



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 ИУ6-31БВ  
(Группа)

\_\_\_\_\_  
(Подпись, дата)

В.О. Бокова  
(И.О. Фамилия)

Преподаватель

\_\_\_\_\_  
(Подпись, дата)

В.Д.Шульман  
(И.О. Фамилия)

Москва, 2024

**Цель работы**— получения первичных навыков использования веб-фрейворков в бекэнд-разработке на golang.

## 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  = "5401"
    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"))
}

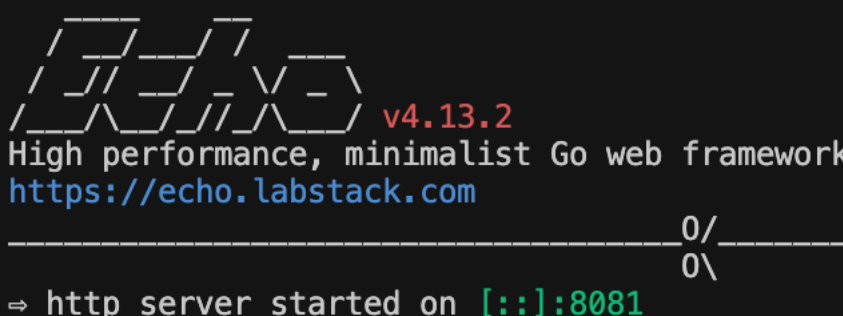
```

## Тестирование:

```

go get github.com/labstack/echo/v4
victoriabokova@MacBook-Pro-Vistoria count % go run main.go

```



High performance, minimalist Go web framework  
<https://echo.labstack.com>  
 -----0/-----  
 0\  
 ⇒ http server started on [::]:8081

Рисунок 1 – запуск сервера

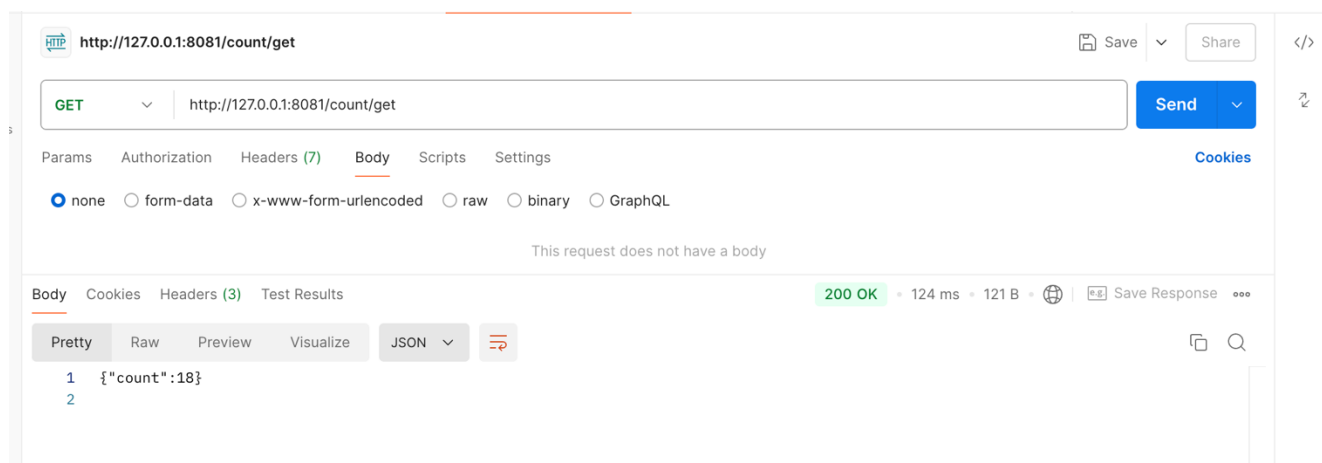


Рисунок 2 – get запрос

## 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  = "5401"
    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"))
}

```

## Тестирование

```
o victoriabokova@MacBook-Pro-Vistoria hello % go run main.go

  _ _ _ _ _
 /_/_/_/_/_ v4.13.2
High performance, minimalist Go web framework
https://echo.labstack.com
-----0/-----
0\
⇒ http server started on [::]:8081
```

