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**

О Т Ч Е Т

по лабораторной работе № 8

Название: Организация клиент-серверного взаимодейсвтия между Golang и PostgreSQL

Дисциплина: Языки интернет программирования

Студент	ИУ6-33Б	21.11.24	Пономаренко В.М.
	(Группа)	(Подпись, дата)	(И.О. Фамилия)
Преподаватель		21.11.24	Шульман В.Д.
		(Подпись, дата)	(И.О. Фамилия)

Москва, 2024

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

Задание:
Доработать сервисы таким образом, чтобы они использовали для хранения данных СУБД PostgreSQL. Каждый сервис должен как добавлять новые данные в БД (insert/update), так и доставать их для предоставления пользователю (select)

Ход работы:

Задание 1. Query

1. Описание реализуемого функционала
   1) Get – выводит приветствие для последнего пользователя, внесенного в БД
   2) Post – создает в БД новую запись для пользователя, чье имя отправлено в Query-параметре name
   3) Put – изменяет имя последнего пользователя в БД на имя в Query-параметре name
2. Создадим БД посредством pgAdmin4

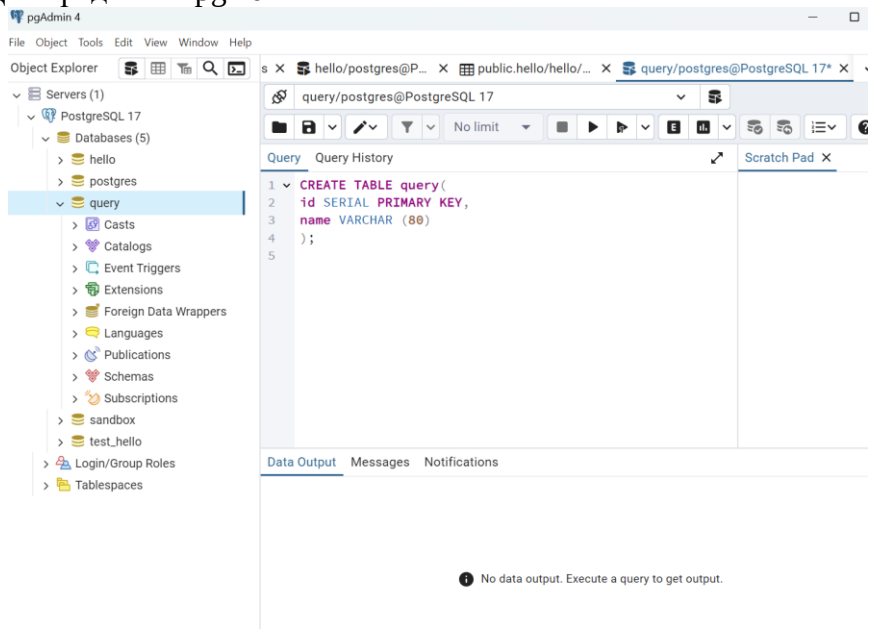

Рисунок 1 - создание БД query

3. Ниже приведен листинг файла query.go

```go
package main

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

    _ "github.com/lib/pq"
)

const (
    host     = "localhost"
    port     = 5432
    user     = "postgres"
    password = "catjkm8800"
    dbname   = "query"
)
```

```go
type Handlers struct {
    dbProvider DatabaseProvider
}

type DatabaseProvider struct {
    db *sql.DB
}

// Обработчики HTTP-запросов
func (h *Handlers) GetQuery(w http.ResponseWriter, r *http.Request) {
    msg, err := h.dbProvider.SelectQuery()
    if err != nil {
        w.WriteHeader(http.StatusInternalServerError)
        w.Write([]byte(err.Error()))
    }
    w.WriteHeader(http.StatusOK)
    w.Write([]byte("Hello " + msg + "!"))
}

func (h *Handlers) PostQuery(w http.ResponseWriter, r *http.Request) {
    nameInput := r.URL.Query().Get("name") // ради разнообразия поработаем с Query-параметром, как в
изначальном условии лр6
    if nameInput == "" {
        w.WriteHeader(http.StatusBadRequest)
        w.Write([]byte("Missing 'name' query parameter"))
    }
    err := h.dbProvider.InsertQuery(nameInput)
    if err != nil {
        w.WriteHeader(http.StatusInternalServerError)
        w.Write([]byte(err.Error()))
        return
    }
