|  |  |
| --- | --- |
| 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 № 8**

**Название:** Организация клиент-серверного взаимодействия между Golang и PostgreSQL

**Дисциплина:** Языки интернет-программирования

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Студент | ИУ6-31Б |  |  | Хан С.Т. |
|  | (Группа) |  | (Подпись, дата) | (И.О. Фамилия) |
|  |  |  |  |  |
| Преподаватель |  |  |  | В.Д.Шульман |
|  |  |  | (Подпись, дата) | (И.О. Фамилия) |

Москва, 2024

Цель работы – получение первичных навыков в организации долгосрочного хранения данных с использованием PostgresSQL и Golang.

**Сервис COUNT:**

package main

import (

    "database/sql"

    "encoding/json"

    "flag"

    "fmt"

    "log"

    "net/http"

    \_ "github.com/lib/pq"

)

const (

    host     = "localhost"

    port     = 5423

    user     = "postgres"

    password = "Salviya542350"

    dbname   = "count"

)

type Handlers struct {

    dbProvider DatabaseProvider

}

type DatabaseProvider struct {

    db \*sql.DB

}

// Обработчик GET для получения значения счетчика

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

    count, err := h.dbProvider.SelectCount()

    if err != nil {

        w.WriteHeader(http.StatusInternalServerError)

        w.Write([]byte(err.Error()))

        return

    }

    w.WriteHeader(http.StatusOK)

    w.Write([]byte(fmt.Sprintf("Текущий счетчик: %d", count)))

}

// Обработчик POST для увеличения счетчика

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

    input := struct {

        Count int `json:"count"`

    }{}

    decoder := json.NewDecoder(r.Body)

    err := decoder.Decode(&input)

    if err != nil {

        http.Error(w, "Ошибка парсинга JSON", http.StatusBadRequest)

        return

    }

    if input.Count <= 0 {

        http.Error(w, "Значение count должно быть положительным числом", http.StatusBadRequest)

        return

    }

    err = h.dbProvider.UpdateCount(input.Count)

    if err != nil {

        w.WriteHeader(http.StatusInternalServerError)

        w.Write([]byte(err.Error()))

        return

    }

    w.WriteHeader(http.StatusOK)

    w.Write([]byte(fmt.Sprintf("Счетчик увеличен на %d", input.Count)))

}

// Методы для работы с базой данных

func (dp \*DatabaseProvider) SelectCount() (int, error) {

    var count int

    row := dp.db.QueryRow("SELECT count FROM counters WHERE id = 1")

    err := row.Scan(&count)

    if err != nil {

        if err == sql.ErrNoRows {

            // Если записи нет, создаем начальный счетчик

            \_, err := dp.db.Exec("INSERT INTO counters (count) VALUES (0)")

            if err != nil {

                return 0, err

            }

            count = 0

        } else {

            return 0, err

        }

    }

    return count, nil

}

func (dp \*DatabaseProvider) UpdateCount(increment int) error {

    \_, err := dp.db.Exec("UPDATE counters SET count = count + $1 WHERE id = 1", increment)

    if err != nil {

        return err

    }

    return nil

}

func main() {

    address := flag.String("address", "127.0.0.1:8081", "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/get", h.GetCount)   // Обработчик GET-запроса

    http.HandleFunc("/count/post", h.PostCount) // Обработчик POST-запроса для увеличения

    err = http.ListenAndServe(\*address, nil)

