**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 **09.04.01 ИУ6-32Б**

# О Т Ч Е Т

**по лабораторной работе № 6**

## Название: Основы Back-End разработки на Golang

**Дисциплина: Языки интернет-программирования**

Студент ИУ6-32Б _____ Кулиев Э.
(Группа) (Подпись, дата) (И.О. Фамилия)

Преподаватель _____ Шульман В.Д.
(Подпись, дата) (И.О. Фамилия)

Цель работы: изучение основ сетевого взаимодействия и серверной разработки с использованием языка Golang.

# Задание 1

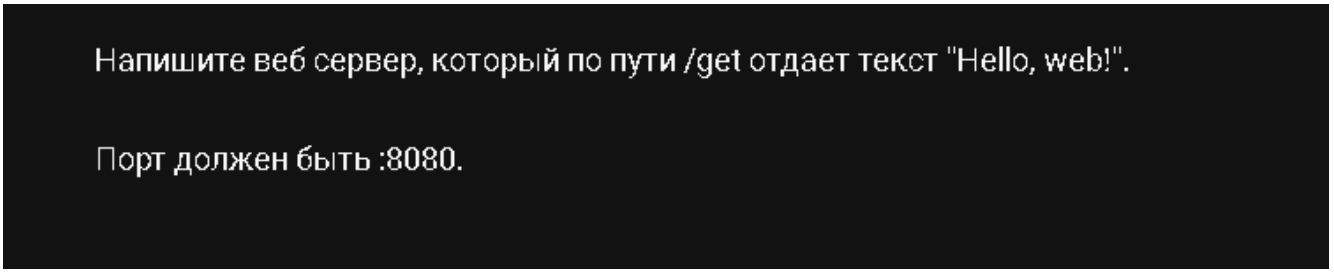


Рисунок 1

```go
package main

import (
    "fmt"
    "net/http"
)

Codeium: Refactor | Explain | Generate GoDoc | ×
func helloHandler(w http.ResponseWriter, r *http.Request) {
    fmt.Fprint(w, "Hello, web!")
}

Codeium: Refactor | Explain | Generate GoDoc | ×
func main() {
    http.HandleFunc("/get", helloHandler)
    fmt.Println("Server is listening on port 8080")
    http.ListenAndServe(":8080", nil)
}
```

Рисунок 2

На рисунке 2 показан мой результат.
Есть функция helloHandler, обрабатывающая запрос на “get” которая возвращает “Hello, web!”/