Рисунок 3 – запуск сервера

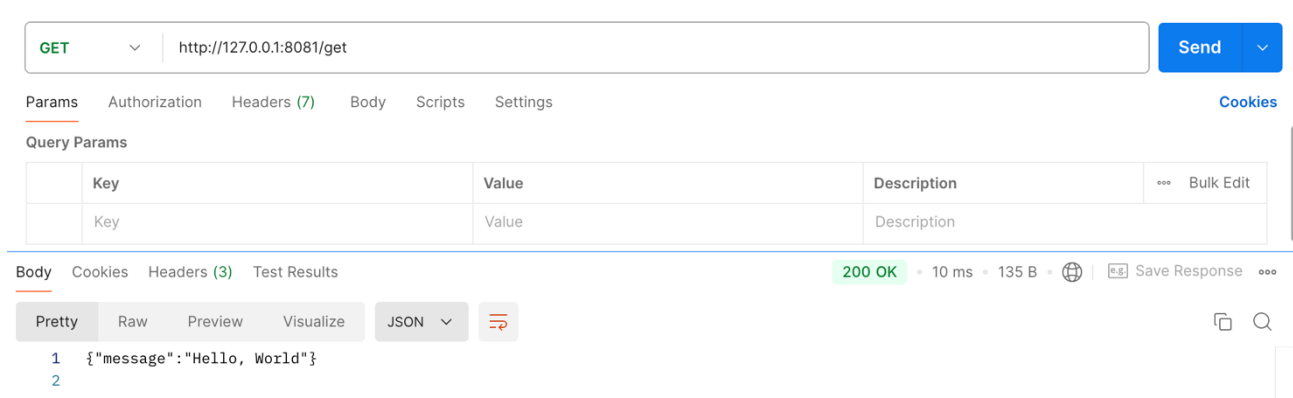


Рисунок 3 – get запрос

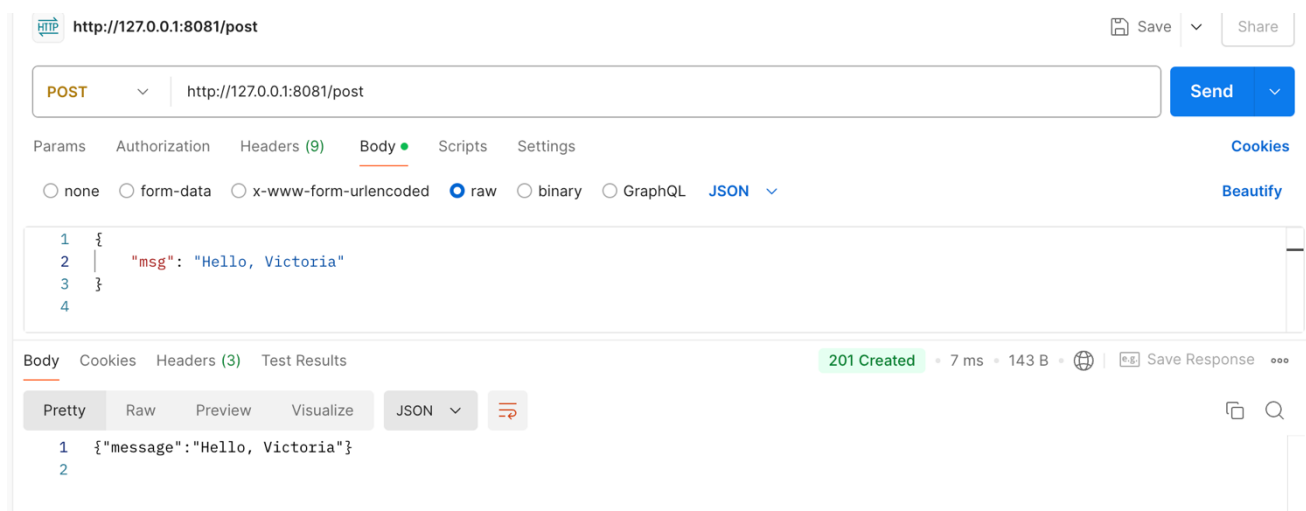


Рисунок 4 – post запрос

## Get запрос выводит или Hello, World или Hello Victoria

The screenshot shows a REST client interface. The top bar displays the method **GET** and the URL `http://127.0.0.1:8081/get`. Below the bar are tabs for Params, Authorization, Headers (9), **Body**, Scripts, and Settings. The **Body** tab is active, showing a JSON body: 

```
1 {
2   "msg": "Hello, Victoria"
3 }
4
```

 The right side of the interface has buttons for **Send**, **Cookies**, and **Beautify**. Below the body editor, there are tabs for Body, Cookies, Headers (3), and Test Results. The **Body** tab is active, showing a status of **200 OK**, a response time of **2 ms**, and a size of **138 B**. Below this are tabs for Pretty, Raw, Preview, Visualize, and JSON. The **Pretty** tab is active, showing the response body: 

```
1 {"message": "Hello, Victoria"}
2
```

Рисунок 5 – get запрос

The screenshot shows a REST client interface. The top bar displays the method **GET** and the URL `http://127.0.0.1:8081/get`. Below the bar are tabs for Params, Authorization, Headers (9), **Body**, Scripts, and Settings. The **Body** tab is active, showing an empty body. The right side of the interface has buttons for **Send**, **Cookies**, and **Beautify**. Below the body editor, there are tabs for Body, Cookies, Headers (3), and Test Results. The **Body** tab is active, showing a status of **200 OK**, a response time of **3 ms**, and a size of **135 B**. Below this are tabs for Pretty, Raw, Preview, Visualize, and JSON. The **Pretty** tab is active, showing the response body: 

```
1 {"message": "Hello, World"}
2
```

Рисунок 6 – get запрос

## 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  = "5401"
    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", postgresInfo)
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"))
}

```

```

victoriabokova@MacBook-Pro-Vistoria query % go run main.go

  ____  _
 / ___|| | | |
| |___| |_| |
 \___ \|  __/
      | |  | |
      |_|  |_| v4.13.2
High performance, minimalist Go web framework
https://echo.labstack.com
-----0/-----
                        0\
⇒ http server started on [::]:8081

```

Рисунок 7 – запуск сервера

The screenshot shows a web browser interface for testing HTTP requests. The address bar displays the URL `http://127.0.0.1:8081/api/user?name=Victoria`. The method is set to `GET`. The response status is `200 OK` with a response time of `16 ms` and a body size of `133 B`. The response body is `Hello, Victoria!`.

Query Params

Key	Value	Description
name	Victoria	

Body

```
1 Hello, Victoria!
```

Рисунок 8 – Get запрос

Заключение – интегрировали echo и модифицировали программы из лабораторной работы №8.