    w.WriteHeader(http.StatusCreated)
}

func (h *Handlers) PutQuery(w http.ResponseWriter, r *http.Request) {
    nameInput := r.URL.Query().Get("name") // ради разнообразия поработаем с Query-параметром, как в
изначальном условии лр6
    if nameInput == "" {
        w.WriteHeader(http.StatusBadRequest)
        w.Write([]byte("Missing 'name' query parameter"))
        return
    }
    err := h.dbProvider.UpdateQuery(nameInput)
    if err != nil {
        w.WriteHeader(http.StatusInternalServerError)
        w.Write([]byte(err.Error()))
        return
    }
    w.WriteHeader(http.StatusCreated)
}

// Методы для работы с базой данных
func (dbp *DatabaseProvider) SelectQuery() (string, error) {
    var msg string
    row := dbp.db.QueryRow("SELECT name FROM query ORDER BY id DESC LIMIT 1")
    err := row.Scan(&msg)
    if err != nil {
        return "", err
    }
    return msg, nil
}

func (dbp *DatabaseProvider) UpdateQuery(n string) error {
    _, err := dbp.db.Exec("UPDATE query SET name = $1 WHERE id = (SELECT MAX(id) FROM query)", n)
    if err != nil {
        return err
    }
    return nil
}
```

```go
func (dbp *DatabaseProvider) InsertQuery(n string) error {
    _, err := dbp.db.Exec("INSERT INTO query (name) VALUES ($1)", n)
    if err != nil {
        return err
    }
    return nil
}

func main() {
    address := flag.String("address", "127.0.0.1:8081", "адрес для запуска сервера")
    flag.Parse()
    // Формирование строки подключения для postgres
    psqlInfo := fmt.Sprintf("host=%s port=%d user=%s "+"password=%s dbname=%s sslmode=disable", host,
port, user, password, dbname)

    // Создание соединения с сервером postgres
    db, err := sql.Open("postgres", psqlInfo)
    if err != nil {
        log.Fatal(err)
    }
    defer db.Close()
    err = db.Ping()
    if err != nil {
        fmt.Println("NO! 2")
    }
    fmt.Println("Connected!")
    // Создаем провайдер для БД с набором методов
    dp := DatabaseProvider{db: db}
    // Создаем экземпляр структуры с набором обработчиков
    h := Handlers{dbProvider: dp}

    // Регистрируем обработчики
    http.HandleFunc("/get", h.GetQuery)
    http.HandleFunc("/post", h.PostQuery)
    http.HandleFunc("/put", h.PutQuery)

    // Запускаем веб-сервер на указанном адресе
    err = http.ListenAndServe(*address, nil)
    if err != nil {
        log.Fatal(err)
    }

}
```

