



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:** Golang & PostgreSQL

**Дисциплина:** Языки интернет программирования

Студент

ИУ6-32Б

(Группа)

\_\_\_\_\_  
(Подпись, дата)

Заушников Л.И.

(И.О. Фамилия)

Преподаватель

\_\_\_\_\_  
(Подпись, дата)

В.Д. Шульман

(И.О. Фамилия)

Москва, 2024

**Цель работы** — получить первичные навыки в организации долгосрочного хранения данных с использованием PostgreSQL и Golang.

## Ход работы

1. Было произведено ознакомление с материалами для подготовки перед выполнением лабораторной работы
2. Был сделан форк репозитория в GitHub (рисунок 1), копия была клонирована локально, была создана от мастера ветка dev и было произведено переключение на неё

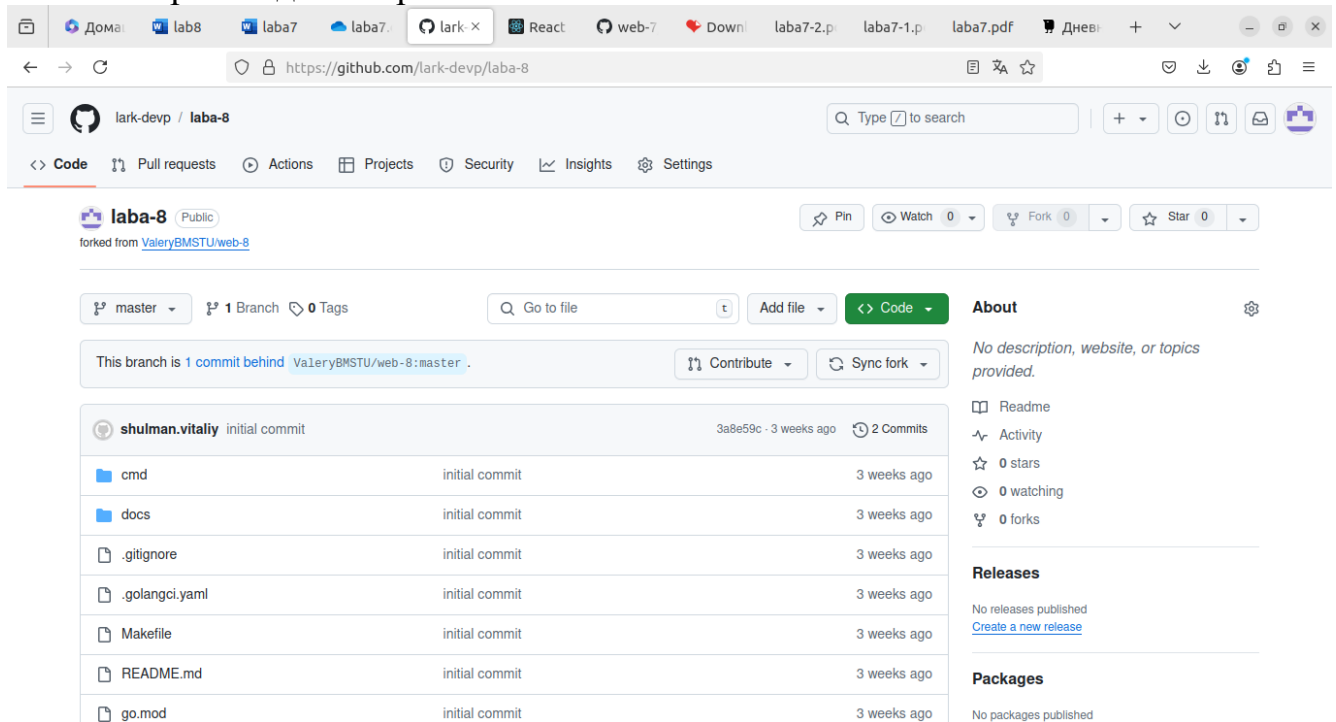


Рисунок 1 - Форкнутый репозиторий

Далее были обновлены и протестированы 3 сервиса, все коды были проверены линтами.

**Сервис hello**  
package main

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

```
    _ "github.com/lib/pq"  
)
```

```
const (  
    host    = "localhost"  
    port    = 5432  
    user    = "lark_dev"  
    password = "Annapetrovna2005"
```

```

dbname = "hello"
)

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)
}
}

