<u>Т.А. Гаджиев</u> (И.О. Фамилия)
Преподаватель		<u>24.11.2024</u> (Подпись, дата)	<u>В.Д. Шульман</u> (И.О. Фамилия)

Москва, 2024

Цель работы — получение первичных навыков использования веб-фреймворков в Backend-разработке на Golang.

Ход работы.

1. Сделали форк данного репозитория в GitHub, клонировали получившуюся копию локально, создали от мастера ветку дев и переключились на нее. Скопировали наши микросервисы из 8-ой лабораторной работы:

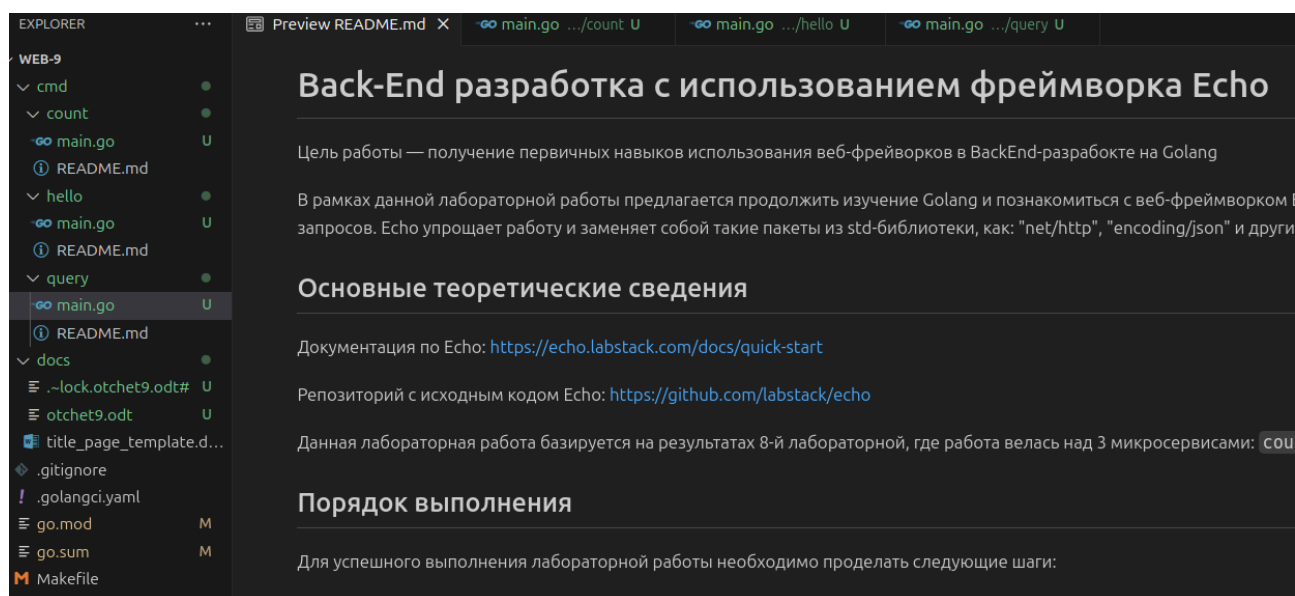


Рисунок 1 — Скопированный репозиторий

2. Установили библиотеки для работы с фреймворком echo и переписали под него наши микросервисы:

**Задача 1**(Вывод строки приветствия):

```
package main
```

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

```
const (  
    host = "localhost"  
    port = 5432  
    user = "postgres"
```

```

password = "postgres"
dbname = "mydb"
)

type Handlers struct {
dbProvider DatabaseProvider
}

type DatabaseProvider struct {
db *sql.DB
}

// Обработчики HTTP-запросов
func (h *Handlers) GetHello(c echo.Context) error {
msg, err := h.dbProvider.SelectHello()
if err != nil {
return c.JSON(http.StatusInternalServerError, map[string]string{"error": err.Error()})
}

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

func (h *Handlers) PostHello(c echo.Context) error {
var input struct {
Msg string `json:"msg"`
}

if err := c.Bind(&input); err != nil {
return c.JSON(http.StatusBadRequest, map[string]string{"error": "Неверный формат запроса"})
}

err := h.dbProvider.InsertHello(input.Msg)
if err != nil {
return c.JSON(http.StatusInternalServerError, map[string]string{"error": err.Error()})
}

return c.JSON(http.StatusCreated, map[string]string{"message": "Сообщение успешно добавлено"})
}

// Методы для работы с базой данных
func (dp *DatabaseProvider) SelectHello() (string, error) {

// Получаем одно сообщение из таблицы hello, отсортированной в случайном порядке
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
}

