



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

ИУ6-33Б

(Группа)

06.09.2024

(Подпись, дата)

Н.Н. Товарас

(И.О. Фамилия)

Преподаватель

14.09.2024

(Подпись, дата)

В.Д. Шульман

(И.О. Фамилия)

Москва, 2024

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

В рамках данной лабораторной работы предлагается продолжить изучение Golang и познакомиться с набором стандартных библиотек, используемых для организации сетевого взаимодействия и разработки серверных приложений.

Ход работы:

Задание 1:

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

Порт должен быть :8080.

Код программы:

```
package main

import (
    "fmt"
    "net/http"
)

func handler(w http.ResponseWriter, r *http.Request) {
    fmt.Fprintf(w, "Hello,web!")
}

func main() {
    // Регистрация обработчика по пути /get
    http.HandleFunc("/get", handler)
```

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

if err != nil {

    fmt.Println(err)

}

}
```

Результат работы:

The screenshot shows a web browser interface with the address bar displaying `http://localhost:8080/get`. The browser has tabs for Params, Authorization, Headers (8), Body (selected), Pre-request Script, Tests, and Settings. The Body tab shows a table with columns Key, Value, and Bulk Edit. The table has one row with the value 'Hello, web!'. The status bar at the bottom indicates a 200 OK response with a 2 ms response time and 127 B of data. The response is saved and can be viewed in Pretty, Raw, Preview, or Visualize format. The response body is displayed as 'Hello, web!'.

Рис.1 — Результат 1

Задание 2:

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

Код программы:

```
package main

import (
    "fmt"
    "net/http"
)

func main() {
    http.HandleFunc("/api/user", userHandler)

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

    if err != nil {
        fmt.Println(err)
    }
}

func userHandler(w http.ResponseWriter, r *http.Request) {
    name := r.URL.Query().Get("name")

    if name == "" {
        name = "Guest"
    }

    fmt.Fprintf(w, "Hello,%s!", name)
```

}

Результат работы:

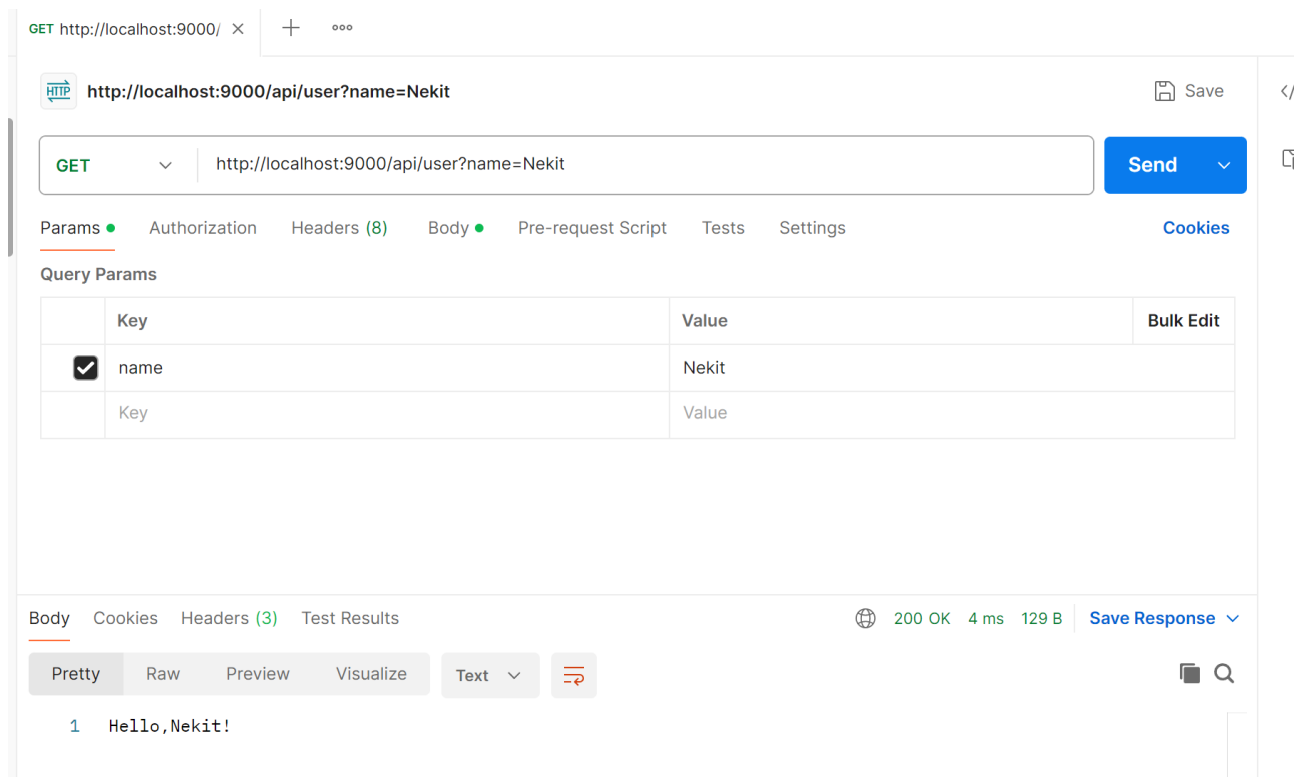


Рис.2 — Результат 2

Задание 3:

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)`.

Код программы:

```
package main
```

```
import (
```

```
    "fmt"
```

```
    "net/http"
```

```
    "strconv"
```

```
)
```

```
var counter int = 0
```

```
func main() {
```

```
    http.HandleFunc("/count", counthHandler)
```

```
    http.ListenAndServe(":3333", nil)
```

```
}
```

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

```
    switch r.Method {
```

```
    case http.MethodGet:
```

```
        fmt.Fprintf(w, "%d", counter)
```

```
    case http.MethodPost:
```

```
        err := r.ParseForm()
```

```
        if err != nil {
```

```
            fmt.Println(err)
```

```
        }
```

```
        countstr := r.FormValue("count")
```

```
        count, err := strconv.Atoi(countstr)
```

```
        if err != nil {
```

```
            http.Error(w, "это не число", http.StatusBadRequest)
```

```
            return
```

```
        }
```

```
        counter += count
```

default:

```
http.Error(w, "Метод не поддерживается", http.StatusMethodNotAllowed)
```

```
}
```

```
}
```

Результат работы:

POST ⌵ http://localhost:3333/count?count=10 Send ⌵

Рис.3 — Отправляем 10

Body Cookies Headers (4) Test Results 400 Bad Request 2 ms 182 B Save Response ⌵

Pretty Raw Preview Visualize Text ⌵ ⌵

1	это не число
2	

Рис.4 — Отправляем не число

GET ⌵ http://localhost:3333/count Send ⌵

Params Authorization Headers (8) Body ● Pre-request Script Tests Settings Cookies

Query Params

	Key	Value	Bulk Edit
	Key	Value	

Body Cookies Headers (3) Test Results 200 OK 3 ms 118 B Save Response ⌵

Pretty Raw Preview Visualize Text ⌵ ⌵

1	10
---	----

Рис.5 — Получаем значение

Вывод: Я научился работать с http запросами на golang.