4. Приведем примеры работы программы

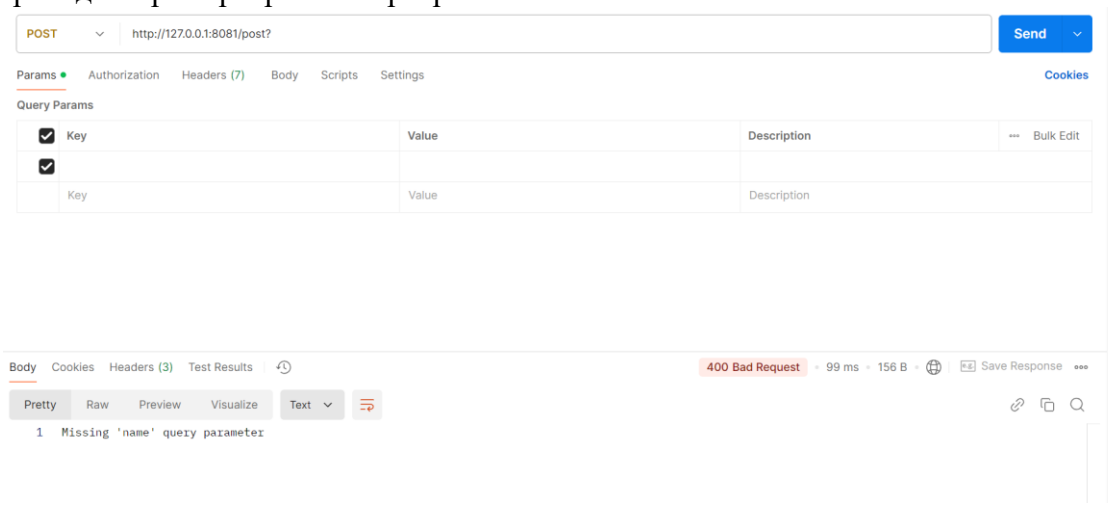

Рисунок 2 - тест 1 - отсутствует параметр

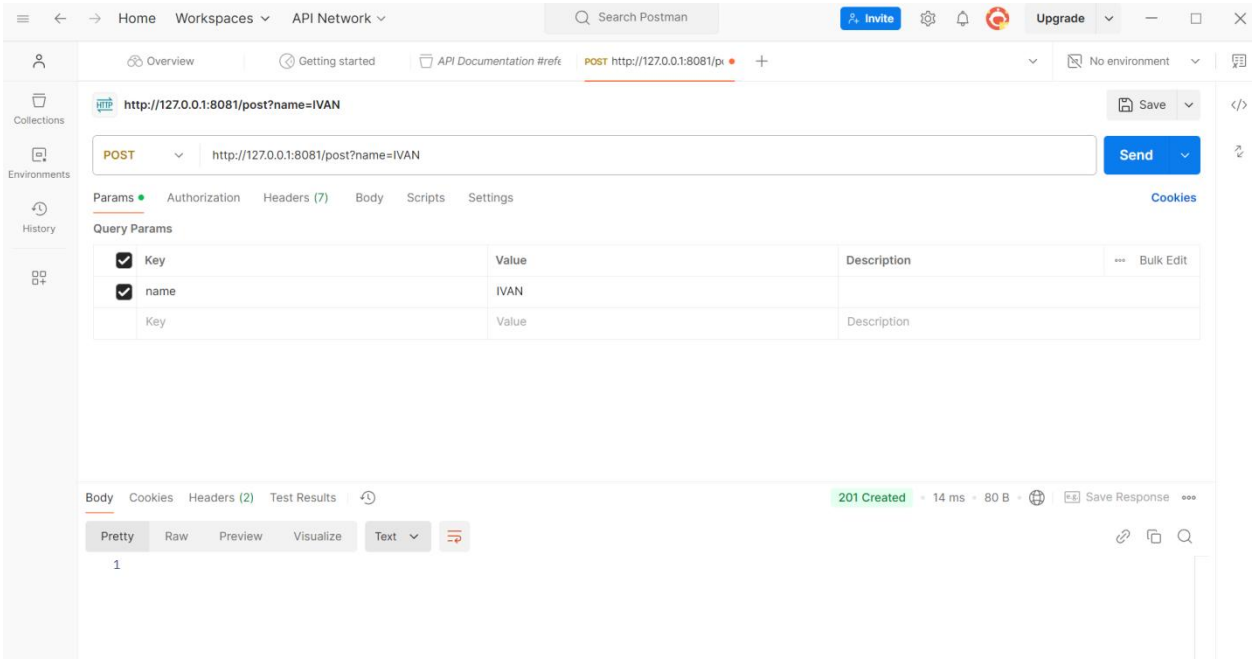

Рисунок 3 - тест 2 (POST)