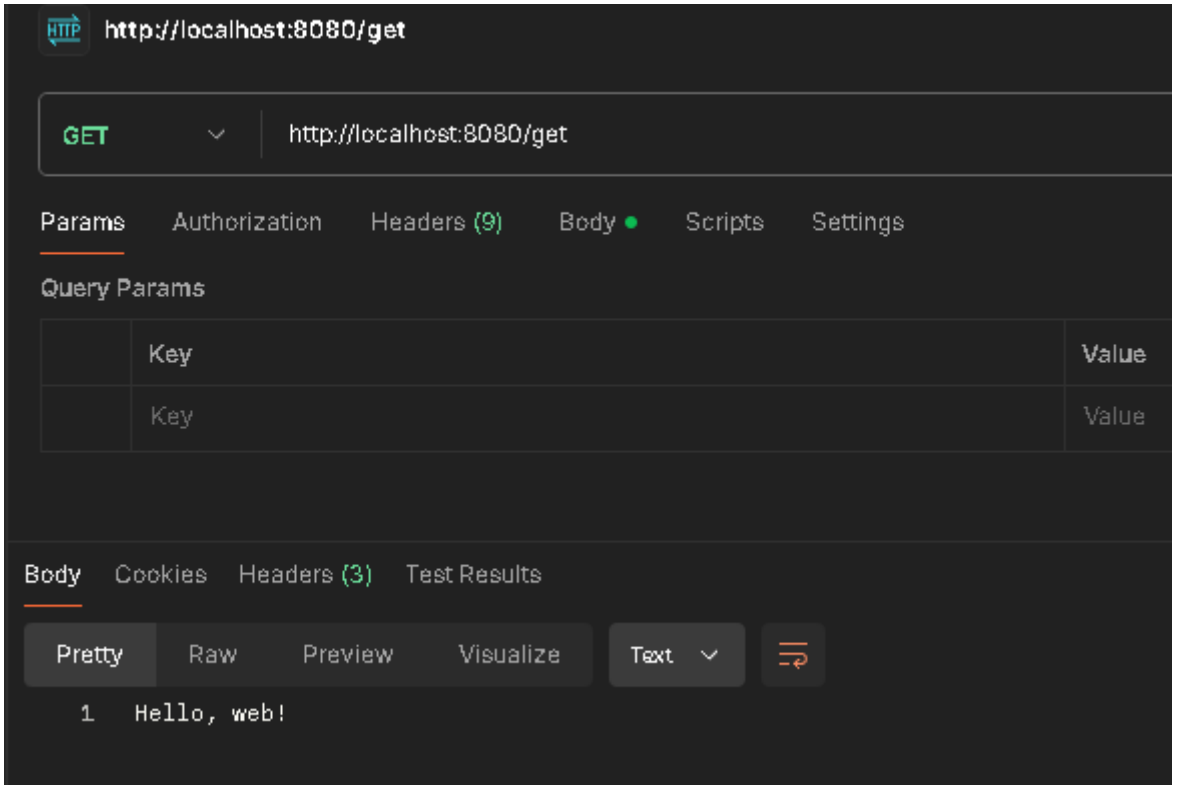


Рисунок 3

На рисунке 3 показан пример вывода.

## Задание 2

Напишите веб-сервер который по пути /api/user приветствует пользователя:
Принимает и парсит параметр `name` и делает ответ "Hello,<name>!"
Пример: /api/user?name=Golang
Ответ: Hello,Golang!

Рисунок 4

Как и в предыдущем задании мы обрабатываем запрос “api/user” но теперь в запросе присутствует параметр name который задается api/user?name=Stepan

```go
package main

import (
    "fmt"
    "net/http"
)

Codeium: Refactor | Explain | Generate GoDoc | ×
func helloHandler(w http.ResponseWriter, r *http.Request) {
    name := r.URL.Query().Get("name")
    if name == "" {
        fmt.Fprint(w, "Hello, stranger!")
        return
    }
    fmt.Fprintf(w, "Hello, %s!", name)
}

Codeium: Refactor | Explain | Generate GoDoc | ×
func main() {
    http.HandleFunc("/api/user", helloHandler)
    fmt.Println("Server is listening on port 8080")
    http.ListenAndServe(":8080", nil)
}
```

Рисунок 5

На рис.5 показан мой результат.

