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 № 6

Название: Основы Back-End разработки на Golang

Дисциплина: Языки интернет-программирования

Студент ИУ6-31Б _____ Л.В. Зимин
(Группа) (Подпись, дата) (И.О. Фамилия)

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

Москва, 2024

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

Задание:

1. Ознакомиться с разделом "4. Списки, сеть и сервера" курса
2. Сделать форк данного репозитория в GitHub, склонировать получившуюся копию локально, создать от мастера ветку дев и переключиться на неё
3. Выполнить задания. Ссылки на задания можно найти в README-файлах в директории projects
4. (опционально) Проверить свой коды линтерами с помощью команды make lint
5. Сделать отчёт и поместить его в директорию docs
6. Зафиксировать изменения, сделать коммит и отправить получившееся состояние ветки дев в личный форк данного репозитория в GitHub
7. Через интерфейс GitHub создать Pull Request dev --> master
8. На защите лабораторной работы продемонстрировать открытый Pull Request. PR должен быть направлен в master ветку форка, а не исходного репозитория

Ход работы

1. Задача Hello

Условие:

Необходимо написать веб-сервер, который по пути «/get» отдает текст «Hello, web!». Порт должен быть :8080. Код должен компилироваться, а сервер запускаться и корректно обрабатывать запросы. Для локальной отладки можно использовать Postman или Insomnia.

Решение:

```
package main
// некоторые импорты нужны для проверки
import (
	"fmt"
	"io"
	"net/http"
	"os"
	"time"
)
func handler(w http.ResponseWriter, r *http.Request) {
	w.Write([]byte("Hello, web!"))
}
func main() {
	http.HandleFunc("/get", handler)
	err := http.ListenAndServe(":8080", nil)
	if err != nil {
```

```
        fmt.Println("Ошибка запуска сервера!")
    }
}
```

Тестирование

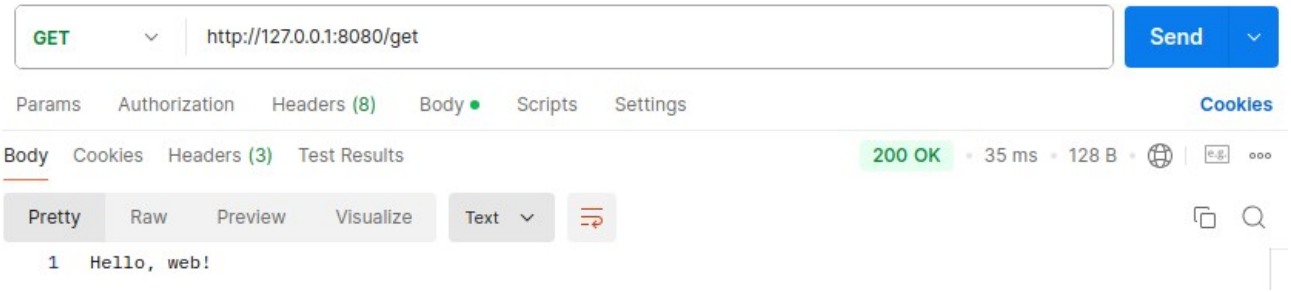

Рисунок 1 — Тестирование задачи Hello

2. Задача Query

Условие:

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!

Порт :9000

Решение:

```
package main
// некоторые импорты нужны для проверки
import (
    "fmt"
    "io"
    "net/http" // пакет для поддержки HTTP протокола
    "os"
    "time"
)
```

```
func handler(w http.ResponseWriter, r *http.Request) {
	name := r.URL.Query().Get("name")
	w.Write([]byte("Hello," + name + "!"))
}

func main() {
	http.HandleFunc("/api/user", handler)
	err := http.ListenAndServe(":9000", nil)
	if err != nil {
		fmt.Println("Ошибка запуска сервера!")
	}
}
```

Тестирование

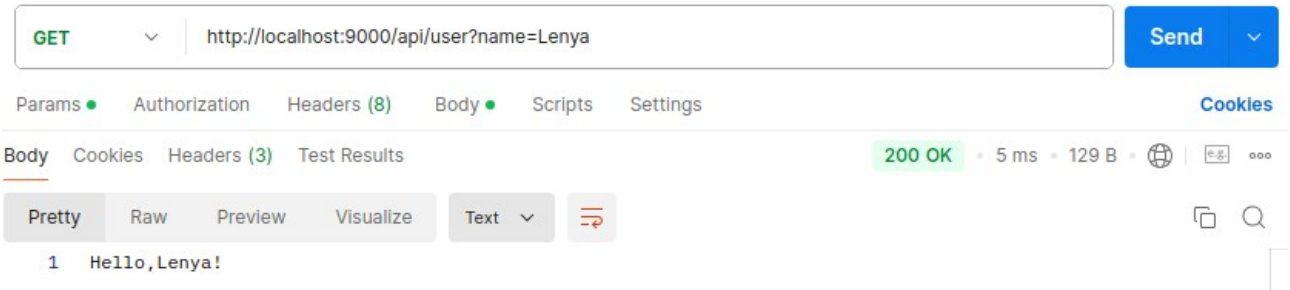

Рисунок 2 — Тестирование задачи Query

3. Задача Count

Условие:

Напиши веб сервер (порт :3333) — счетчик который будет обрабатывать GET (/count) и POST (/count) запросы:

GET: возвращает счетчик

POST: увеличивает ваш счетчик на значение (с ключом "count") которое вы получаете из формы, но если пришло НЕ число то нужно ответить клиенту: "это не число" со статусом http.StatusBadRequest (400).

