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		О.И.Ельничных
	(Группа)	(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.

Ход работы

Делаем fork репозитория (Рисунок 1).

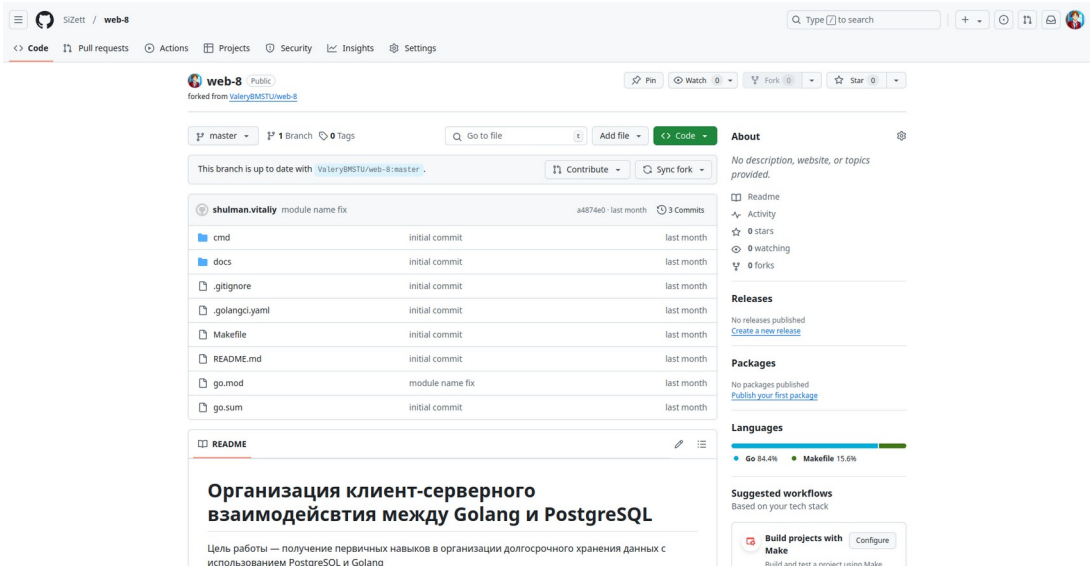

Рисунок 1

Код задания hello:

```
package main

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

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

```
)

const (
host = "localhost"
port = 5432
user = "username"
password = "password"
dbname = "lab8"
)

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 name_hello 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 (name_hello) VALUES ($1)", msg)
if err != nil {
return err
}

return nil
}

func main() {
// Считываем аргументы командной строки
address := flag.String("address", "127.0.0.1:8085", "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)
```

```
}
}
```

Код задания query:

```
package main

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

_ "github.com/lib/pq"
)

const (
host = "localhost"
port = 5432
user = "username"
password = "password"
dbname = "lab8"
)

type Handlers struct {
dbProvider DatabaseProvider
}

type DatabaseProvider struct {
db *sql.DB
}

// Обработчики HTTP-запросов
func (h *Handlers) GetQuery(w http.ResponseWriter, r *http.Request) {
name := r.URL.Query().Get("name")

if name == "" {
w.WriteHeader(http.StatusBadRequest)
w.Write([]byte("Не введен параметр!"))
return
}

test, err := h.dbProvider.SelectQuery(name)
if !test {
w.WriteHeader(http.StatusBadRequest)
w.Write([]byte("Запись не добавлена в БД!"))
return
}
if err != nil {
w.WriteHeader(http.StatusInternalServerError)
w.Write([]byte(err.Error()))
}
```