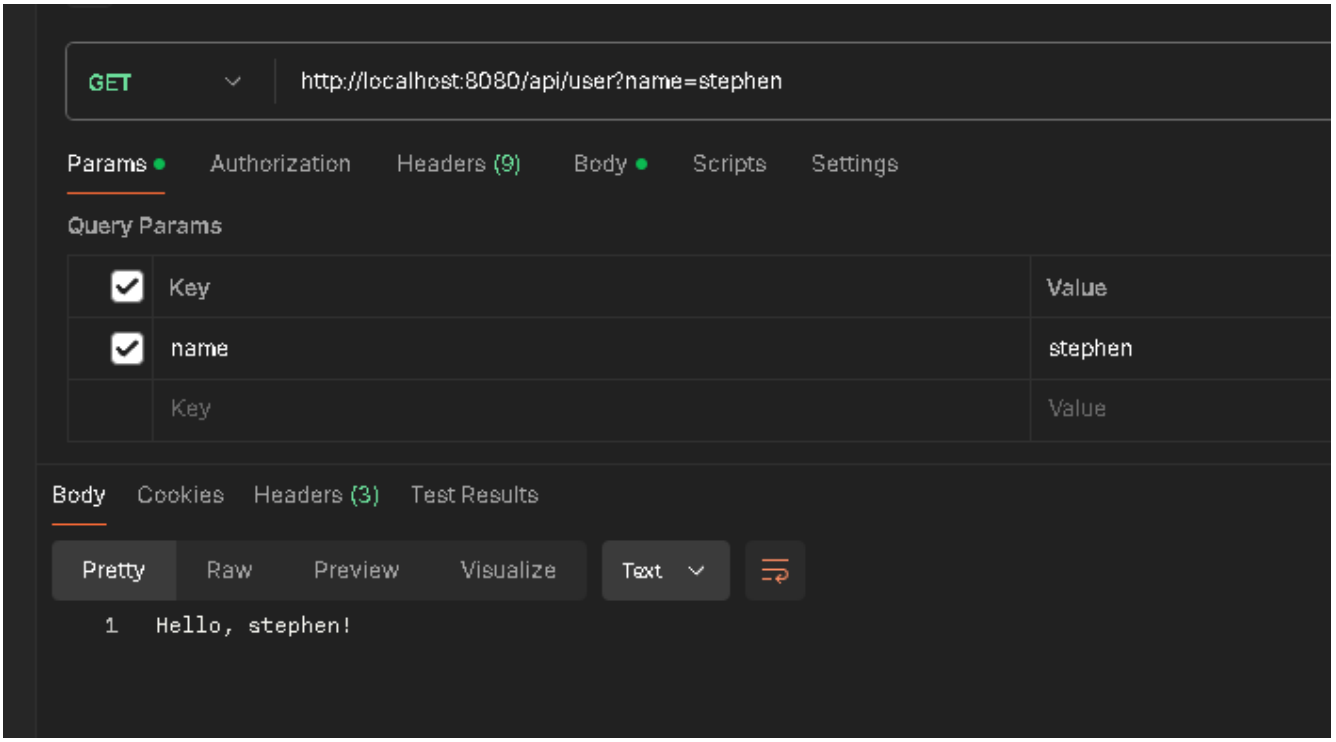


Рисунок 6 (Пример вывода)

# Задание 3

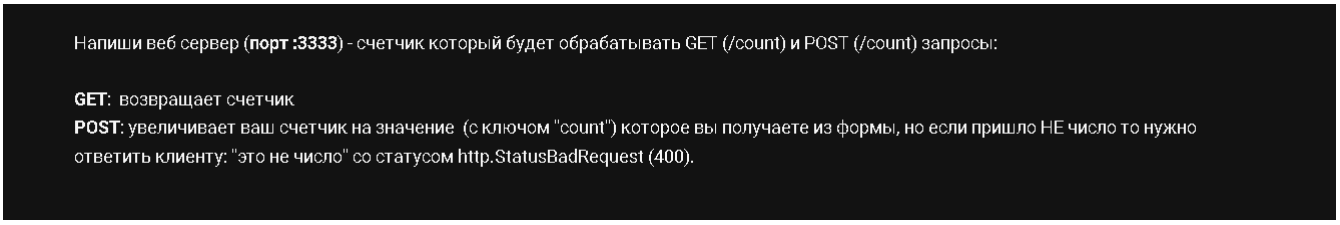


Рисунок 7

Для выполнения задания нам придется создать 2 фунцкии обработчики, для POST и GET метода.

```go
func updateCountHandler(w http.ResponseWriter, r *http.Request) {
    if r.Method != http.MethodPost {
        http.Error(w, "Invalid request method", http.StatusBadRequest)
        return
    }

    var update struct {
        Count int `json:"count"`
    }

    err := json.NewDecoder(r.Body).Decode(&update)
    if err != nil {
        http.Error(w, "Invalid request body", http.StatusBadRequest)
        return
    }

    if update.Count == 0 {
        http.Error(w, "Это не число", http.StatusBadRequest)
        return
    }

    counter.Value += update.Count
    w.WriteHeader(http.StatusOK)
}
```