Решение:

```
package main
// некоторые импорты нужны для проверки
import (
        "fmt"
        "io"
        "log"
        "net/http"
        "net/url"
        "os"
        "time"
    "strconv" // вдруг понадобиться вам ;)
)
var count1 int = 0
func handler(w http.ResponseWriter, r *http.Request) {
        if r.Method == "GET" {
       w.WriteHeader(http.StatusOK)
                w.Write([]byte(strconv.Itoa(count1)))
                return
        } else if r.Method == "POST" {
                r.ParseForm()
                s := r.FormValue("count")
                if s == "" {
                        w.WriteHeader(http.StatusBadRequest)
```

```
                w.Write([]byte("это не число"))
                return
            }
            number, err := strconv.Atoi(s)
            if err != nil {
                w.WriteHeader(http.StatusBadRequest)
                w.Write([]byte("это не число"))
                return
            }
            count1 += number
            return
        } else {
            w.WriteHeader(http.StatusMethodNotAllowed)
            w.Write([]byte("Метод не поддерживается"))
            return
        }
}
func main() {
        http.HandleFunc("/count", handler)
        err := http.ListenAndServe(":3333", nil)
        if err != nil {
            fmt.Println("Ошибка запуска сервера!")
        }
}
```

Тестирование

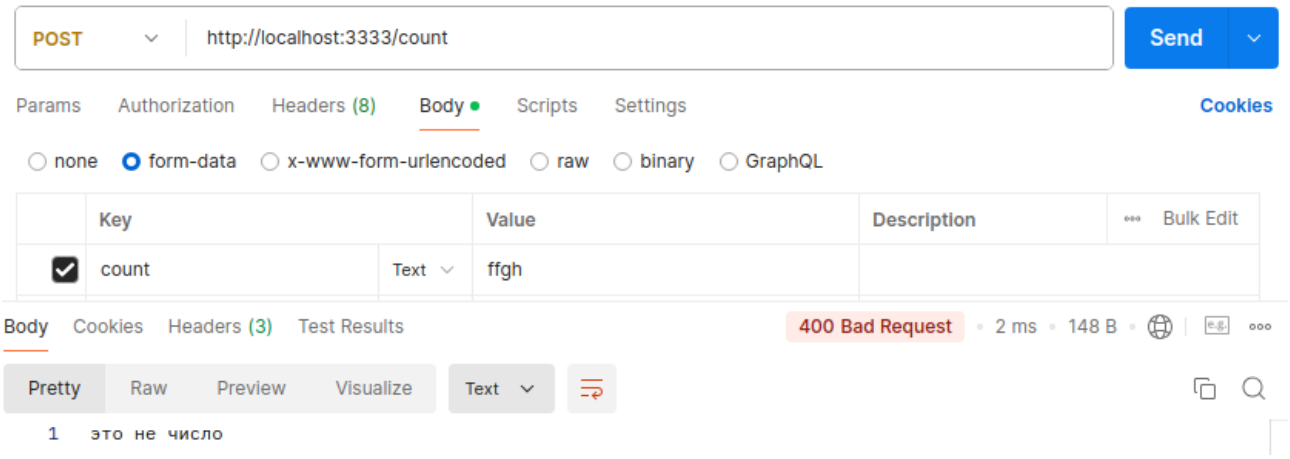

Рисунок 3 — Тестирование 1 задачи Count

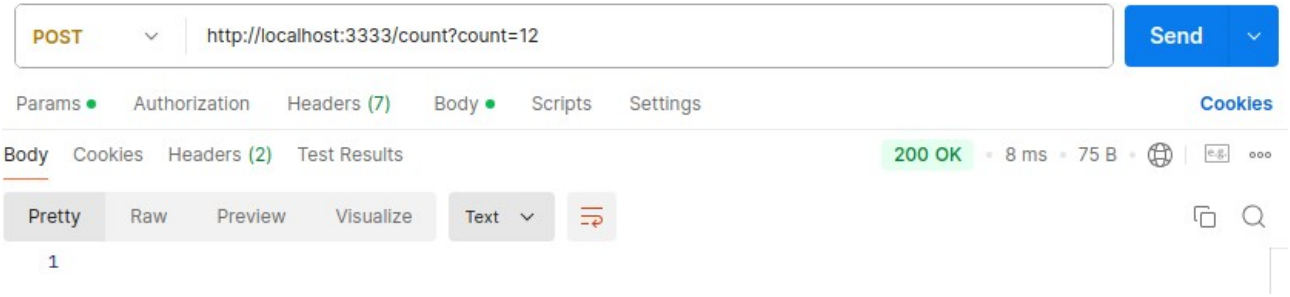

Рисунок 4 — Тестирование 2 задачи Count

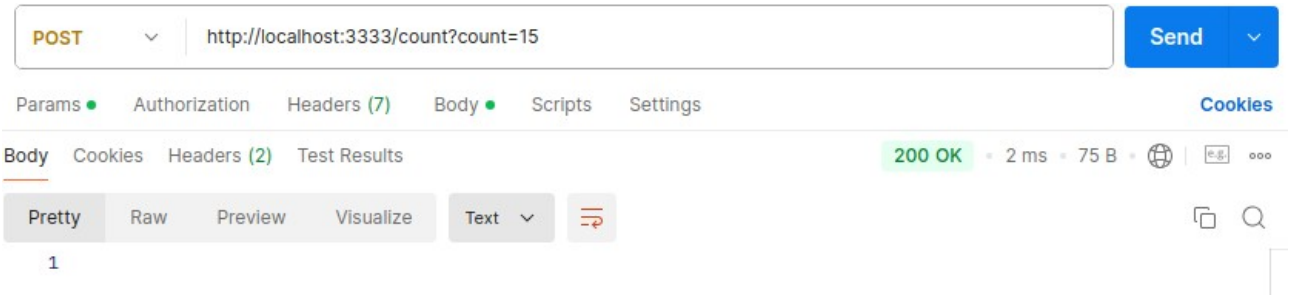

Рисунок 5 — Тестирование 3 задачи Count

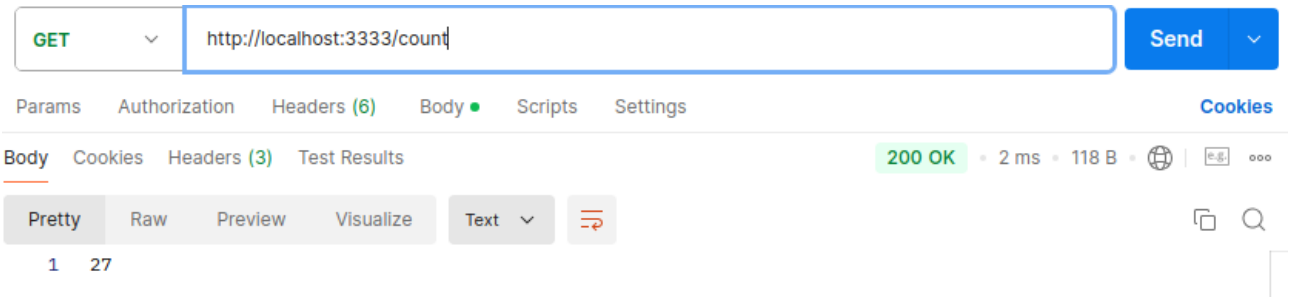

Рисунок 6 — Тестирование 4 задачи Count

4. Загрузим решения на GitHub и сделаем Pull request из dev в мастер с помощью интерфейса GitHub.

Контрольные вопросы

1. В чём разница между протоколами TCP и UDP ?

