**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 взаимодейсвтия между Golang и PostgreSQL

**Дисциплина:** Языки интернет-программирования

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Студент | ИУ6-33Б |  | |  | Д.А. Лазутин |
|  | (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 PostgreSQL и Golang

В рамках данной лабораторной работы предлагается продолжить изучение Golang и познакомиться с набором стандартных библиотек, используемых для организации клиент-серверного взаимодействия между Golang и Postgresql, где в роли клиента выступает сервис Golang, а в роли сервера СУБД Postgresql.

Ход работы:

1)Модифицируем код микросервиса query для сохранения имен(пользователей) в БД с ID.

Создаем таблицу:

CREATE TABLE users (

id SERIAL PRIMARY KEY,

name TEXT NOT NULL

);

Код программы:

package main

import (

    "database/sql"

    "encoding/json"

    "fmt"

    "log"

    "net/http"

    \_ "github.com/lib/pq"

)

const (

    host     = "localhost"

    port     = 5432

    user     = "postgres"

    password = "postgres"

    dbname   = "sandbox"

)

type DatabaseProvider struct {

    db \*sql.DB

}

type User struct {

    ID   int    `json:"id"`

    Name string `json:"name"`

}

func main() {CREATE TABLE counter (

id SERIAL PRIMARY KEY,

value INT NOT NULL

);

    // Подключение к PostgreSQL

    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()

    // Проверяем соединение

    err = db.Ping()

    if err != nil {

        log.Fatal(err)

    }

    fmt.Println("Successfully connected to the database!")

    // Инициализируем провайдер БД

    dbProvider := &DatabaseProvider{db: db}

    // Регистрируем маршруты

    http.HandleFunc("/api/user", func(w http.ResponseWriter, r \*http.Request) {

        userHandler(w, r, dbProvider)

    })

    // Запускаем сервер

    err = http.ListenAndServe(":9000", nil)

    if err != nil {

        log.Fatal(err)

    }

}

// Обработчик запросов

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

    w.Header().Set("Access-Control-Allow-Origin", "\*")

    w.Header().Set("Access-Control-Allow-Methods", "GET, POST, OPTIONS")

    w.Header().Set("Access-Control-Allow-Headers", "Content-Type")

    switch r.Method {

    case http.MethodGet:

        // Получение информации о пользователе по имени

        name := r.URL.Query().Get("name")

        if name == "" {

            http.Error(w, "Parameter 'name' is required", http.StatusBadRequest)

            return

        }

        user, err := dbProvider.GetUser(name)

        if err != nil {

            http.Error(w, err.Error(), http.StatusInternalServerError)

            return

        }

        if user == nil {

            http.Error(w, "User not found", http.StatusNotFound)

            return

        }

        if err := json.NewEncoder(w).Encode(user); err != nil {

            http.Error(w, err.Error(), http.StatusInternalServerError)

        }

    case http.MethodPost:

        // Добавление нового пользователя

        var user User

        err := json.NewDecoder(r.Body).Decode(&user)

        if err != nil {

            http.Error(w, "Invalid JSON format", http.StatusBadRequest)

            return

        }

        err = dbProvider.AddUser(user.Name)

        if err != nil {

            http.Error(w, err.Error(), http.StatusInternalServerError)

            return

        }

        w.WriteHeader(http.StatusCreated)

        fmt.Fprintf(w, "User %s added successfully", user.Name)

    default:

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

    }

}

// Методы работы с базой данных

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

    query := "SELECT id, name FROM users WHERE name = $1"

    row := dp.db.QueryRow(query, name)

    var user User

    err := row.Scan(&user.ID, &user.Name)