Рисунок 8 (Фунцкия для POST запроса)

```go
package main

import (
    "encoding/json"
    "fmt"
    "net/http"
)

Codeium: Refactor | Explain
type Counter struct {
    Value int
}

var counter Counter

Codeium: Refactor | Explain | Generate GoDoc | ×
func getCountHandler(w http.ResponseWriter, _ *http.Request) {
    json.NewEncoder(w).Encode(counter)
}
```

Рисунок 9 (Функция для GET запроса)

```go
Codeium: Refactor | Explain | Generate GoDoc | ×
func main() {
    http.HandleFunc("/count", func(w http.ResponseWriter, r *http.Request) {
        if r.Method == http.MethodGet {
            getCountHandler(w, r)
        } else if r.Method == http.MethodPost {
            updateCountHandler(w, r)
        }
    })

    fmt.Println("Server is listening on port 3333")
    http.ListenAndServe(":3333", nil)
}
```

Рисунок 11 (Основная функция main)

В ней мы сначала обрабатываем тип запроса и в зависимости от этого выбираем функцию-обработчик.

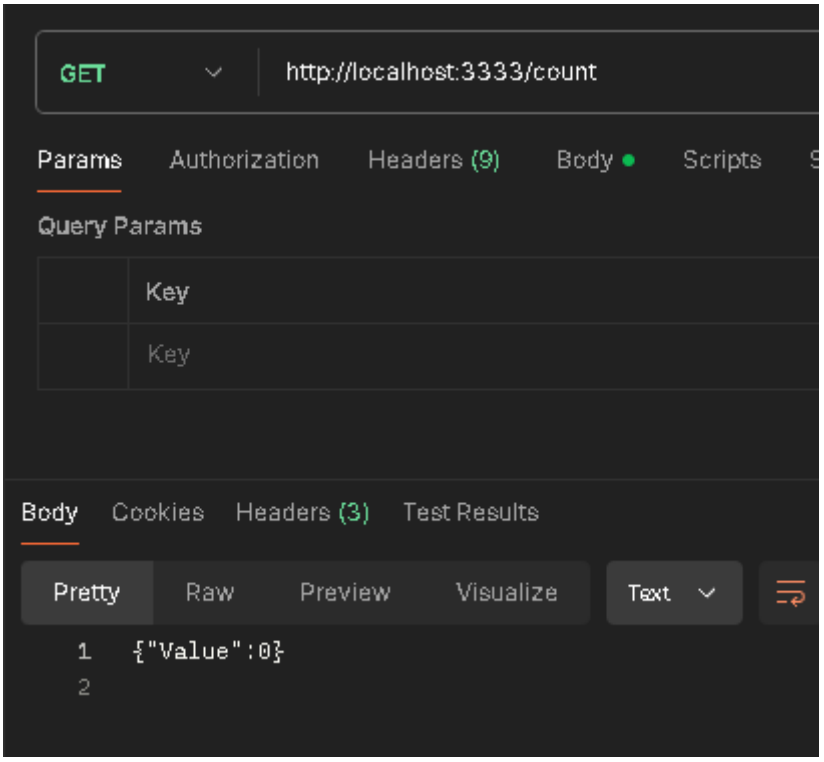


Рисунок 12 (GET запрос)

