|  |  |
| --- | --- |
| 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 | ИУ6-31БВ |  |  | В.О. Бокова |
|  | (Группа) |  | (Подпись, дата) | (И.О. Фамилия) |
|  |  |  |  |  |
| Преподаватель |  |  |  | В.Д. Шульман |
|  |  |  | (Подпись, дата) | (И.О. Фамилия) |

Москва, 2024

**Цель работы**— изучение основ сетевого взаимодествия и серверной разработки с

использованием языка Golang.

**Порядок выполнения**

1. 2. 3. 4. 5. 6. 7. 8. Ознакомиться с разделом "4. Списки, сеть и сервера" курса

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

Сделать форк данного репозитория в GitHub, склонировать получившуюся копию

локально, создать от мастера ветку дев и переключиться на неё

Выполнить задания. Ссылки на задания можно найти в README-файлах в директории

projects

(опционально) Проверить свой коды линтерами с помощью команды make lint

Сделать отчёт и поместить его в директорию docs

Зафиксировать изменения, сделать коммит и отправить получившееся состояние ветки

дев в личный форк данного репозитория в GitHub

Через интерфейс GitHub создать Pull Request dev --> master

На защите лабораторной работы продемонстрировать открытый Pull Request. PR должен

быть направлен в master ветку форка, а не исходного репозитория

**Ход работы:**

**A) Задание «1\_hello»**

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

Порт должен быть :8080.

Решение:

package main

import (

"fmt"

"net/http"

)

// Обработчик HTTP-запросов

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

w.Write([]byte("Hello world!"))

}

func main() {

// Регистрируем обработчик для пути "/"

http.HandleFunc("/", handler)

// Запускаем веб-сервер на порту 8080

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

if err != nil {

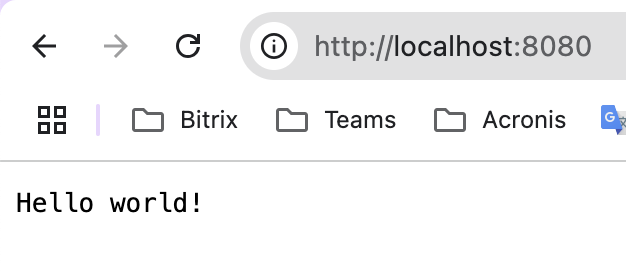
fmt.Println("Ошибка запуска сервера:", err)

}

}

Ссылка: http://localhost:8080/

**Тестирование:**



**B) Задание «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

Решение:

package main

import (

"fmt"

"net/http"

)

func main() {

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

fmt.Println("Server listening on port 9000")

http.ListenAndServe(":9000", nil)

}

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

query := r.URL.Query()

name := query.Get("name")

if name == "" {

fmt.Fprintln(w, "Please provide a name using the 'name' parameter in the URL.")

return

}

// Используйте переданное имя, а не "Salviya"

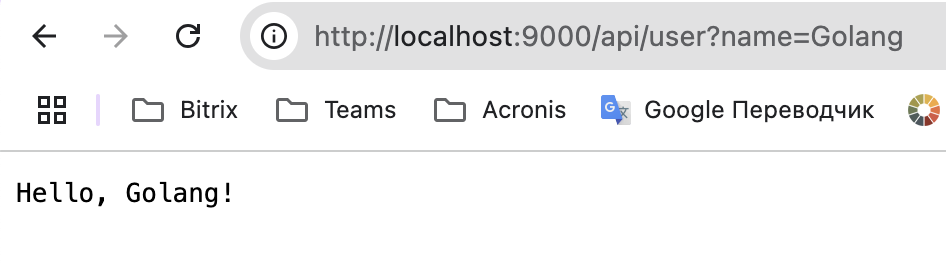
greeting := fmt.Sprintf("Hello, %s!", name)

fmt.Fprintln(w, greeting)

}

Ссылка: http://localhost:9000/api/user?name=Golang

**Тестирование:**



**C) Задание «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).

Решение:

package main

import (

"fmt"

"net/http"

"strconv"

"sync"

)

// Переменная для счетчика

var counter int

var mu sync.Mutex // Для синхронизации доступа к счетчику

func main() {

// Обработчик GET и POST запросов

http.HandleFunc("/count", countHandler)

// Запуск сервера nа порту 3333

fmt.Println("Server is running on port 3333...")

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

if err != nil {

fmt.Println("Error starting server:", err)

}

}

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

switch r.Method {

case "GET":

// Возвращаем текущее значение счетчика

mu.Lock()

fmt.Fprintf(w, "Counter: %d", counter)

mu.Unlock()

case "POST":

// Читаем значение "count" из формы

err := r.ParseForm()

if err != nil {

fmt.Println("Error parsing form: ", err)

http.Error(w, "Failed to parse form", http.StatusBadRequest)

return

}

if err != nil {fmt.Println("Invalid count value: ", err)

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

return

}

// Получаем значение из формы

countStr := r.FormValue("count")

// Конвертируем строку в число

count, err := strconv.Atoi(countStr)

if err != nil {

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

return

}

// Увеличиваем счетчик

mu.Lock()

counter += count

mu.Unlock()

// Возвращаем обновленное значение счетчика

fmt.Fprintf(w, "Counter updated: %d", counter)

default:

// Если метод не поддерживается

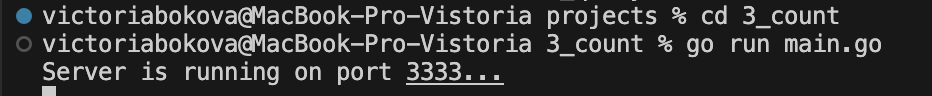
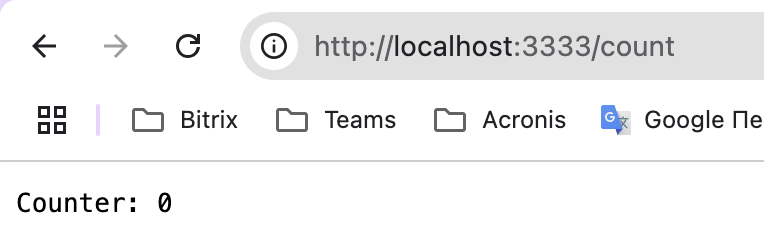
http.Error(w, "Method not allowed", http.StatusMethodNotAllowed)

}

}

Ссылка: http://localhost:3333/count

**Тестирование:**

****

**Вывод:**

Изучил основы сетевого взаимодействия и серверной разработки с

использованием языка Golang.

Список использованных источников:

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