



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 09.04.01 ИУ6-32Б

О Т Ч Е Т

по лабораторной работе № 8

Название: Организация клиент-серверного
взаимодействия между Golang и PostgreSQL

Дисциплина: Языки интернет-программирования

Студент

ИУ6-32Б

Юдаков Г.Д.

(Группа)

(Подпись, дата)

(И.О. Фамилия)

Преподаватель

Шульман В.Д.

(Подпись, дата)

(И.О. Фамилия)

Цель работы: изучение основ разработки
SPAприложение на JavaScript.

Задание 1

Переделать сервисы, полученные в 6 лабораторной работе, так чтобы они использовали бд

Сервис Count

Для начала создадим все необходимые таблицы в бд

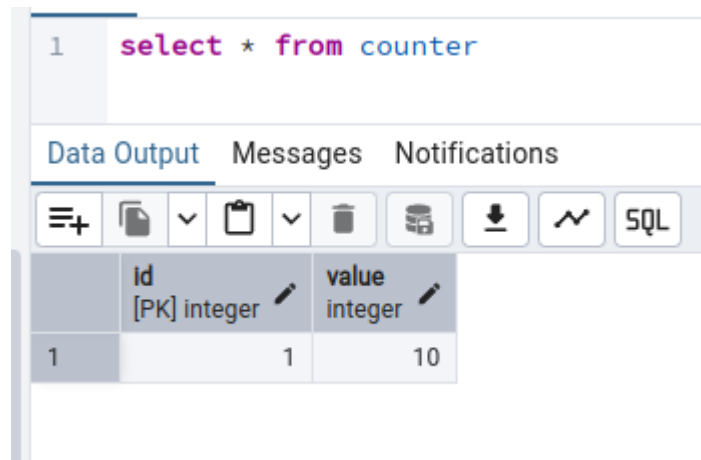


Рисунок 1

```
package main

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

    _ "github.com/lib/pq"
)

const (
    host = "localhost"
    port = 5432
    user = "postgres"
    password = "1234"
    dbname = "sandbox"
)

var (
    db *sql.DB
```

```

mu sync.Mutex
)

func initDB() {
var err error
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.Fatalf("Failed to connect to the database: %v", err)
}

// Проверяем соединение
if err := db.Ping(); err != nil {
log.Fatalf("Database is not reachable: %v", err)
}

// Создаём таблицу, если её нет
_, err = db.Exec(`
CREATE TABLE IF NOT EXISTS public.counter (
id SERIAL PRIMARY KEY,
value INTEGER NOT NULL DEFAULT 0
)
`)
if err != nil {
log.Fatalf("Failed to create table: %v", err)
}

// Инициализация счётчика, если таблица пуста
_, err = db.Exec(`INSERT INTO public.counter (value) VALUES (0) ON CONFLICT DO
NOTHING`)
if err != nil {
log.Fatalf("Failed to initialize counter: %v", err)
}

log.Println("Database initialized successfully")
}

func getCountHandler(w http.ResponseWriter, r *http.Request) {
mu.Lock()
defer mu.Unlock()

var counter int
err := db.QueryRow(`SELECT value FROM public.counter WHERE id = 1`).Scan(&counter)
if err != nil {
http.Error(w, "Failed to fetch counter value", http.StatusInternalServerError)
return
}

fmt.Fprintf(w, "Current count: %d", counter)
}

```

```

func postCountHandler(w http.ResponseWriter, r *http.Request) {
var data struct {
Count int `json:"count"`
}
err := json.NewDecoder(r.Body).Decode(&data)
if err != nil {
http.Error(w, "Invalid JSON format", http.StatusBadRequest)
return
}
defer r.Body.Close()

mu.Lock()
defer mu.Unlock()

// Увеличиваем значение счётчика в базе данных
_, err = db.Exec(`UPDATE public.counter SET value = value + $1 WHERE id = 1`, data.Count)
if err != nil {
http.Error(w, "Failed to update counter", http.StatusInternalServerError)
return
}

var newCounter int
err = db.QueryRow(`SELECT value FROM public.counter WHERE id = 1`).Scan(&newCounter)
if err != nil {
http.Error(w, "Failed to fetch updated counter value", http.StatusInternalServerError)
return
}

fmt.Fprintf(w, "Count incremented by %d, new count: %d", data.Count, newCounter)
}

func main() {
initDB()
defer db.Close()

http.HandleFunc("/count", func(w http.ResponseWriter, r *http.Request) {
switch r.Method {
case http.MethodGet:
getCountHandler(w, r)
case http.MethodPost:
postCountHandler(w, r)
default:
http.Error(w, "Method not allowed", http.StatusMethodNotAllowed)
}
})

fmt.Println("Server is running on http://localhost:3333")
if err := http.ListenAndServe(":3333", nil); err != nil {
log.Fatalf("Failed to start server: %v", err)
}
}

