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 № 8

Название: Организация клиент-серверного взаимодействия между Golang и PostgreSQL

Дисциплина: Языки интернет-программирования

Студент	ИУ6-32Б	23.11.2024	Т.А. Гаджиев
	(Группа)	(Подпись, дата)	(И.О. Фамилия)
Преподаватель		23.11.2024	В.Д. Шульман
		(Подпись, дата)	(И.О. Фамилия)

Москва, 2024

Цель работы — получение первичных навыков в организации долгосрочного хранения данных с использованием PostgreSQL и Golang.

Ход работы.

1. Сделали форк данного репозитория в GitHub, склонировали получившуюся копию локально, создали от мастера ветку дев и переключились на нее. Скопировали наши микросервисы из шестой лабораторной работы:

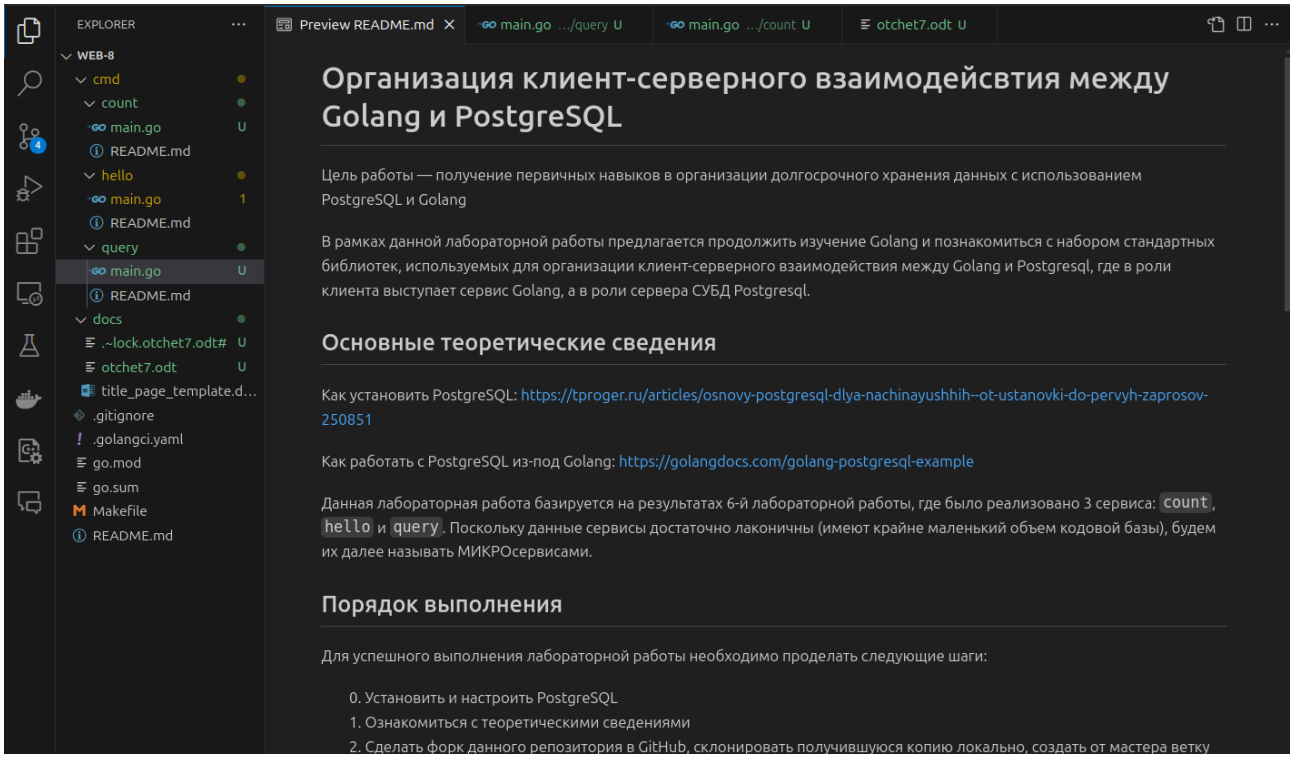

Рисунок 1 — Скопированный репозиторий

2. Установили postgreSQL и создали новые базы данных для каждого из сервисов

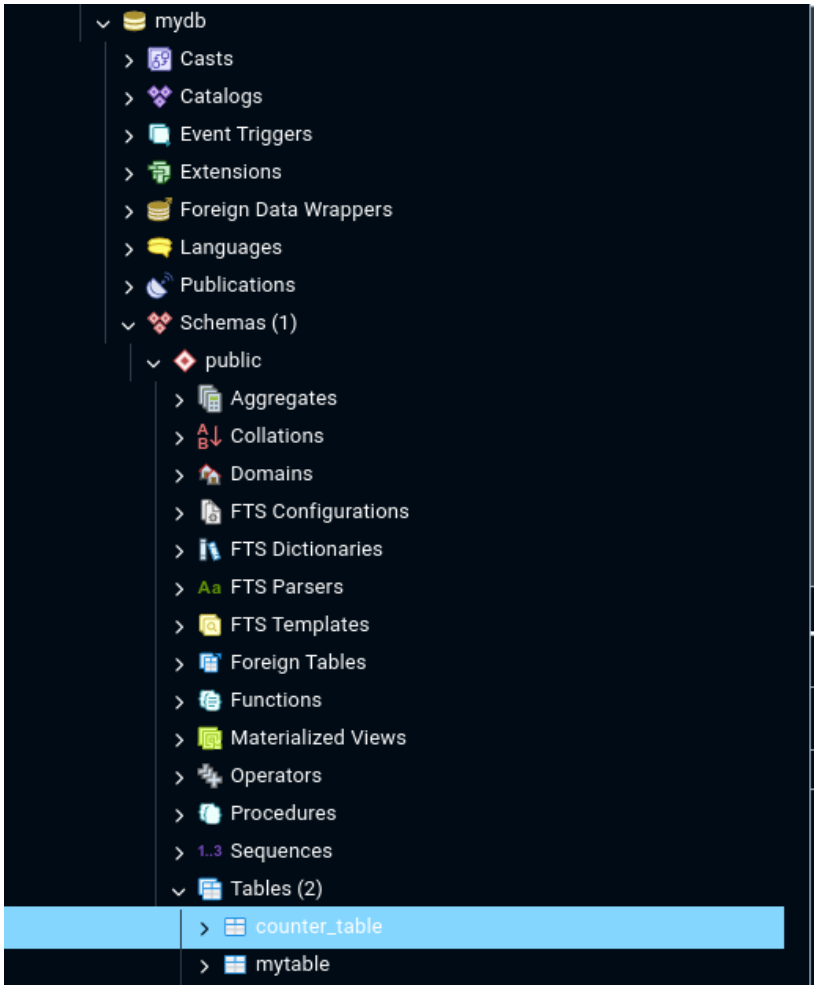

Рисунок 2 — Созданная база данных

Дописали наши микросервисы для работы с базой данных.

