|  |  |
| --- | --- |
| 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)**

**Отчет**

**по лабораторной работе № 10**

**Дисциплина:** Языки интернет-программирования

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

Москва, 2024

Цель работы: получение первичных навыков организации кодовой базы проекта на Golang

В рамках данной лабораторной работы предлагается ознакомиться с набором рекомендаций для разработки поддерживаемых и расширяемых backend-сервисов на golang

Вот архитектура приложения. Она будет использоваться во всех задачах.

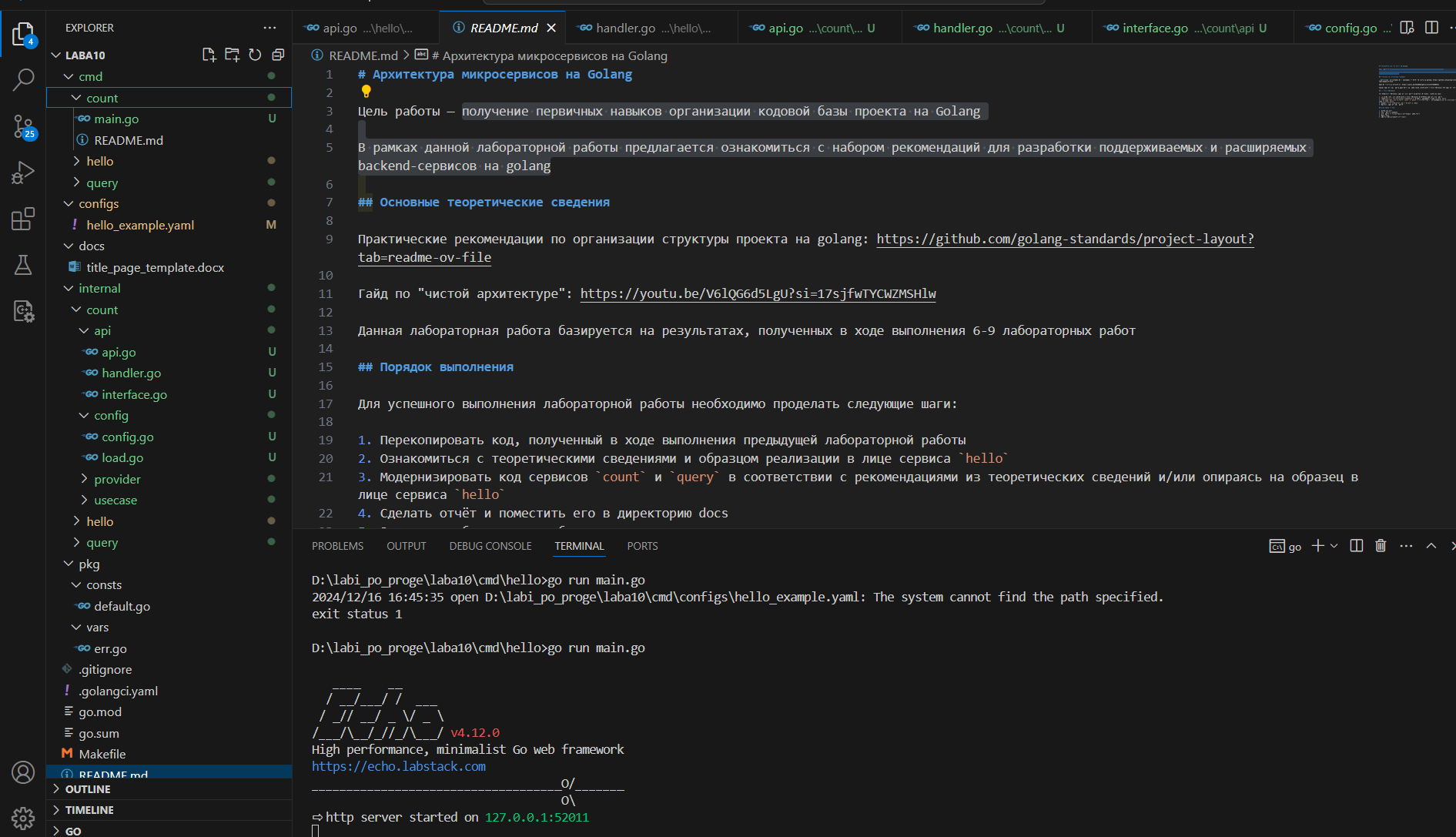


Рисунок 1 – архитектура программы

Задача 1 (hello):

Создать архитектуру и переписать под нее код из 9 лабораторной для соответствующей задачи.

