|  |  |
| --- | --- |
| Gerb-BMSTU_01 | **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**

**Отчет**

**по лабораторной работе № 6**

**Название:** Основы Back-End разработки на Golang

**Дисциплина:** Языки интернет-программирования

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Студент | ИУ6-32Б |  | 10.10.2024 | Л.И. Заушников |
|  | (Группа) |  | (Подпись, дата) | (И.О. Фамилия) |
|  |  |  |  |  |
| Преподаватель |  |  | 10.10.2024 | В.Д. Шульман |
|  |  |  | (Подпись, дата) | (И.О. Фамилия) |

Москва, 2024

**Цель работы:** изучить основы сетевого взаимодействия и серверной разработки с использованием языка Golang.

Ход работы

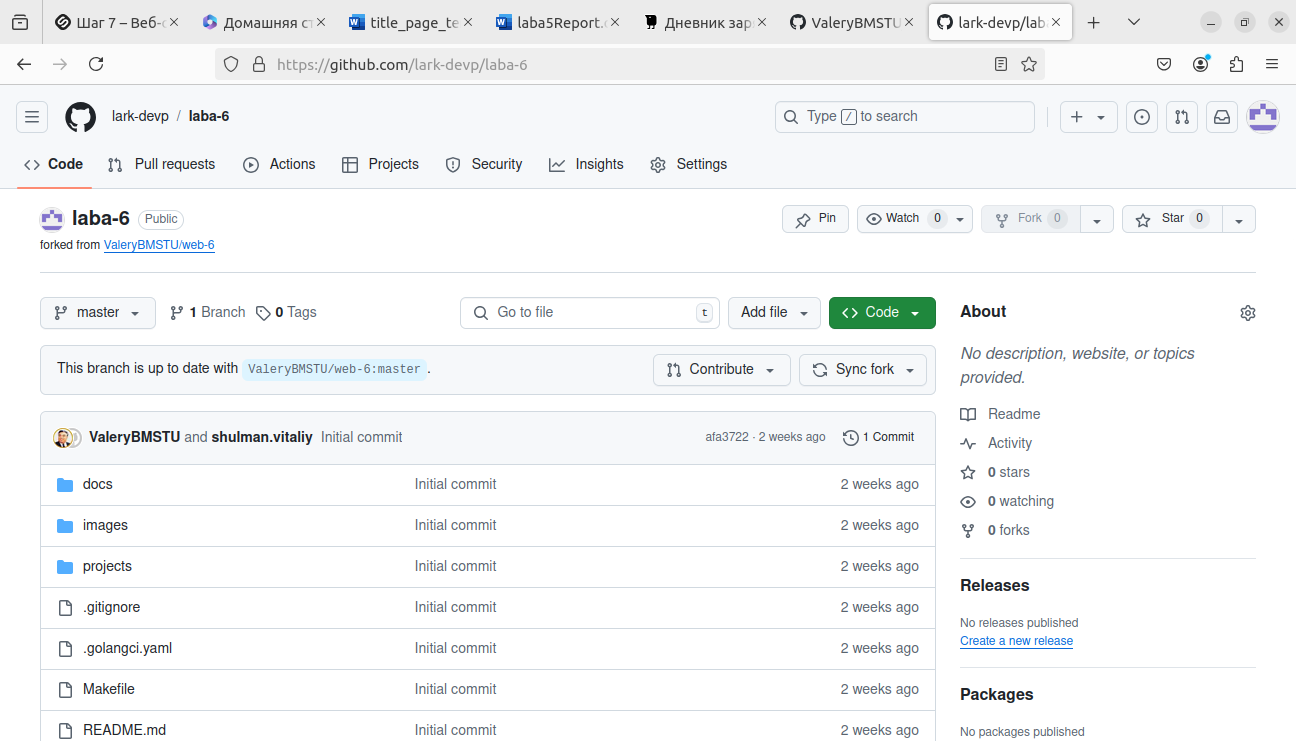
1. Первым делом было проведено ознакомление с разделом курса https://stepik.org/course/54403/info "4. Списки, сеть и сервера". Здесь основное внимание было уделено урокам "4.2 Работа с сетью" и "4.3 Веб-сервера".
2. Далее был сделан форк данного репозитория в GitHub (рисунок 1), склонирована получившаяся копия локально, создана от мастера ветка dev и было произведено переключение на неё.