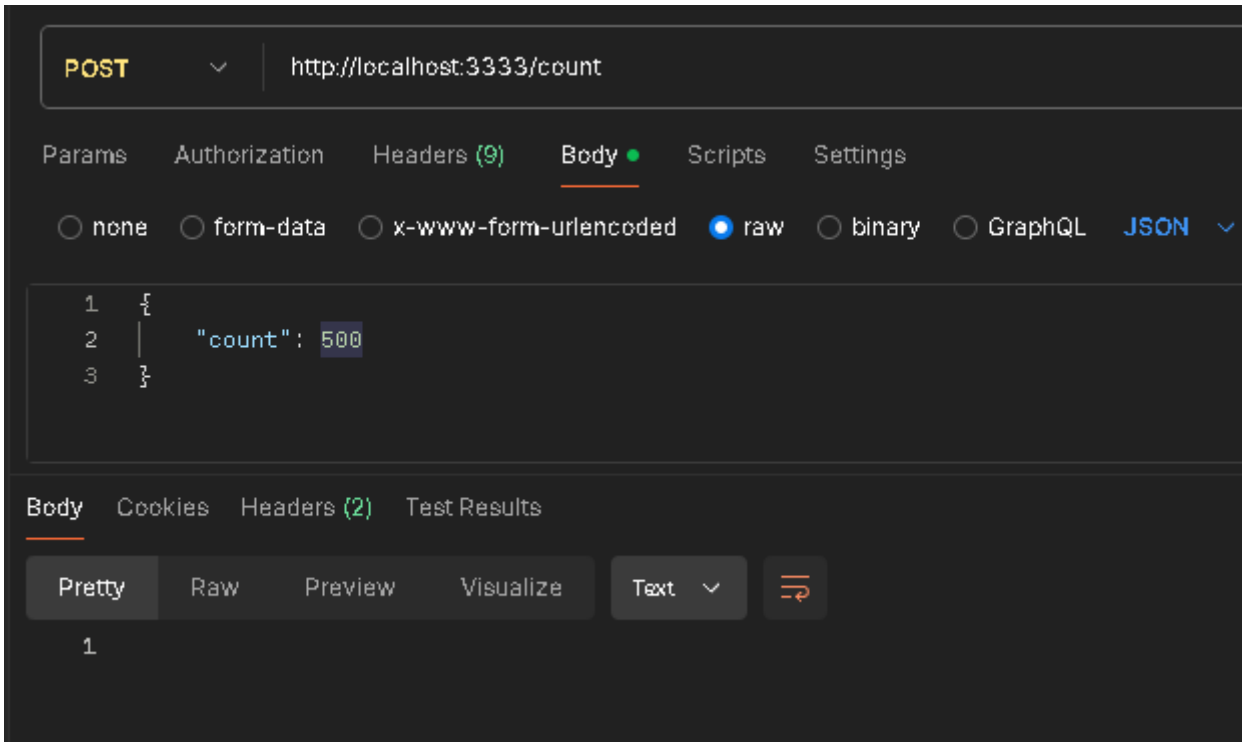


Рисунок 13 (POST запрос)