```

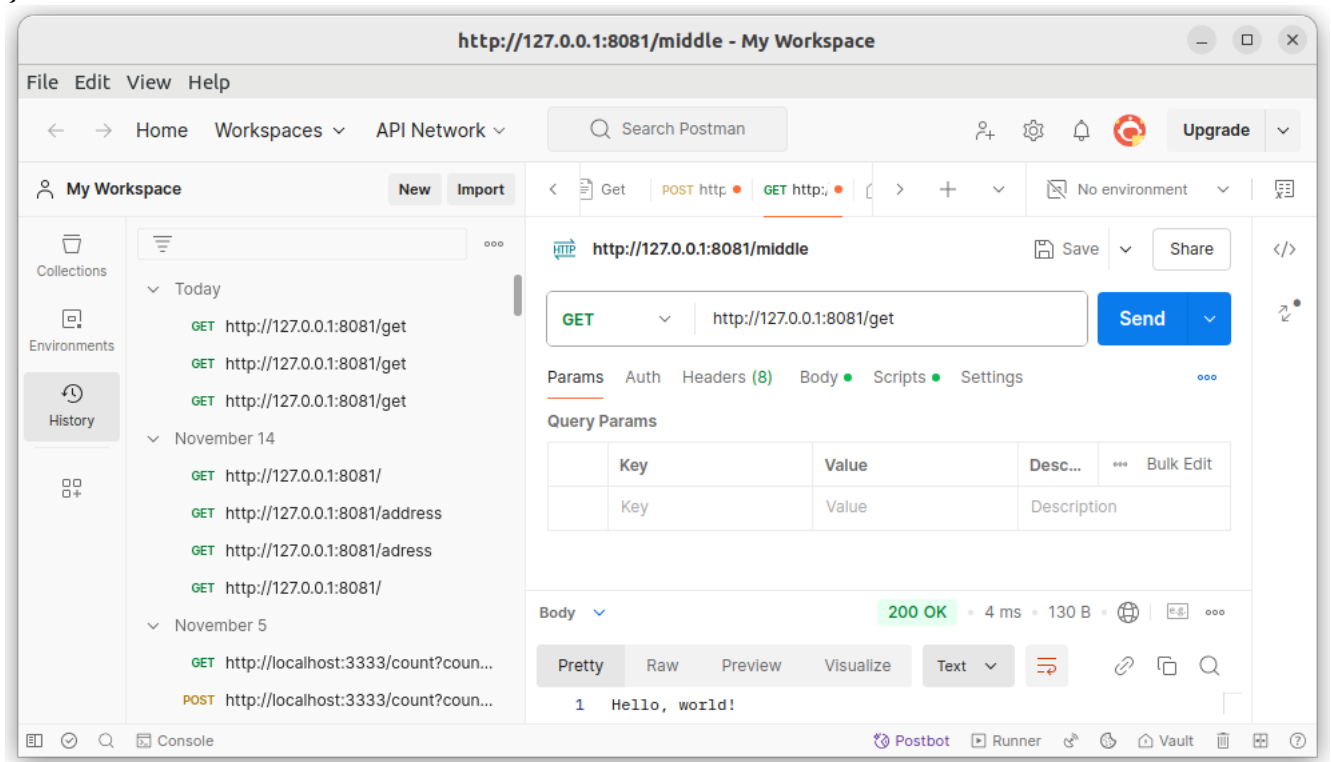


Рисунок 2 - Тестирование сервиса hello

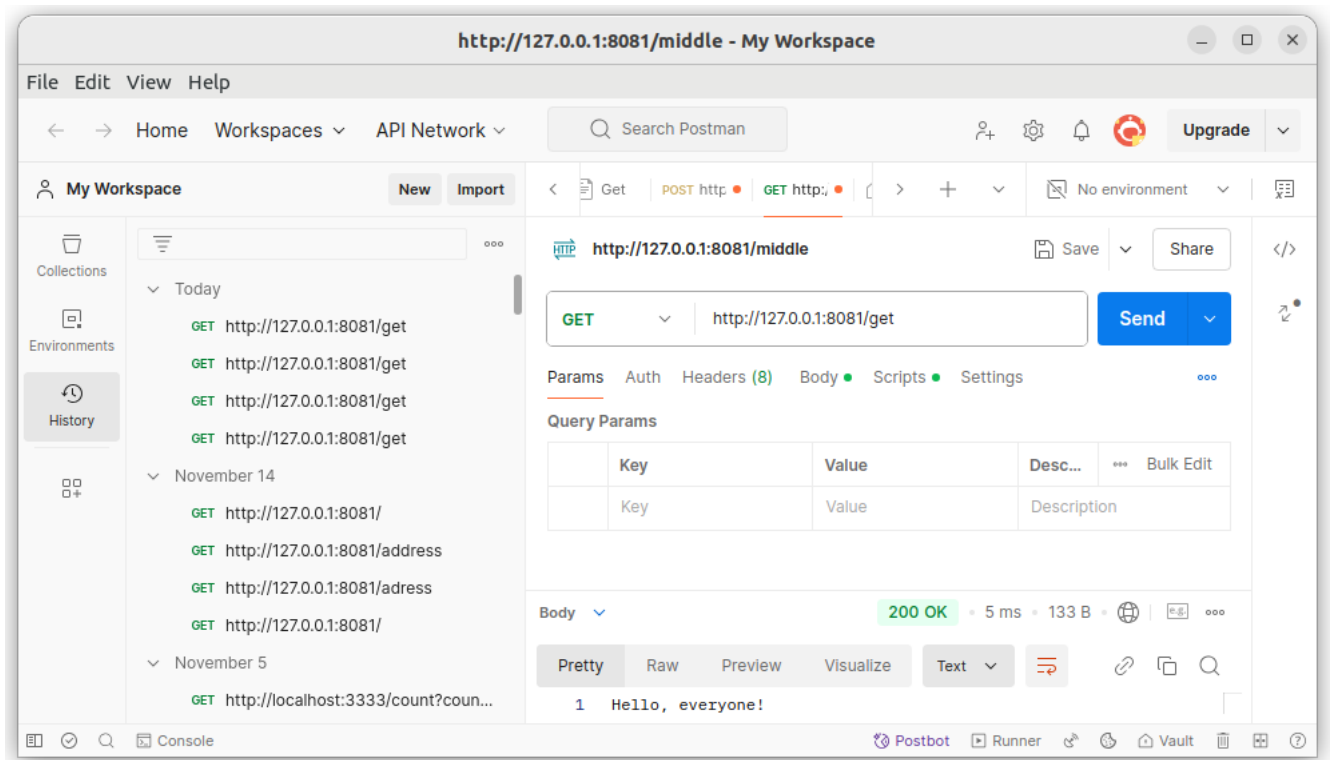


Рисунок 3 - Тестирование сервиса hello

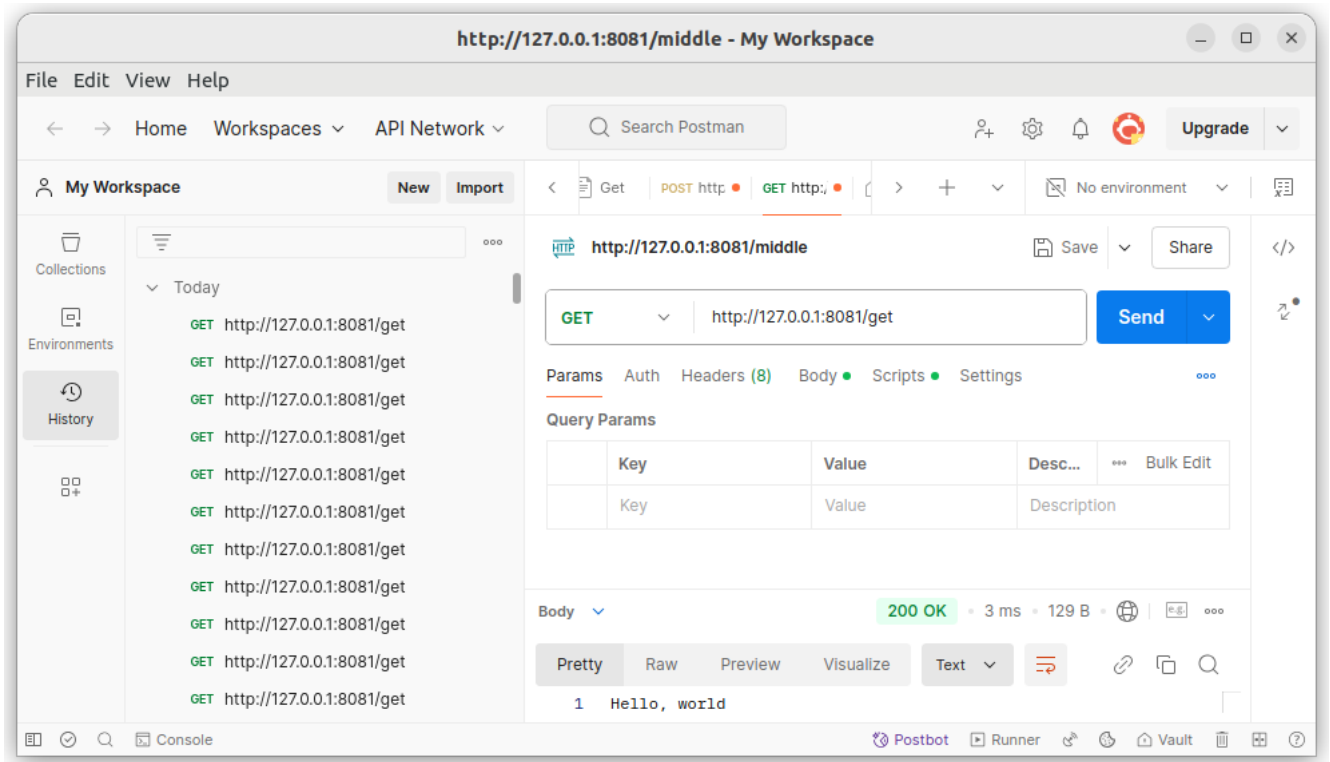


Рисунок 4 - Тестирование сервиса hello

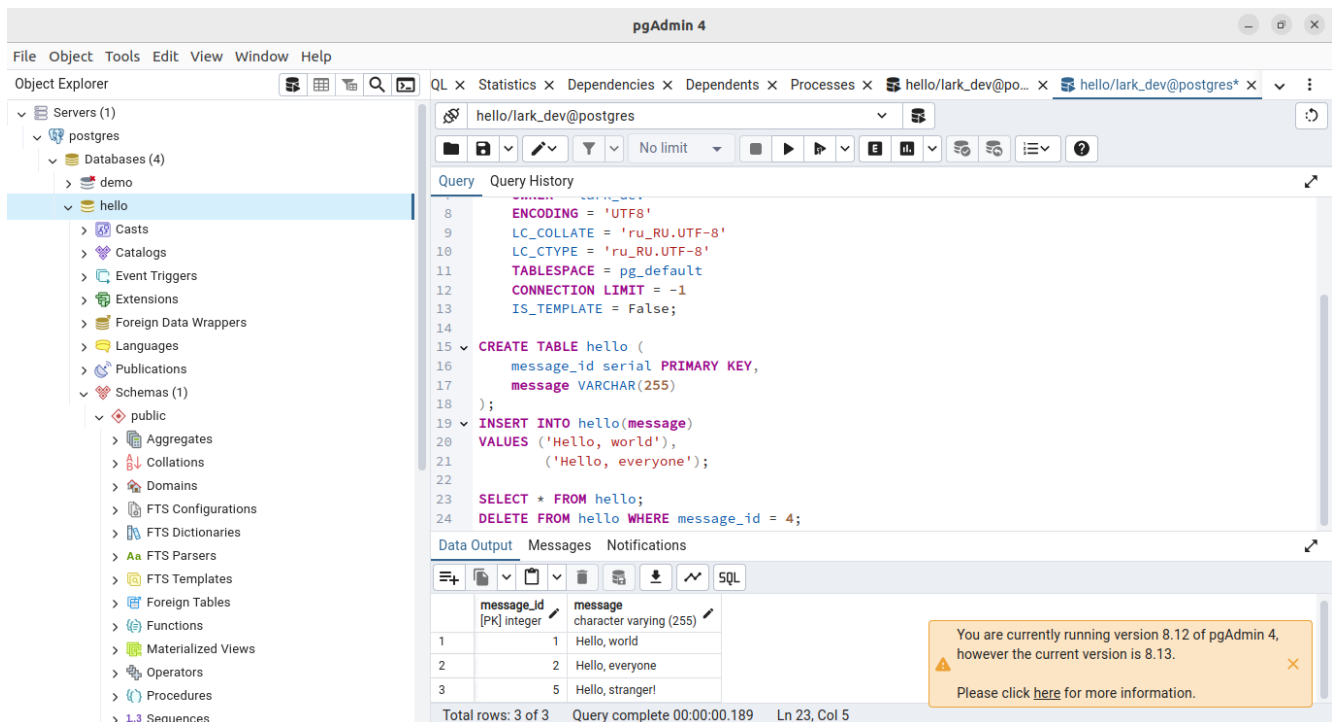


Рисунок 5 - Тестирование сервиса hello

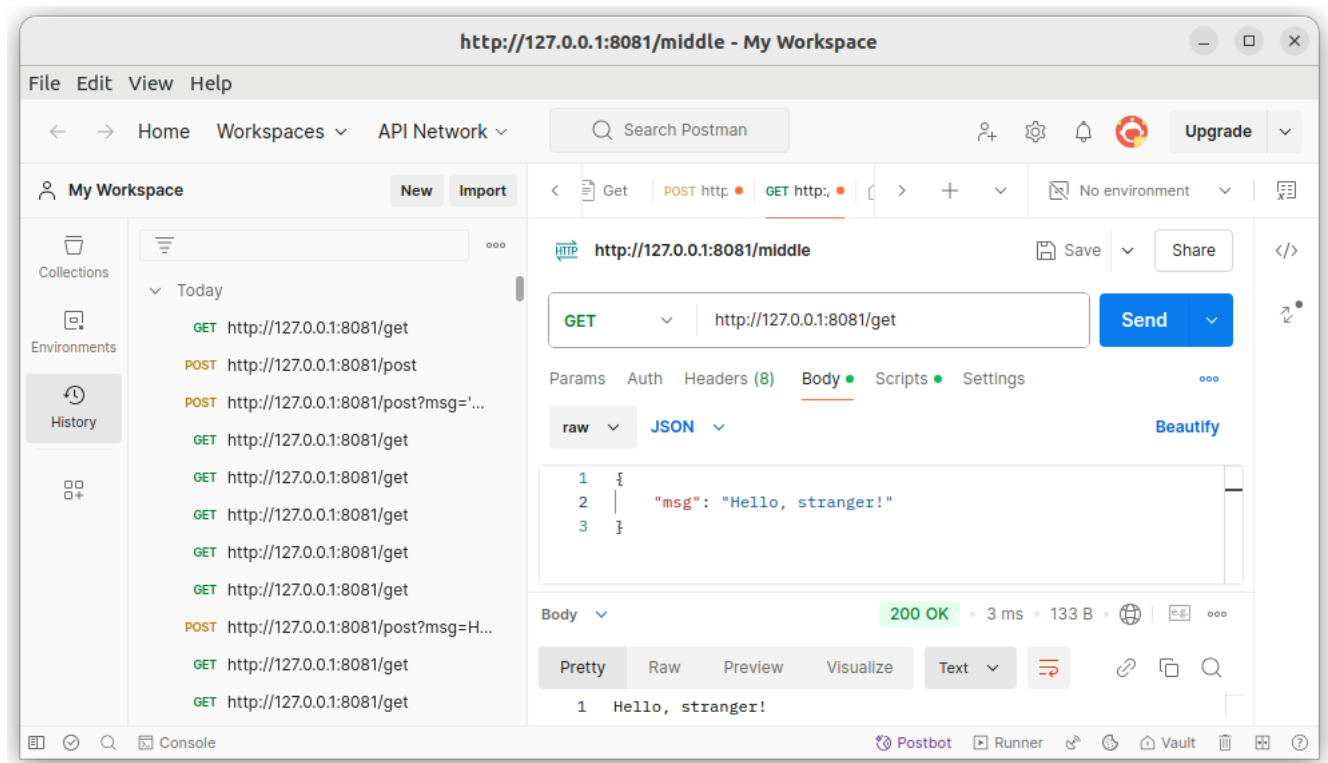


Рисунок 6 - Тестирование сервиса hello

## Сервис query package main

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

    _ "github.com/lib/pq" // подключение пакета для работы с PostgreSQL
)

// Константы для подключения к базе данных
const (
    dbHost    = "localhost"
    dbPort    = 5432
    dbUser    = "lark_dev"
    dbPassword = "Annapetrovna2005"
    dbName    = "query"
)

// User представляет пользователя в системе
type User struct {
    ID   int   `json:"id"`
    Name string `json:"name"`
}
```

```

}

// DatabaseProvider содержит соединение с базой данных
type DatabaseProvider struct {
db *sql.DB
}