```

```

}

men-fish@Men-fish:~$ curl -X POST -H "Content-Type: application/json" -d '{"count": 4323}' http://localhost:3333/count
Count incremented by curl -X GET http://localhost:3333/count-$ curl -X GET http://localhost:3333/count
Current count: 4554men-fish@Men-fish:~$

```

Рисунок 2

На рисунке показан пример get и роые запроса для сервиса count.

	Id [PK] integer	value integer
1	1	4554

Рисунок 3

На рисунке 4 показано то, как это хранится в базе данных

Сервис Hello

DB:

	message character varying[] (255)
1	{Имя2}
2	{Имя1}

Рисунок 4

Здесь показано как возможные сообщения хранятся в бд.

На get запрос сервер отправляет приветствующее сообщение

```

men-fish@Men-fish:~$ curl -X POST -H "Content-Type: application/json" -d '{"msg": "Привет, мир!"}' http://127.0.0.1:8081/post
pq: malformed array literal: "Привет, мир!"men-fish@Men-fish:~$ curl -X GET http://127.0.0.1:8081/get
{"Имя1"}men-fish@Men-fish:~$ curl -X GET http://127.0.0.1:8081/get
{"Имя1"}men-fish@Men-fish:~$ curl -X GET http://127.0.0.1:8081/get
{"Имя1"}men-fish@Men-fish:~$ curl -X POST -H "Content-Type: application/json" -d '{"msg": "Привет, мир!"}' http://127.0.0.1:8081/post
pq: malformed array literal: "Привет, мир!"men-fish@Men-fish:~$ curl -X GET http://127.0.0.1:8081/get
{"Имя2"}men-fish@Men-fish:~$ curl -X GET http://127.0.0.1:8081/get
{"Имя2"}men-fish@Men-fish:~$ curl -X GET http://127.0.0.1:8081/get
{"Имя2"}men-fish@Men-fish:~$ curl -X POST -H "Content-Type: application/json" -d '{"msg": "Привет, мир!"}' http://127.0.0.1:8081/post
men-fish@Men-fish:~$

```

```

main.go - web-8 - Visual Studio Code
File Edit Selection View Go Run Terminal Help
cmd > hello > main.go 1, M X
41 func (h *Handlers) PostHello(w http.ResponseWriter, r *http.Request) {
42     input := struct {
43         Msg string `json:"msg"`
44     }{}
45
46     decoder := json.NewDecoder(r.Body)
47     err := decoder.Decode(&input)
48     if err != nil {
49         if err != nil {
50             w.WriteHeader(http.StatusBadRequest)
51             w.Write([]byte(err.Error()))
52         }
53     }
54
55     err = h.dbProvider.InsertHello(input.Msg)
56     if err != nil {
57         w.WriteHeader(http.StatusInternalServerError)
58         w.Write([]byte(err.Error()))
59     }
60     w.WriteHeader(http.StatusCreated)
61
62 }
63
64 // Методы для работы с базой данных
65 func (dp *DatabaseProvider) SelectHello() (string, error) {
66     var msg string
67
68     // Получаем одно сообщение из таблицы hello, отсортированной в случайном порядке
69     row := dp.db.QueryRow("SELECT message FROM hello ORDER BY RANDOM() LIMIT 1")
70     err := row.Scan(&msg)
71     if err != nil {
72         return "", err
73     }
74     return msg, nil
75 }
76
77 func (dp *DatabaseProvider) InsertHello(msg string) error {

```

Рисунок 5 – Тестирование программы

Сервис Query

В данном сервисе мы должны возвращать Hello <username> на get запросы пользователя если такой уже существует в бд.

