*Федеральное государственное бюджетное образовательное учреждение* *высшего образования*

|  |  |
| --- | --- |
| **Изображение выглядит как эмблема, герб, нашивка, символ  Автоматически созданное описание** | ***«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 №**

**Название лабораторной работы: Организация клиент-серверного взаимодейсвтия между Golang и PostgreSQL**



**Дисциплина: Языки интернет программирования**



Студент гр. ИУ6-32Б **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Суворов Вакао А.**

(Подпись, дата) (И.О. Фамилия)



Преподаватель  **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

(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.

1. Разбираемся с данным кодом серверной части для микросервиса «hello»

Скачиваем недостающие драйвера через терминал Goland (рис. 1): Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Рис.1

Далее создаем таблицу в PgAdmin(также PostgreSQL, дальше СУБД) (рис. 2)

Изображение выглядит как текст, Шрифт, снимок экрана, белый

Автоматически созданное описание

Рис. 2

Проверяем успешность создания ‘select’(рис. 3):

Изображение выглядит как текст, снимок экрана, программное обеспечение, дисплей

Автоматически созданное описание

Рис. 3

Тут я уже добавил один элемент во время выполнения работы.

Обращение к СУБД будем совершать через Postman, также и возвращать значения из СУБД.

Использую cURL запрос формата: *«curl -X POST -H "Content-Type: application/json" -d '{"msg": "Hello from client!"}'* [*http://localhost:8081/post»*](http://localhost:8081/post)

Отправим новый запрос изменив JSON составляющую, запустив GO сервер (рис. 4):

Изображение выглядит как текст, снимок экрана, линия, Шрифт

Автоматически созданное описание

Рис. 4

Проверим СУБД отправив select запрос в нашу таблицу (рис. 5):

Изображение выглядит как текст, снимок экрана, программное обеспечение, Значок на компьютере

Автоматически созданное описание

Рис. 5

Далее запустим GET запрос в Postman для проверки возвращаемых значений (рис. 6, 7):

Изображение выглядит как текст, снимок экрана, программное обеспечение, число

Автоматически созданное описание

Рис. 6

Изображение выглядит как текст, снимок экрана, дисплей, программное обеспечение

Автоматически созданное описание

Рис. 7

1. Перейдем к реализации query микросервиса с СУБД, сначала добавим таблицу в нашу СУБД (рис. 8):

Изображение выглядит как текст, снимок экрана, дисплей, программное обеспечение

Автоматически созданное описание

Рис. 8

Представлю код микросервиса со всеми пояснениями:

*package main  
  
import (  
 "database/sql"  
 "encoding/json"  
 "flag"  
 "fmt"  
 "log"  
 "net/http"  
 "net/url"  
  
 \_ "github.com/lib/pq" // Импорт драйвера для PostgreSQL  
)  
  
// Структура для параметров подключения к базе данных  
type dbConfig struct {  
 Host string  
 Port int  
 User string  
 Password string  
 DBName string  
 SSLMode string // Добавлено для безопасности  
}  
  
// Структура для хранения приветственных сообщений  
type Greeting struct {  
 ID int `json:"id"`  
 Message string `json:"message"`  
}  
  
// DatabaseProvider обрабатывает взаимодействие с базой данных  
type DatabaseProvider struct {  
 db \*sql.DB  
}  
  
// NewDatabaseProvider создает новый экземпляр DatabaseProvider  
func NewDatabaseProvider(cfg dbConfig) (\*DatabaseProvider, error) {  
 psqlInfo := fmt.Sprintf("host=%s port=%d user=%s password=%s dbname=%s sslmode=%s",  
 cfg.Host, cfg.Port, cfg.User, cfg.Password, cfg.DBName, cfg.SSLMode)  
  
 db, err := sql.Open("postgres", psqlInfo)  
 if err != nil {  
 return nil, fmt.Errorf("не удалось подключиться к базе данных: %w", err)  
 }  
  
 err = db.Ping()  
 if err != nil {  
 return nil, fmt.Errorf("не удалось проверить соединение с базой данных: %w", err)  
 }  
  
 return &DatabaseProvider{db: db}, nil  
}  
  
// GreetUser приветствует пользователя и сохраняет приветствие в базе данных.  
func (dp \*DatabaseProvider) GreetUser(name string) (\*Greeting, error) {  
 // Подготавливаем SQL-запрос для предотвращения SQL-инъекций.  
 stmt, err := dp.db.Prepare("INSERT INTO greetings (message) VALUES ($1) RETURNING id, message")  
 if err != nil {  
 return nil, fmt.Errorf("не удалось подготовить запрос: %w", err)  
 }  
 defer stmt.Close() // Важно: Закрываем запрос после использования.  
  