// NewDatabaseProvider создает новый экземпляр DatabaseProvider
func NewDatabaseProvider() (*DatabaseProvider, error) {
// Формируем строку подключения
connStr := fmt.Sprintf("host=%s port=%d user=%s password=%s dbname=%s\nsslmode=disable",
dbHost, dbPort, dbUser, dbPassword, dbName) // sslmode будет проверять
подлинность сервера, проверяя цепочку доверия до корневого сертификата,

db, err := sql.Open("postgres", connStr)
if err != nil {
return nil, err
}

// Проверяем подключение
if err := db.Ping(); err != nil {
return nil, err
}

return &DatabaseProvider{db: db}, nil
}

// InsertUser добавляет нового пользователя в базу данных
func (dp *DatabaseProvider) InsertUser(name string) (int, error) {
var id int
err := dp.db.QueryRow("INSERT INTO users(name) VALUES($1) RETURNING id",
name).Scan(&id)
if err != nil {
return 0, err
}
return id, nil
}

// GetUser извлекает пользователя из базы данных по ID
func (dp *DatabaseProvider) GetUser(id int) (User, error) {
var user User
err := dp.db.QueryRow("SELECT id, name FROM users WHERE id = $1",
id).Scan(&user.ID, &user.Name)
if err != nil {
return User{}, err
}
return user, nil
}

```



```

}

// addUserHandler обрабатывает добавление пользователя
func (dp *DatabaseProvider) addUserHandler(w http.ResponseWriter, r *http.Request)
{
    if r.Method != http.MethodPost {
        http.Error(w, "Метод не разрешен", http.StatusMethodNotAllowed)
        return
    }

    var user User
    err := json.NewDecoder(r.Body).Decode(&user)
    if err != nil {
        http.Error(w, "Не удалось прочитать тело запроса", http.StatusBadRequest)
        return
    }

    id, err := dp.InsertUser(user.Name)
    if err != nil {
        http.Error(w, "Не удалось добавить пользователя", http.StatusInternalServerError)
        return
    }

    w.WriteHeader(http.StatusCreated)
    fmt.Fprintf(w, "Создан пользователь с ID: %d", id)
}

// getUserHandler обрабатывает извлечение пользователя по ID
func (dp *DatabaseProvider) getUserHandler(w http.ResponseWriter, r *http.Request) {
    if r.Method != http.MethodGet {
        http.Error(w, "Метод не разрешен", http.StatusMethodNotAllowed)
        return
    }
    idStr := r.URL.Query().Get("id")
    if idStr == "" {
        http.Error(w, "Отсутствует ID пользователя", http.StatusBadRequest)
        return
    }

    var id int
    _, err := fmt.Sscanf(idStr, "%d", &id) //Функция fmt.Sscanf() в языке Go сканирует
    указанную строку и сохраняет последовательные значения, разделенные
    пробелами, в последовательные аргументы, как определено форматом.
    if err != nil {
        http.Error(w, "Некорректный формат ID", http.StatusBadRequest)
        return
    }
}

```

```
user, err := dp.GetUser(id)
if err != nil {
    http.Error(w, "Пользователь не найден", http.StatusNotFound)
    return
}

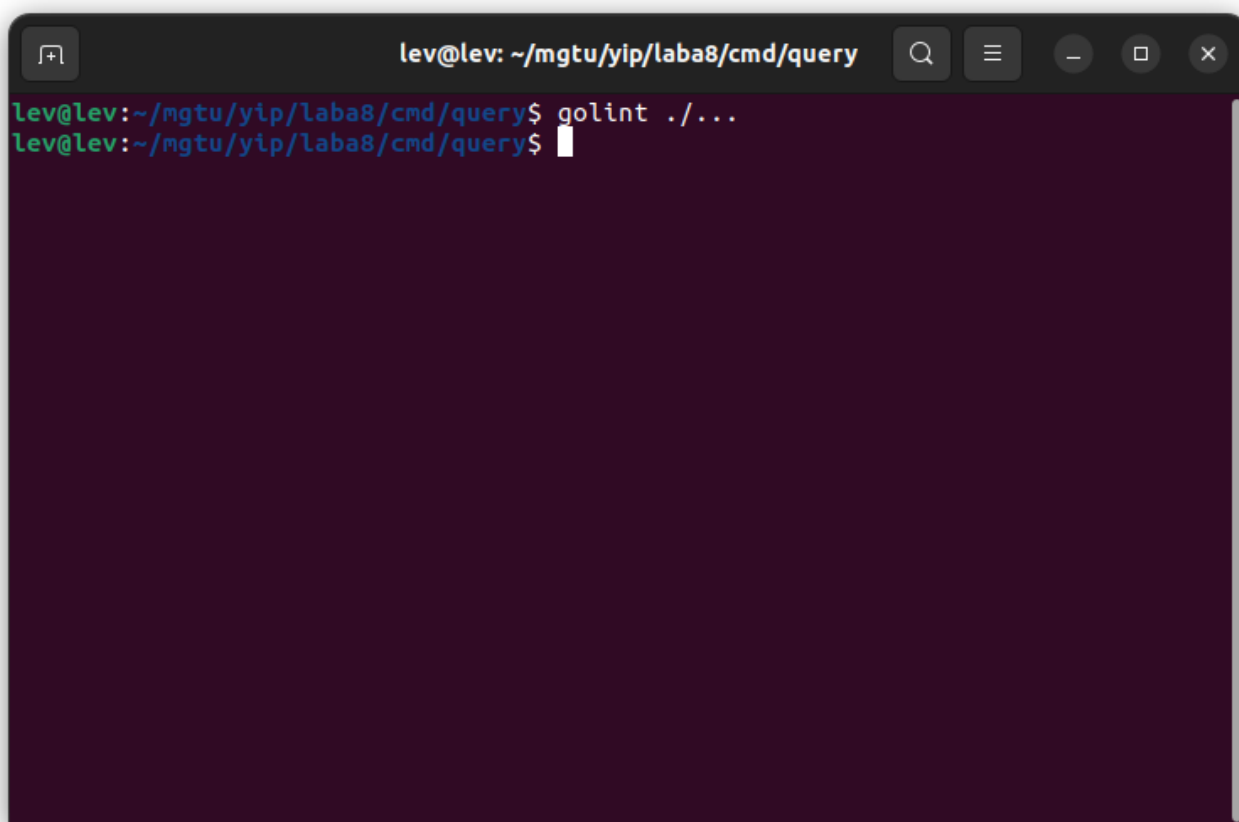
w.Header().Set("Content-Type", "application/json")
json.NewEncoder(w).Encode(user)
}

func main() {
    // Подключение к базе данных
    dbProvider, err := NewDatabaseProvider()
    if err != nil {
        log.Fatalf("Ошибка подключения к БД: %v", err) //используется для записи
        сообщений об ошибках в резервный журнал и завершения работы программы.
    }
    defer dbProvider.db.Close()

    // Регистрация обработчиков
    http.HandleFunc("/api/user/post", dbProvider.addUserHandler)
    http.HandleFunc("/api/user/get", dbProvider.getUserHandler)

    // Запуск сервера

    log.Println("Сервер запущен на порту 9000...")
    err = http.ListenAndServe(":9000", nil)
    if err != nil {
        log.Fatalf("Ошибка запуска сервера: %v", err)
    }
}
```