Рисунок 1 - Форкнутый репозиторий

1. Было написано на языке Golang 3 HTTP сервера.

**Задача 1**

Напишите веб сервер, который по пути /get отдает текст "Hello, web!".

Порт должен быть :8080.

Код, решающий задачу выше представлен ниже.

package main

import (

"fmt"

"net/http"

)

func helloHandler(w http.ResponseWriter, r \*http.Request) {

w.Write([]byte("Hello, web!"))

}

func main() {

// здесь ваш код

http.HandleFunc("/get", helloHandler)

err := http.ListenAndServe(":8080", nil)

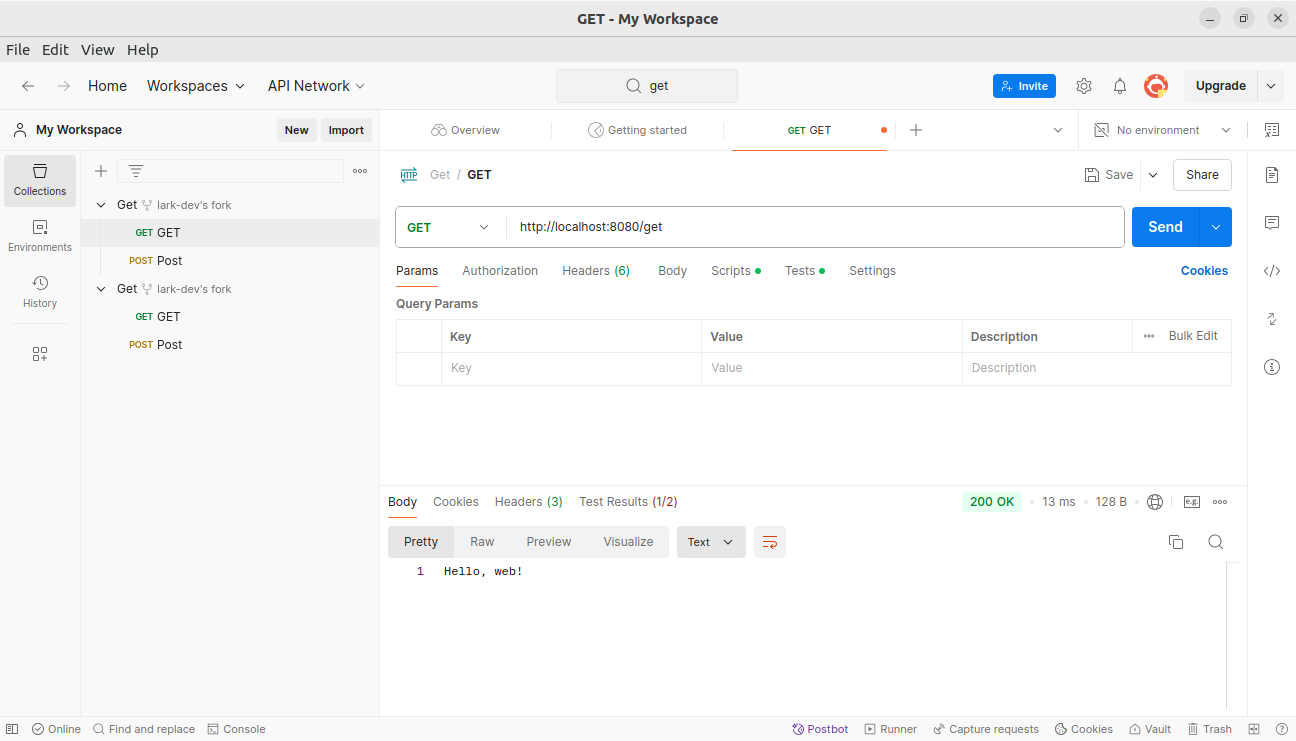
if err != nil {

fmt.Println("Ошибка запуска сервера", err)

}

}

Результаты тестирования программы представлены на рисунке 2.

