



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 Компьютерные системы и сети

**О Т Ч Е Т**

**по лабораторной работе № 6**

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

**Дисциплина:** Языки интернет-программирования

Студент	<u>ИУ6-32Б</u> (Группа)	<u>11.11.2024</u> (Подпись, дата)	<u>Г.Д. Юдаков</u> (И.О. Фамилия)
Преподаватель		<u>11.11.2024</u> (Подпись, дата)	<u>В.Д. Шульман</u> (И.О. Фамилия)

Москва, 2024

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

Ход работы.

1. Ознакомились с курсом <https://stepik.org/course/54403/info>
2. Сделали форк данного репозитория в GitHub, клонировали получившуюся копию локально, создали от мастера ветку дев и переключились на нее:

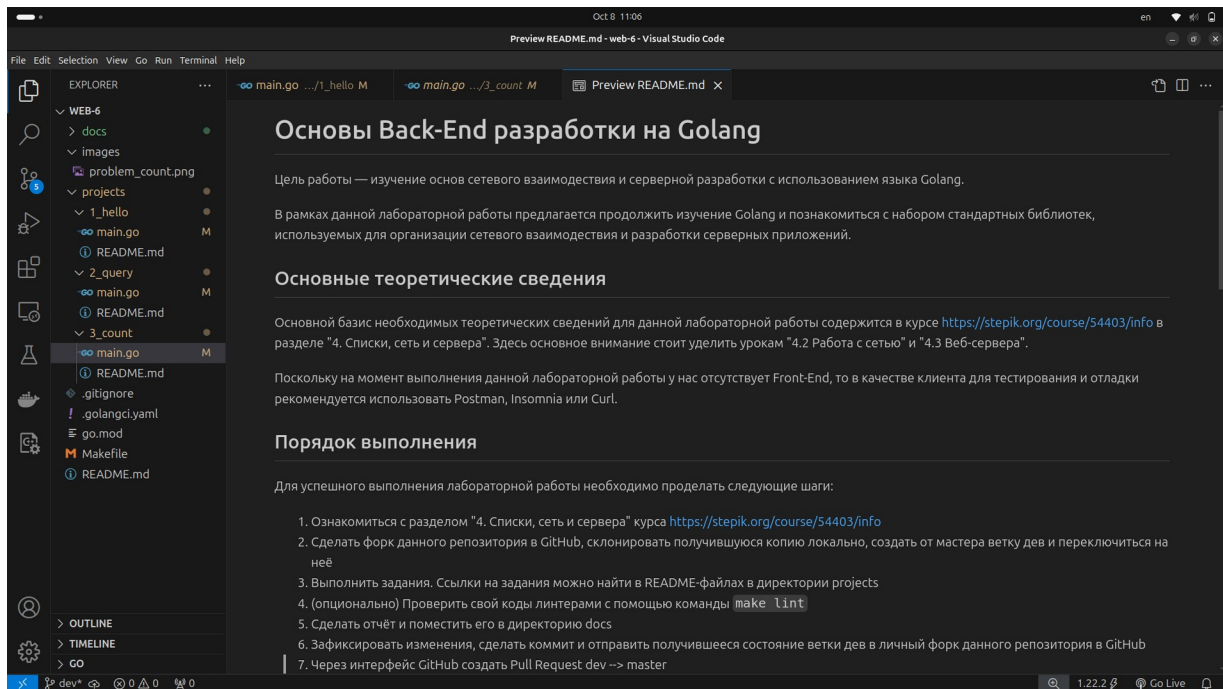


Рисунок 1 — Скопированный репозиторий

3. Написали 3 HTTP сервера на GoLang. Код серверов и результаты запросов в Postman прикрепили ниже:

