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 взаимодействия между Golang и PostgreSQL

Дисциплина: Языки интернет-программирования

Студент ИУ6-31Б _____ Д.Ю. Воронин
(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

Задание:

1. Установить и настроить PostgreSQL
2. Ознакомиться с теоретическими сведениями
3. Сделать форк данного репозитория в GitHub, склонировать получившуюся копию локально, создать от мастера ветку dev и переключиться на неё
4. Перекопировать код сервисов, полученный в ходе выполнения 6-й лабораторной работы, в соответствующие поддиректории в директории cmd (кроме кода сервиса hello, т.к. он уже реализован в качестве примера)
5. 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)
6. Проверить свой код линтерами с помощью команды make lint
7. Сделать отчёт и поместить его в директорию docs
8. Зафиксировать изменения, сделать коммит и отправить получившееся состояние ветки дев в личный форк данного репозитория в GitHub
9. Через интерфейс GitHub создать Pull Request dev --> master
10. На защите лабораторной работы продемонстрировать открытый Pull Request. PR должен быть направлен в master ветку форка, а не исходного репозитория

Ход работы

1. **Задача query**

Доработаем код согласно условию:

```
package main

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

	_ "github.com/lib/pq"
)

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

type Handlers struct {
	dbProvider DatabaseProvider
}

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

```
// Обработчики HTTP-запросов
func (h *Handlers) GetQuery(w http.ResponseWriter, r *http.Request) {
	if r.Method != http.MethodGet {
		w.WriteHeader(http.StatusMethodNotAllowed)
		return
	}
	msg, err := h.dbProvider.SelectQuery()
	if err != nil {
		w.WriteHeader(http.StatusInternalServerError)
		w.Write([]byte(err.Error()))
	} else if msg == "" {
		w.WriteHeader(http.StatusBadRequest)
		w.Write([]byte("Неинициализированное значение в базе данных!"))
	} else {
		w.WriteHeader(http.StatusOK)
		w.Write([]byte("Hello, " + msg + "!"))
	}
}
func (h *Handlers) PostQuery(w http.ResponseWriter, r *http.Request) {
	if r.Method != http.MethodPost {
		w.WriteHeader(http.StatusMethodNotAllowed)
		return
	}
	input := struct {
		Msg string `json:"name"`
	}{}

	decoder := json.NewDecoder(r.Body)
	err := decoder.Decode(&input)
	if err != nil {
		w.WriteHeader(http.StatusBadRequest)
```

```
            w.Write([]byte(err.Error()))
        } else if input.Msg == "" {
            w.WriteHeader(http.StatusBadRequest)
            w.Write([]byte("Пустая строка!"))
        } else {
            err = h.dbProvider.UpdateQuery(input.Msg)
            if err != nil {
                w.WriteHeader(http.StatusInternalServerError)
                w.Write([]byte(err.Error()))
            } else {
                w.WriteHeader(http.StatusAccepted)
            }
        }
}

// Методы для работы с базой данных
func (dp *DatabaseProvider) SelectQuery() (string, error) {
        var msg string

        row := dp.db.QueryRow("SELECT name FROM query")
        err := row.Scan(&msg)
        if err != nil {
            return "", err
        }

        return msg, nil
}
func (dp *DatabaseProvider) UpdateQuery(msg string) error {
        _, err := dp.db.Exec("UPDATE query SET name = $1", msg)
        if err != nil {
            return err
```

```
	}

	return nil
}

func main() {

	// 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.GetQuery)
	http.HandleFunc("/post", h.PostQuery)

	fmt.Println("Сервер запущен")
	// Запускаем веб-сервер на указанном адресе
	err = http.ListenAndServe(":8082", nil)
```