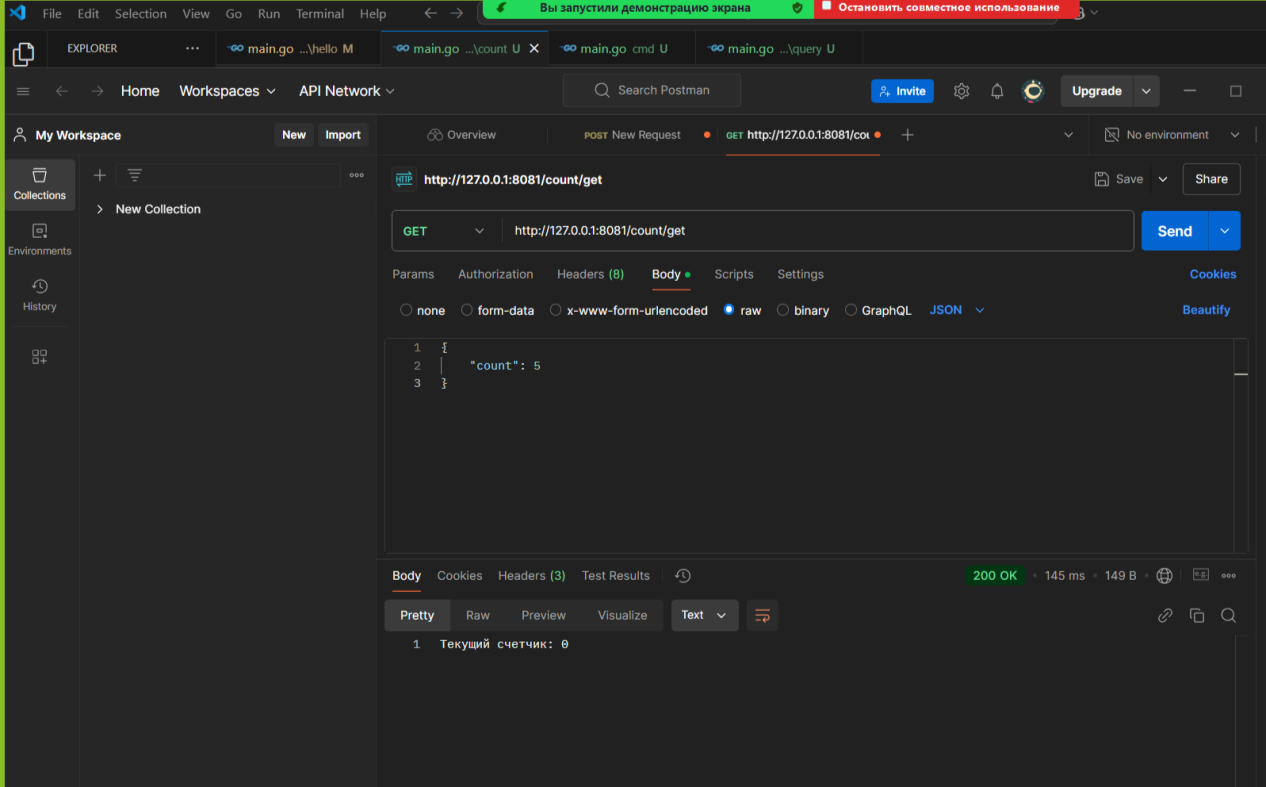
    if err != nil {

        log.Fatal(err)

    }

}

**Тестирование:**



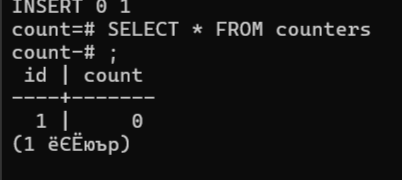


Рисунок 1 – БД count

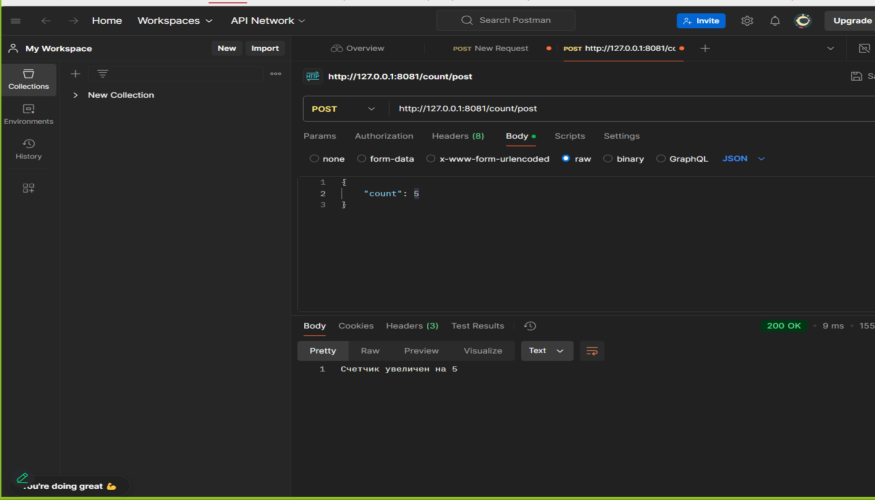


Рисунок 2 – изменение счетчика

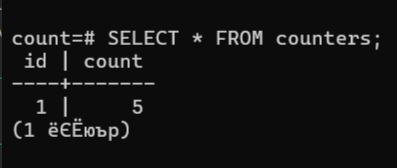


Рисунок 3 – БД count

**Сервис Query:**

package main

import (

    "database/sql"

    "flag"

    "fmt"

    "log"

    "net/http"

    \_ "github.com/lib/pq"

)

const (

    host     = "localhost"

    port     = 5423

    user     = "postgres"

    password = "Salviya542350"

    dbname   = "query"