    if err == sql.ErrNoRows {

        return nil, nil // Пользователь не найден

    } else if err != nil {

        return nil, err

    }

    return &user, nil

}

func (dp \*DatabaseProvider) AddUser(name string) error {

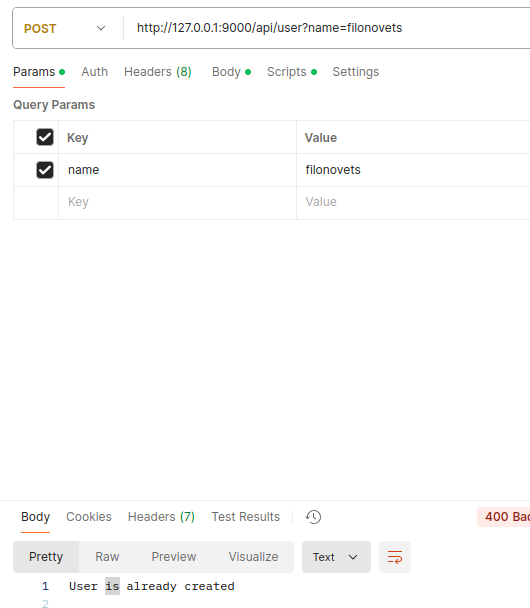
    query := "INSERT INTO users (name) VALUES ($1)"

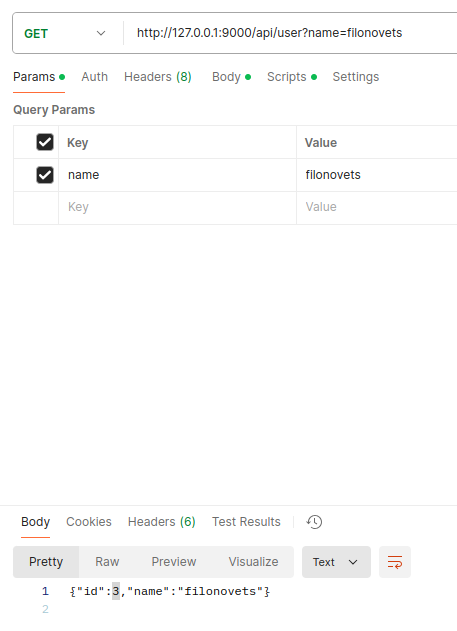
    \_, err := dp.db.Exec(query, name)

    return err

}

Результат работы:

Рисунок 1 — результат добавления (ранее добавленного пользователя)

Рисунок 2 – поиск пользователя по имени.

2) Модифицируем код микросервиса count для хранения значения счетчика в БД.

Создание таблицы в БД:

CREATE TABLE counter (

id SERIAL PRIMARY KEY,

value INT NOT NULL

);

Код программы:

package main

import (

    "database/sql"

    "encoding/json"

    "fmt"

    "log"

    "net/http"

    "strconv"

    \_ "github.com/lib/pq"

)

const (

    host     = "localhost"

    port     = 5432

    user     = "postgres"

    password = "postgres"

    dbname   = "sandbox"

)

type DatabaseProvider struct {

    db \*sql.DB

}

type Counter struct {

    ID    int `json:"id"`

    Value int `json:"value"`

}

func main() {

    // Подключение к PostgreSQL

    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()

    // Проверяем соединение

    err = db.Ping()

    if err != nil {

        log.Fatal(err)

    }

    fmt.Println("Successfully connected to the database!")

    // Инициализируем провайдер БД

    dbProvider := &DatabaseProvider{db: db}

    // Добавляем начальное значение счетчика если оно отсутствует

    err = dbProvider.initializeCounter()

    if err != nil {

        log.Fatal(err)

    }

    // Регистрируем маршруты

    http.HandleFunc("/count", func(w http.ResponseWriter, r \*http.Request) {

        countHandler(w, r, dbProvider)

    })

    // Запускаем сервер

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

    if err != nil {

        log.Fatal(err)

    }

}

// Обработчик запросов

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

    w.Header().Set("Access-Control-Allow-Origin", "\*")

    w.Header().Set("Access-Control-Allow-Methods", "GET, POST, OPTIONS")

    w.Header().Set("Access-Control-Allow-Headers", "Content-Type")

    switch r.Method {

    case http.MethodGet:

        // Получение текущего значения счетчика

        counter, err := dbProvider.GetCounter()

        if err != nil {

            http.Error(w, err.Error(), http.StatusInternalServerError)

            return

        }

        if counter == nil {

            http.Error(w, "Counter not found", http.StatusNotFound)

            return

        }

        if err := json.NewEncoder(w).Encode(counter); err != nil {

            http.Error(w, err.Error(), http.StatusInternalServerError)