```
        if err != nil {
                log.Fatal(err)
        }
}
```

Проверим работоспособность программы – рисунки 1-4.

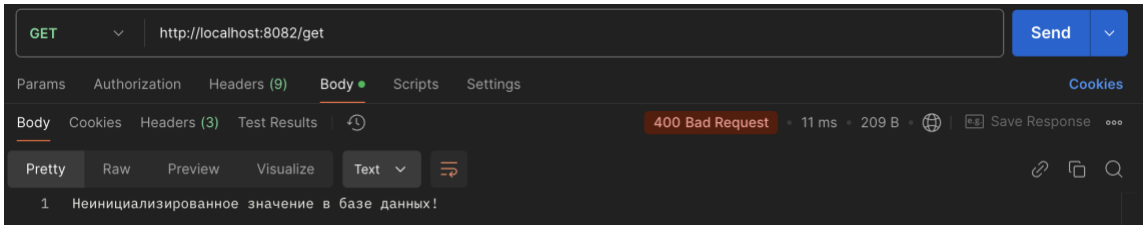

Рисунок 1 – Неинициализированное значение в базе данных

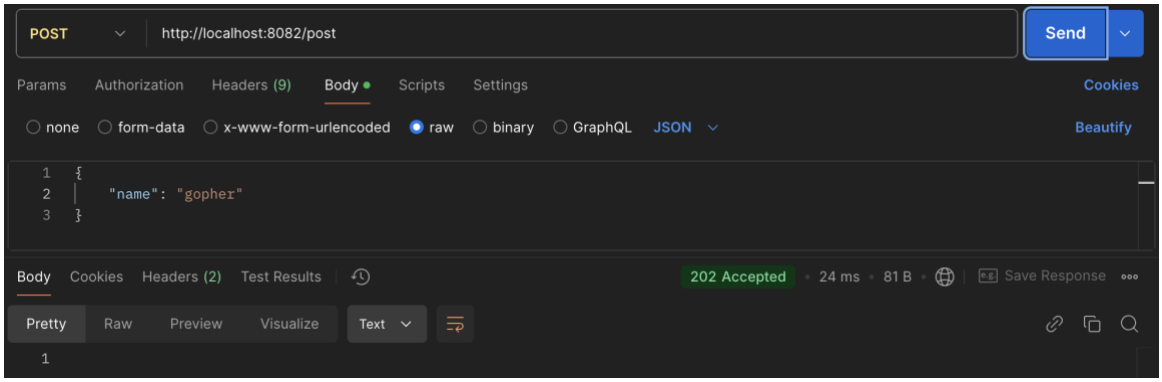

Рисунок 2 – POST запрос с параметром “gopher”

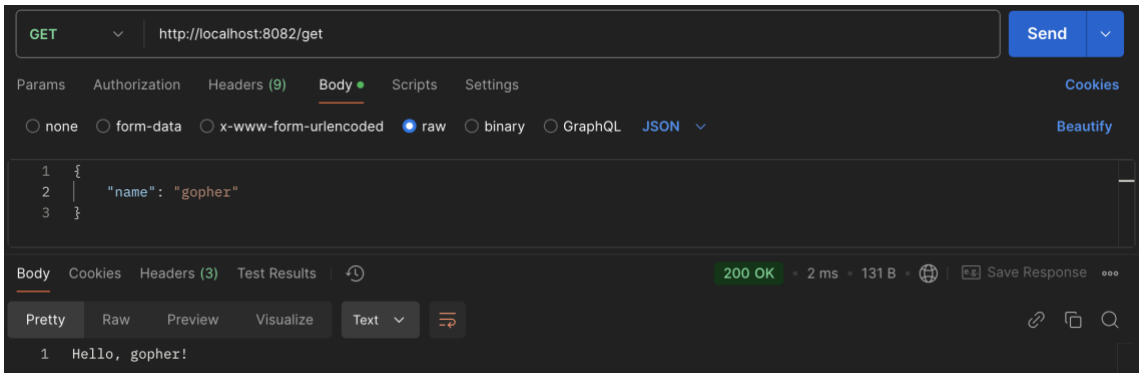

Рисунок 3 – Успешный GET запрос

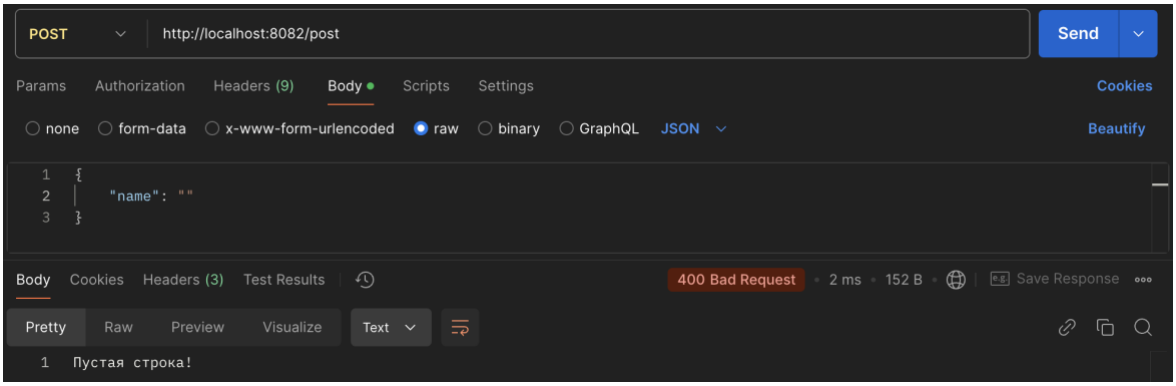

Рисунок 4 – Неудачный POST запрос

2. Задача count

Доработаем код согласно условию:

```
package main

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

	_ "github.com/lib/pq"
)

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

type Handlers struct {
	dbProvider DatabaseProvider
}

type DatabaseProvider struct {
	db *sql.DB
}

// Обработчики HTTP-запросов
```

```
func (h *Handlers) GetCount(w http.ResponseWriter, r *http.Request) {
	if r.Method != http.MethodGet {
		w.WriteHeader(http.StatusMethodNotAllowed)
		return
	}
	msg, err := h.dbProvider.SelectCount()
	if err != nil {
		w.WriteHeader(http.StatusInternalServerError)
		w.Write([]byte(err.Error()))
	} else {
		w.WriteHeader(http.StatusOK)
		w.Write([]byte(msg))
	}
}
func (h *Handlers) PostCount(w http.ResponseWriter, r *http.Request) {
	if r.Method != http.MethodPost {
		w.WriteHeader(http.StatusMethodNotAllowed)
		return
	}
	input := struct {
		Msg *int `json:"count"`
	}{}
	decoder := json.NewDecoder(r.Body)
	err := decoder.Decode(&input)
	if input.Msg == nil {
		w.WriteHeader(http.StatusBadRequest)
		w.Write([]byte("Отсутствует поле 'count'!"))
	} else if err != nil {
		w.WriteHeader(http.StatusBadRequest)
		w.Write([]byte(err.Error()))
	} else {
```

```
                err = h.dbProvider.UpdateCount(*input.Msg)
                if err != nil {
                        w.WriteHeader(http.StatusInternalServerError)
                        w.Write([]byte(err.Error()))
                } else {
                        w.WriteHeader(http.StatusAccepted)
                }
        }
}

// Методы для работы с базой данных
func (dp *DatabaseProvider) SelectCount() (string, error) {
        var msg string

        row := dp.db.QueryRow("SELECT num FROM counter")
        err := row.Scan(&msg)
        if err != nil {
                return "", err
        }

        return msg, nil
}
func (dp *DatabaseProvider) UpdateCount(msg int) error {
        _, err := dp.db.Exec("UPDATE counter SET num = num + $1", msg)
        if err != nil {
                return err
        }

        return nil
}
```

```
func main() {

	// 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.GetCount)
	http.HandleFunc("/post", h.PostCount)