Задача 1(Вывод строки с ключом):

```
package main

import (
"database/sql"
"encoding/json"
"flag"
"fmt"
"log"
"net/http"

_ "github.com/lib/pq"
)

const (
host = "localhost"
port = 5432
user = "postgres"
```

```
password = "postgres"
dbname = "mydb"
)

type Handlers struct {
dbProvider DatabaseProvider
}

type DatabaseProvider struct {
db *sql.DB
}

func (h *Handlers) GetUser(w http.ResponseWriter, r *http.Request) {
name := r.URL.Query().Get("name")
if name == "" {
http.Error(w, "Name parameter is required", http.StatusBadRequest)
return
}
user, err := h.dbProvider.SelectUser(name)
if err != nil {
w.WriteHeader(http.StatusInternalServerError)
w.Write([]byte(err.Error()))
return
}

w.WriteHeader(http.StatusOK)
w.Write([]byte("Hello, " + user + "!"))
}
func (h *Handlers) PostUser(w http.ResponseWriter, r *http.Request) {
var input struct {
Name string `json:"name"`
}
decoder := json.NewDecoder(r.Body)
err := decoder.Decode(&input)
if err != nil {
http.Error(w, err.Error(), http.StatusBadRequest)
return
}
err = h.dbProvider.InsertUser(input.Name)
if err != nil {
w.WriteHeader(http.StatusInternalServerError)
w.Write([]byte(err.Error()))
return
}

w.WriteHeader(http.StatusCreated)
w.Write([]byte("Запись добавлена!"))
}
func (dp *DatabaseProvider) SelectUser(name string) (string, error) {
var user string
```

```
row := dp.db.QueryRow("SELECT name FROM mytable WHERE name = $1", name)
err := row.Scan(&user)
if err != nil {
return "", err
}

return user, nil
}

func (dp *DatabaseProvider) InsertUser(name string) error {
_, err := dp.db.Exec("INSERT INTO mytable (name) VALUES ($1)", name)
if err != nil {
return err
}
return nil
}
func main() {
address := flag.String("address", "127.0.0.1:8080", "адрес для запуска сервера")
flag.Parse()
psqlInfo := fmt.Sprintf("host=%s port=%d user=%s password=%s dbname=%s sslmode=disable",
host, port, user, password, dbname)
db, err := sql.Open("postgres", psqlInfo)
if err != nil {
log.Fatal(err)
}
defer db.Close()
dp := DatabaseProvider{db: db}
h := Handlers{dbProvider: dp}
http.HandleFunc("/api/user", h.GetUser)
http.HandleFunc("/api/user/create", h.PostUser)

err = http.ListenAndServe(*address, nil)
if err != nil {
log.Fatal(err)
}
}
```