```
lev@lev: ~/mgtu/yip/laba8/cmd/query
lev@lev:~/mgtu/yip/laba8/cmd/query$ golangci-lint ./...
lev@lev:~/mgtu/yip/laba8/cmd/query$
```

Рисунок 7 - Проверка кода сервиса query линтами

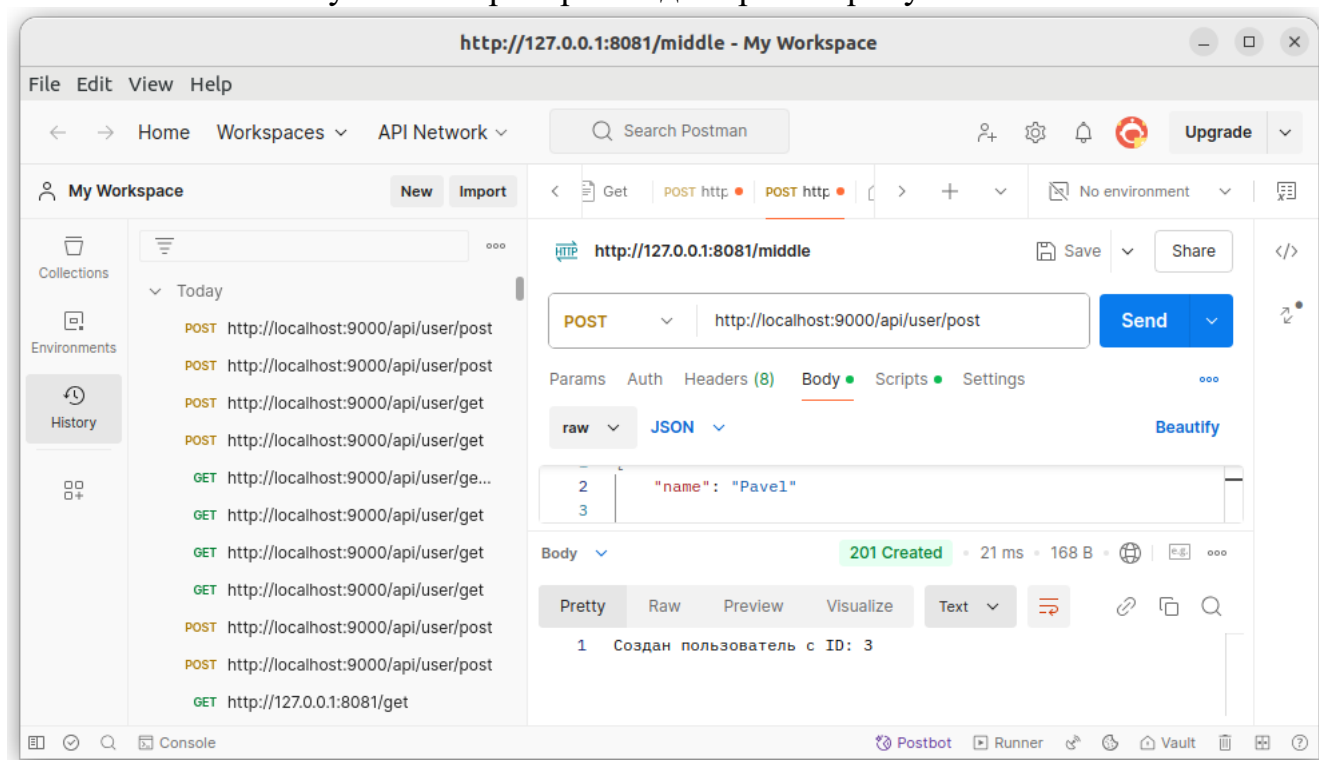


Рисунок 8 - Тестирование сервиса query

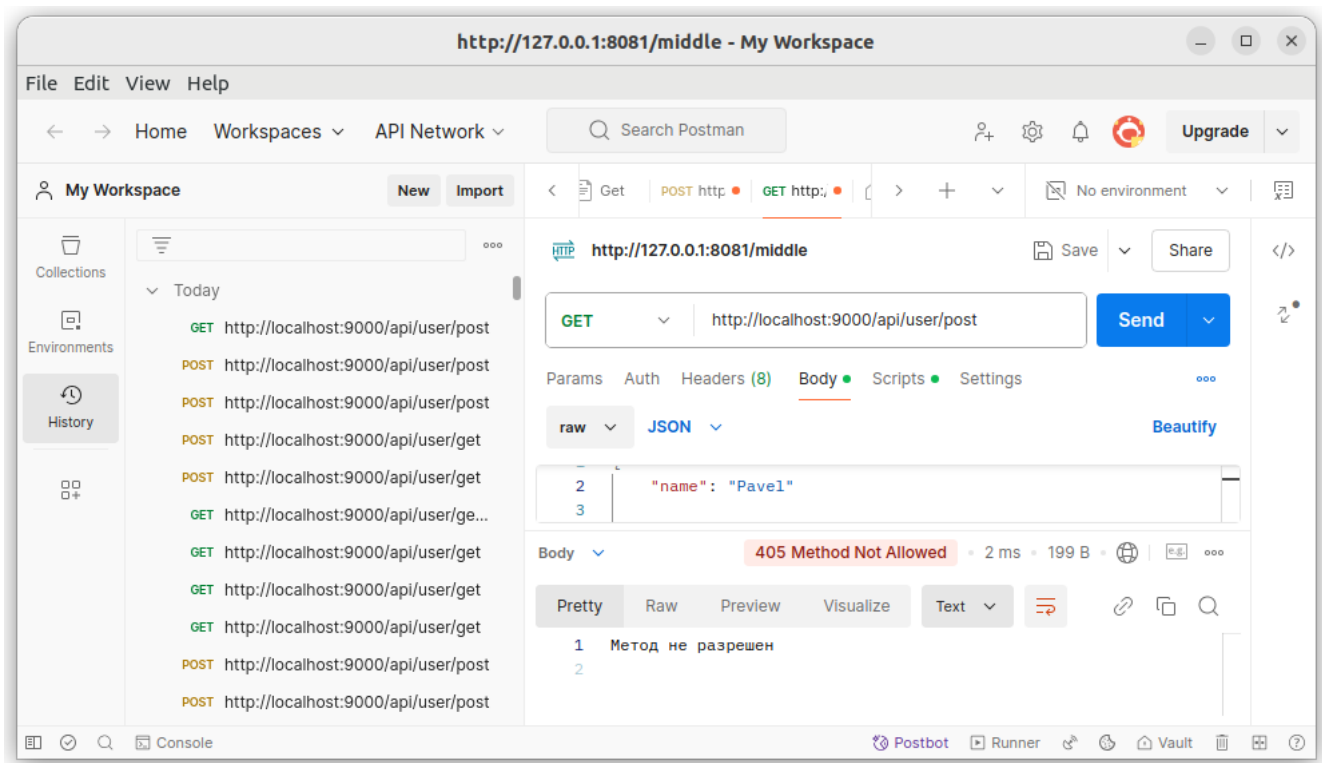


Рисунок 9 - Тестирование сервиса query

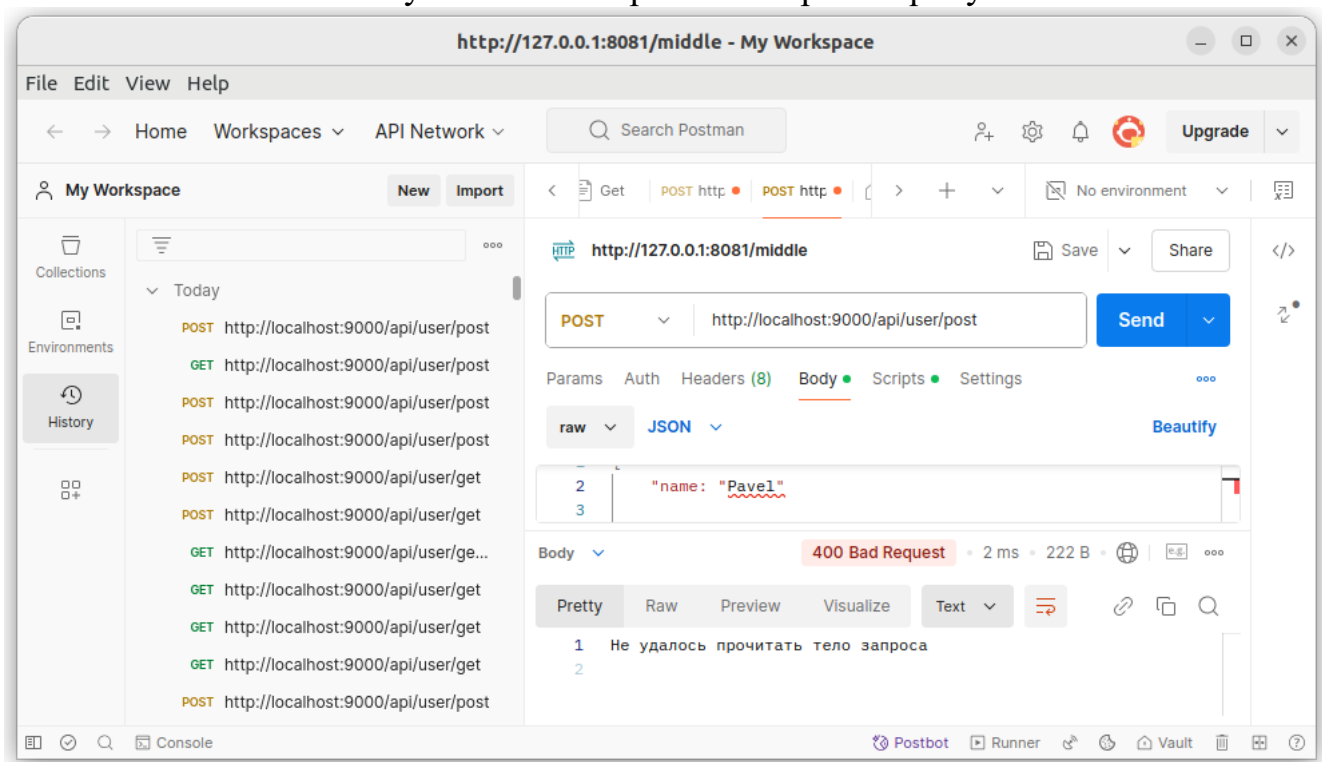


Рисунок 10 - Тестирование сервиса query

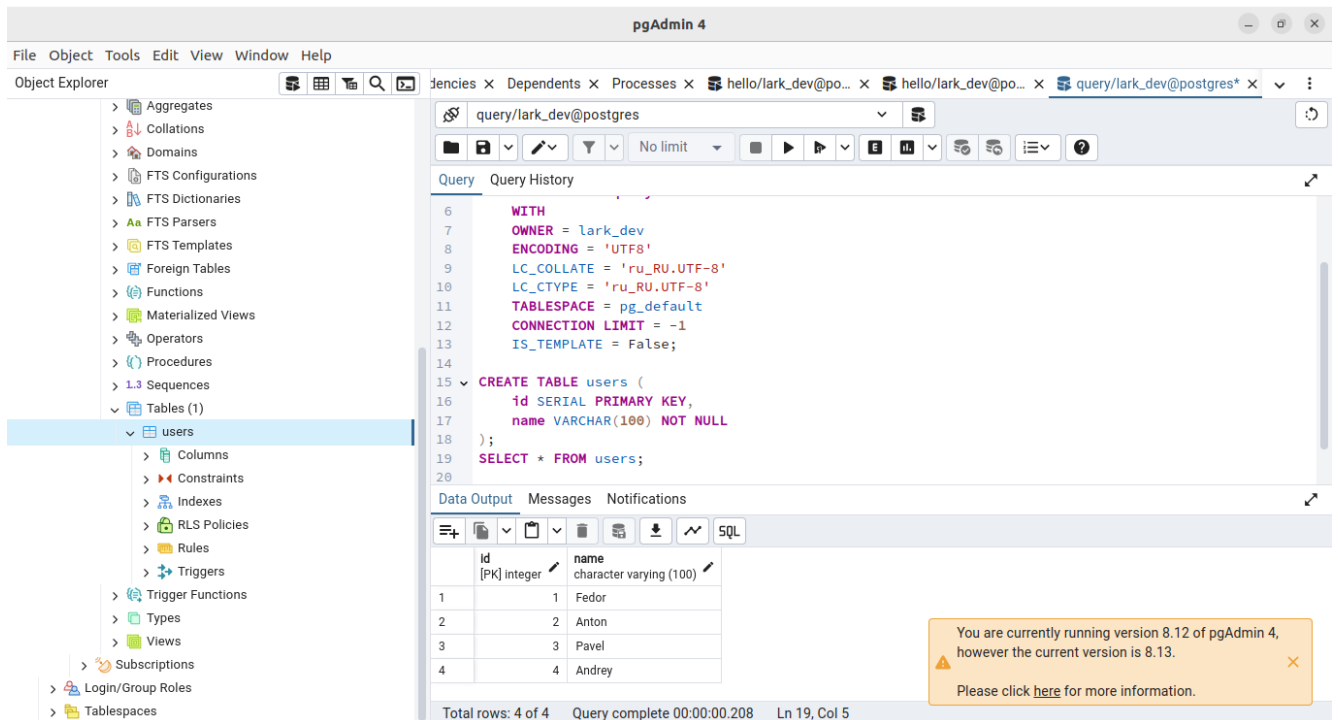