```
w.WriteHeader(http.StatusOK)
w.Write([]byte("Hello," + name + "!"))
}

func (h *Handlers) PostQuery(w http.ResponseWriter, r *http.Request) {
name := r.URL.Query().Get("name")
if name == "" {
w.WriteHeader(http.StatusBadRequest)
w.Write([]byte("Не введен параметр!"))
return
}

test, err := h.dbProvider.SelectQuery(name)
if test && err == nil {
w.WriteHeader(http.StatusBadRequest)
w.Write([]byte("Запись уже добавлена БД!"))
return
}

err = h.dbProvider.InsertQuery(name)
if err != nil {
w.WriteHeader(http.StatusInternalServerError)
w.Write([]byte(err.Error()))
}

w.WriteHeader(http.StatusCreated)
w.Write([]byte("Добавили запись!"))
}

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

row := dp.db.QueryRow("SELECT name_query FROM query WHERE name_query = ($1)", msg)
err := row.Scan(&rec)
if err != nil {
return false, err
}

return true, nil
}

func (dp *DatabaseProvider) InsertQuery(msg string) error {
_, err := dp.db.Exec("INSERT INTO query (name_query) VALUES ($1)", msg)
if err != nil {
return err
}

return nil
}
```

```
func main() {
// Считываем аргументы командной строки
address := flag.String("address", "127.0.0.1:8086", "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.GetQuery)
http.HandleFunc("/post", h.PostQuery)

// Запускаем веб-сервер на указанном адресе
err = http.ListenAndServe(*address, nil)
if err != nil {
log.Fatal(err)
}
}
```

Код задания count:

```
package main

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

_ "github.com/lib/pq"
)

const (
host = "localhost"
port = 5432
user = "username"
```

```
password = "password"
dbname = "lab8"
)

type Handlers struct {
dbProvider DatabaseProvider
}

type DatabaseProvider struct {
db *sql.DB
}

// обработчики http-запросов
func (h *Handlers) GetCount(w http.ResponseWriter, r *http.Request) {
msg, err := h.dbProvider.SelectCount()
if err != nil {
w.WriteHeader(http.StatusInternalServerError)
w.Write([]byte(err.Error()))
}

w.WriteHeader(http.StatusOK)
w.Write([]byte("count: " + strconv.Itoa(msg)))
}

func (h *Handlers) PostCount(w http.ResponseWriter, r *http.Request) {
input := struct {
Msg int `json:"msg"`
}{}

decoder := json.NewDecoder(r.Body)
err := decoder.Decode(&input)
if err != nil {
w.WriteHeader(http.StatusBadRequest)
w.Write([]byte(err.Error()))
}

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

w.WriteHeader(http.StatusCreated)
w.Write([]byte("count changed"))
}

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

row := dp.db.QueryRow("SELECT number FROM count WHERE id_number = 1")
err := row.Scan(&msg)
```

```
if err != nil {
return -1, err
}

return msg, nil
}

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

return nil
}

func main() {
// Считываем аргументы командной строки
address := flag.String("address", "127.0.0.1:8084", "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.GetCount)
http.HandleFunc("/post", h.PostCount)

// Запускаем веб-сервер на указанном адресе
err = http.ListenAndServe(*address, nil)
if err != nil {
log.Fatal(err)
}
}
```

Бд с соответствующими таблицами:

REATE TABLE IF NOT EXISTS public.count

```
(
    id_number serial NOT NULL,
    "number" integer,
    CONSTRAINT count_pkey PRIMARY KEY (id_number)
);

CREATE TABLE IF NOT EXISTS public.hello
(
    id_hello serial NOT NULL,
    name_hello character varying(255) COLLATE pg_catalog."default",
    CONSTRAINT hello_pkey PRIMARY KEY (id_hello)
);

CREATE TABLE IF NOT EXISTS public.query
(
    id_query serial NOT NULL,
    name_query character varying(255) COLLATE pg_catalog."default",
    CONSTRAINT query_pkey PRIMARY KEY (id_query)
);
```

Заключение

При выполнении заданий лабораторной работы №8 мы получили первичные навыки в организации долгосрочного хранения данных с использованием PostgreSQL и Golang и выполнили задание основанное на результатах лабораторной работы №6.