База данных была уже создана ранее.

Часть кода программы (представлен только код функции main()):

func main() {

    package main

import (

    "flag"

    "log"

    "github.com/ValeryBMSTU/web-10/internal/hello/api"

    "github.com/ValeryBMSTU/web-10/internal/hello/config"

    "github.com/ValeryBMSTU/web-10/internal/hello/provider"

    "github.com/ValeryBMSTU/web-10/internal/hello/usecase"

    \_ "github.com/lib/pq"

)

func main() {

    // Считываем аргументы командной строки

    configPath := flag.String("config-path", "../../configs/hello\_example.yaml", "путь к файлу конфигурации")

    flag.Parse()

    cfg, err := config.LoadConfig(\*configPath)

    if err != nil {

        log.Fatal(err)

    }

    prv := provider.NewProvider(cfg.DB.Host, cfg.DB.Port, cfg.DB.User, cfg.DB.Password, cfg.DB.DBname)

    use := usecase.NewUsecase(cfg.Usecase.DefaultMessage, prv)

    srv := api.NewServer(cfg.IP, cfg.Port, cfg.API.MaxMessageSize, use)

    srv.Run()

}

Запуск программы:

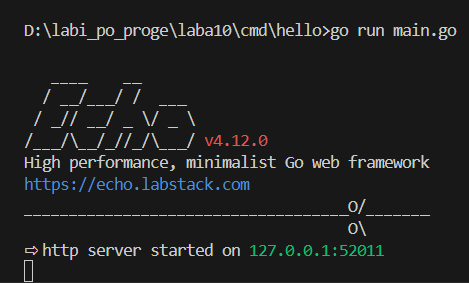


Рисунок 2 – запуск сервера

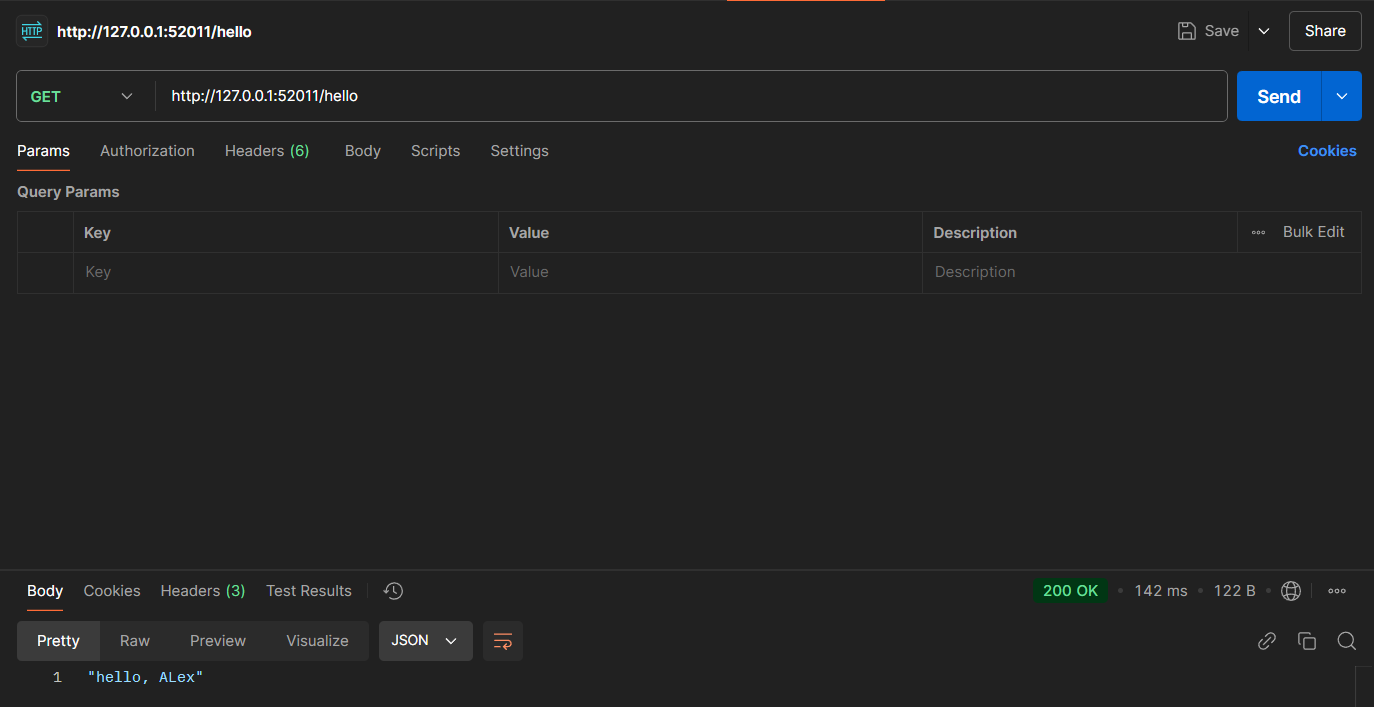


Рисунок 3 – отправка сообщений через postman

Задача 2 (count):

Создать архитектуру и переписать под нее код из 9 лабораторной для соответствующей задачи.

База данных была уже создана ранее.

Часть кода программы (представлен только код функции main()):

package main

import (

    "flag"

    "log"

    "github.com/ValeryBMSTU/web-10/internal/count/api"

    "github.com/ValeryBMSTU/web-10/internal/count/config"

    "github.com/ValeryBMSTU/web-10/internal/count/provider"

    "github.com/ValeryBMSTU/web-10/internal/count/usecase"

    \_ "github.com/lib/pq"

)

func main() {

    // Считываем аргументы командной строки

    configPath := flag.String("config-path", "configs/hello\_example.yaml", "путь к файлу конфигурации")

    flag.Parse()

    cfg, err := config.LoadConfig(\*configPath)

    if err != nil {

        log.Fatal(err)

    }

    prv := provider.NewProvider(cfg.DB.Host, cfg.DB.Port, cfg.DB.User, cfg.DB.Password, cfg.DB.DBname)

    use := usecase.NewUsecase(cfg.Usecase.DefaultMessageCount, prv)

    srv := api.NewServer(cfg.IP, cfg.Port, cfg.API.MaxMessageSize, use)

    srv.Run()

}

Получим следующий результат:

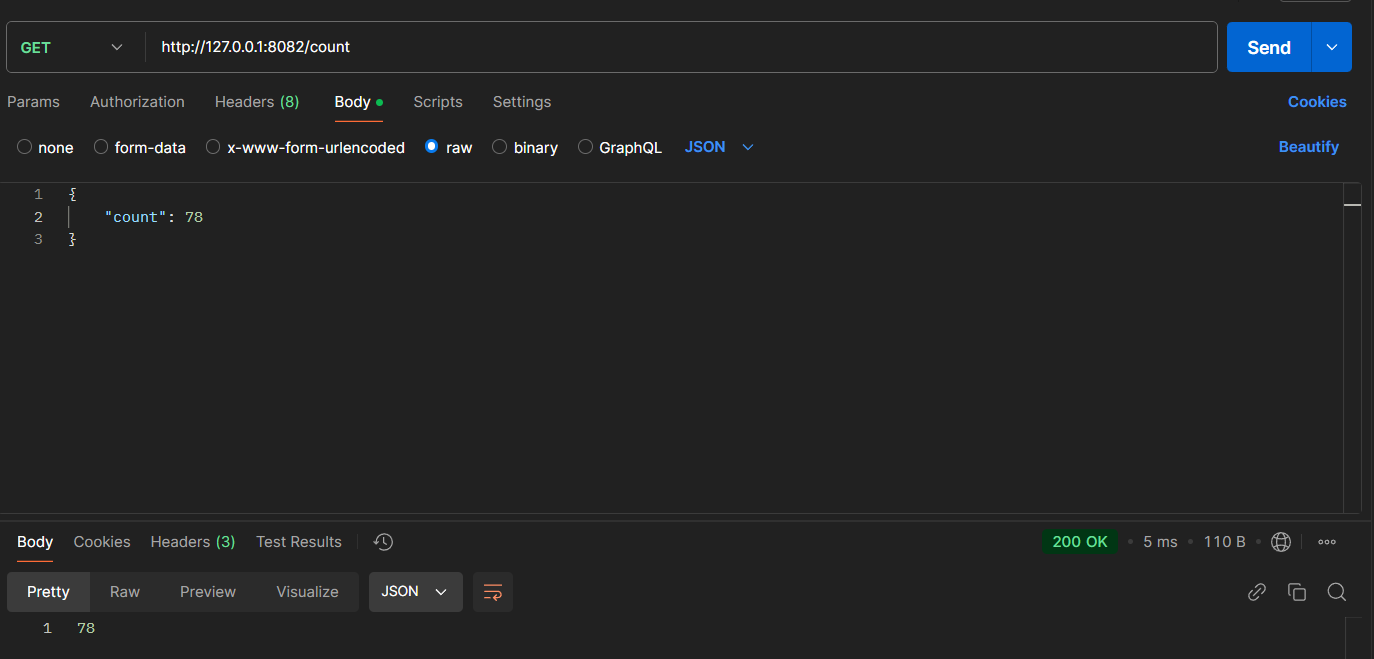


Рисунок 4 - Начальное значение в бд при запросе GET (Оно не равно 0, так я сам уже проводил тесты)

Теперь увеличим его с помощью команды POST:

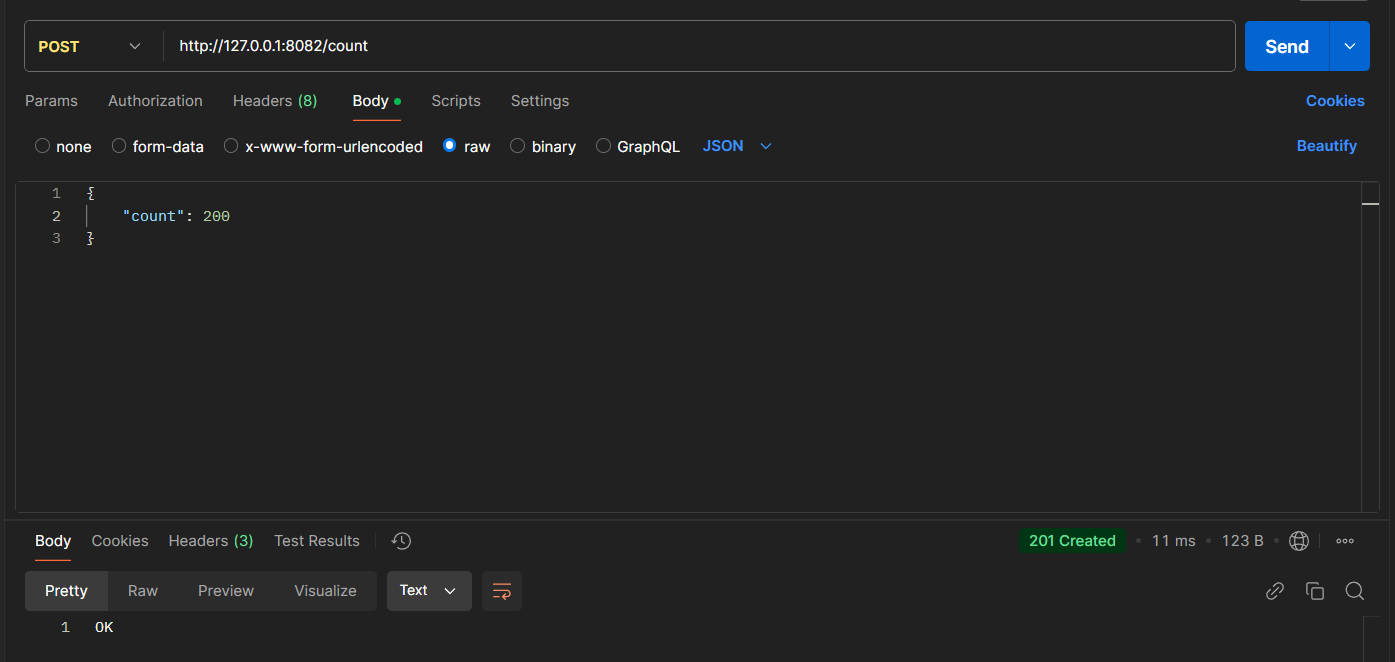


Рисунок 5 – изменение значения

И снова посмотрим на значение с помощью запроса GET:

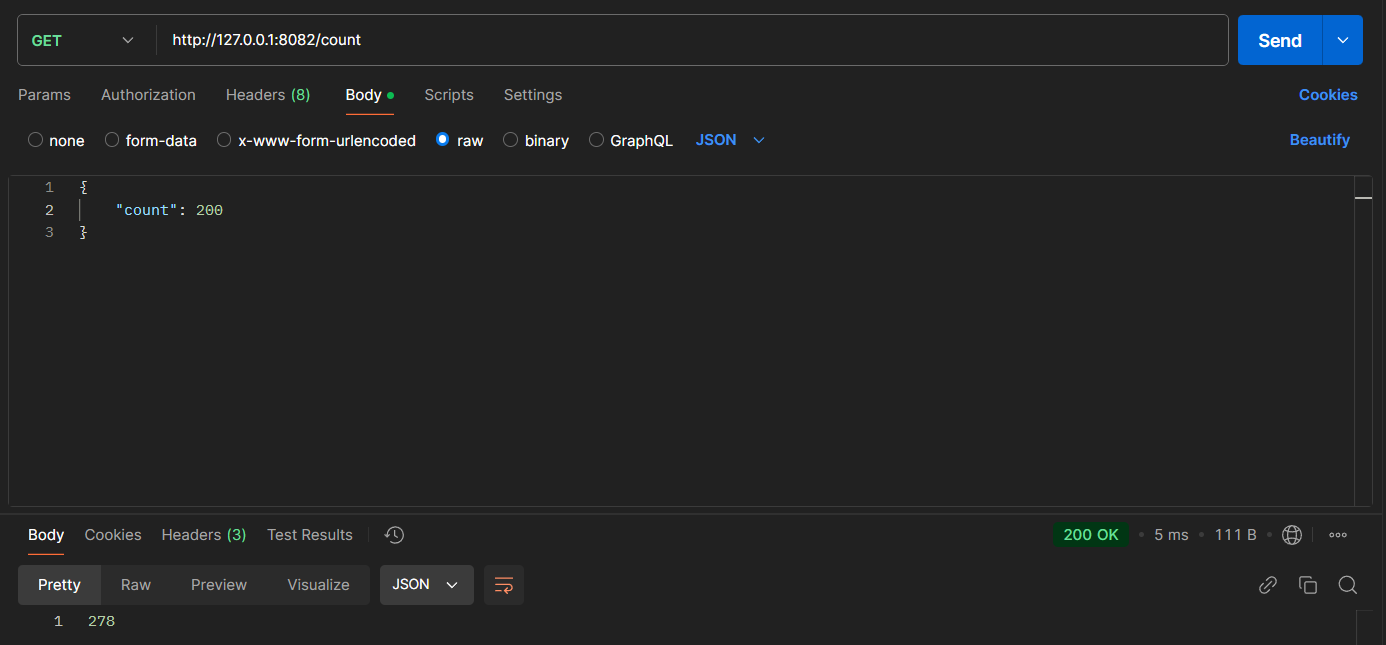


Рисунок 6 – результат работы программы

Как мы видим программа работает исправно.

Задача 3 (query):

Создать архитектуру и переписать под нее код из 9 лабораторной для соответствующей задачи.