Рисунок 11 - Тестирование сервиса query

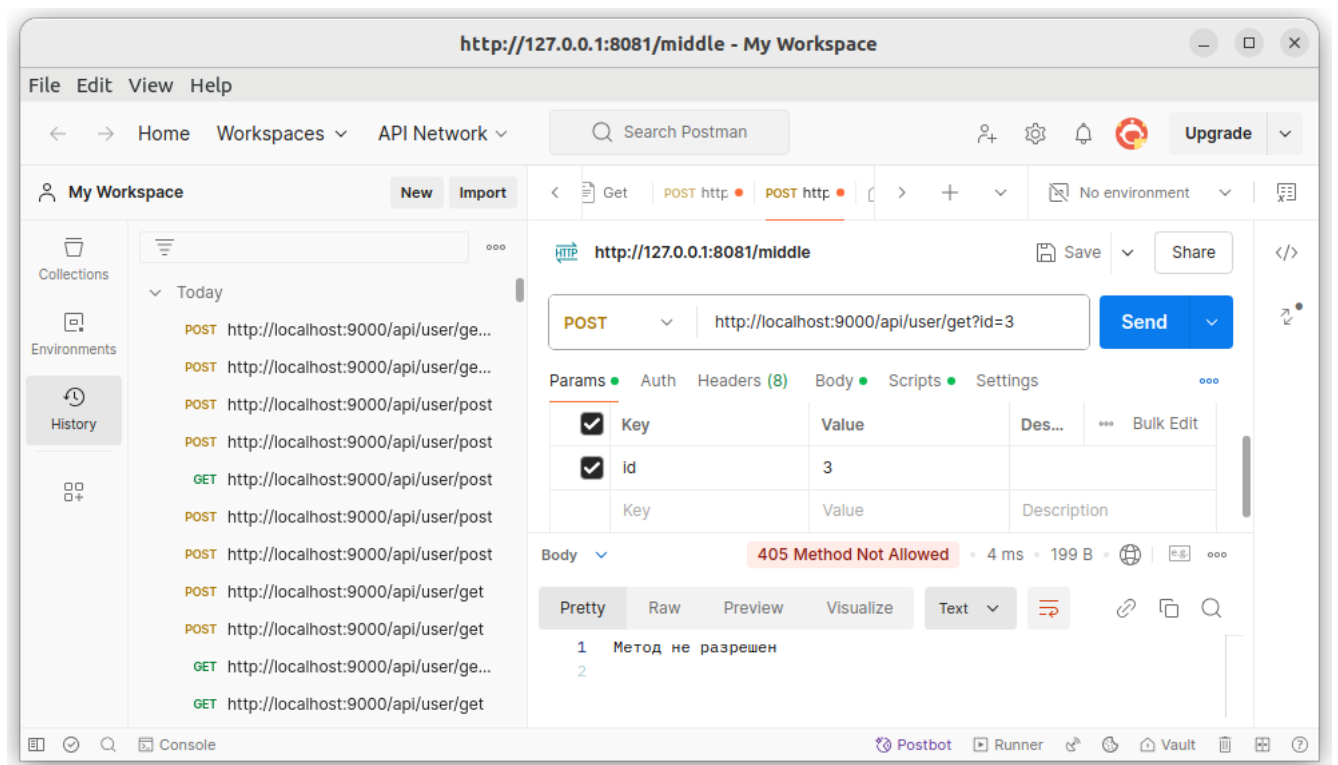


Рисунок 12 - Тестирование сервиса query

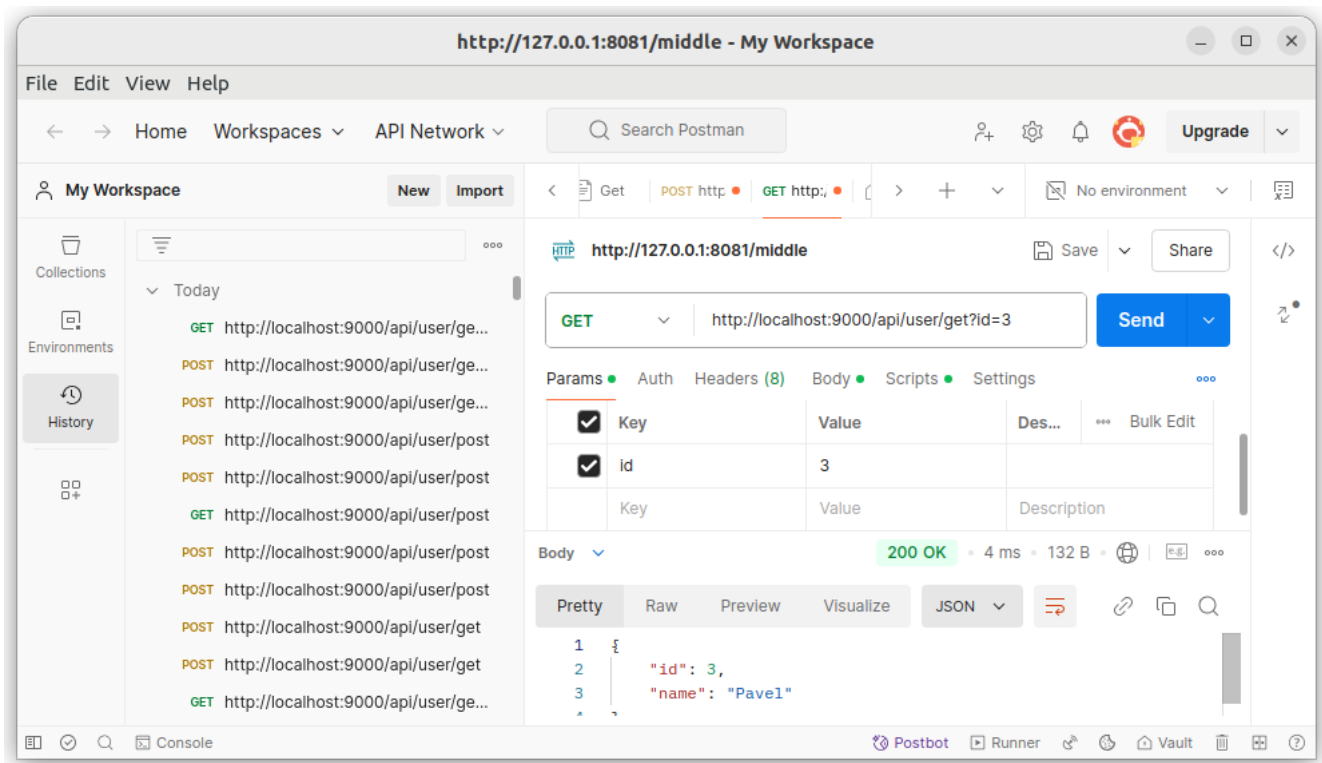


Рисунок 13 - Тестирование сервиса query

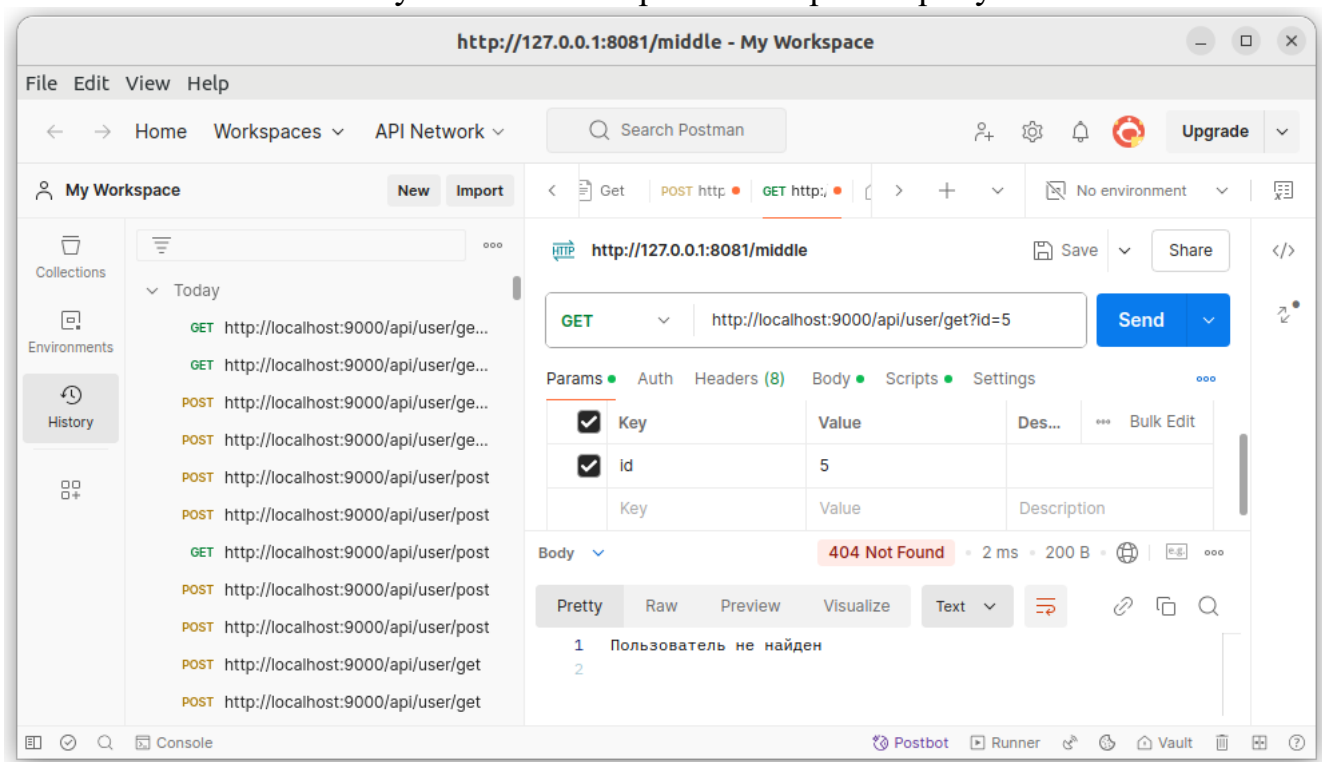


Рисунок 14 - Тестирование сервиса query

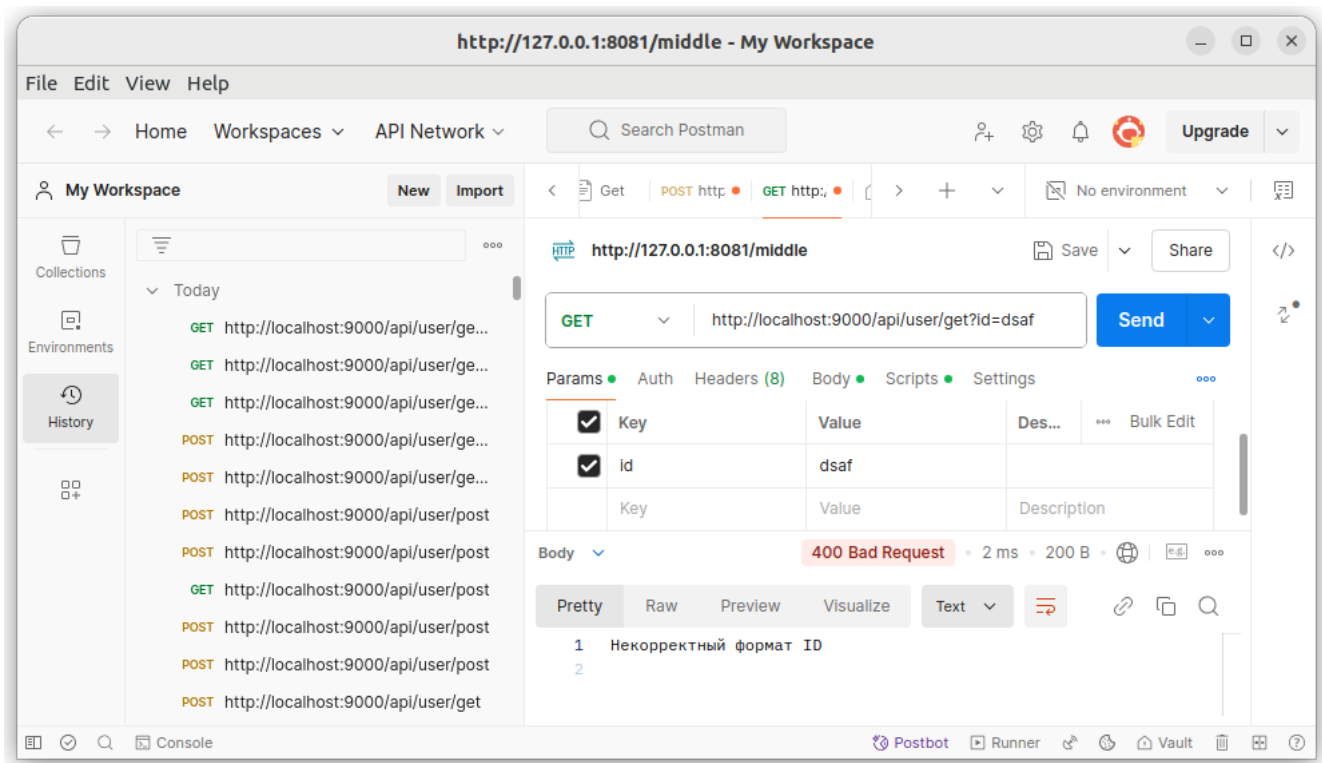


Рисунок 15 - Тестирование сервиса query

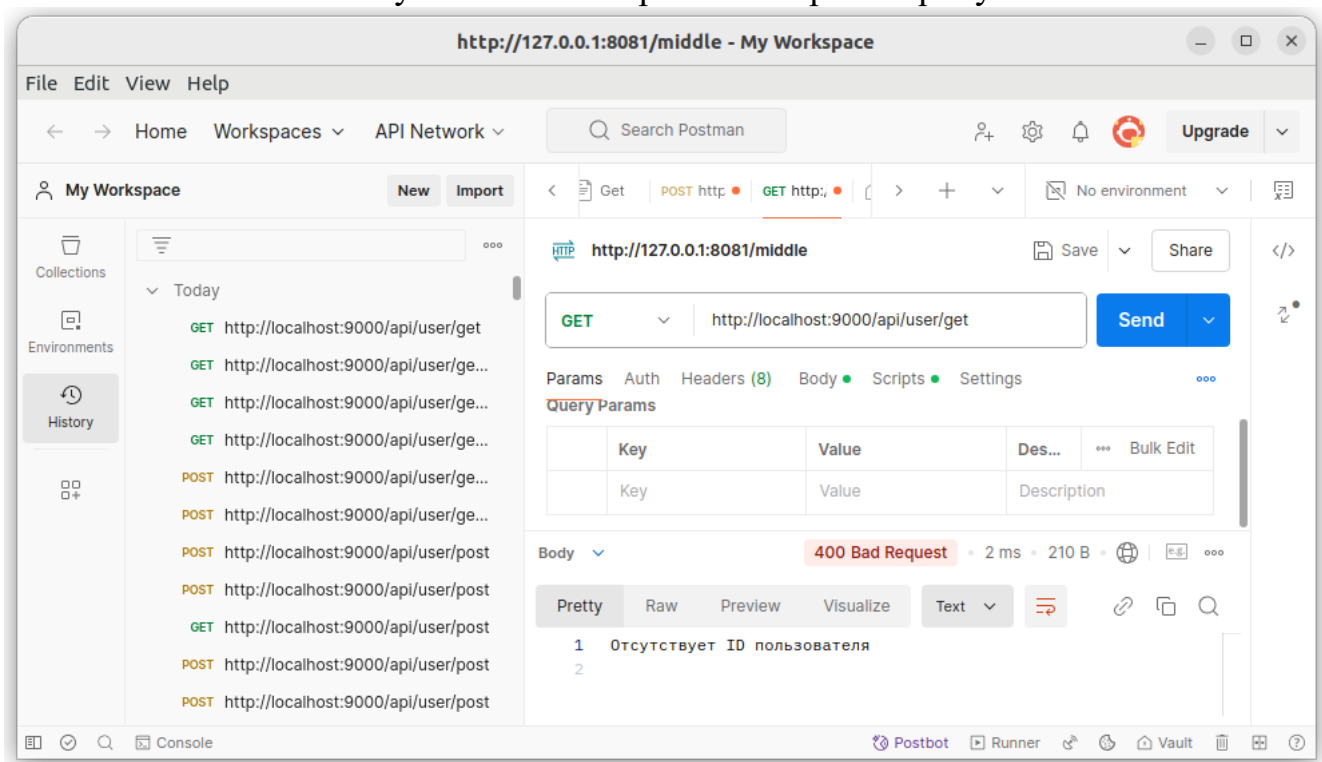


Рисунок 16 - Тестирование сервиса query

## Сервис count

package main