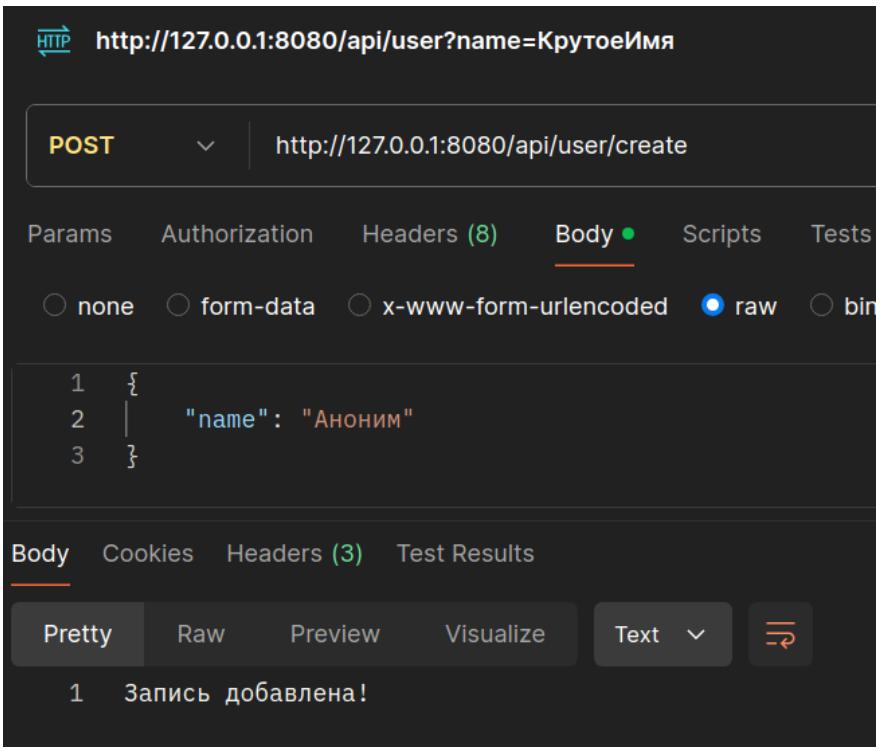

Рисунок 3 — Post запрос в postman

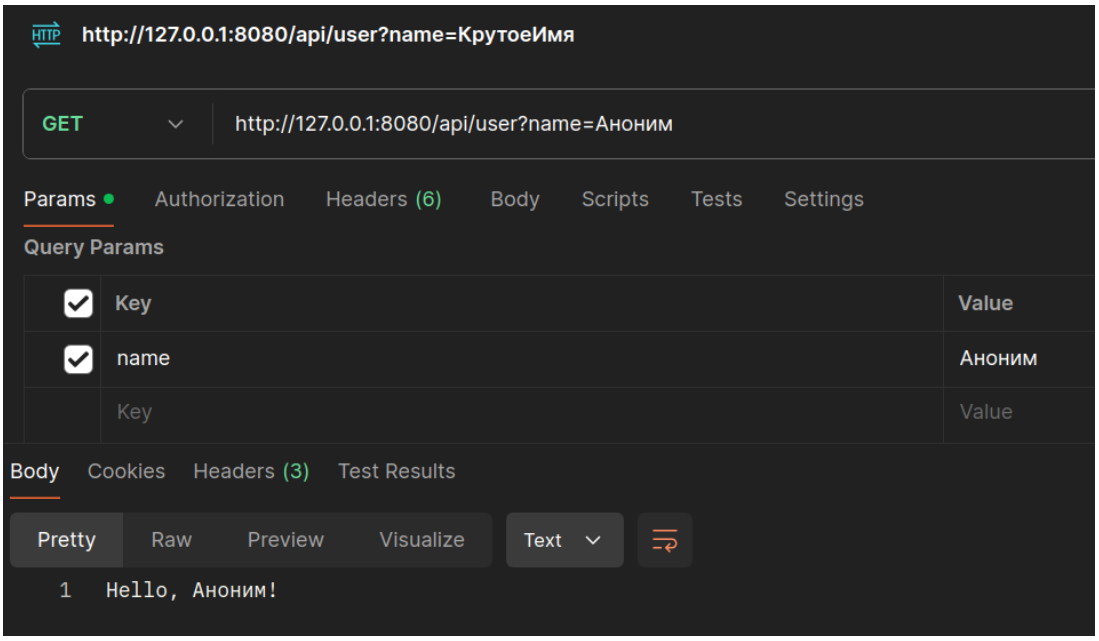

Рисунок 4 — Get запрос в postman

Query Query History

```
1 SELECT * FROM public.mytable
2 ORDER BY id ASC
```

Data Output Messages Notifications

	id [PK] integer	name character varying (255)
1	1	John Doe
2	2	КрутоеИмя
3	3	ЛучшееИмя
4	4	CoolName
5	5	Аноним

Рисунок 5 — Содержимое DB

Задача 2(Count):

```
package main

import (
"database/sql"
"flag"
"fmt"
"log"
"net/http"
"strconv"

_ "github.com/lib/pq"
)

const (
host = "localhost"
port = 5432
user = "postgres"
password = "postgres"
dbname = "mydb"
)

type Handlers struct {
dbProvider DatabaseProvider
}

type DatabaseProvider struct {
db *sql.DB
}
```

```
func (dp *DatabaseProvider) GetCounter() (int, error) {
var counter int
row := dp.db.QueryRow("SELECT value FROM counter_table LIMIT 1")
err := row.Scan(&counter)
if err != nil {
return 0, err
}
return counter, nil
}

func (dp *DatabaseProvider) UpdateCounter(value int) error {
_, err := dp.db.Exec("UPDATE counter_table SET value = value + $1", value)
return err
}

func (h *Handlers) HandleCount(w http.ResponseWriter, r *http.Request) {
switch r.Method {
case "GET":