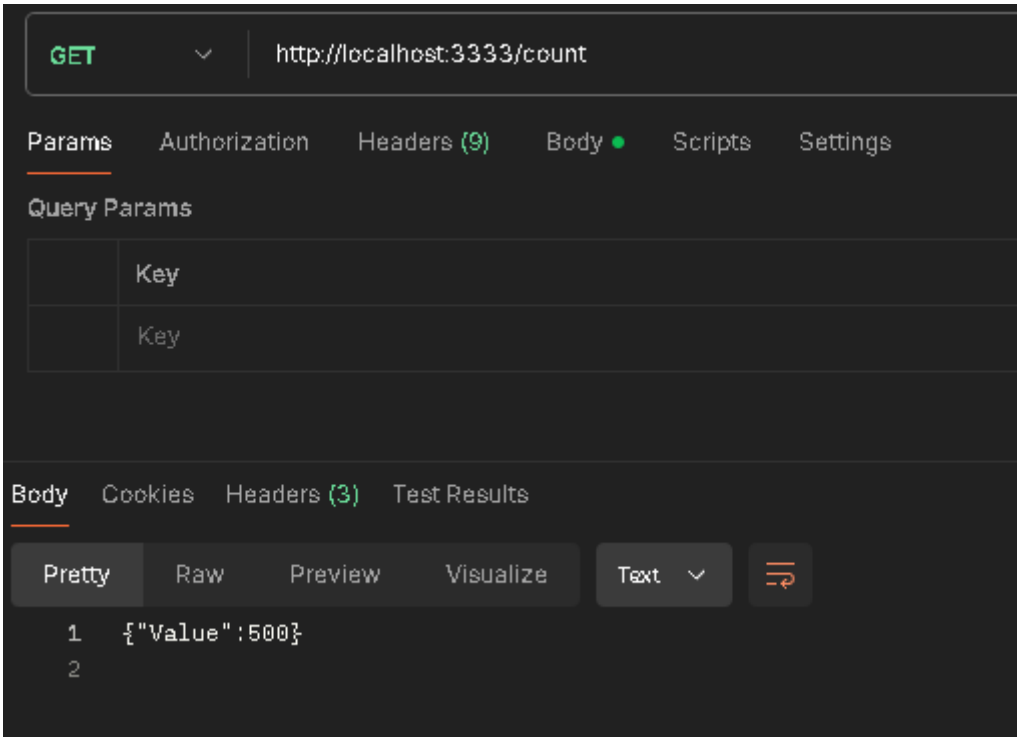

Рисунок 14 (GET запрос)

# Контрольные вопросы

1. **В чём разница между протоколами TCP и UDP**

TCP (Transmission Control Protocol) и UDP (User Datagram Protocol) - это два основных протокола сетевого уровня, используемых в Интернете. Они отличаются по своей надежности, производительности и способу обработки ошибок.
Вот некоторые общие различия между TCP и UDP:
Надежность: TCP обеспечивает надежную передачу данных, а UDP - нет.
Последовательность: TCP обеспечивает последовательную доставку данных, а UDP - нет.
Подтверждения: TCP использует подтверждения, а UDP - нет.
Скорость: TCP медленнее, чем UDP.

2. **Для чего нужны IP Address и Port Number у веб-сервера и в чём разница?**

IP-адрес определяет местоположение веб-сервера в сети, а порт определяет, какое приложение или службу он обслуживает.

3. **Какой набор методов в HTTP-request в полной мере релализует семантику CRUD ?**

Набор методов HTTP, реализующий семантику CRUD:

* CREATE - POST
* READ - GET

* UPDATE - PUT или PATCH
* DELETE – DELETE

**4. Какие группы status code существуют у HTTP-response**
Существуют следующие группы статусов HTTP:

* 1xx - Информационные (100 Continue, 101 Switching Protocols)
* 2xx - Успешные (200 OK, 201 Created, 204 No Content)
* 3xx - Перенаправления (301 Moved Permanently, 302 Found, 304 Not Modified)
* 4xx - Ошибки клиента (400 Bad Request, 401 Unauthorized, 404 Not Found)
* 5xx - Ошибки сервера (500 Internal Server Error, 502 Bad Gateway, 503 Service Unavailable)

**5.Из каких составных элементов состоит HTTP-request и HTTP-response ?**

HTTP-запрос (HTTP-request) состоит из:

* Метод (Method, например GET, POST, PUT)
* URI (Uniform Resource Identifier)
* Версия HTTP (HTTP-Version)
* Заголовки (Headers)
* Тело запроса (Body)

HTTP-ответ (HTTP-response) состоит из:

* Версия HTTP (HTTP-Version)
* Статусный код (Status Code)
* Текстовое описание статуса (Status Text)
* Заголовки (Headers)
* Тело ответа (Body)

## Заключение: Я научился создавать веб-сервер и обрабатывать GET и POST запросы, при этом получая параметры различными способами.

Москва, 2024