```

```

}

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

func main() {
// Считываем аргументы командной строки
address := flag.String("address", "127.0.0.1:8081", "адрес для запуска сервера")
flag.Parse()

// Формирование строки подключения для postgres
psqlInfo := fmt.Sprintf("host=%s port=%d user=%s password=%s dbname=%s sslmode=disable",
host, port, user, password, dbname)

// Создание соединения с сервером postgres
db, err := sql.Open("postgres", psqlInfo)
if err != nil {
log.Fatal(err)
}
defer db.Close()

// Создаем провайдер для БД с набором методов
dp := DatabaseProvider{db: db}
// Создаем экземпляр структуры с набором обработчиков
h := Handlers{dbProvider: dp}

// Создаем экземпляр Echow
e := echo.New()

// Регистрируем обработчики
e.GET("/get", h.GetHello)
e.POST("/post", h.PostHello)

// Запускаем веб-сервер на указанном адресе
fmt.Printf("Сервер запущен на %s\n", *address)
if err := e.Start(*address); err != nil {
log.Fatal(err)
}
}

```

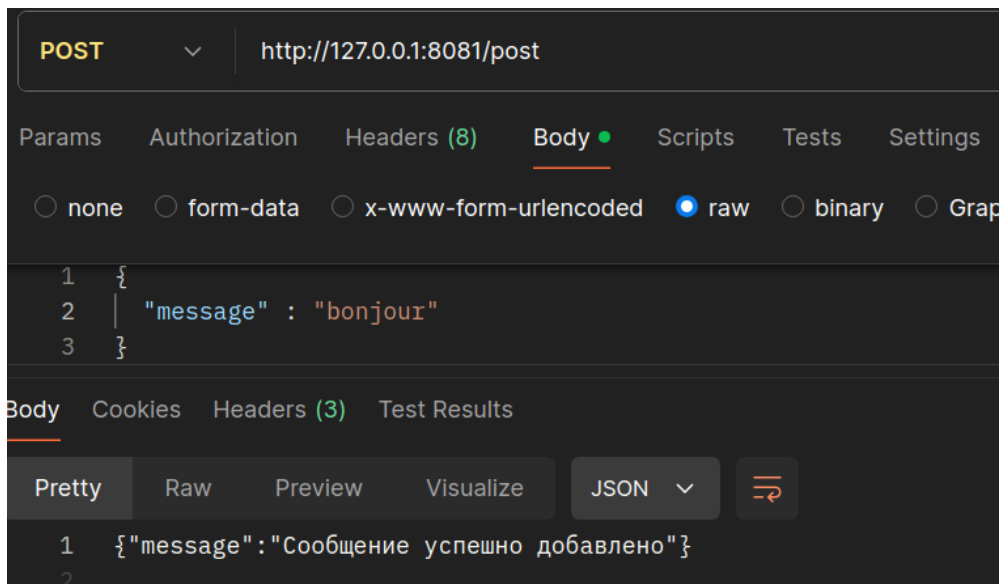


Рисунок 2 — Post запрос в Postman

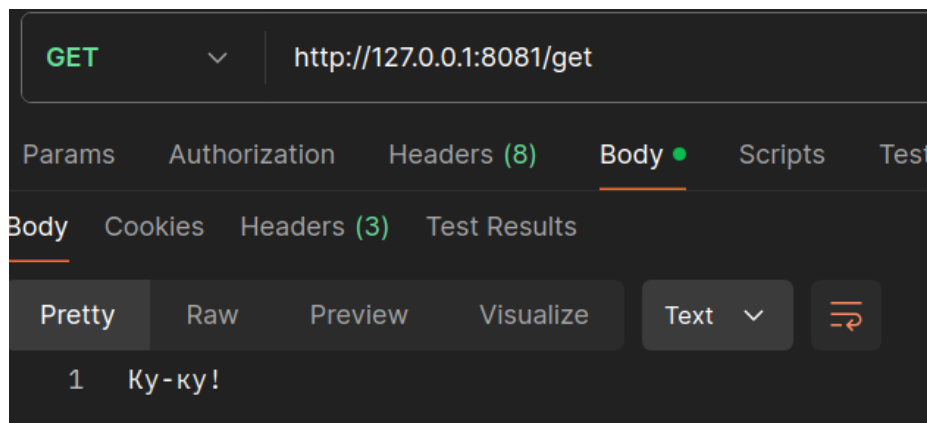


Рисунок 3- Get запрос в postman

Data Output		Messa
		<div> <div>≡+</div> <div>📄</div> <div>▼</div> <div>📋</div> <div>▼</div> </div>
	message	text 🔒
1	Привет, мир!	
2	hello world!	
3	Gutten Acht!	
4	Ку-ку!	
5	bonjour	

Рисунок 4 — Содержимое таблицы hello

## Задача 2(Вывод строки приветствия с ключом)

```
package main
```

```
import (  
    "database/sql"  
    "flag"  
    "fmt"  
    "log"  
    "net/http"  
  
    "github.com/labstack/echo/v4"  
    _ "github.com/lib/pq"  
)  
  
const (  
    host = "localhost"  
    port = 5432  
    user = "postgres"  
    password = "postgres"  
    dbname = "mydb"  
)  
  
type Handlers struct {  
    dbProvider DatabaseProvider  
}  
  
type DatabaseProvider struct {  