**Задача 1(Вывод строки):**

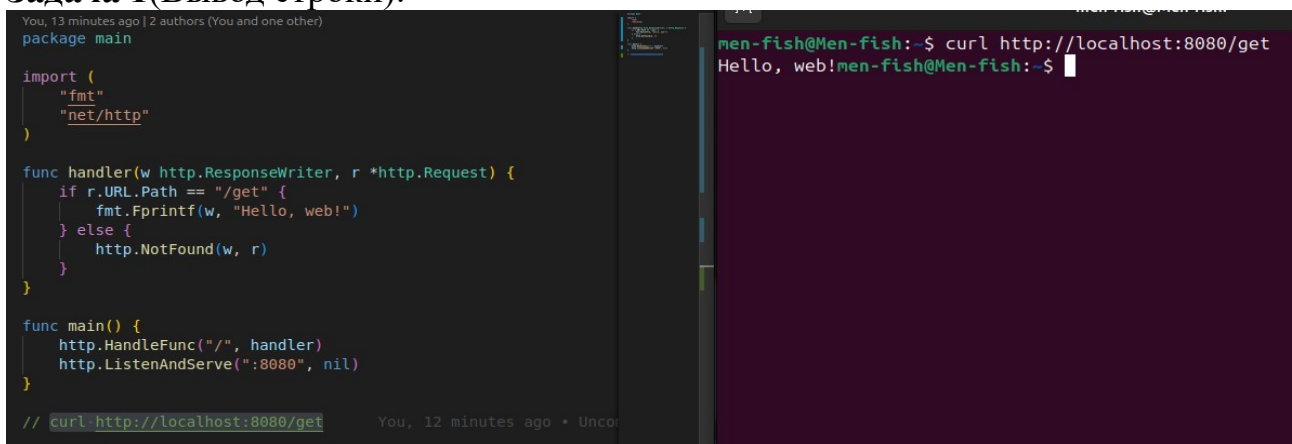
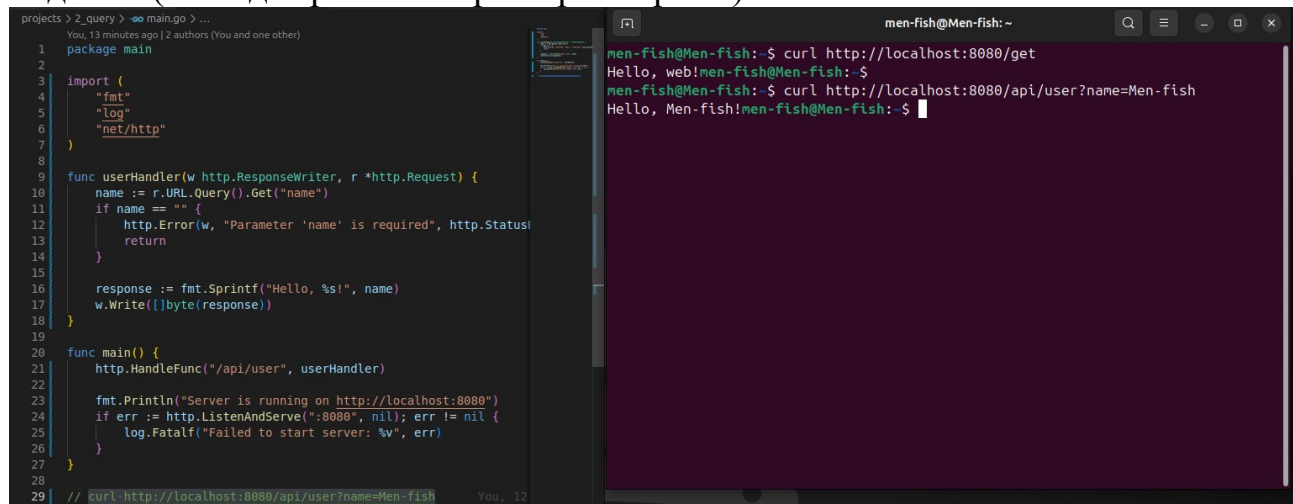