TCP (Transmission Control Protocol) – это надежный и устойчивый протокол передачи данных в сетях. Он обеспечивает установление соединения между отправителем и получателем, а также обеспечивает гарантию доставки данных в правильном порядке и контроль ошибок. TCP используется для приложений, которым важна надежная передача данных, таких как веб-серверы, электронная почта и файловые передачи.

UDP (User Datagram Protocol) – это простой и быстрый протокол передачи данных в сетях. Он не гарантирует надежную доставку данных, не устанавливает соединение и не контролирует порядок доставки. UDP используется в приложениях, где небольшая потеря данных не критична, например, в видеозвонках и стриминге.

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

IP – это уникальный адрес, который присваивается устройству при подключении компьютерной сети. То есть с помощью IP можно идентифицировать устройство в сети.

Port Number используется для идентификации у устройства программы, которая осуществляет работу с данными в сети.

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

Create – POST

Read – GET

Update – PUT

Delete - DELETE

4. Какие группы status code существуют у HTTP-response (желательно, с примерами) ?

1xx — информационные. Например, 102 – Идёт обработка

2xx — успешные. Например, 202 - Принято

3xx — перенаправление. Например, 305 – Использовать прокси

4xx — клиентские ошибки. Например, 404 – Не найдено

5xx — серверные ошибки. Например, 501 – Не реализовано

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

Структура HTTP запроса и ответа:

1) Стартовая строка

2) Заголовки

3) Тело

Заключение:

Язык программирования Golang позволяет полноценно работать с сетью. Например, есть возможность создать веб-сервер без подключения дополнительных сторонних библиотек с возможностью обработки HTTP запросов.

Список использованных источников:

https://stepik.org/course/54403/syllabus

https://pkg.go.dev/