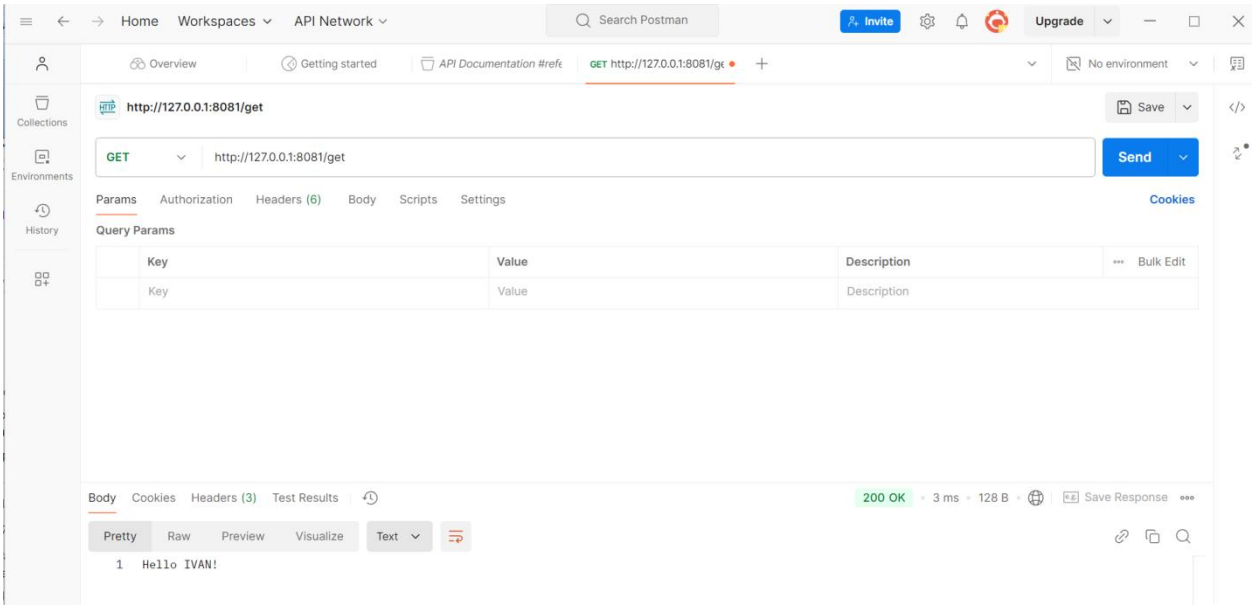

Рисунок 4 - тест 3 (GET)

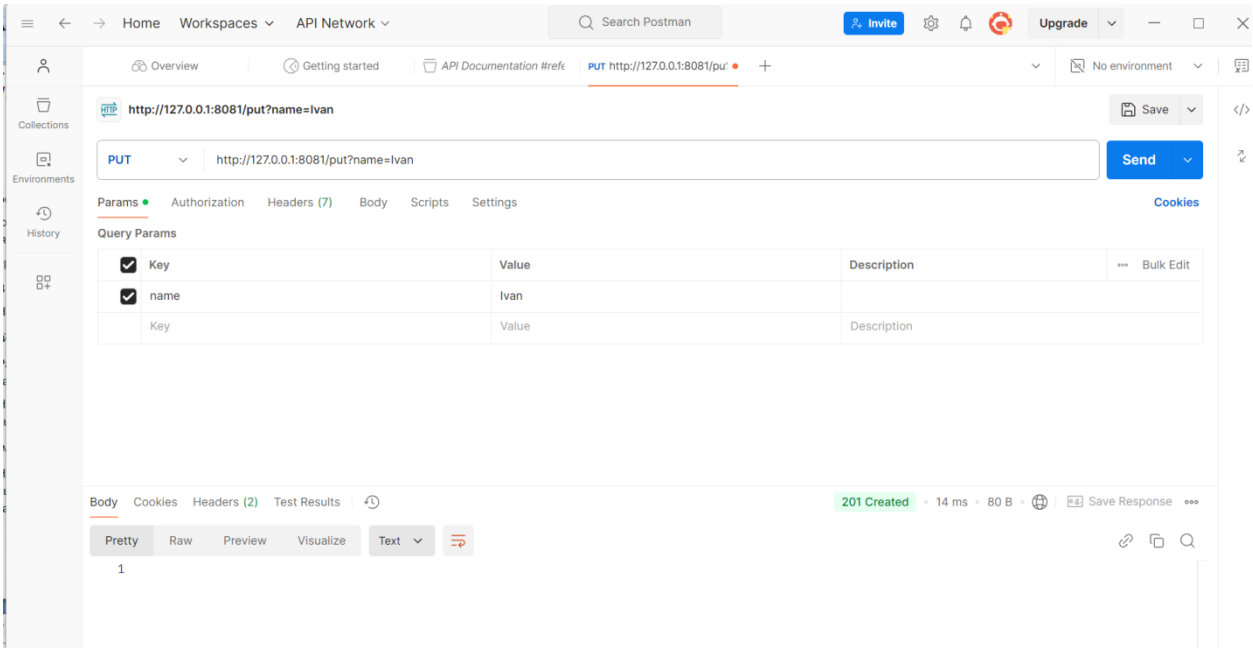

Рисунок 5 - тест 4 (PUT)

Для демонстрации функционала добавим post-запросом пользователя IVAN, через get-запрос видим приветствие для этого пользователя. Теперь исправим это имя на обычное Ivan через put-запрос. В качестве подтверждения приведем саму БД: последняя запись IVAN была заменена на Ivan

	id [PK] integer	name character varying (80)
1	1	Vera
2	2	Вера
3	3	Иван
4	4	Ivan
5	5	IVAN

	id [PK] integer	name character varying (80)
1	1	Vera
2	2	Вера
3	3	Иван
4	4	Ivan
5	5	Ivan

Рисунок 6 – демонстрация изменения БД

5. Так как работа производится под ОС Windows, для которой команда make lint не является традиционной, то проверка осуществлялась альтернативным способом.