```
import (
    "database/sql"
    "fmt"
```

```

"net/http"
"strconv"

_ "github.com/lib/pq" // PostgreSQL driver
)

// Константы подключения к базе данных
const (
dbUser    = "lark_dev"
dbPassword = "Annapetrovna2005"
dbName    = "count"
dbHost    = "localhost"
dbPort    = 5432 // стандартный порт PostgreSQL
)

var db *sql.DB

// Инициализируем базу данных
func initDB() error {
var err error
connStr := fmt.Sprintf("user=%s password=%s dbname=%s host=%s port=%d sslmode=disable",
dbUser, dbPassword, dbName, dbHost, dbPort)
db, err = sql.Open("postgres", connStr)
if err != nil {
return err
}

// Проверка подключения к базе данных
if err := db.Ping(); err != nil {
return err
}

// Создание таблицы, если она не существует
_, err = db.Exec(`
    CREATE TABLE IF NOT EXISTS counter (
        id SERIAL PRIMARY KEY,
        value INTEGER
    )
`)
if err != nil {
return err
}

// Инициализация значения счетчика в БД, если таблица пуста
var count int
err = db.QueryRow("SELECT value FROM counter WHERE id = 1").Scan(&count)
if err == sql.ErrNoRows {

```



```

// Таблица пустая, добавляем начальное значение
_, err = db.Exec("INSERT INTO counter (value) VALUES (0)")
} else if err != nil {
return err
} // иначе, значение счётчика уже инициализировано в базе данных
return err
}

func countHandler(w http.ResponseWriter, r *http.Request) {
switch r.Method {
case http.MethodGet:
var count int
err := db.QueryRow("SELECT value FROM counter WHERE id = 1").Scan(&count)
if err != nil {
http.Error(w, "Ошибка получения счётчика", http.StatusInternalServerError)
return
}
w.WriteHeader(http.StatusOK)
w.Write([]byte(strconv.Itoa(count)))
case http.MethodPost:
err := r.ParseForm()
if err == nil {
countStr := r.FormValue("count")
if countStr == "" {
w.WriteHeader(http.StatusBadRequest)
w.Write([]byte("Введите, пожалуйста, число"))
return
}

count, err := strconv.Atoi(countStr)
if err != nil {
w.WriteHeader(http.StatusBadRequest)
w.Write([]byte("это не число"))
return
}

// Обновляем значение счетчика в БД
_, err = db.Exec("UPDATE counter SET value = value + $1 WHERE id = 1", count)
if err != nil {
http.Error(w, "Ошибка обновления счётчика", http.StatusInternalServerError)
return
}
w.WriteHeader(http.StatusOK)
w.Write([]byte("Счетчик обновлен"))
} else {
w.WriteHeader(http.StatusBadRequest)
return
}
}

```

```
}  
default:  
w.WriteHeader(http.StatusMethodNotAllowed)  
w.Write([]byte("Метод не поддерживается"))  
}  
}  
  
func main() {  
// Инициализация базы данных  
err := initDB()  
if err != nil {  
fmt.Println("Ошибка инициализации базы данных:", err)  
return  
}  
  
defer db.Close()  
  
http.HandleFunc("/count", countHandler)  
fmt.Println("Сервер запущен на порту: 3333")  
// Запуск сервера  
err = http.ListenAndServe(":3333", nil)  
if err != nil {  
  