)

type Handlers struct {

    dbProvider DatabaseProvider

}

type DatabaseProvider struct {

    db \*sql.DB

}

// Обработчик GET для получения приветствия по имени

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

    name := r.URL.Query().Get("name")

    if name == "" {

        http.Error(w, "Нет параметра 'name'", http.StatusBadRequest)

        return

    }

    greeting, err := h.dbProvider.SelectGreeting(name)

    if err != nil {

        w.WriteHeader(http.StatusInternalServerError)

        w.Write([]byte(err.Error()))

        return

    }

    w.WriteHeader(http.StatusOK)

    w.Write([]byte(greeting))

}

// Методы для работы с базой данных

func (dp \*DatabaseProvider) SelectGreeting(name string) (string, error) {

    var greeting string

    row := dp.db.QueryRow("SELECT greeting FROM greetings WHERE name = $1", name)

    err := row.Scan(&greeting)

    if err != nil {

        if err == sql.ErrNoRows {

            \_, err := dp.db.Exec("INSERT INTO greetings (name, greeting) VALUES ($1, $2)", name, fmt.Sprintf("Hello, %s!", name))

            if err != nil {

                return "", err

            }

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

        } else {

            return "", err

        }

    }

    return greeting, nil

}

func main() {

    address := flag.String("address", "127.0.0.1:8081", "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}

    // Регистрируем обработчик для /api/user

    http.HandleFunc("/api/user", h.GetGreeting)

    // Запускаем веб-сервер на указанном адресе

    err = http.ListenAndServe(\*address, nil)

    if err != nil {

        log.Fatal(err)