        }

    case http.MethodPost:

        // Увеличение счетчика

        err := r.ParseForm()

        if err != nil {

            http.Error(w, "Invalid form data", http.StatusBadRequest)

            return

        }

        countStr := r.FormValue("count")

        count, err := strconv.Atoi(countStr)

        if err != nil {

            http.Error(w, "Count must be a number", http.StatusBadRequest)

            return

        }

        err = dbProvider.IncreaseCounter(count)

        if err != nil {

            http.Error(w, err.Error(), http.StatusInternalServerError)

            return

        }

        fmt.Fprintf(w, "Counter increased by %d", count)

    default:

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

    }

}

// Методы работы с базой данных

func (dp \*DatabaseProvider) GetCounter() (\*Counter, error) {

    query := "SELECT id, value FROM counter LIMIT 1"

    row := dp.db.QueryRow(query)

    var counter Counter

    err := row.Scan(&counter.ID, &counter.Value)

    if err == sql.ErrNoRows {

        return nil, nil // Счетчик не найден

    } else if err != nil {

        return nil, err

    }

    return &counter, nil

}

func (dp \*DatabaseProvider) IncreaseCounter(value int) error {

    query := "UPDATE counter SET value = value + $1 WHERE id = 1"

    \_, err := dp.db.Exec(query, value)

    return err

}

// Инициализация счетчика, если он отсутствует

func (dp \*DatabaseProvider) initializeCounter() error {

    var count Counter

    query := "SELECT id FROM counter LIMIT 1"

    err := dp.db.QueryRow(query).Scan(&count.ID)

    if err == sql.ErrNoRows {

        // Счетчик не найден, добавляем начальное значение

        insertQuery := "INSERT INTO counter (value) VALUES ($1)"

        \_, err := dp.db.Exec(insertQuery, 0)

        if err != nil {

            return err

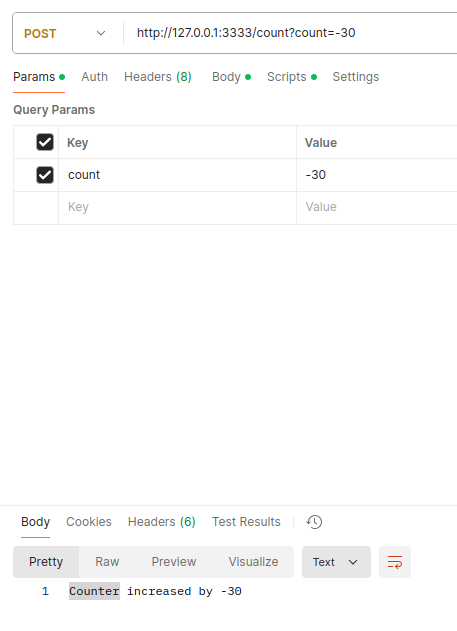
        }

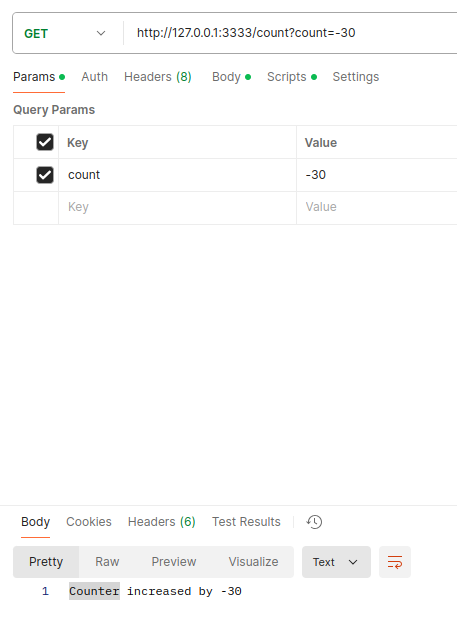
    }

    return nil

}

Результат работы:

Рисунок 3 — тестирование увеличения счетчинка.

Рисунок 4 — проверка счетчика.

Вывод: Я научился организовывать долгосрочное хранение данных с помощью PostgreSQL и Golang.