Рисунок 2 — Вывод задачи 1

## Задача 2(Вывод строки с квери парамеирами):



The screenshot shows a Go IDE with a file named `main.go` and a terminal window. The code in `main.go` is a simple web server that listens on `localhost:8080` and responds to `GET` requests at `/api/user` with a greeting. The terminal shows the server running and two `curl` commands being executed to test the endpoint.

```
projects > 2_query > main.go > ...
You, 13 minutes ago | 2 authors (You and one other)
package main

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

func userHandler(w http.ResponseWriter, r *http.Request) {
    name := r.URL.Query().Get("name")
    if name == "" {
        http.Error(w, "Parameter 'name' is required", http.StatusBadRequest)
        return
    }
    response := fmt.Sprintf("Hello, %s!", name)
    w.Write([]byte(response))
}

func main() {
    http.HandleFunc("/api/user", userHandler)
    fmt.Println("Server is running on http://localhost:8080")
    if err := http.ListenAndServe(":8080", nil); err != nil {
        log.Fatalf("Failed to start server: %v", err)
    }
}

// curl http://localhost:8080/api/user?name=Men-fish
You, 12
```

```
men-fish@Men-fish:~$ curl http://localhost:8080/get
Hello, web!men-fish@Men-fish:~$
men-fish@Men-fish:~$ curl http://localhost:8080/api/user?name=Men-fish
Hello, Men-fish!men-fish@Men-fish:~$
```

Рисунок 3 — Вывод задачи 2

## Задача 3(Store):

```
package main

import (
    "encoding/json"
    "fmt"
    "io"
    "log"
    "net/http"
    "sync"
)

var (
    counter int
    mu sync.Mutex
)

func getCountHandler(w http.ResponseWriter, r *http.Request) {
    mu.Lock()
    defer mu.Unlock()
    fmt.Fprintf(w, "Current count: %d", counter)
}

func postCountHandler(w http.ResponseWriter, r *http.Request) {
    body, err := io.ReadAll(r.Body)
    if err != nil {
        http.Error(w, "Unable to read request body", http.StatusBadRequest)
        return
    }
    defer r.Body.Close()
    var data map[string]interface{}
    if err := json.Unmarshal(body, &data); err != nil {
        http.Error(w, "Invalid JSON format", http.StatusBadRequest)
        return
    }
}
```

```

countValue, ok := data["count"]
if !ok {
    http.Error(w, "Key 'count' not found", http.StatusBadRequest)
    return
}
countFloat, ok := countValue.(float64)
if !ok {
    http.Error(w, "это не число", http.StatusBadRequest)
    return
}
countInt := int(countFloat)
mu.Lock()
counter += countInt
mu.Unlock()
fmt.Fprintf(w, "Count incremented by %d, new count: %d", countInt, counter)
}

func main() {
    http.HandleFunc("/count", func(w http.ResponseWriter, r *http.Request) {
        switch r.Method {
        case http.MethodGet:
            getCountHandler(w, r)
        case http.MethodPost:
            postCountHandler(w, r)
        default:
            http.Error(w, "Method not allowed", http.StatusMethodNotAllowed)
        }
    })
    fmt.Println("Server is running on http://localhost:3333")
    if err := http.ListenAndServe(":3333", nil); err != nil {
        log.Fatalf("Failed to start server: %v", err)
    }
}

// curl -X POST -H "Content-Type: application/json" -d '{"count": 5}'
http://localhost:3333/count
// curl http://localhost:3333/count