counter, err := h.dbProvider.GetCounter()
if err != nil {
http.Error(w, err.Error(), http.StatusInternalServerError)
return
}

err = h.dbProvider.UpdateCounter(1)
if err != nil {
http.Error(w, err.Error(), http.StatusInternalServerError)
return
}

fmt.Fprintf(w, "%d", counter+1)
case "POST":
count, err := strconv.Atoi(r.FormValue("count"))
if err != nil {
http.Error(w, "это не число", http.StatusBadRequest)
return
}
err = h.dbProvider.UpdateCounter(count)
if err != nil {
http.Error(w, err.Error(), http.StatusInternalServerError)
return
}
fmt.Fprintf(w, "Success")
default:
http.Error(w, "Неизвестный метод", http.StatusMethodNotAllowed)
}
}

func main() {
address := flag.String("address", "127.0.0.1:3333", "адрес для запуска сервера")
flag.Parse()
```

```
psqlInfo := fmt.Sprintf("host=%s port=%d user=%s password=%s dbname=%s sslmode=disable",
host, port, user, password, dbname)

db, err := sql.Open("postgres", psqlInfo)
if err != nil {
log.Fatal(err)
}
defer db.Close()

dp := DatabaseProvider{db: db}
h := Handlers{dbProvider: dp}

http.HandleFunc("/count", h.HandleCount)

fmt.Println("Сервер запущен на порту :3333")
err = http.ListenAndServe(*address, nil)
if err != nil {
log.Fatal(err)
}
}
```

