



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 Компьютерные системы и сети

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

(Группа)

(Подпись, дата)

К.Д. Коротаев

(И.О. Фамилия)

Преподаватель

(Подпись, дата)

И.О. Фамилия

(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, клонировать 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 dev в личный форк данного репозитория в GitHub
 9. Через интерфейс GitHub создать Pull Request dev --> master
- На защите лабораторной работы продемонстрировать открытый Pull Request. PR должен быть направлен в master ветку форка, а не исходного репозитория

Ход работы:

Код для работы с сервисом Hello:

```
package main
```

```
import (  
    "database/sql"  
    "encoding/json"  
    "flag"  
    "fmt"  
    "log"  
    "net/http"  
  
    _ "github.com/lib/pq"  
)
```

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

```
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 "+
    "dbname=%s sslmode=disable",
    host, port, user, 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 для работы с сервисом Count:

```

package main

// некоторые импорты нужны для проверки
import (
    "database/sql"
    "encoding/json"
    "flag"
    "fmt"
    "log"
    "net/http"
    "strconv"

    _ "github.com/lib/pq"
)

```

```

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

type Handlers struct {
    dbProvider DatabaseProvider
}

type DatabaseProvider struct {
    db *sql.DB
}

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

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

func (h *Handlers) PostCounter(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.UpdateCounter(input.Msg)
    if err != nil {
        w.WriteHeader(http.StatusInternalServerError)
        w.Write([]byte(err.Error()))
    }

    w.WriteHeader(http.StatusCreated)
    w.Write([]byte("Изменили счетчик!"))
}

```

```

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

    row := dp.db.QueryRow("SELECT number FROM counter WHERE id_number
= 1")
    err := row.Scan(&msg)
    if err != nil {
        return -1, err
    }

    return msg, nil
}

func (dp *DatabaseProvider) UpdateCounter(msg int) error {
    _, err := dp.db.Exec("UPDATE counter 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", "127.0.0.1:8082", "адрес для запуска сервера")
    flag.Parse()

    // Формирование строки подключения для postgres
    psqlInfo := fmt.Sprintf("host=%s port=%d user=%s "+
        "dbname=%s sslmode=disable",
        host, port, user, 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.GetCounter)
    http.HandleFunc("/post", h.PostCounter)

```

```

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

```

Код для работы с сервисом Query:

```

package main
import (
    "database/sql"
    "flag"
    "fmt"
    "log"
    "net/http"
    _ "github.com/lib/pq"
)

const (
    host  = "localhost"
    port  = 5432
    user  = "postgres"
    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) {
    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 record FROM query WHERE record = ($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 (record) VALUES ($1)", msg)
    if err != nil {
        return err
    }

    return nil
}

func main() {
    // Считываем аргументы командной строки
    address := flag.String("address", "127.0.0.1:8083", "адрес для запуска сервера")
    flag.Parse()