Рисунок 7 (Пример бд)

Рисунок 8 (пример запроса GET)

Код:

```
package main

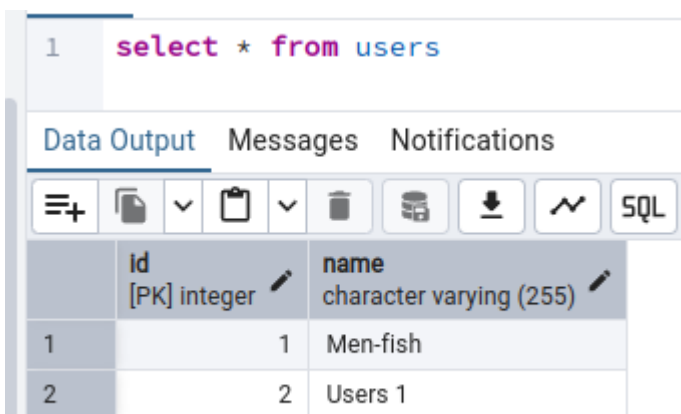
import (
    "database/sql"
    "fmt"
    "log"
    "net/http"

    _ "github.com/lib/pq"
)

const (
    host = "local-
port = 5432
user = "postgres"
password = "1234"
dbname = "sandbox"
)

var db *sql.DB

func initDB() {
```



The screenshot shows a database query tool interface. At the top, a SQL query is entered: `1 select * from users`. Below the query bar, there are tabs for "Data Output", "Messages", and "Notifications". Under the "Data Output" tab, there is a toolbar with icons for expand, save, refresh, copy, paste, delete, insert, and a "SQL" button. Below the toolbar is a table with two columns: "Id" and "name". The "Id" column has a sub-header "[PK] integer" and an edit icon. The "name" column has a sub-header "character varying (255)" and an edit icon. The table contains two rows of data: the first row has Id 1 and name "Men-fish"; the second row has Id 2 and name "Users 1".

	Id [PK] integer	name character varying (255)
1	1	Men-fish
2	2	Users 1

"github.com/lib/pq"

host"

```
men-fish@Men-fish:~$ curl "http://localhost:8083/api/user?name=Men-fish"
Hello, Men-fish!men-f$ curl "http://localhost:8083/api/user?name=Unknown"er?name=Unknown"
User 'Unknown' not found
men-fish@Men-fish:~$
```

```
var err error
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.Fatalf("Failed to connect to the database: %v", err)
}

// Проверяем соединение
if err := db.Ping(); err != nil {
log.Fatalf("Database is not reachable: %v", err)
}

// Создаём таблицу, если её нет
_, err = db.Exec(`
CREATE TABLE IF NOT EXISTS public.users (
id SERIAL PRIMARY KEY,
name VARCHAR(255) UNIQUE NOT NULL
)
`)
if err != nil {
log.Fatalf("Failed to create table: %v", err)
}

log.Println("Database initialized successfully")
}

func userHandler(w http.ResponseWriter, r *http.Request) {
name := r.URL.Query().Get("name")
if name == "" {
http.Error(w, "Parameter 'name' is required", http.StatusBadRequest)
return
}

var exists bool
err := db.QueryRow(`SELECT EXISTS(SELECT 1 FROM public.users WHERE name = $1)`,
name).Scan(&exists)
if err != nil {
http.Error(w, "Database error", http.StatusInternalServerError)
return
}

if !exists {
http.Error(w, fmt.Sprintf("User '%s' not found", name), http.StatusNotFound)
return
}
}
```

```
response := fmt.Sprintf("Hello, %s!", name)
w.Write([]byte(response))
}

func main() {
    initDB()
    defer db.Close()

    http.HandleFunc("/api/user", userHandler)

    fmt.Println("Server is running on http://localhost:8083")
    if err := http.ListenAndServe(":8083", nil); err != nil {
        log.Fatalf("Failed to start server: %v", err)
    }
}
```

Заключение: Я научился создавать веб-сервер и обрабатывать GET и POST запросы, при этом получая параметры различными способами и хранить данные в БД.