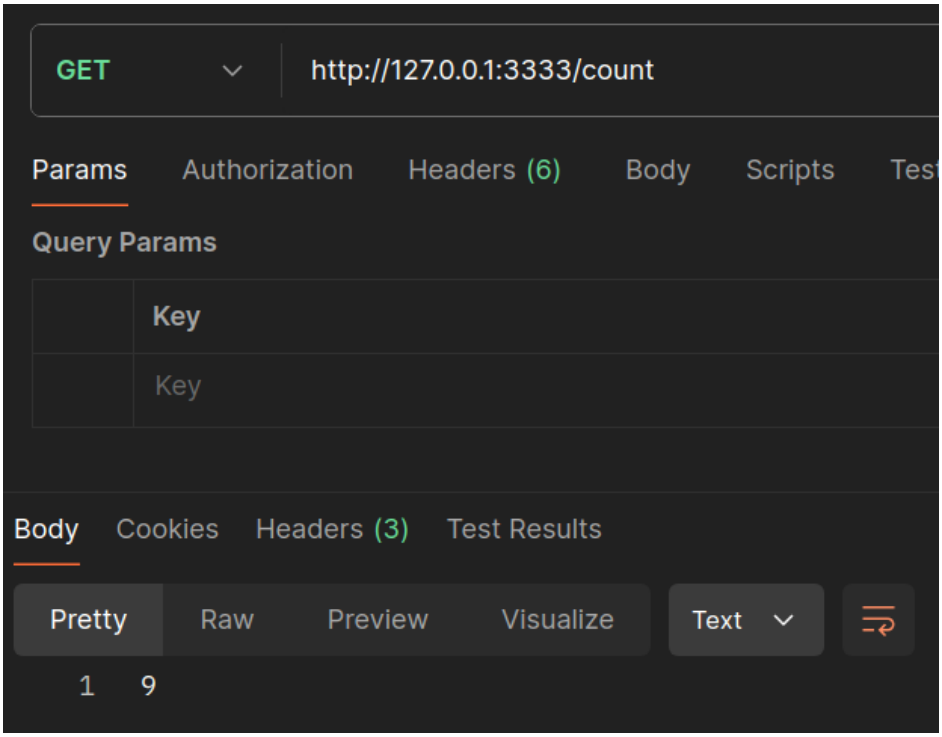

Рисунок 6 — Get запрос в Postman

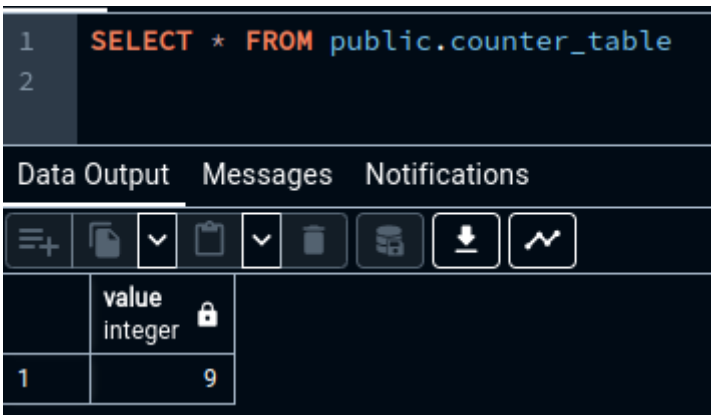

Рисунок 7 — Содержимое DB

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

Заключение: в ходе лабораторной работы получили первичные навыки в организации долгосрочного хранения данных с использованием PostgreSQL и Golang.