База данных была уже создана ранее.

Часть кода программы (представлен только код функции main()):

package main

package main

import (

    "flag"

    "log"

    "web-10/internal/query/api"

    "web-10/internal/query/config"

    "web-10/internal/query/provider"

    "web-10/internal/query/usecase"

    \_ "github.com/lib/pq"

)

func main() {

    // Считываем аргументы командной строки

    configPath := flag.String("config-path", "../../configs/hello\_example.yaml", "путь к файлу конфигурации")

    flag.Parse()

    cfg, err := config.LoadConfig(\*configPath)

    if err != nil {

        log.Fatal(err)

    }

    prv := provider.NewProvider(cfg.DB.Host, cfg.DB.Port, cfg.DB.User, cfg.DB.Password, cfg.DB.DBname)

    use := usecase.NewUsecase(cfg.Usecase.DefaultMessageQuery, prv)

    srv := api.NewServer(cfg.IP, cfg.Port, cfg.API.MaxMessageSize, use)

    srv.Run()

}

При вводе post запроса получим:

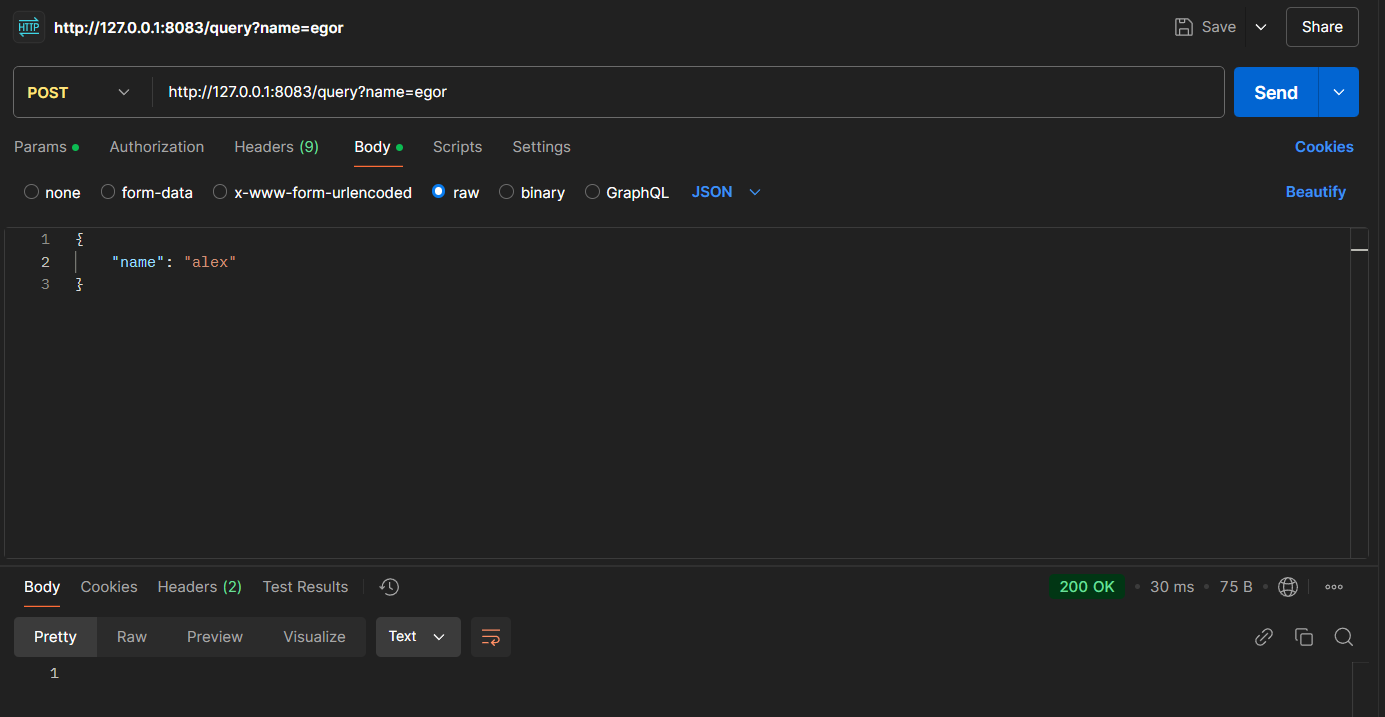


Рисунок 7 – результат post запроса

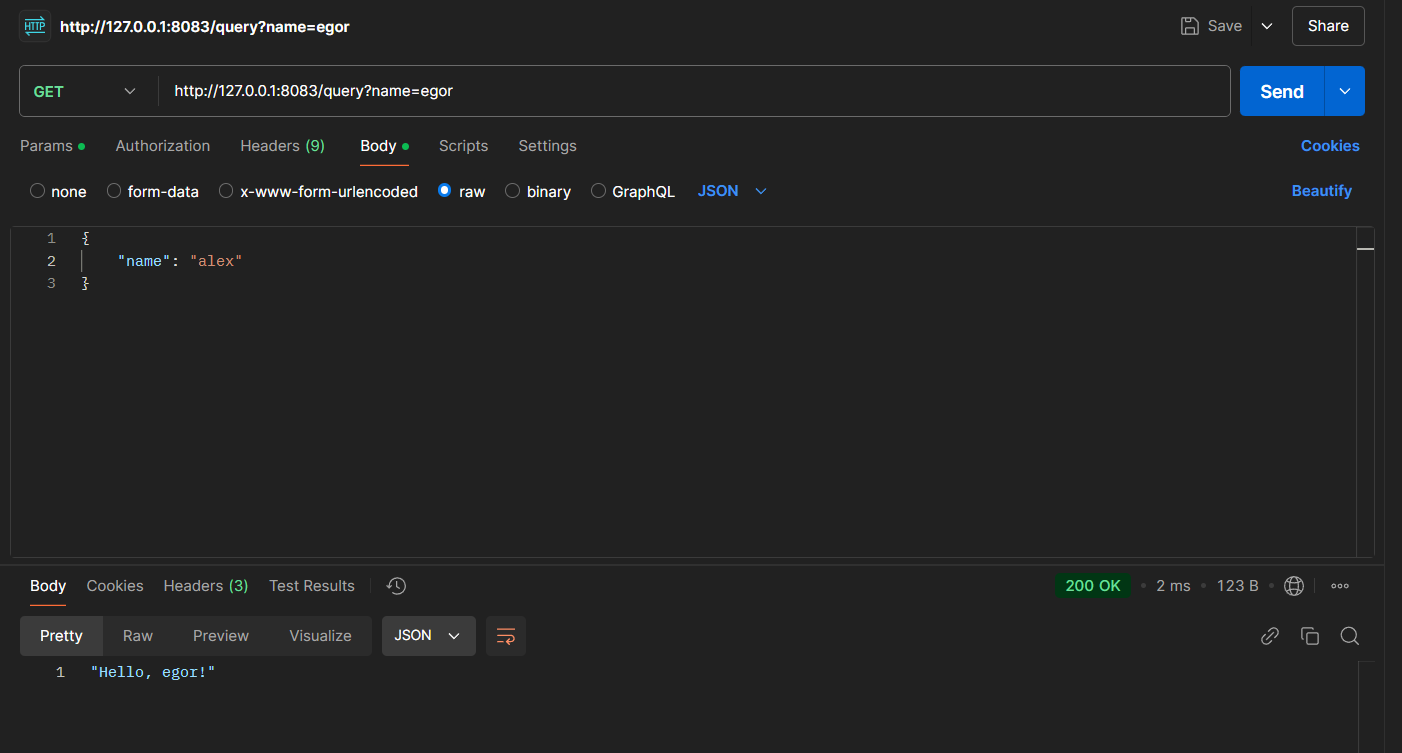


Рисунок 8 – результат get запроса

Вывод: в ходе лабораторной работы я получил первичные навыки организации кодовой базы проекта на Golang.

В рамках данной лабораторной работы я ознакомился с набором рекомендаций для разработки поддерживаемых и расширяемых backend-сервисов на golang.