



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

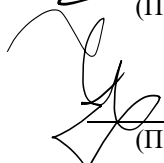
ИУ6-33Б

(Группа)

 21.10.2024
(Подпись, дата)

А.Д. Мартыненко
(И.О. Фамилия)

Преподаватель

 09.11.2024
(Подпись, дата)

В.Д. Шульман
(И.О. Фамилия)

Москва, 2024

1) Цель лабораторной работы

Изучение основ сетевого взаимодействия и серверной разработки с использованием языка Golang.

2) Задание

Продолжить изучение Golang, познакомиться с набором стандартных библиотек, используемых для организации сетевого взаимодействия и разработки серверных приложений.

3) Ход работы

1. *1_hello*. Напишите веб сервер, который по пути /get отдает текст "Hello, web!". Порт должен быть :8080.

Код на языке Go:

```
package main

import (
    "fmt"
    "net/http"
)

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", err)
    }
}
```

Результат:

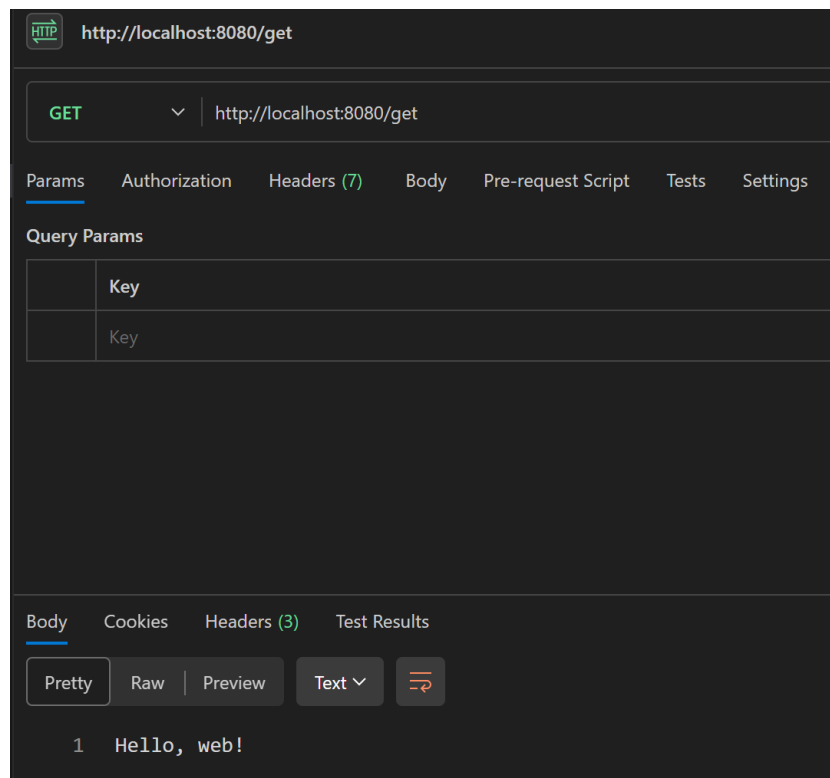


Рис. 1. Результат работы программы

2. *2_query*. 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 на языке Go:

```
package main

import (
    "fmt"
    "net/http"
)

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

```

    name := r.URL.Query().Get("name")
    if name == "" {
        name = "гость"
    }
    fmt.Fprintf(w, "Hello,%s!", name)
}

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

```

Результат:

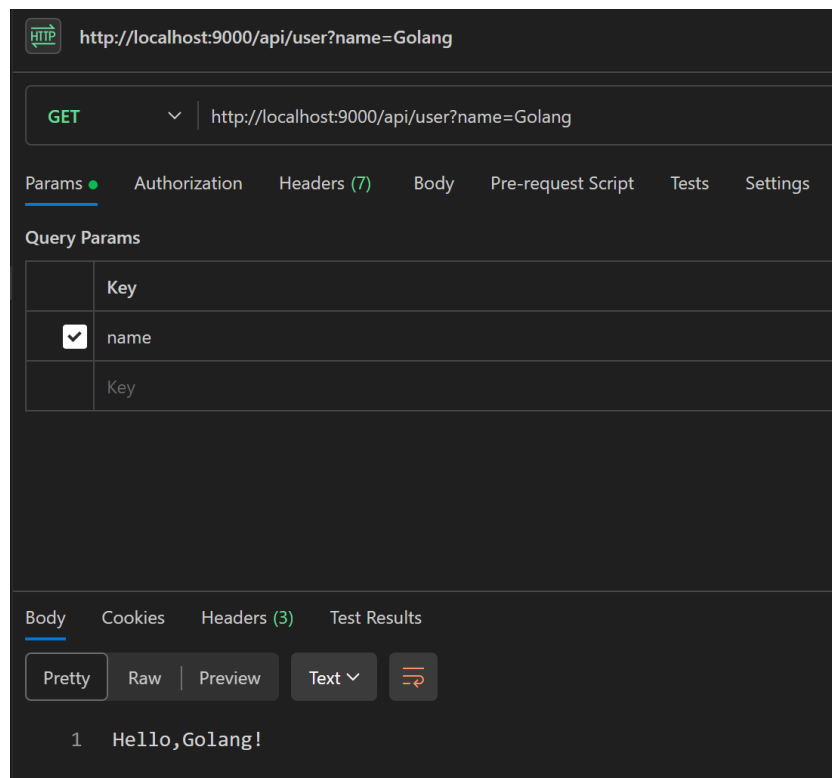


Рис. 2. Результат работы программы

3. *3_count*. 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 на языке Go:

```
package main

import (
    "fmt"
    "net/http"
    "strconv"
)

var k int

func handler(w http.ResponseWriter, r *http.Request) {
    r.ParseForm()
    if r.Method == "GET" {
        fmt.Fprintf(w, "%d", k)
    } else if r.Method == "POST" {
        delta := r.Form.Get("count")
        idelta, err := strconv.Atoi(delta)
        if err != nil {
            w.WriteHeader(http.StatusBadRequest)
            fmt.Fprintf(w, "это не число")
            return
        }
        k += idelta
        w.WriteHeader(http.StatusOK)
    } else {
        w.WriteHeader(http.StatusMethodNotAllowed)
        fmt.Fprintf(w, "Метод не поддерживается")
    }
}
```

```
func main() {
    http.HandleFunc("/", handler)
    err := http.ListenAndServe(":3333", nil)
    if err != nil {
        fmt.Println("Ошибка запуска сервера", err)
    }
}
```

Результат:

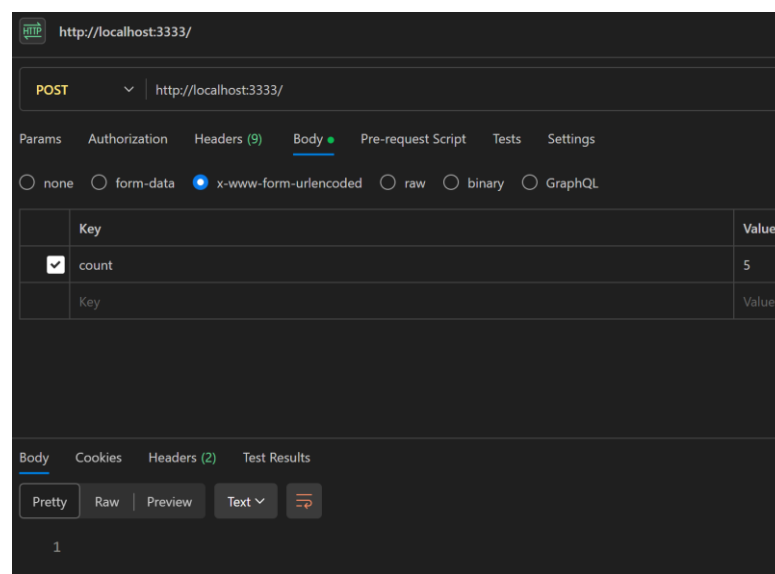


Рис. 3. Изменение счётчика посредством POST-запроса

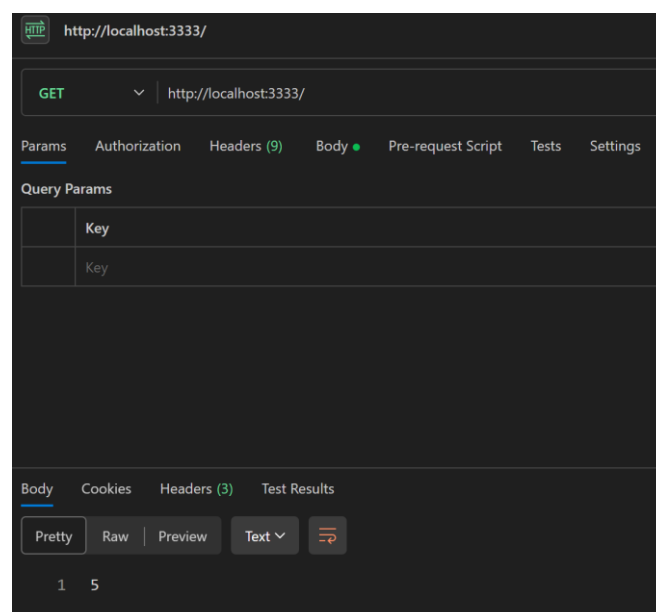


Рис. 4. Получение значения счётчика с сервера посредством GET-запроса

4) Контрольные вопросы

1. В чём разница между протоколами TCP и UDP?
2. Для чего нужны IP Address и Port Number у веб-сервера и в чём разница?
3. Какой набор методов в HTTP-request в полной мере реализует семантику CRUD?
4. Какие группы status code существуют у HTTP-response (желательно, с примерами)?
5. Из каких составных элементов состоит HTTP-request и HTTP-response?

Ответы:

1. TCP — это протокол с установлением соединения, который гарантирует надежную доставку данных. Перед передачей информации он устанавливает соединение между отправителем и получателем через процесс, известный как трёхстороннее рукопожатие. Это означает, что TCP обеспечивает подтверждение получения данных, повторную передачу потерянных пакетов и контроль целостности передаваемой информации. Благодаря этим механизмам TCP может адаптировать скорость передачи данных в зависимости от состояния сети, что делает его более надежным, но и более медленным.

UDP — это протокол без установления соединения. Он отправляет данные без предварительного рукопожатия, что делает его быстрее, но менее надежным. UDP не гарантирует доставку пакетов, и они могут быть потеряны или доставлены в неправильном порядке. Этот протокол не использует механизмы подтверждения или управления потоком, что позволяет ему работать с высокой скоростью, но при этом увеличивает риск потери данных.

2. IP-адрес — это уникальный числовой идентификатор, присвоенный каждому устройству в сети, который позволяет определить его местоположение и обеспечить маршрутизацию данных. Существует два

основных типа IP-адресов: IPv4, состоящий из четырех чисел, и более новый IPv6, который решает проблему исчерпания адресов.

Номер порта служит для идентификации конкретного приложения или службы на устройстве. Он позволяет нескольким приложениям использовать одно и то же сетевое соединение, избегая конфликтов. Например, веб-серверы обычно используют порт 80 для HTTP и порт 443 для HTTPS.

Таким образом, IP-адрес определяет устройство в сети, а номер порта — приложение на этом устройстве. Эти два компонента вместе обеспечивают эффективный обмен данными, позволяя идентифицировать не только устройства, но и конкретные процессы, с которыми необходимо взаимодействовать.

3. Семантика CRUD (Create, Read, Update, Delete) в контексте HTTP-запросов реализуется с помощью следующих методов:

1) Create — POST:

используется для создания нового ресурса на сервере.

2) Read — GET:

используется для получения данных о ресурсе или списка ресурсов.

3) Update — PUT и PATCH:

PUT используется для обновления существующего ресурса (полная замена);

PATCH: используется для частичного обновления существующего ресурса.

4) Delete — DELETE:

используется для удаления ресурса с сервера.

Таким образом, комбинация этих методов позволяет полностью реализовать операции CRUD в веб-приложениях.

4. 1) Информационные (1xx). Эти коды указывают на то, что запрос был получен и обрабатывается:

- **100 Continue:** Сервер принял начальную часть запроса и ожидает дальнейшие данные.
- **101 Switching Protocols:** Сервер принимает запрос на переключение протоколов.

2) **Успех (2xx).** Эти коды указывают на успешное выполнение запроса:

- **200 OK:** Запрос выполнен успешно.
- **201 Created:** Запрос выполнен, и ресурс был создан.
- **204 No Content:** Запрос выполнен, но нет содержимого для возврата.

3) **Перенаправление (3xx).** Эти коды указывают на необходимость дальнейших действий для завершения запроса:

- **301 Moved Permanently:** Запрашиваемый ресурс был перемещен на новый URL.
- **302 Found:** Запрашиваемый ресурс временно доступен по другому URL.
- **304 Not Modified:** Ресурс не изменился с последнего запроса.

4) **Ошибка клиента (4xx).** Эти коды указывают на ошибки, связанные с запросом клиента:

- **400 Bad Request:** Сервер не может обработать запрос из-за неверного синтаксиса.
- **401 Unauthorized:** Необходима аутентификация для доступа к ресурсу.
- **404 Not Found:** Запрашиваемый ресурс не найден.

5) **Ошибка сервера (5xx).** Эти коды указывают на ошибки, произошедшие на стороне сервера:

- **500 Internal Server Error:** Произошла ошибка на сервере, обработка запроса не удалась.
- **502 Bad Gateway:** Сервер, действующий как шлюз или прокси, получил недопустимый ответ от вышестоящего сервера.
- **503 Service Unavailable:** Сервер временно недоступен (например, из-за перегрузки или технического обслуживания).

5. HTTP-запрос начинается с метода, который указывает, какое действие клиент хочет выполнить, например, GET для получения данных или POST для их отправки. Затем идет URL, который представляет собой адрес ресурса на сервере. Важным элементом является версия протокола, например, HTTP/1.1 или HTTP/2. Дополнительно запрос может содержать заголовки, которые предоставляют дополнительную информацию о запросе, такие как тип содержимого или данные для аутентификации. В некоторых случаях запрос включает тело, где могут находиться данные, отправляемые на сервер.

HTTP-ответ состоит из версии протокола, статуса, который информирует о результате обработки запроса (например, код 200 означает успешное выполнение), и статус-фразы, которая дает краткое описание этого статуса. Ответ также содержит заголовки, которые могут указывать на тип данных, возвращаемых в теле ответа, и другие важные сведения. Наконец, тело ответа включает основное содержимое, которое может быть представлено в различных форматах, таких как HTML или JSON.

5) Заключение

В процессе выполнения лабораторной работы были изучены основы Back-End разработки на Golang, а также получены практические навыки написания программ в данной области программирования.

6) Использованные источники

- <https://github.com/ValeryBMSTU/web-6>
- <https://stepik.org/course/54403/info>