fmt.Println("Ошибка запуска сервера:", err)  
}  
}
```

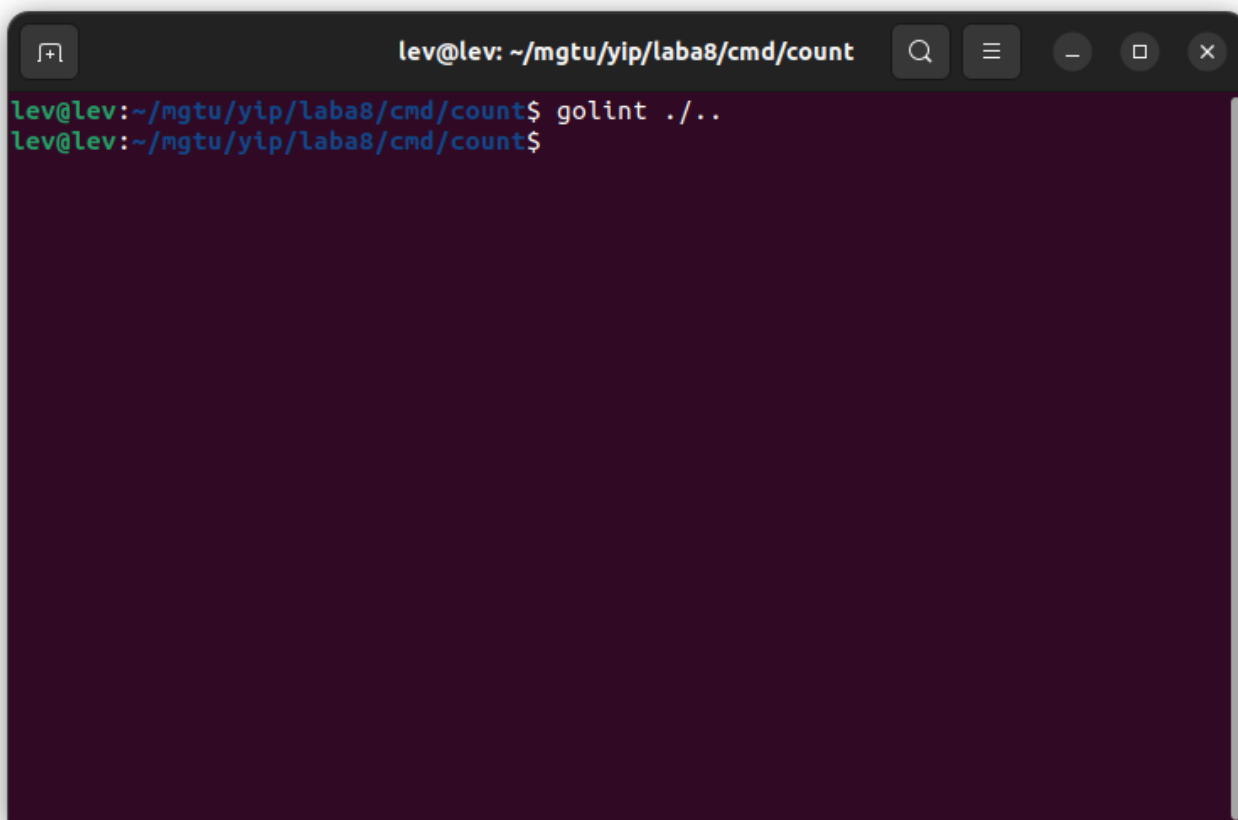


Рисунок 17 - Тестирование сервиса count линтами

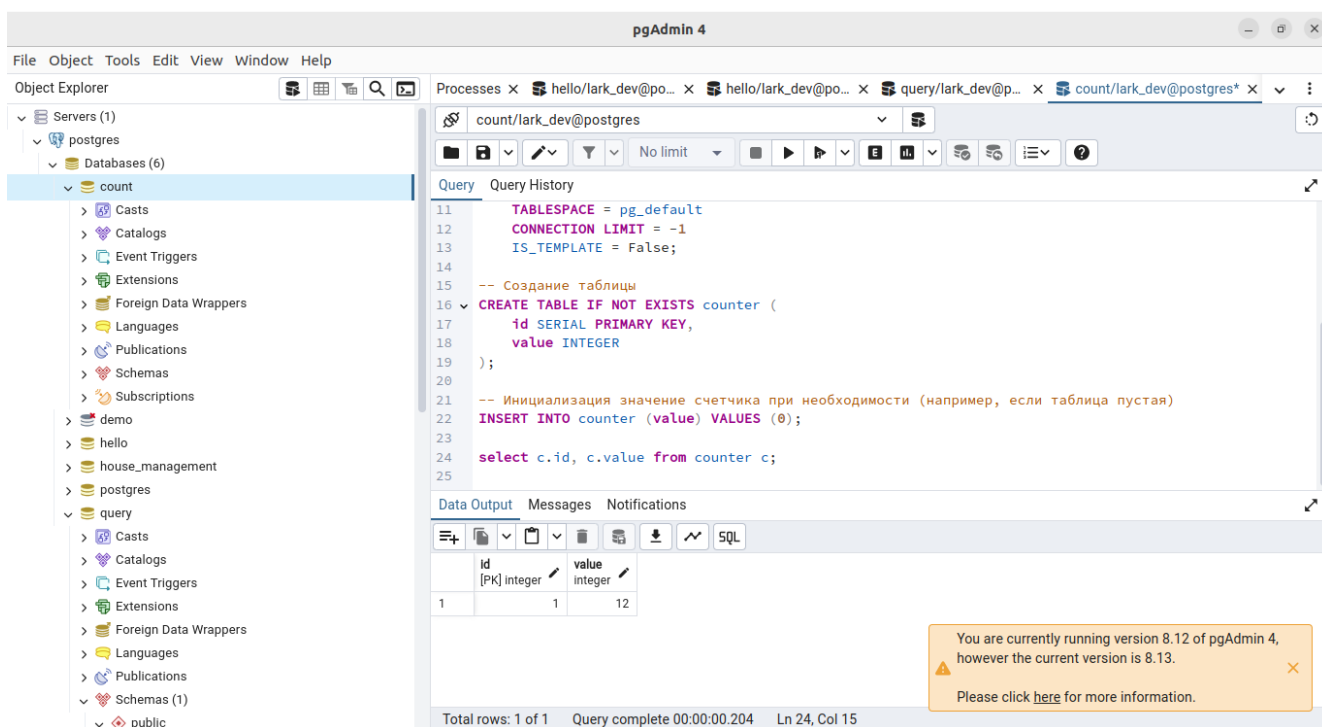


Рисунок 18 - Тестирование сервиса count

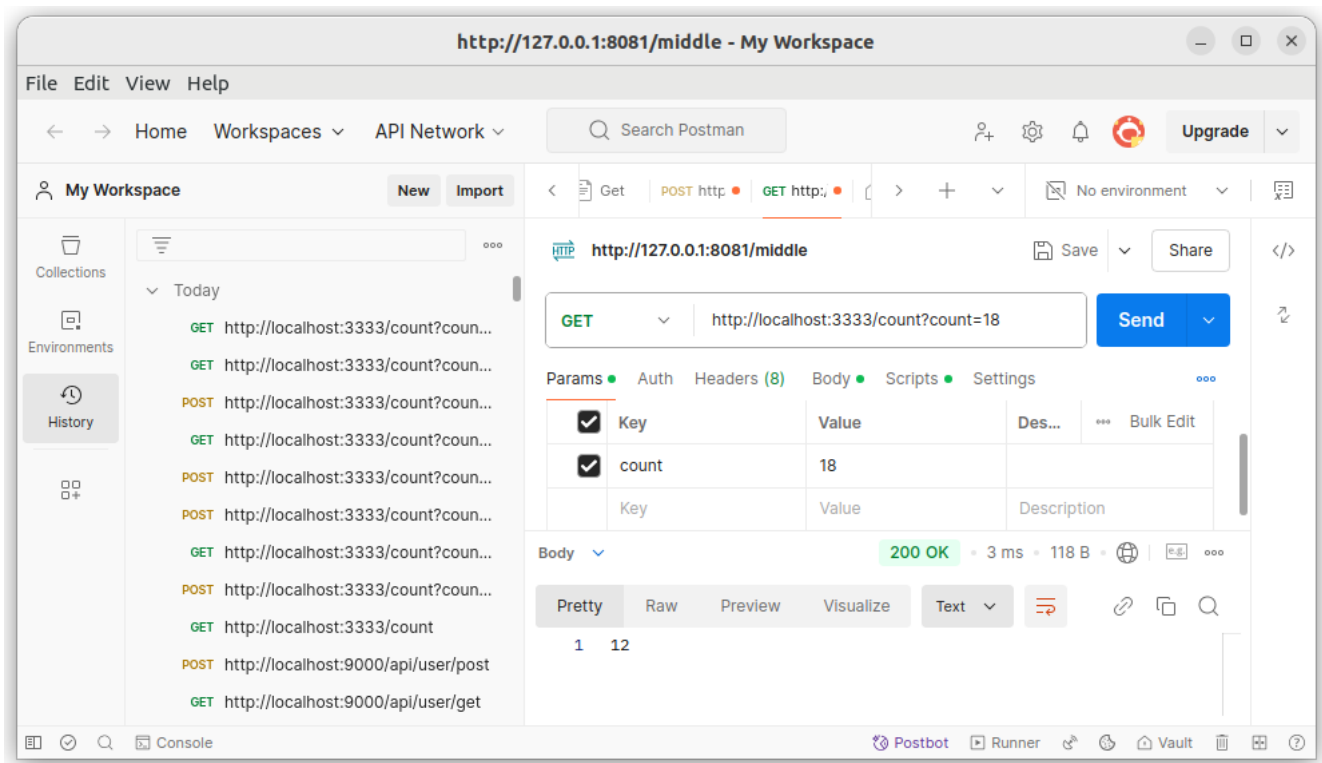


Рисунок 19 - Тестирование сервиса count

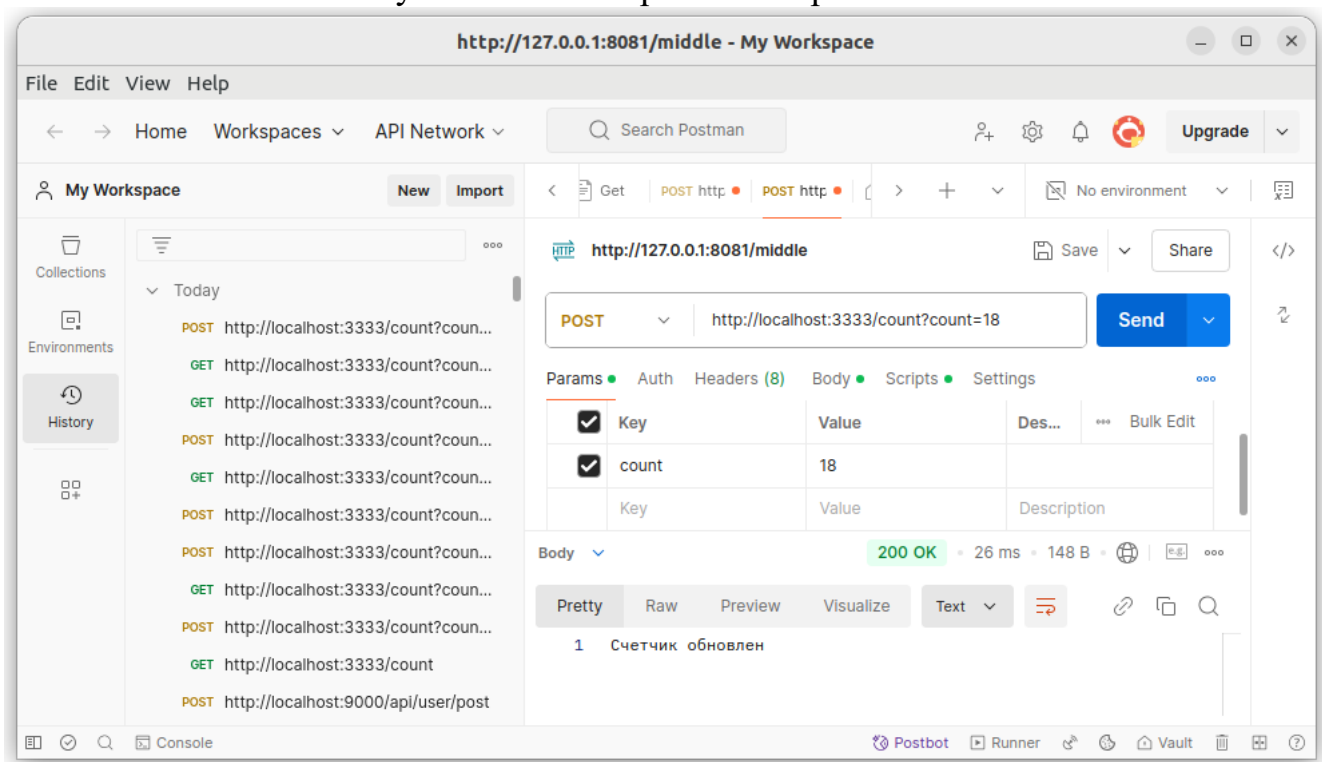


Рисунок 20 - Тестирование сервиса count

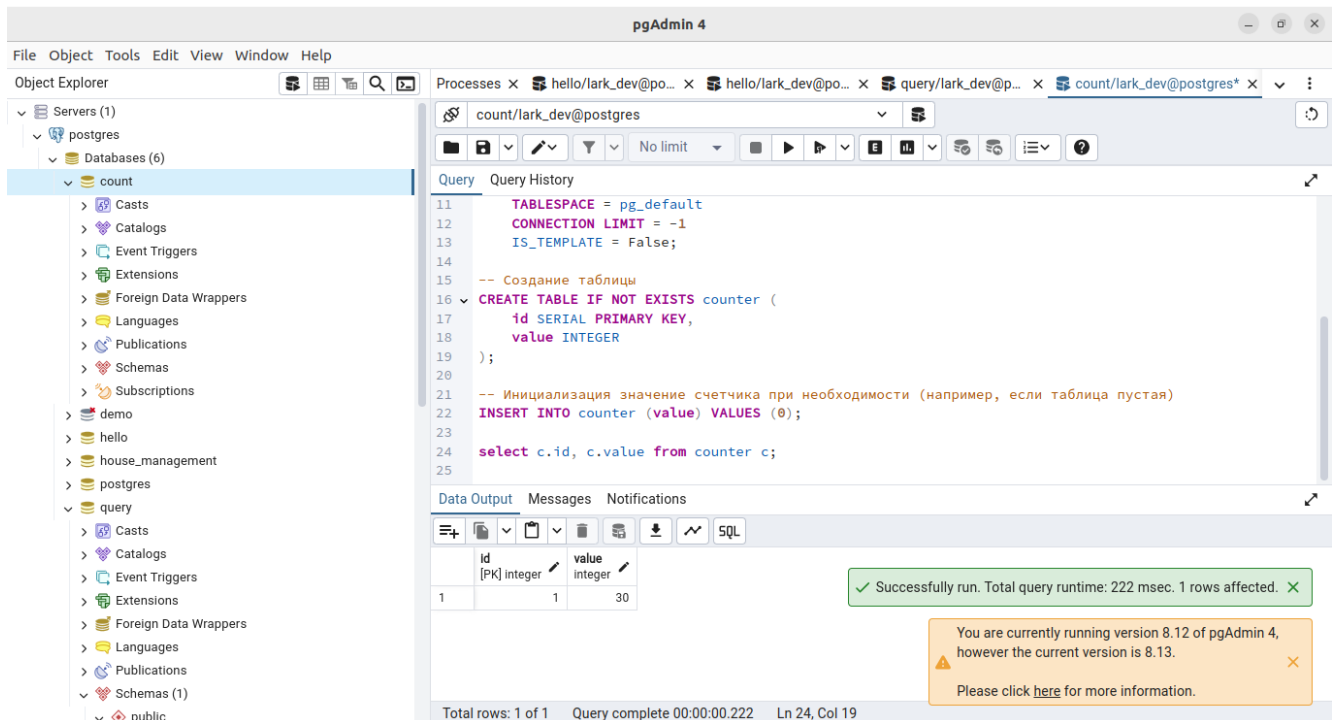