    // Формирование строки подключения для postgres
    psqlInfo := fmt.Sprintf("host=%s port=%d user=%s "+
        "dbname=%s sslmode=disable",
        host, port, user, 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)
    }
}

```

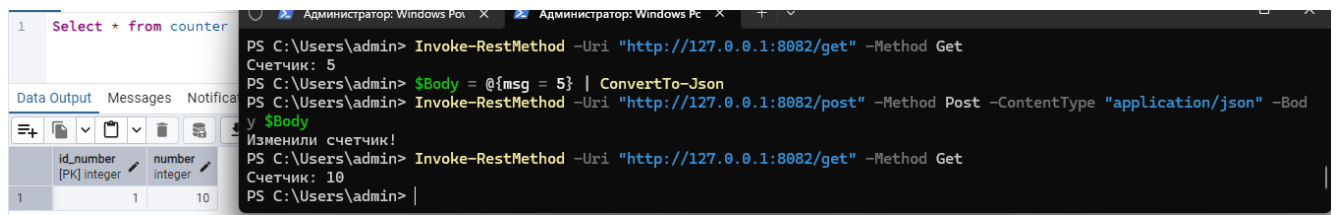
Тестирование к сервису Hello:

| | id [PK] integer | message text | |
|---|--------------------|------------------------|---|
| 1 | 1 | Hello from PowerShell! | PS C:\Users\admin> \$message = @{"msg" = "Hello from PowerShell!"} |
| 2 | 2 | Hello from PowerShell! | >> } |
| 3 | 3 | Hello from PowerShell! | PS C:\Users\admin> Invoke-RestMethod -Uri "http://127.0.0.1:8081/post" -Method Post -Body (\$message ConvertTo-Json) -Content-Type "application/json" |
| 4 | 4 | Hello from PowerShell! | PS C:\Users\admin> Invoke-RestMethod -Uri "http://127.0.0.1:8081/get" -Method Get |
| 5 | 5 | Hello from PowerShell! | Hello from PowerShell! |

Тестирование к сервису Query:

| | | |
|---|------------------------------------|--|
| 1 | Select * from query | Hello from PowerShell! |
| | | PS C:\Users\admin> Invoke-WebRequest -Uri "http://127.0.0.1:8083/post?name=Kostya" -Method Post |
| | Data Output Messages Notifications | StatusCode : 201 |
| | | StatusDescription : Created |
| | | Content : Добавили запись! |
| | | RawContent : HTTP/1.1 201 Created |
| | | Content-Length: 30 |
| | | Content-Type: text/plain; charset=utf-8 |
| | | Date: Sun, 08 Dec 2024 11:57:56 GMT |
| | | Добавили запись! |
| | | Forms : {} |
| | | Headers : {[Content-Length, 30], [Content-Type, text/plain; charset=utf-8], [Date, Sun, 08 Dec 2024 11:57:56 GMT]} |
| | | Images : {} |
| | | InputFields : {} |
| | | Links : {} |
| | | ParsedHtml : System.__ComObject |
| | | RawContentLength : 30 |
| | | PS C:\Users\admin> Invoke-WebRequest -Uri "http://127.0.0.1:8083/get?name=Kostya" -Method Get |
| | | StatusCode : 200 |
| | | StatusDescription : OK |
| | | Content : Hello, Kostya! |
| | | RawContent : HTTP/1.1 200 OK |
| | | Content-Length: 13 |
| | | Content-Type: text/plain; charset=utf-8 |
| | | Date: Sun, 08 Dec 2024 11:58:30 GMT |
| | | Hello, Kostya! |
| | | Forms : {} |
| | | Headers : {[Content-Length, 13], [Content-Type, text/plain; charset=utf-8], [Date, Sun, 08 Dec 2024 11:58:30 GMT]} |

Тестирование к сервису Count:



The screenshot displays a REST client interface on the left and a PowerShell terminal on the right. The REST client shows a SQL query `Select * from counter` and a table with two columns: `id_number` (integer, primary key) and `number` (integer). The table contains one row with values 1 and 10. The PowerShell terminal shows a series of REST API calls to a service at `http://127.0.0.1:8082`. The first call is a GET request to `/get`, which returns a count of 5. The second call is a POST request to `/post` with a JSON body `{ "msg": 5 }`, which returns a count of 10. The third call is a GET request to `/get`, which returns a count of 10.

| id_number | number |
|-----------|--------|
| 1 | 10 |

```
PS C:\Users\admin> Invoke-RestMethod -Uri "http://127.0.0.1:8082/get" -Method Get
Счетчик: 5
PS C:\Users\admin> $Body = @{"msg" = 5} | ConvertTo-Json
PS C:\Users\admin> Invoke-RestMethod -Uri "http://127.0.0.1:8082/post" -Method Post -ContentType "application/json" -Body $Body
Изменили счетчик!
PS C:\Users\admin> Invoke-RestMethod -Uri "http://127.0.0.1:8082/get" -Method Get
Счетчик: 10
PS C:\Users\admin>
```

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