 var greeting Greeting  
 err = stmt.QueryRow(fmt.Sprintf("Hello, %s!", name)).Scan(&greeting.ID, &greeting.Message)  
 if err != nil {  
 return nil, fmt.Errorf("не удалось вставить приветствие: %w", err)  
 }  
  
 return &greeting, nil  
}  
  
// Функция для получения всех приветствий из базы данных  
func (dp \*DatabaseProvider) GetAllGreetings() ([]Greeting, error) {  
 rows, err := dp.db.Query("SELECT id, message FROM greetings")  
 if err != nil {  
 return nil, fmt.Errorf("failed to query greetings: %w", err)  
 }  
 defer rows.Close()  
  
 var greetings []Greeting  
 for rows.Next() {  
 var greeting Greeting  
 err := rows.Scan(&greeting.ID, &greeting.Message)  
 if err != nil {  
 return nil, fmt.Errorf("failed to scan row: %w", err)  
 }  
 greetings = append(greetings, greeting)  
 }  
  
 return greetings, nil  
}  
  
// Новый обработчик для получения всех приветствий  
func getAllGreetingsHandler(dbProvider \*DatabaseProvider) http.HandlerFunc {  
 return func(w http.ResponseWriter, r \*http.Request) {  
 greetings, err := dbProvider.GetAllGreetings()  
 if err != nil {  
 http.Error(w, fmt.Sprintf("Database error: %v", err), http.StatusInternalServerError)  
 return  
 }  
  
 w.Header().Set("Content-Type", "application/json")  
 json.NewEncoder(w).Encode(greetings)  
 }  
}  
  
// Функция обработчика HTTP-запросов  
func greetHandler(dbProvider \*DatabaseProvider) http.HandlerFunc {  
 return func(w http.ResponseWriter, r \*http.Request) {  
 query, err := url.ParseQuery(r.URL.RawQuery)  
 if err != nil {  
 http.Error(w, "Неверные параметры запроса", http.StatusBadRequest)  
 return  
 }  
  
 name := query.Get("name")  
 if name == "" {  
 http.Error(w, "Отсутствует параметр 'name'", http.StatusBadRequest)  
 return  
 }  
  
 greeting, err := dbProvider.GreetUser(name)  
 if err != nil {  
 http.Error(w, fmt.Sprintf("Ошибка базы данных: %v", err), http.StatusInternalServerError)  
 return  
 }  
  
 w.Header().Set("Content-Type", "application/json")  
 json.NewEncoder(w).Encode(greeting)  
 }  
}  
  
func main() {  
 // Настройка с помощью флагов  
 var dbConfig dbConfig  
 flag.StringVar(&dbConfig.Host, "db-host", "localhost", "Хост базы данных")  
 flag.IntVar(&dbConfig.Port, "db-port", 5432, "Порт базы данных")  
 flag.StringVar(&dbConfig.User, "db-user", "postgres", "Пользователь базы данных")  
 flag.StringVar(&dbConfig.Password, "db-password", "", "Пароль базы данных")  
 flag.StringVar(&dbConfig.DBName, "db-dbname", "sandbox", "Имя базы данных")  
 flag.StringVar(&dbConfig.SSLMode, "db-sslmode", "disable", "Режим SSL для базы данных (disable, require, verify-ca, verify-full)")  
 flag.Parse()  
  
 // Создаем провайдер для базы данных  
 dbProvider, err := NewDatabaseProvider(dbConfig)  
 if err != nil {  
 log.Fatalf("Ошибка при создании провайдера базы данных: %v", err)  
 }  
 defer dbProvider.db.Close()  
  
 http.HandleFunc("/api/user", greetHandler(dbProvider))  
 http.HandleFunc("/api/greetings", getAllGreetingsHandler(dbProvider))  
 fmt.Println("Сервер запущен на порту :8080")  
 log.Fatal(http.ListenAndServe(":8080", nil))  
}*

Пополняем БД через POST запросы вида «*http://localhost:8080/api/user?name=John%20Doe*» (рис. 9):

Изображение выглядит как текст, снимок экрана, программное обеспечение, число

Автоматически созданное описание

Рис. 9

Проверяем результаты в СУБД (мною была допущена ошибка и по случайности добавлены одинаковые значения 6 раз) (рис. 10):

Изображение выглядит как текст, снимок экрана, программное обеспечение, число

Автоматически созданное описание

Рис. 10

Возвращаем все значения через GET запрос (рис. 11):

Изображение выглядит как текст, снимок экрана, программное обеспечение, число

Автоматически созданное описание

Рис. 11

Нам возвращает JSON массив со всеми значениями состоящими в нашей таблице