    db *sql.DB  
}  
  
// Обработчики HTTP-запросов  
func (h *Handlers) GetUser(c echo.Context) error {  
    name := c.QueryParam("name")  
    if name == "" {  
        return c.JSON(http.StatusBadRequest, map[string]string{"error": "Name parameter is required"})  
    }  
  
    user, err := h.dbProvider.SelectUser(name)  
    if err != nil {  
        return c.JSON(http.StatusInternalServerError, map[string]string{"error": err.Error()})  
    }  
  
    return c.String(http.StatusOK, "Hello, "+user+"!")  
}  
  
func (h *Handlers) PostUser(c echo.Context) error {  
    var input struct {  
        Name string `json:"name"`  
    }
```

```

}

if err := c.Bind(&input); err != nil {
return c.JSON(http.StatusBadRequest, map[string]string{"error": err.Error()})
}

err := h.dbProvider.InsertUser(input.Name)
if err != nil {
return c.JSON(http.StatusInternalServerError, map[string]string{"error": err.Error()})
}

return c.JSON(http.StatusCreated, map[string]string{"message": "Запись добавлена!"})
}

func (dp *DatabaseProvider) SelectUser(name string) (string, error) {
var user string

row := dp.db.QueryRow("SELECT name FROM mytable WHERE name = $1", name)
err := row.Scan(&user)
if err != nil {
return "", err
}

return user, nil
}

func (dp *DatabaseProvider) InsertUser(name string) error {
_, err := dp.db.Exec("INSERT INTO mytable (name) VALUES ($1)", name)
return err
}

func main() {
address := flag.String("address", "127.0.0.1:8080", "адрес для запуска сервера")
flag.Parse()

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()

dp := DatabaseProvider{db: db}
h := Handlers{dbProvider: dp}

e := echo.New()

e.GET("/api/user", h.GetUser)
e.POST("/api/user/create", h.PostUser)

```

```

fmt.Printf("Сервер запущен на %s\n", *address)
if err := e.Start(*address); err != nil {
log.Fatal(err)
}
}