Рисунок 2 - Тестирование программы для решения задачи 1

**Задача 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!

**порт** :9000

Код, решающий задачу выше представлен ниже.

package main

import (

"fmt"

"net/http" // пакет для поддержки HTTP протокола

)

func handler(w http.ResponseWriter, r \*http.Request) {

\_, err := w.Write([]byte("Hello," + r.URL.Query().Get("name") + "!"))

if err != nil {

panic(err)

}

}

func main() {

// здесь ваш код

http.HandleFunc("/api/user", handler)

err := http.ListenAndServe(":9000", nil)

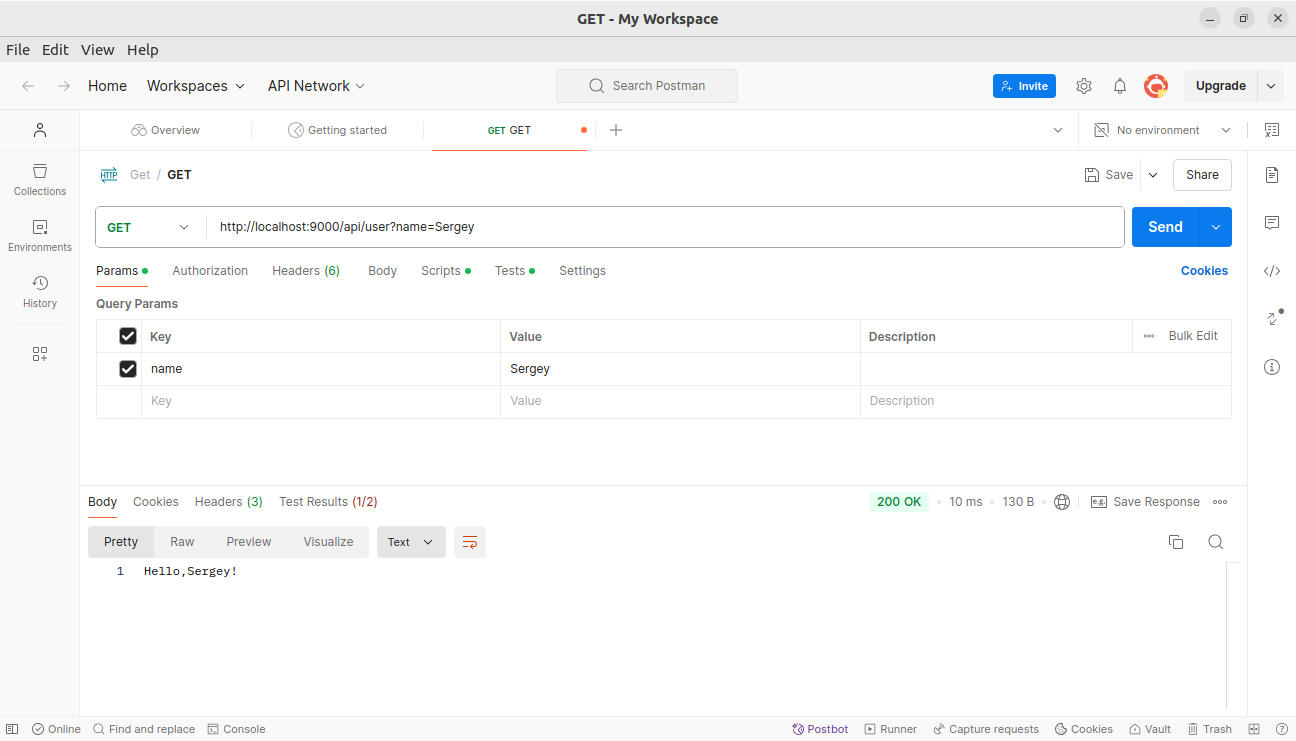
if err != nil {

fmt.Println("Ошибка запуска сервера: ", err)

}

}

Результаты тестирования программы представлены на рисунке 3.

Рисунок 3 - Тестирование программы для решения задачи 2

**Задача 3**

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).