    }

}

**Тестирование:**

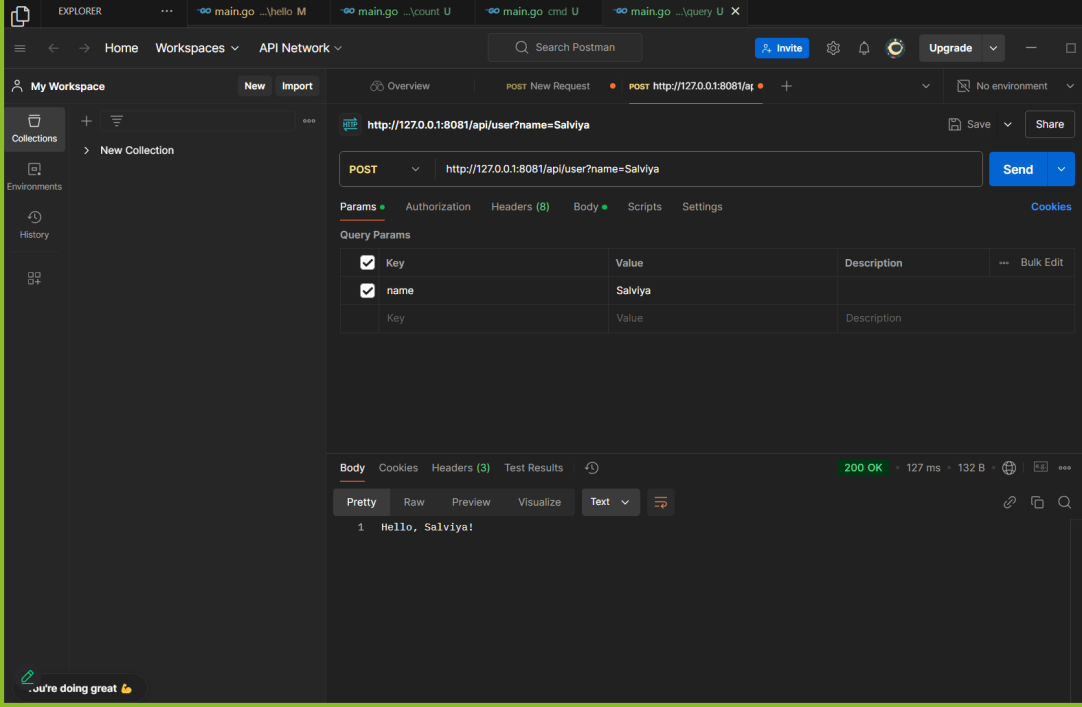


Рисунок 4 – post запрос

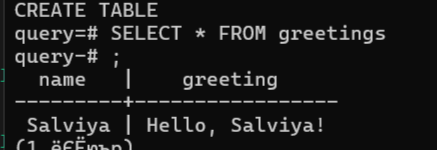


Рисунок 5 – БД query

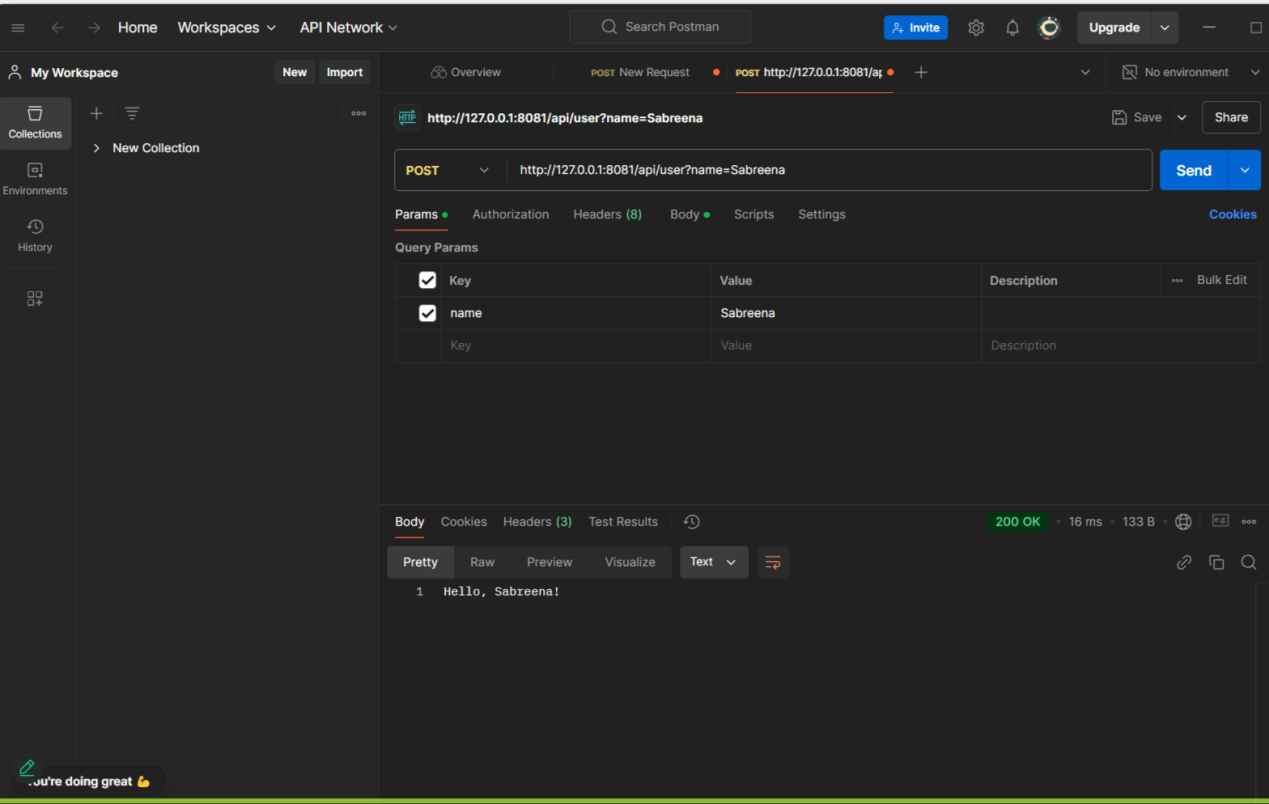


Рисунок 6 – post запрос

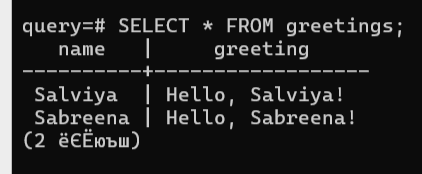


Рисунок 7 – БД query

**Сервис Hello:**

package main

import (

    "database/sql"

    "encoding/json"

    "flag"

    "fmt"

    "log"

    "net/http"

    \_ "github.com/lib/pq"

)

const (

    host     = "localhost"

    port     = 5423

    user     = "postgres"

    password = "Salviya542350"

    dbname   = "sandbox"

)

type Handlers struct {

    dbProvider DatabaseProvider

}

type DatabaseProvider struct {

    db \*sql.DB

}

// Обработчики HTTP-запросов

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

    msg, err := h.dbProvider.SelectHello()

    if err != nil {

        w.WriteHeader(http.StatusInternalServerError)

        w.Write([]byte(err.Error()))

    }

    w.WriteHeader(http.StatusOK)

    w.Write([]byte(msg))