```

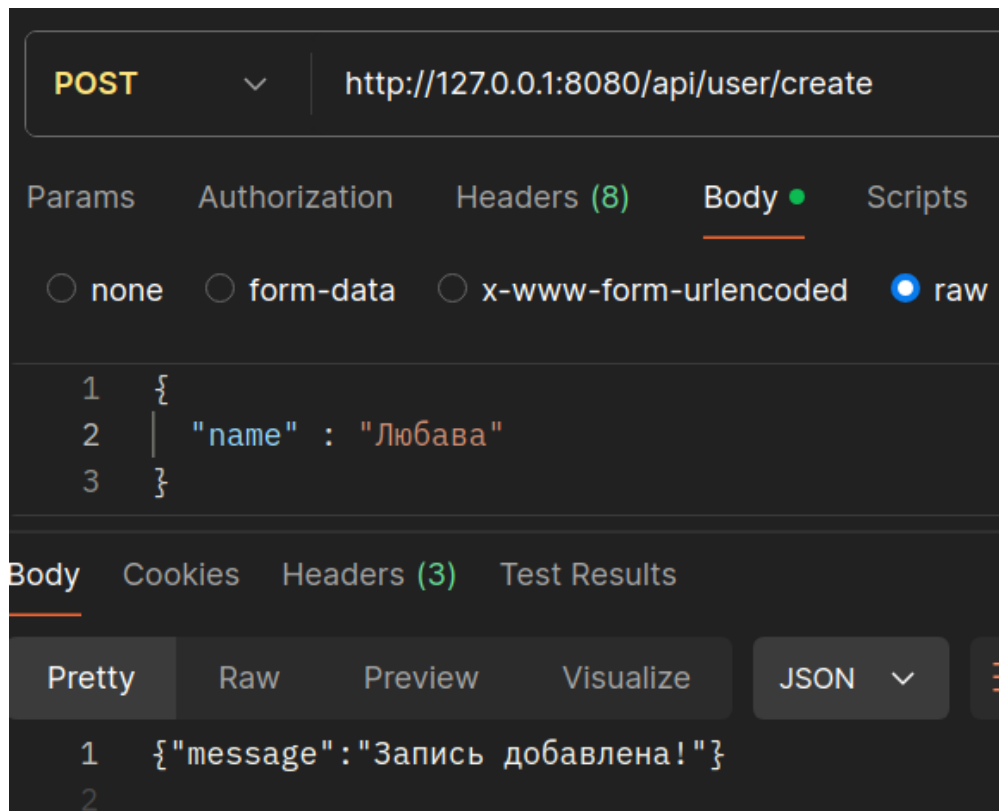


Рисунок 5 — Post запрос в postman

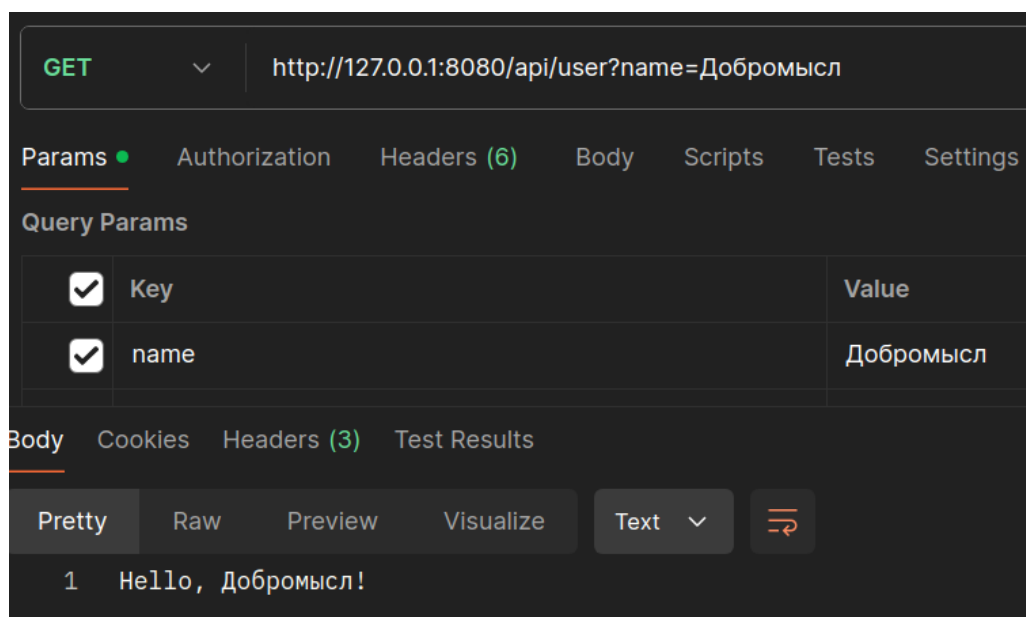


Рисунок 6 — Get запрос в postman



Data Output

Messages

Notifications

	id [PK] integer	name character varying (255)
1	1	John Doe
2	2	КрутоеИмя
3	3	ЛучшееИмя
4	4	CoolName
5	5	Аноним
6	6	Добромысл
7	7	Любава