Код, решающий задачу выше представлен ниже.

package main

// некоторые импорты нужны для проверки

import (

"fmt"

"net/http"

"strconv" // вдруг понадобиться вам ;)

)

var counter int = 0

func countHandler(w http.ResponseWriter, r \*http.Request) {

switch r.Method {

case http.MethodGet:

w.WriteHeader(http.StatusOK)

w.Write([]byte(strconv.Itoa(counter)))

case http.MethodPost:

err := r.ParseForm()

if err == nil {

countStr := r.FormValue("count")

if countStr == "" {

w.WriteHeader(http.StatusBadRequest)

w.Write([]byte("это не число"))

return

}

count, err := strconv.Atoi(countStr)

if err != nil {

w.WriteHeader(http.StatusBadRequest)

w.Write([]byte("это не число"))

return

}

counter += count

} else {

w.WriteHeader(http.StatusBadRequest)

return

}

default:

w.WriteHeader(http.StatusMethodNotAllowed)

w.Write([]byte("Метод не поддерживается"))

}

}

func main() {

http.HandleFunc("/count", countHandler)

err := http.ListenAndServe(":3333", nil)

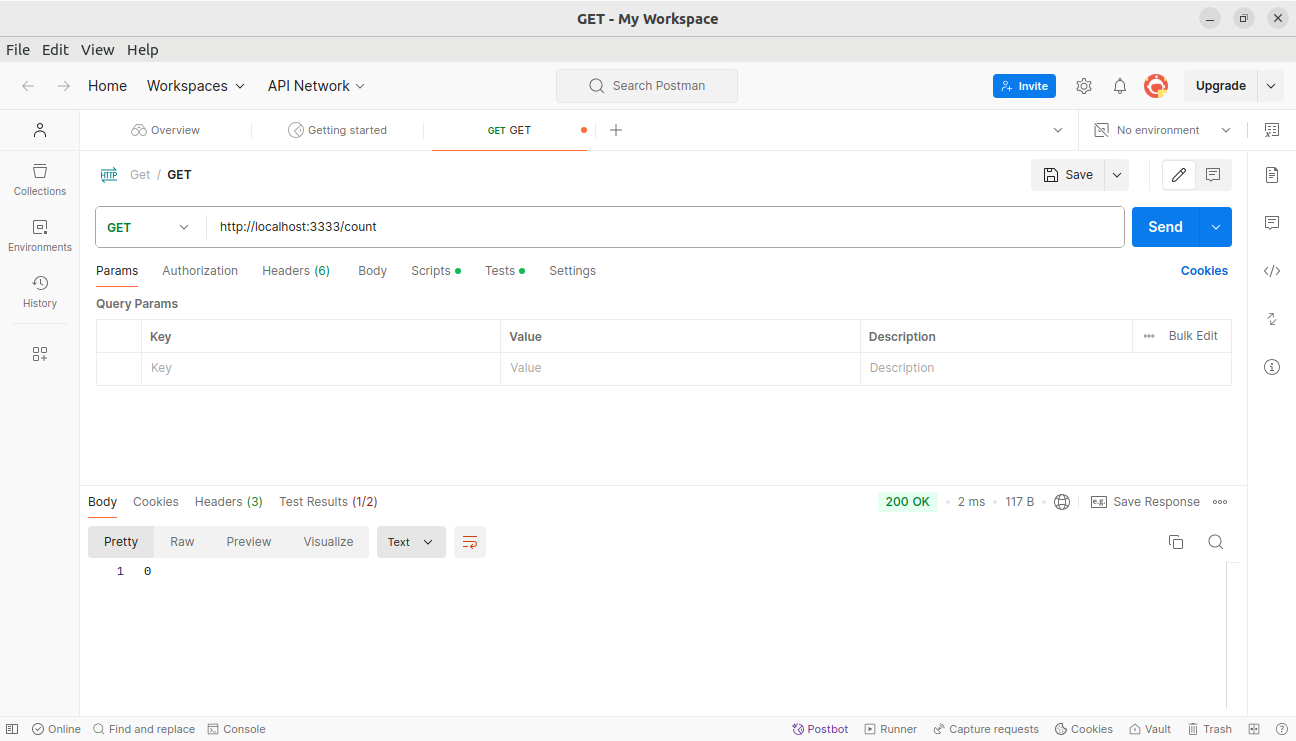
if err != nil {

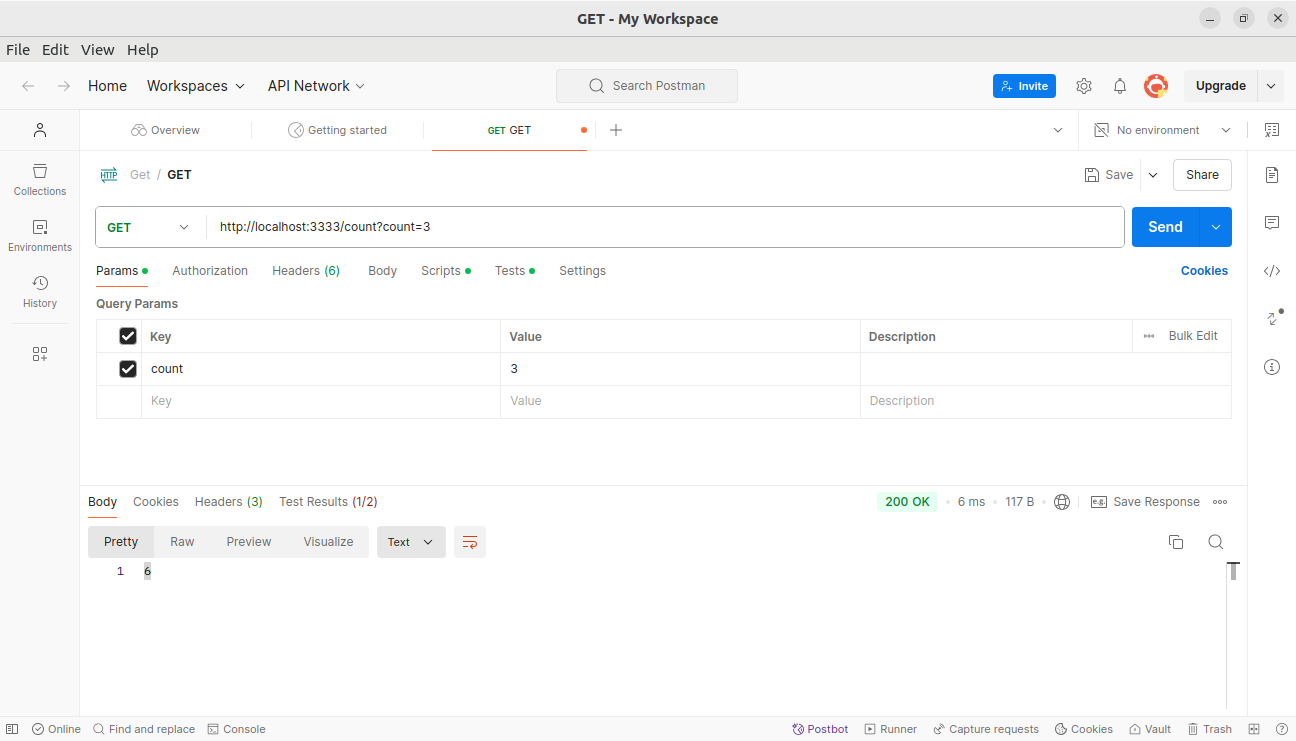
fmt.Println("Ошибка запуска сервера:", err)