Приведем листинг специального файла build.ps1

```
# build.ps1

# Запуск golangci-lint
Write-Host "Запуск golangci-lint..."
golangci-lint run ./...

# Проверка статуса выполнения линтера
if ($LASTEXITCODE -ne 0) {
    Write-Host "Ошибка при выполнении golangci-lint."
    exit $LASTEXITCODE
}

# Компиляция проекта
Write-Host "Компиляция проекта..."
go build -o myapp.exe .

# Проверка статуса компиляции
if ($LASTEXITCODE -ne 0) {
    Write-Host "Ошибка при компиляции проекта."
    exit $LASTEXITCODE
}
```

```
Write-Host "Скрипт завершен."
```

```
PS D:\Go\лр_8> .\build.ps1
Р—Р°РїСѓСЃРє golangci-lint...
query.go:34:10: Error return value of `w.Write` is not checked (errcheck)
                w.Write([]byte(err.Error()))
                        ^
query.go:37:9: Error return value of `w.Write` is not checked (errcheck)
        w.Write([]byte("Hello " + msg + "!"))
                ^
query.go:44:10: Error return value of `w.Write` is not checked (errcheck)
                w.Write([]byte("Missing 'name' query parameter"))
                        ^
РћС€РёР±РєР° РїСЂРё PIC‹РїРѕР»РЅРµРЅРё golangci-lint.
```

Рисунок 7 - результат проверки через PowerShell

Выведенные ошибки указывают на то, что в коде не проверяются возвращаемые значения функции w.Write. Так как реализация является учебной и основная цель работы – получение практических навыков по работе с СУБД, то оставим эти недочеты во избежание излишнего нагромождения кода.

Задание 2. Query

1. Описание реализуемого функционала
   1) Get – выводит текущее состояние счетчика
   2) Post – прибавление к счетчику значения, передаваемого через json
   3) Put – изменяет значение последнего отправленного значения и как следствие итоговую сумму
2. Создадим БД посредством pgAdmin4

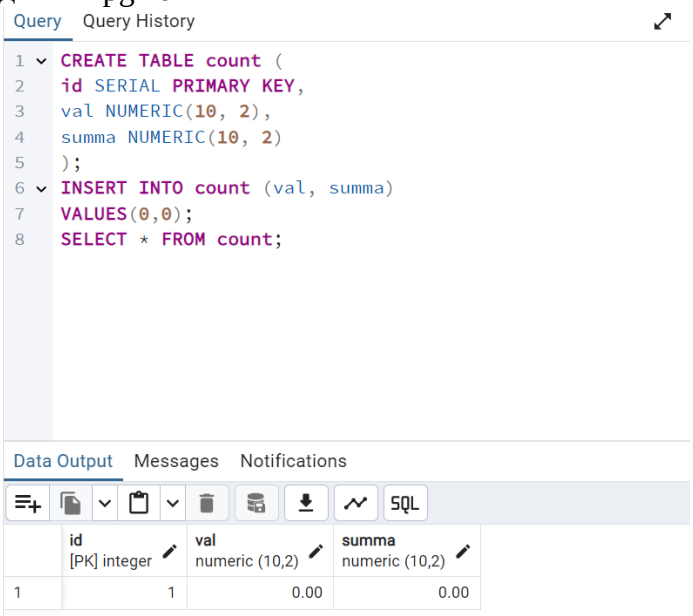

Рисунок 8 - создание БД count

```
PS D:\Go\лр_8_2> go mod init count
go: creating new go.mod: module count
go: to add module requirements and sums:
        go mod tidy
PS D:\Go\лр_8_2> go get github.com/lib/pq@latest
go: added github.com/lib/pq v1.10.9
PS D:\Go\лр_8_2>
```