```

The screenshot displays a Go IDE with a source editor on the left and a terminal on the right. The source editor shows a Go program for a simple web server. The `postCountHandler` function increments a counter and returns a JSON response. The `main` function sets up the server to listen on `localhost:3333`. The terminal on the right shows the execution of the server and several `curl` commands that interact with the `/count` endpoint. The output shows the counter being incremented from 5 to 15. The bottom status bar indicates the project is named `go-3_count`.

```
24 func postCountHandler(w http.ResponseWriter, r *http.Request) {
46     http.Error(w, "это не число", http.StatusBadRequest)
47     return
48 }
49 countInt := int(countFloat)
50 mu.Lock()
51 counter += countInt
52 mu.Unlock()
53
54 fmt.Fprintf(w, "Count incremented by %d, new count: %d", countInt, counter)
55 }
56
57 func main() {
58     http.HandleFunc("/count", func(w http.ResponseWriter, r *http.Request) {
59         switch r.Method {
60             case http.MethodGet:
61                 getCountHandler(w, r)
62             case http.MethodPost:
63                 postCountHandler(w, r)
64             default:
65                 http.Error(w, "Method not allowed", http.StatusMethodNotAllowed)
66         }
67     })
68
69     fmt.Println("Server is running on http://localhost:3333")
70     if err := http.ListenAndServe(":3333", nil); err != nil {
71         log.Fatalf("Failed to start server: %v", err)
72     }
73 }
74
75 // curl -X POST -H "Content-Type: application/json" -d '{"count": 5}' http://localhost:3333/count
76
```

men-fish@Men-fish:~\$ curl -X POST -H "Content-Type: application/json" -d '{"count": 5}' http://localhost:3333/count  
Count incremented by 5, new count: 5  
men-fish@Men-fish:~\$ curl -X POST -H "Content-Type: application/json" -d '{"count": 5}' http://localhost:3333/count  
Count incremented by 5, new count: 10  
men-fish@Men-fish:~\$ curl -X POST -H "Content-Type: application/json" -d '{"count": 5}' http://localhost:3333/count  
Count incremented by 5, new count: 15  
men-fish@Men-fish:~\$ curl -X POST -H "Content-Type: application/json" -d '{"count": dfdsfsdf}' http://localhost:3333/count  
Invalid JSON format  
men-fish@Men-fish:~\$ curl http://localhost:3333/count  
Current count: 15  
men-fish@Men-fish:~\$

men-fish@Men-fish:~/web-6/projects/2\$query\$ cd /home/men-fish/web-6/projects/3\_count  
men-fish@Men-fish:~/web-6/projects/3\_count\$ go run main.go  
Server is running on http://localhost:3333

Рисунок 4 — Вывод к задаче 3

4. Зафиксировали изменения, сделали коммит и отправили полученное состояние ветки dev в удаленный репозиторий GitHub. Через интерфейс GitHub создали Pull Request dev --> master

Заключение: в ходе лабораторной работы изучили основы сетевого взаимодействия и серверной разработки с использованием языка Golang. Освоили работу с Postman.

### Контрольные вопросы:

1. Разница между протоколами TCP и UDP:

- TCP (Transmission Control Protocol) - надежный, ориентированный на соединение протокол. Он обеспечивает гарантированную доставку данных, контроль потока и порядка пакетов.
- UDP (User Datagram Protocol) - ненадежный, без установления соединения протокол. Он не гарантирует доставку пакетов и не контролирует их порядок. Применяется для передачи данных, где важна скорость, а не надежность (например, потоковое видео).

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

- IP-адрес (Internet Protocol address) - уникальный идентификатор устройства в сети Интернет. Он позволяет маршрутизировать трафик до нужного устройства.
- Номер порта (Port number) - номер логического "канала" на хосте, используемый для идентификации приложения, принимающего и отправляющего сетевые пакеты. Это позволяет нескольким приложениям на одном хосте обмениваться данными независимо.

3. Методы HTTP, реализующие CRUD:

- Create (POST)
- Read (GET)
- Update (PUT/PATCH)
- Delete (DELETE)

#### 4. Группы кодов состояния HTTP-ответов:

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

#### 5. Элементы HTTP-запроса и HTTP-ответа:

HTTP-запрос:

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

HTTP-ответ:

- Версия протокола
- Код состояния (Status Code)
- Заголовки (Headers)
- Тело (Body)