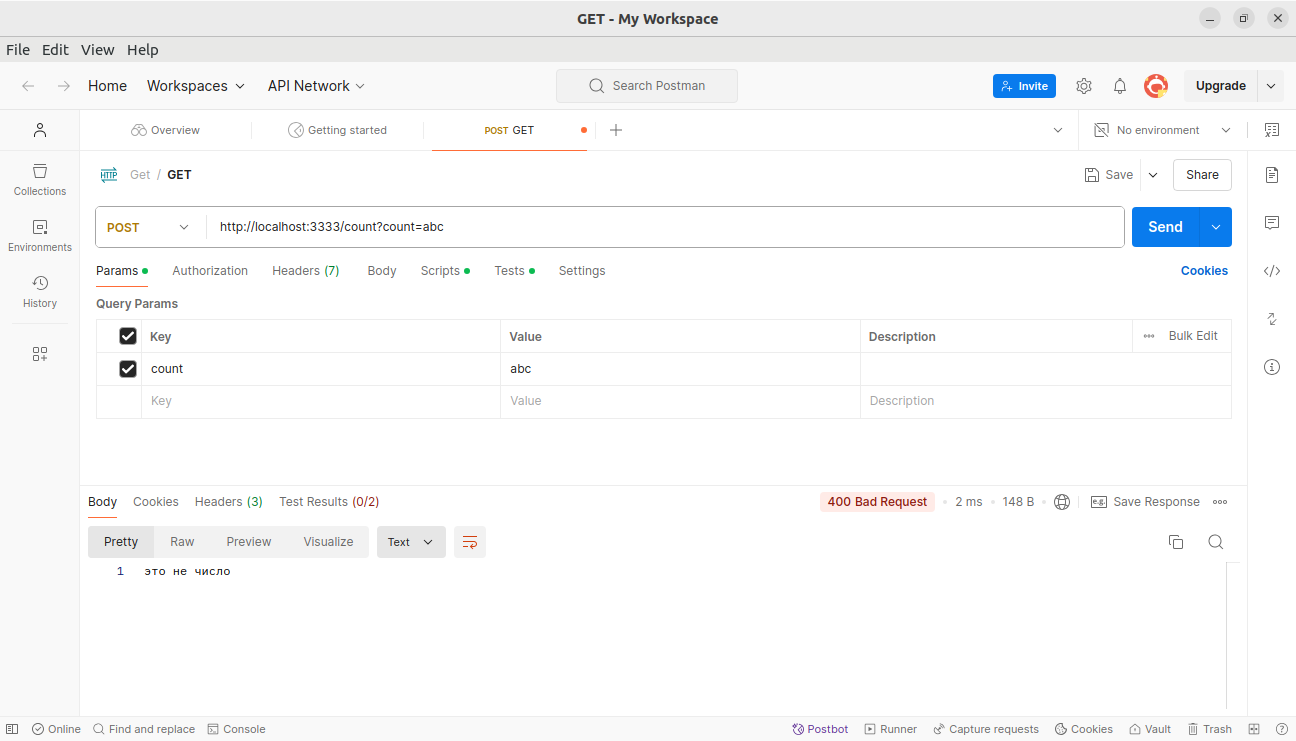
}

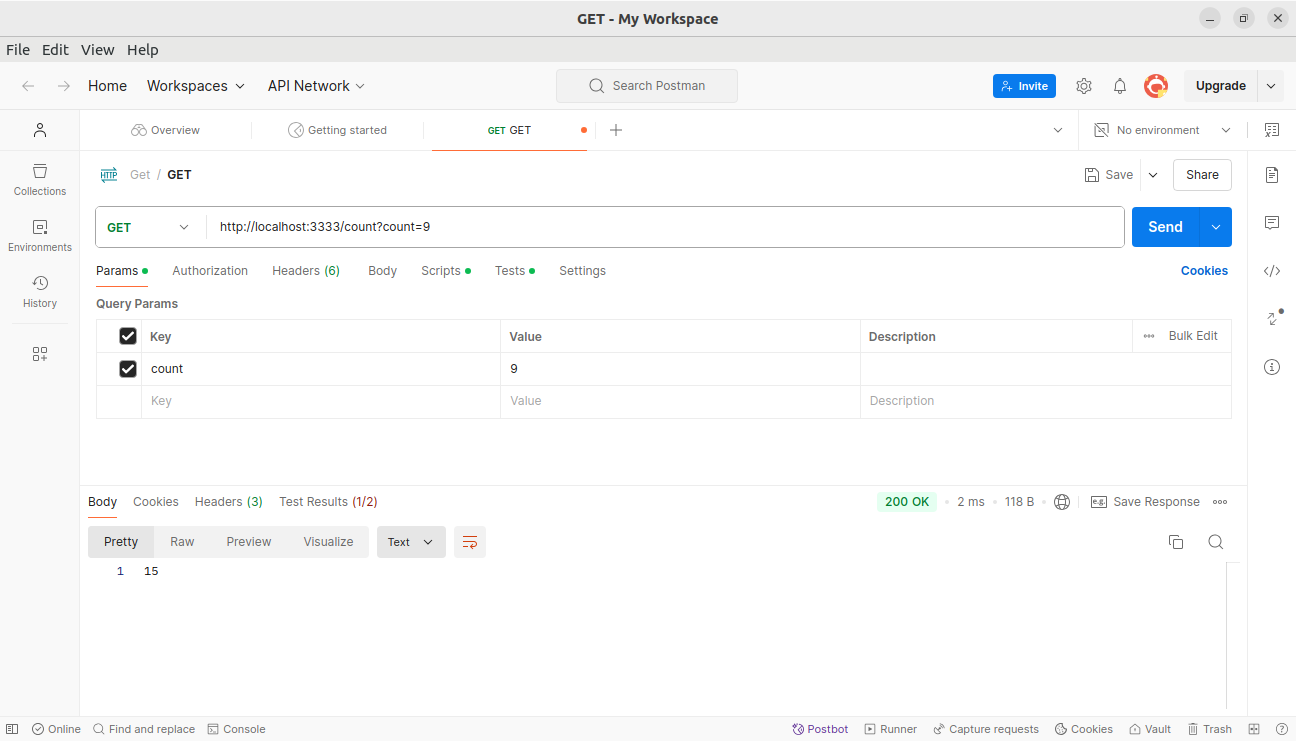
}

Результаты тестирования программы представлены на рисунках 4-7.

Рисунок 4 - Тестирование программы для решения задачи 3

Рисунок 5 - Тестирование программы для решения задачи 3

Рисунок 6 - Тестирование программы для решения задачи 3

Рисунок 7 - Тестирование программы для решения задачи 3

1. Были зафиксированы все изменения файлов исходного репозитория, сделаны необходимые коммиты и отправлено текущее состояние ветки dev в удалённый репозиторий GitHub. Далее через интерфейс Github был создан Pull Request dev --> master.

**Ответы на контрольные вопросы**

1. Разница между протоколами TCP и UDP:

* UDP (User Datagram Protocol) - ненадежный, без установления соединения протокол. В силу этого он не гарантирует доставку пакетов а также не контролирует их порядок. Применяется данный протокол для передачи данных, когда важна не надёжность, а скорость (передача видео).
* TCP (Transmission Control Protocol) - ориентированный на соединение, надежный протокол. Он гарантирует доставку данных, контроль потока и порядок пакетов (почтовые сообщения).

1. IP-адрес и номер порта веб-сервера:

* Port number (номер порта) - номер логического “канала” на хосте, используемый для идентификации приложения, которое отправляет и принимает сетевые пакеты. Это позволяет нескольким приложениям независимо друг от друга на одном хосте обмениваться данными.