	fmt.Println("Сервер запущен")
	// Запускаем веб-сервер на указанном адресе
	err = http.ListenAndServe(":8083", nil)
	if err != nil {
		log.Fatal(err)
	}
}
```

Проверим работоспособность программы – рисунки 5-11

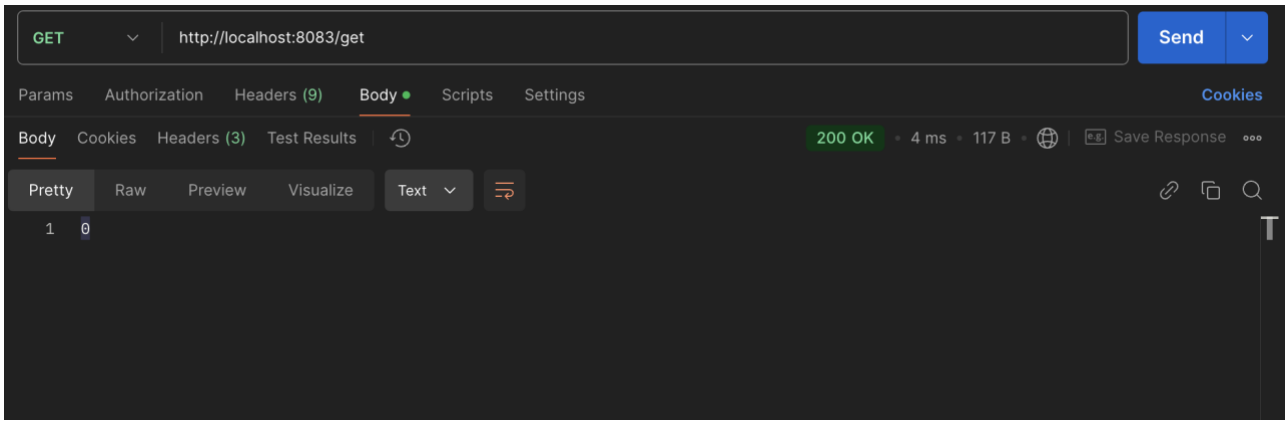

Рисунок 5 – GET запрос для определения начального значения

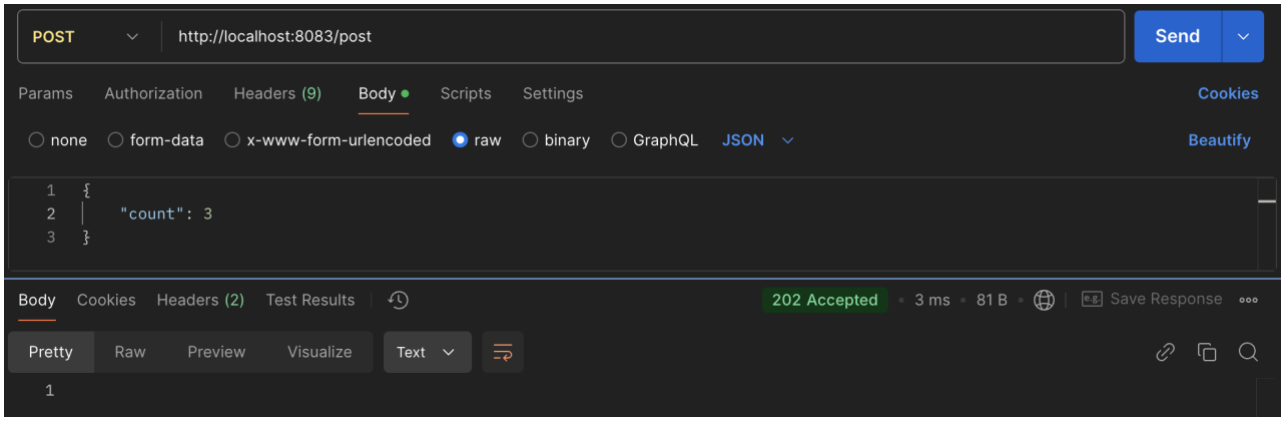

Рисунок 6 – POST запрос с положительным параметром

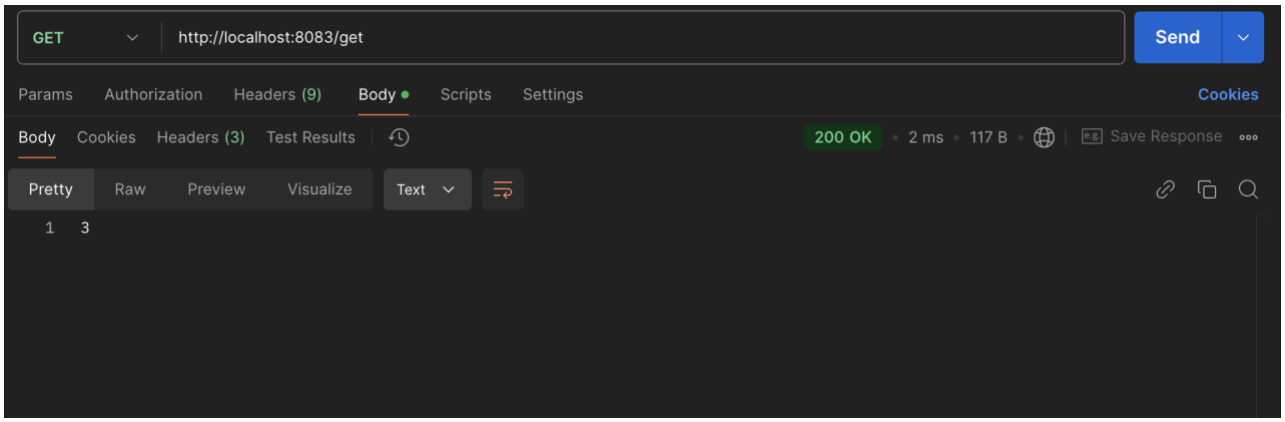