Рисунок 20 - Тестирование сервиса count

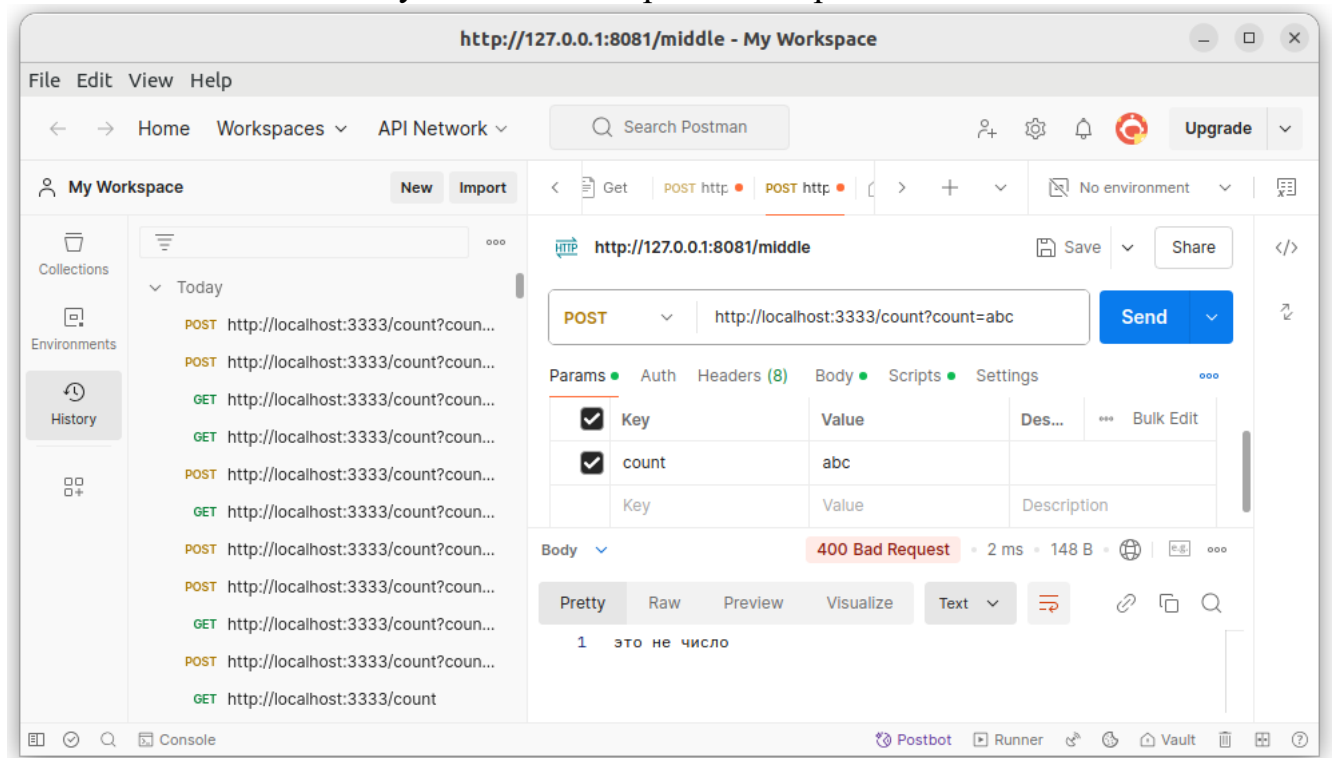


Рисунок 21 - Тестирование сервиса count

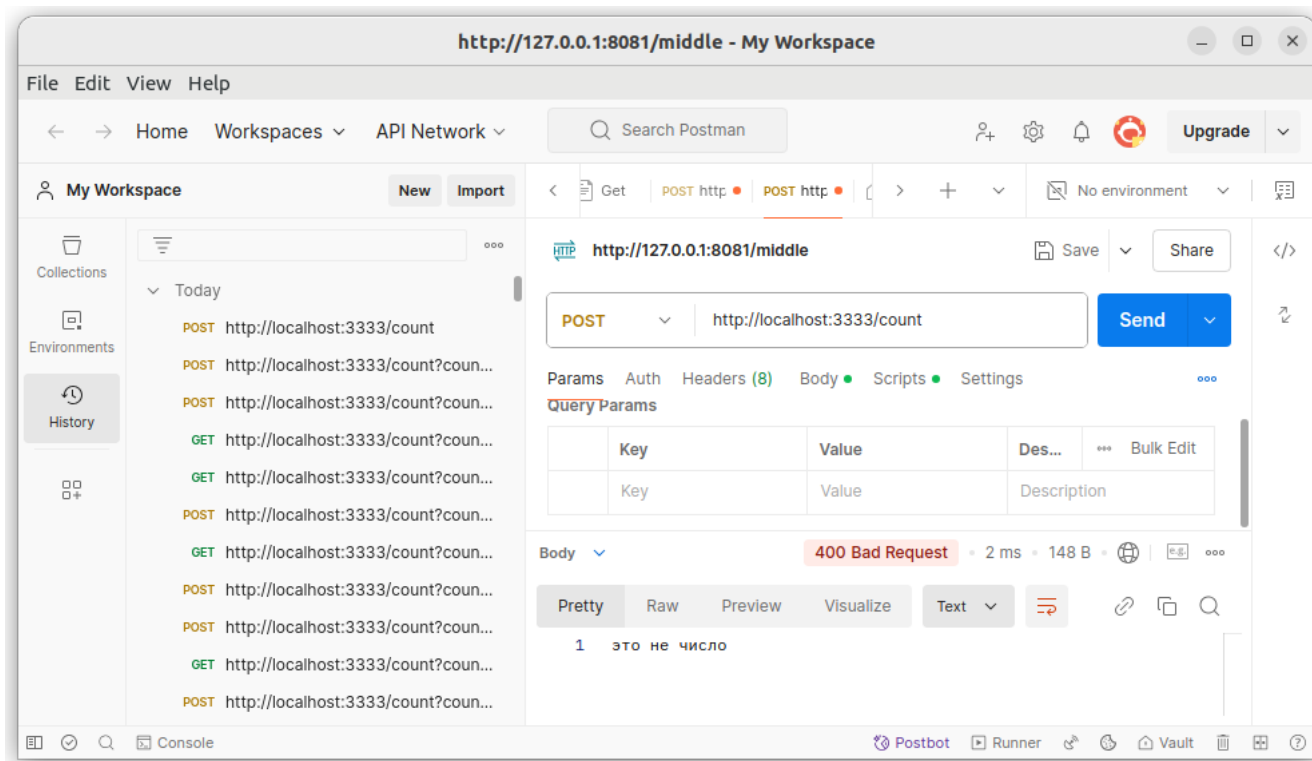


Рисунок 22 - Тестирование сервиса count

**Заключение:** были получены первичные навыки в организации долгосрочного хранения данных с использованием PostgreSQL и Golang.

### Список источников:

1. <https://golangdocs.com/golang-postgresql-example>