* IP-адрес (Internet Protocol address) - уникальный идентификатор устройства в сети Интернет. Он позволяет маршрутизировать трафик до нужного устройства.

1. Методы HTTP, реализующие CRUD:

CRUD – create, read, update, delete

Create – POST

Read – GET

Update – PATCH/PUT (patch для частичного изменения, put создаёт новый ресурс или заменят представление целевого ресурса, данными, представленными в теле запроса).

Delete – DELETE

1. Группы кодов состояния HTTP-ответов:

* 1xx (Informational) - запрос принят, продолжается обработка
* 2xx (Success) - запрос успешно обработан (пример - код 200 StatusOk)
* 3xx (Redirection) - клиенту требуется выполнить дополнительные действия (пример - ошибка 301 Moved Permanently)
* 4xx (Client Error) - ошибка на стороне клиента (пример - ошибка 404 Not Found)
* 5xx (Server Error) - ошибка на строне сервера ( пример - ошибка 500 Internal Server Error)

5. Элементы HTTP-запроса и HTTP-ответа:

HTTP-запрос:

* Метод (GET, PUT, POST, DELETE...)
* URL
* Заголовки (Headers)
* Тело (Body)

HTTP-ответ:

- Версия протокола

- Код состояния (Status Code)

- Заголовки (Headers)

- Тело (Body)

**Заключение**

В рамках данной лабораторной работы было продолжено изучение Golang и произошло знакомство с набором стандартных библиотек, используемых для организации сетевого взаимодествия и разработки серверных приложений.

**Список использованных источников**

1. https://stepik.org/course/54403/info