}

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

    input := struct {

        Msg string `json:"msg"`

    }{}

    decoder := json.NewDecoder(r.Body)

    err := decoder.Decode(&input)

    if err != nil {

        if err != nil {

            w.WriteHeader(http.StatusBadRequest)

            w.Write([]byte(err.Error()))

        }

    }

    err = h.dbProvider.InsertHello(input.Msg)

    if err != nil {

        w.WriteHeader(http.StatusInternalServerError)

        w.Write([]byte(err.Error()))

    }

    w.WriteHeader(http.StatusCreated)

}

// Методы для работы с базой данных

func (dp \*DatabaseProvider) SelectHello() (string, error) {

    var msg string

    // Получаем одно сообщение из таблицы hello, отсортированной в случайном порядке

    row := dp.db.QueryRow("SELECT message FROM hello ORDER BY RANDOM() LIMIT 1")

    err := row.Scan(&msg)

    if err != nil {

        return "", err

    }

    return msg, nil

}

func (dp \*DatabaseProvider) InsertHello(msg string) error {

    \_, err := dp.db.Exec("INSERT INTO hello (message) VALUES ($1)", msg)

    if err != nil {

        return err

    }

    return nil

}

func main() {

    // Считываем аргументы командной строки

    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()

    // 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.GetHello)

    http.HandleFunc("/post", h.PostHello)

    // Запускаем веб-сервер на указанном адресе

    err = http.ListenAndServe(\*address, nil)

    if err != nil {

        log.Fatal(err)

    }

}

**Тестирование:**

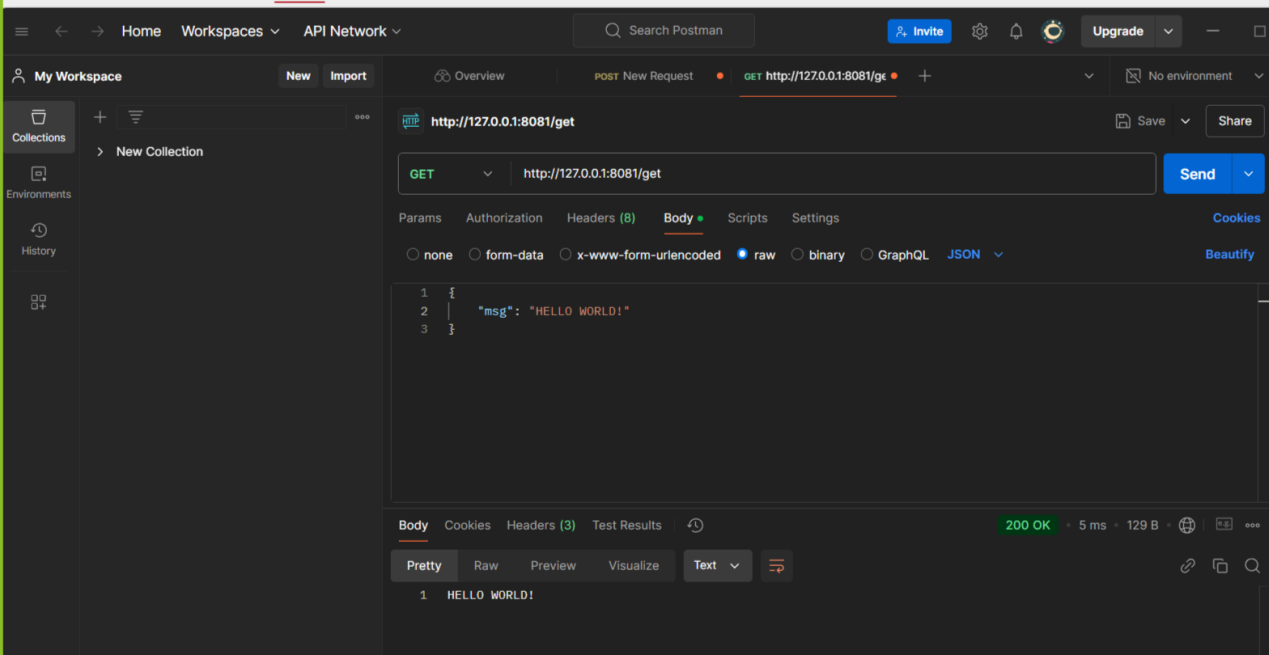


Рисунок 8 – запрос

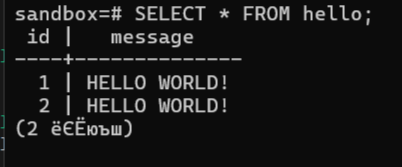


Рисунок 10 – БД sandbox

Вывод: научились интегрировать базы данных в разработку приложений на postgres.