Рисунок 7 – GET запрос для проверки корректности данных после POST запроса

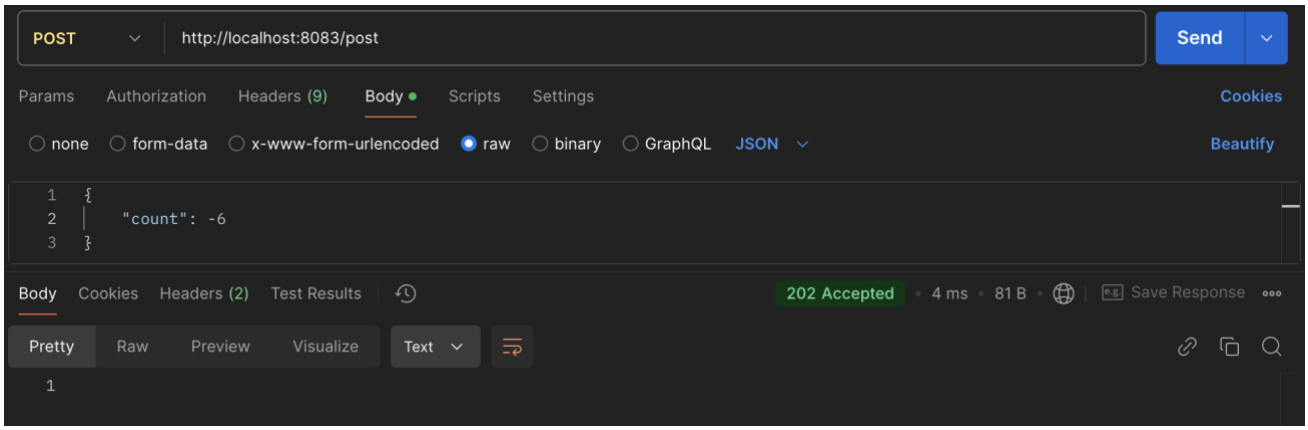

Рисунок 8 – POST запрос с отрицательным параметром

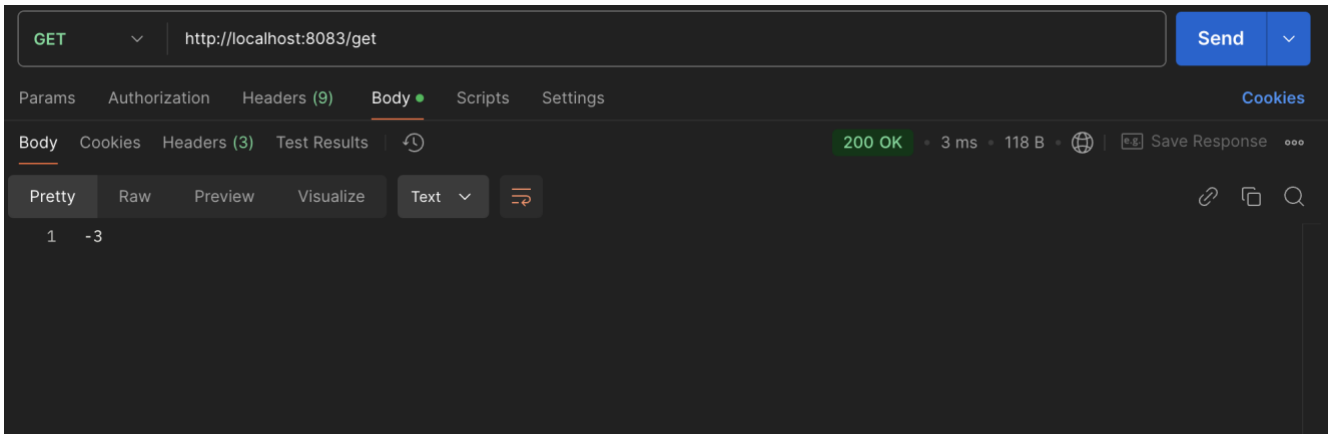

Рисунок 9 – GET запрос для проверки корректности данных после POST запроса с с отрицательным числом

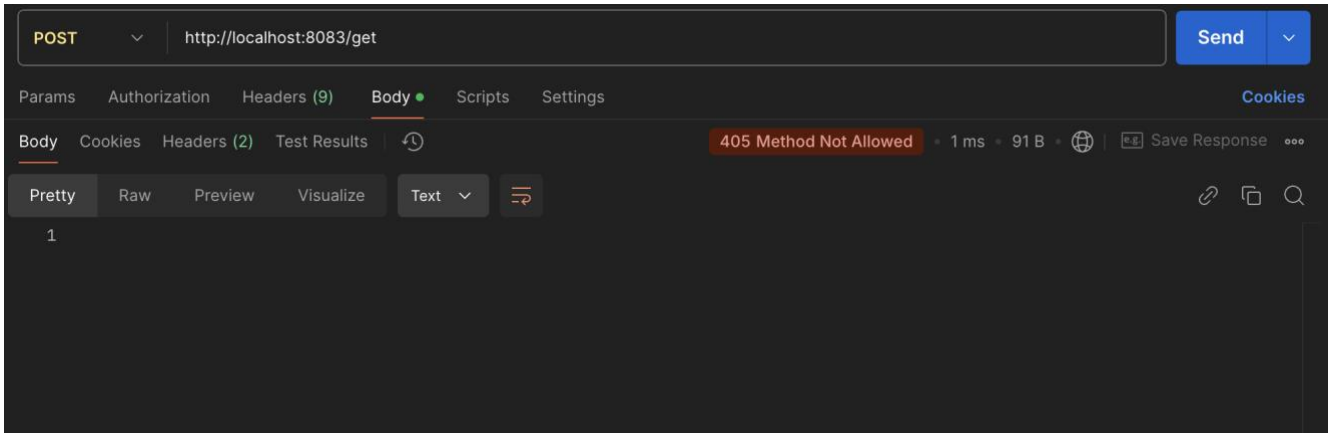

Рисунок 10 – Ошибка выбора неверного метода

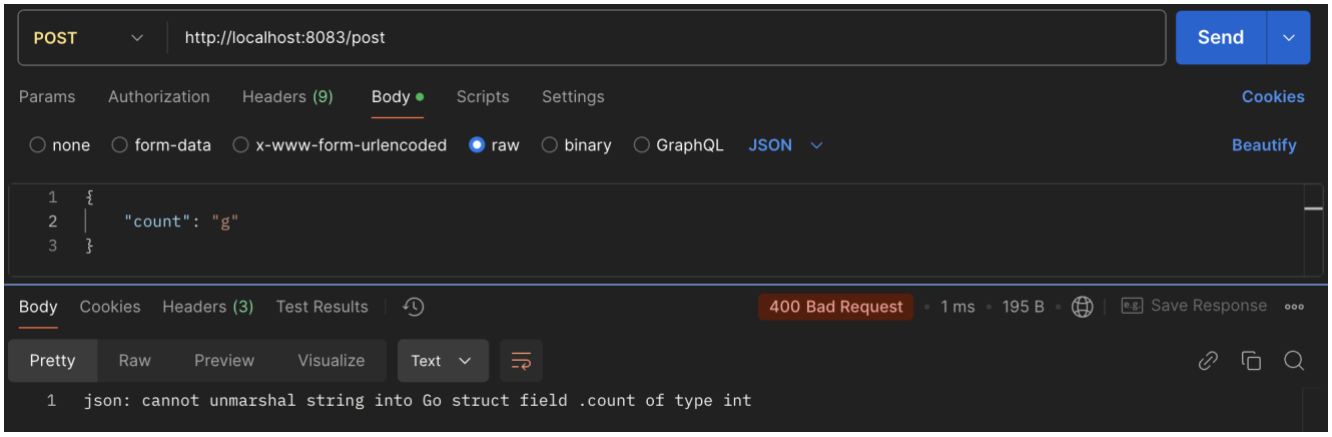

Рисунок 11 – Неверный тип данных в параметрах POST запроса

Заключение:

В результате данной лабораторной работы были получены базовые навыки организации долгосрочного хранения данных с использованием PostgreSQL и Go. Это включает в себя создание базы данных и управление её, проектирование

таблиц, выполнение SQL-запросов, а также интеграцию PostgreSQL с приложением на Go для эффективного взаимодействия с хранилищем данных.

Список использованных источников:

https://github.com/ValeryBMSTU/web-8