Рисунок 9 - подключение БД к проекту

Ниже приведен листинг программы

```go
package main

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

    _ "github.com/lib/pq"
)

const (
    host     = "localhost"
    port     = 5432
    user     = "postgres"
    password = "catjkm8800"
    dbname   = "count"
)

type Handlers struct {
    dbProvider DatabaseProvider
}
type DatabaseProvider struct {
    db *sql.DB
}

// Обработчики HTTP-запросов
func (h *Handlers) GetCount(w http.ResponseWriter, r *http.Request) {
    msg, err := h.dbProvider.SelectCount()
    if err != nil {
        w.WriteHeader(http.StatusInternalServerError)
        w.Write([]byte(err.Error()))
    }
    w.WriteHeader(http.StatusOK)
    w.Write([]byte(msg))
}

func (h *Handlers) PostCount(w http.ResponseWriter, r *http.Request) {
    input := struct {
        Val float32 `json:"val"`
    }{}
    decoder := json.NewDecoder(r.Body)
    err := decoder.Decode(&input)
    if err != nil {
        w.WriteHeader(http.StatusBadRequest)
        w.Write([]byte(err.Error()))
    }
    err = h.dbProvider.InsertCount(input.Val)
    if err != nil {
        w.WriteHeader(http.StatusInternalServerError)
        w.Write([]byte(err.Error()))
```

```
    }
    w.WriteHeader(http.StatusCreated)
}

func (h *Handlers) PutCount(w http.ResponseWriter, r *http.Request) {

    input := struct {
        Val float32 `json:"val"`
    }{}
    decoder := json.NewDecoder(r.Body)
    err := decoder.Decode(&input)
    if err != nil {
        w.WriteHeader(http.StatusBadRequest)
        w.Write([]byte(err.Error()))
    }
    err = h.dbProvider.UpdateCount(input.Val)
    if err != nil {
        w.WriteHeader(http.StatusInternalServerError)
        w.Write([]byte(err.Error()))
    }
    w.WriteHeader(http.StatusCreated)
}

// Методы для работы с базой данных
func (dbp *DatabaseProvider) SelectCount() (string, error) {
    var msg string
    row := dbp.db.QueryRow("SELECT summa FROM count ORDER BY id DESC LIMIT 1")
    err := row.Scan(&msg)
    if err != nil {
        return "", err
    }
    return msg, nil
}

func (dbp *DatabaseProvider) InsertCount(v float32) error {
    _, err := dbp.db.Exec("INSERT INTO count (val, summa) VALUES ($1, $1+(SELECT summa FROM count ORDER BY id
DESC LIMIT 1))", v)
    if err != nil {
        return err
    }

    return nil
}

func (dbp *DatabaseProvider) UpdateCount(v float32) error {
    _, err := dbp.db.Exec("UPDATE count SET val = $1, summa = (val + (SELECT summa FROM count WHERE id =
((SELECT MAX(id) FROM count) - 1))) WHERE id = (SELECT MAX(id) FROM count)", v)
    if err != nil {
        return err
    }
    return nil
}

func main() {
    address := flag.String("address", "127.0.0.1:8081", "адрес для запуска сервера")
    flag.Parse()
    // Формирование строки подключения для postgres
    psqlInfo := fmt.Sprintf("host=%s port=%d user=%s "+"password=%s dbname=%s sslmode=disable", host, port,
user, password, dbname)

    // Создание соединения с сервером postgres
    db, err := sql.Open("postgres", psqlInfo)
    if err != nil {
        log.Fatal(err)
    }
    defer db.Close()
    err = db.Ping()
    if err != nil {
        fmt.Println("NO! 2")
    }
    fmt.Println("Connected!")
```

```
    // Создаем провайдер для БД с набором методов
    dp := DatabaseProvider{db: db}
    // Создаем экземпляр структуры с набором обработчиков
    h := Handlers{dbProvider: dp}

    // Регистрируем обработчики
    http.HandleFunc("/get", h.GetCount)
    http.HandleFunc("/post", h.PostCount)
    http.HandleFunc("/put", h.PutCount)

    // Запускаем веб-сервер на указанном адресе
    err = http.ListenAndServe(*address, nil)
    if err != nil {
        log.Fatal(err)
    }

}
```

Представим примеры работы:

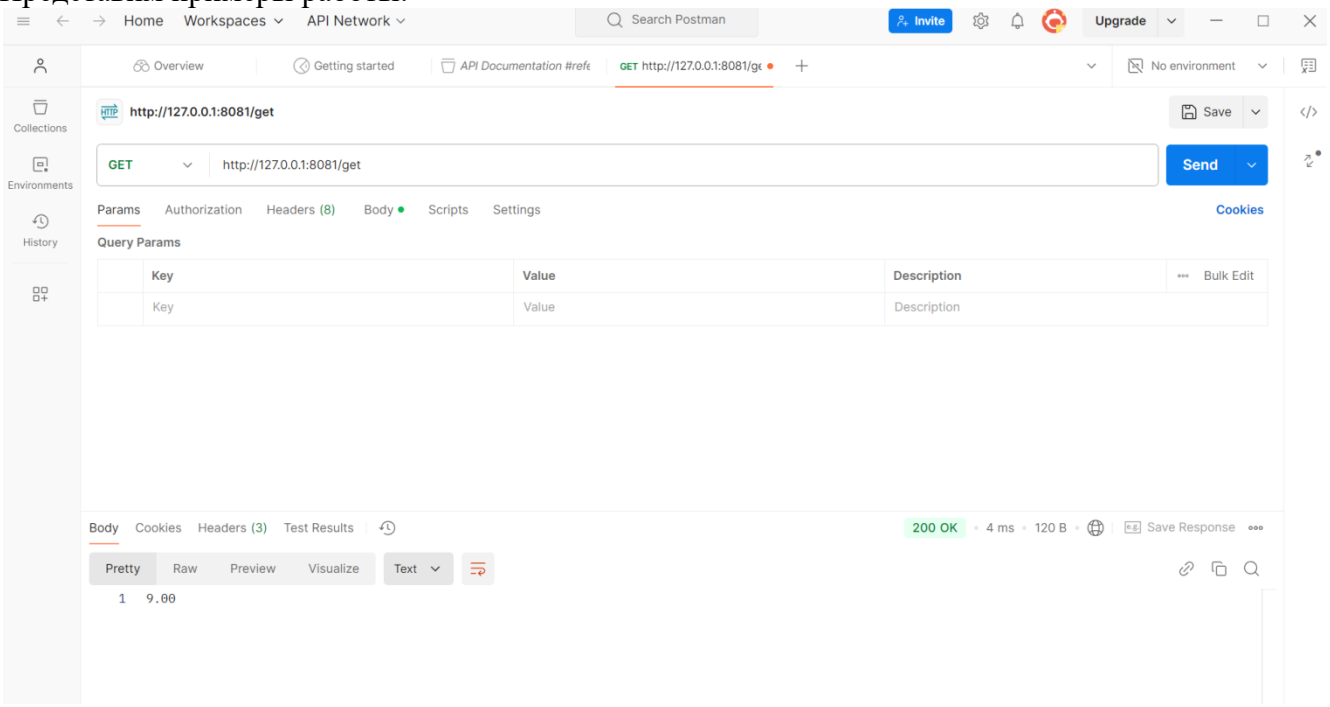

Рисунок 10 - тест 1 (GET)

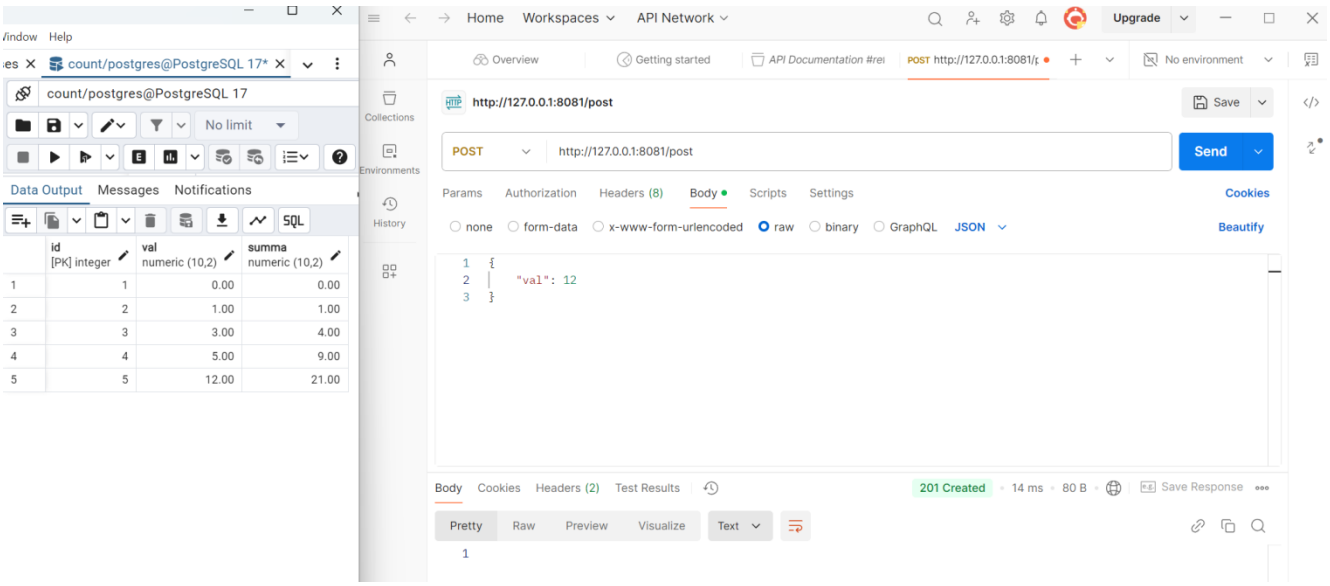

Рисунок 11 - тест 2 (POST)