Рисунок 7 — Содержимое таблицы имён

### Задача 3(Count):

```
package main

import (
    "database/sql"
    "flag"
    "fmt"
    "log"
    "net/http"

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

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

type Handlers struct {
    dbProvider DatabaseProvider
}

type DatabaseProvider struct {
    db *sql.DB
}

func (dp *DatabaseProvider) GetCounter() (int, error) {
    var counter int
```

```

row := dp.db.QueryRow("SELECT value FROM counter_table LIMIT 1")
err := row.Scan(&counter)
if err != nil {
    return 0, err
}
return counter, nil
}

func (dp *DatabaseProvider) UpdateCounter(value int) error {
_, err := dp.db.Exec("UPDATE counter_table SET value = value + $1", value)
return err
}

func (h *Handlers) HandleCount(c echo.Context) error {
switch c.Request().Method {
case http.MethodGet:
    counter, err := h.dbProvider.GetCounter()
    if err != nil {
        return c.JSON(http.StatusInternalServerError, map[string]string{"error": err.Error()})
    }

    err = h.dbProvider.UpdateCounter(1)
    if err != nil {
        return c.JSON(http.StatusInternalServerError, map[string]string{"error": err.Error()})
    }

    return c.String(http.StatusOK, fmt.Sprintf("%d", counter+1)) // Увеличиваем на 1 для ответа

case http.MethodPost:
    var requestBody struct {
        Count int `json:"count"`
    }

    if err := c.Bind(&requestBody); err != nil {
        return c.JSON(http.StatusBadRequest, map[string]string{"error": "это не число"})
    }

    err := h.dbProvider.UpdateCounter(requestBody.Count)
    if err != nil {
        return c.JSON(http.StatusInternalServerError, map[string]string{"error": err.Error()})
    }
    return c.JSON(http.StatusOK, map[string]string{"message": "Success"})

default:
    return c.JSON(http.StatusMethodNotAllowed, map[string]string{"error": "Неизвестный метод"})
}
}

func main() {
    address := flag.String("address", "127.0.0.1:3333", "адрес для запуска сервера")
    flag.Parse()
}

```

```

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

db, err := sql.Open("postgres", pgsqlInfo)
if err != nil {
log.Fatal(err)
}
defer db.Close()

dp := DatabaseProvider{db: db}
h := Handlers{dbProvider: dp}

e := echo.New()

e.Logger.SetLevel(2)

e.GET("/count", h.HandleCount)
e.POST("/count", h.HandleCount)

fmt.Println("Сервер запущен на порту :3333")
if err := e.Start(*address); err != nil {
log.Fatal(err)
}
}

```

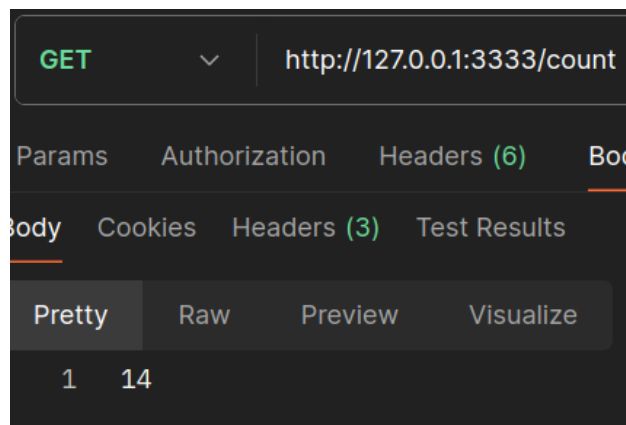


Рисунок 6 — Get запрос в Postman


	value 
1	14

Рисунок 7 — Содержимое таблицы counter\_table

4. Зафиксировали изменения, сделали коммит и отправили полученное состояние ветки dev в удаленный репозиторий GitHub. Через интерфейс GitHub создали Pull Request dev --> master

Заключение: в ходе лабораторной работы получили первичные навыки использования веб-фреймворков в BackEnd-разработке на Golang.