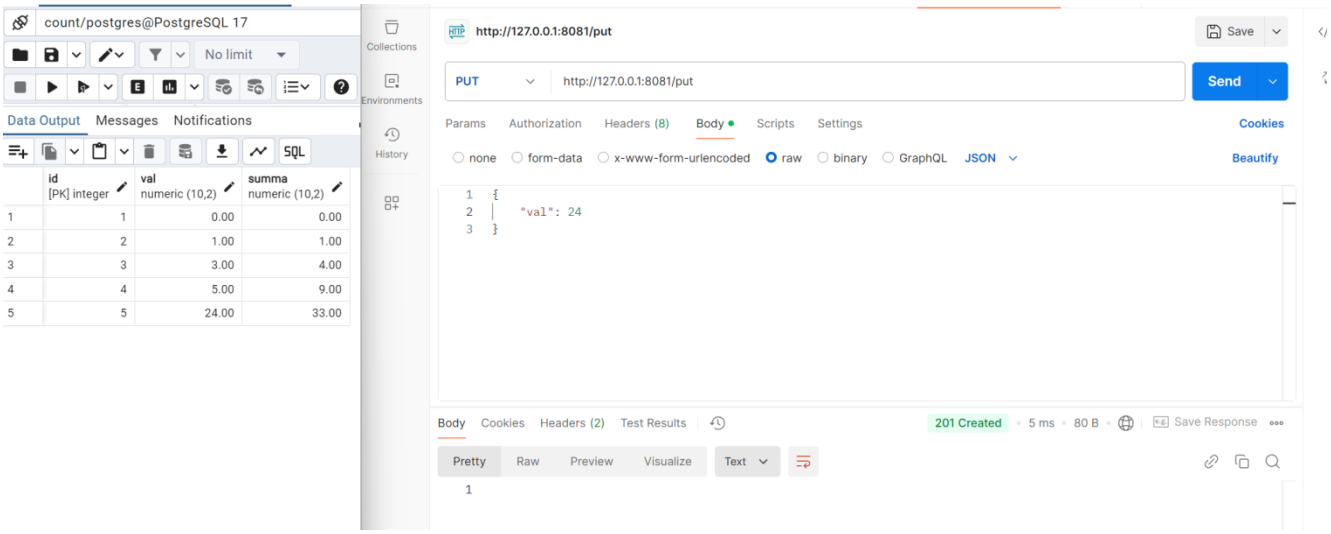

Рисунок 12 - тест 3 (PUT)

Для демонстрации работы обработчиков PUT и POST приведены скриншоты с состоянием БД. Из них виден пересчет значений при изменении последнего отправленного числа.

```
PS D:\Go\лр_8_2>  .\build.ps1
Р–РºРїСѓСЃРє golangci-lint...
count.go:34:10: Error return value of `w.Write` is not checked (errcheck)
                w.Write([]byte(err.Error()))
                        ^
count.go:37:9: Error return value of `w.Write` is not checked (errcheck)
        w.Write([]byte(msg))
                ^
count.go:48:10: Error return value of `w.Write` is not checked (errcheck)
                w.Write([]byte(err.Error()))
                        ^
РћС€РёР±РєР° РїСЂРё PIC‹РїРѕР»РЅРµРЅРё golangci-lint.
```

Рисунок 13 - проверка через .\build.ps1

Результат проверки с помощью линкера аналогичен предыдущему заданию.

Выводы: В ходе выполнения лабораторной работы были изучены стандартные библиотеки, используемые для организации клиент-серверного взаимодействия между Golang и Postgresql, где в роли клиента выступает сервис Golang, а в роли сервера СУБД Postgresql.

Список использованных источников:

1) https://tproger.ru/articles/osnovy-postgresql-dlya-nachinayushhih--ot-ustanovki-do-pervyh-zaprosov-250851
2) https://golangdocs.com/golang-postgresql-example
3) https://stepik.org/course/63054/syllabus