



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 № 10

Название: Архитектура микросервисов на Golang

Дисциплина: Языки интернет-программирования

Студент	<u>ИУ6-32Б</u> (Группа)	<u>02.12.2024</u> (Подпись, дата)	<u>Т.А. Гаджиев</u> (И.О. Фамилия)
Преподаватель		<u>02.12.2024</u> (Подпись, дата)	<u>В.Д. Шульман</u> (И.О. Фамилия)

Москва, 2024

Цель работы — получение первичных навыков организации кодовой базы проекта на Golang.

Ход работы.

1. Сделали форк данного репозитория в GitHub, клонировали получившуюся копию локально, создали от мастера ветку дев и переключились на нее. Скопировали наши микросервисы из 8-ой лабораторной работы:

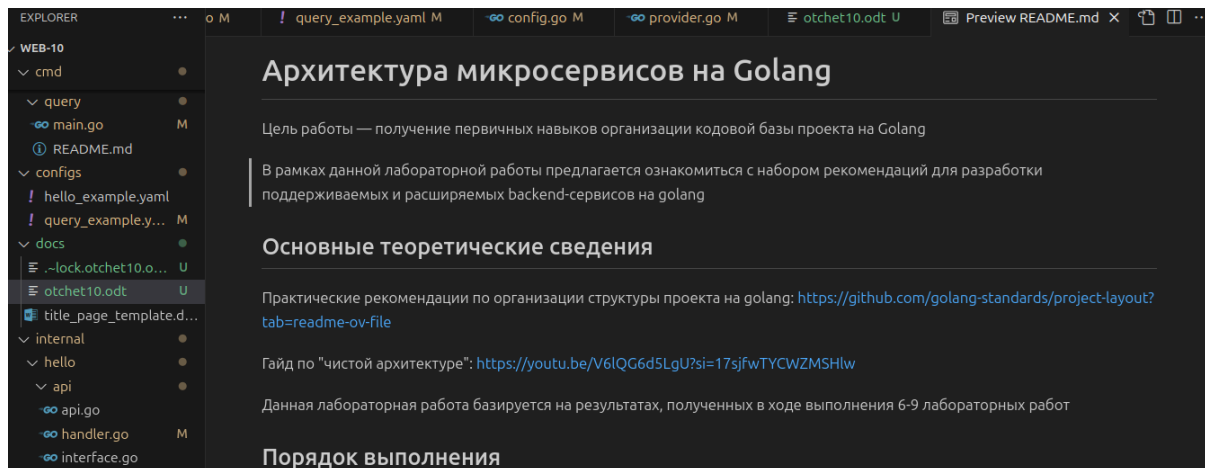


Рисунок 1 — Скопированный репозиторий

2. Модернизировали наши микросервисы, созданные в лабораторной работе №9, следуя гайду по «чистой архитектуре»:

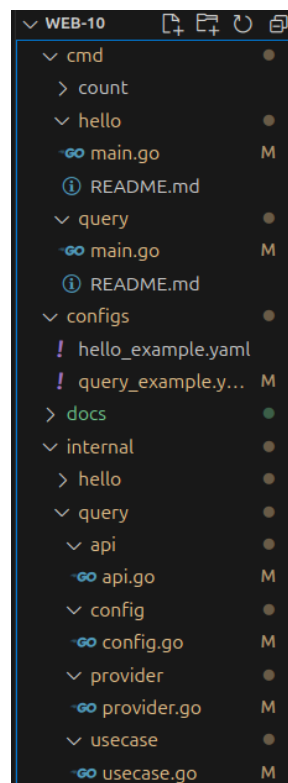


Рисунок 2 — Архитектура сервисов

Задача 1(Вывод строки приветствия):

Дана изначально в качестве примера, изменили лишь конфигурацию для подключения к нашей базе данных.

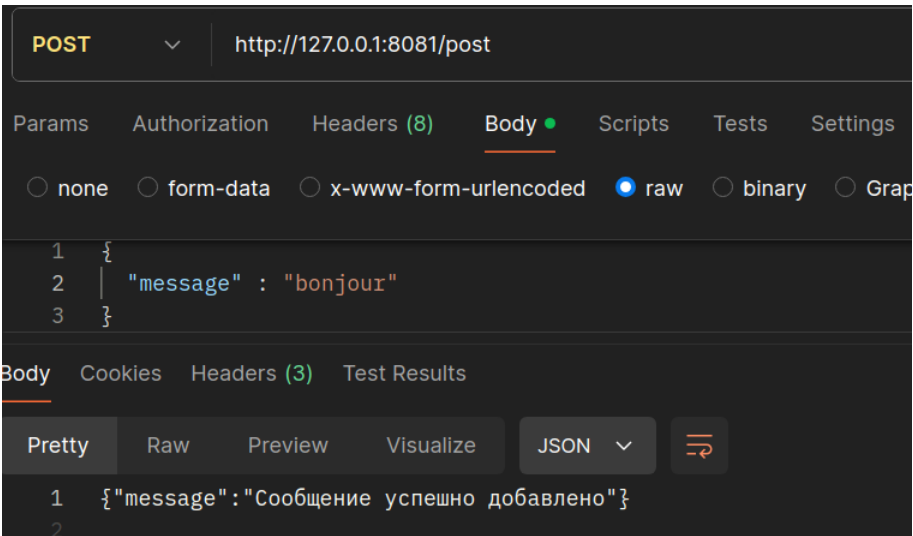


Рисунок 2 — Post запрос в Postman

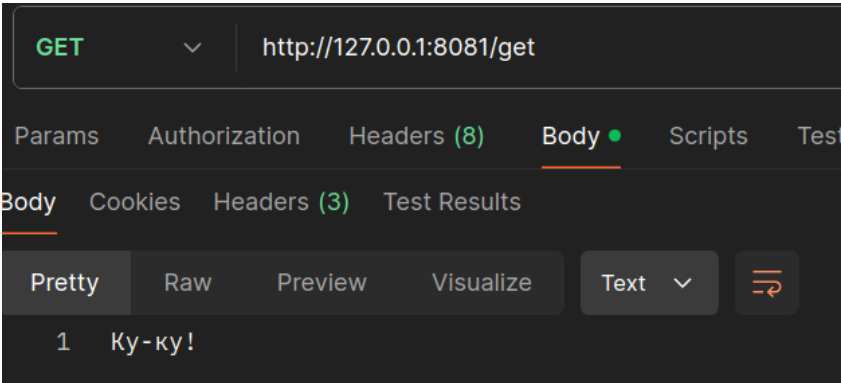


Рисунок 3- Get запрос в postman

Data Output		Messa
		message text
1	Привет, мир!	
2	hello world!	
3	Gutten Acht!	
4	Ку-ку!	
5	bonjour	

Рисунок 4 — Содержимое таблицы hello

Задача 2(Вывод строки приветствия с ключом)

main.go

```
package main
```

```
import (  
    "flag"  
    "log"  
    "net/http"  
    "web-10/internal/query/api"  
    "web-10/internal/query/config"  
    "web-10/internal/query/provider"  
    "web-10/internal/query/usecase"  
  
    _ "github.com/lib/pq"  
)  
  
func main() {  
    configPath := flag.String("config-path", "../configs/query_example.yaml",  
        "путь к файлу конфигурации")  
    flag.Parse()  
  
    cfg, err := config.LoadConfig(*configPath)  
    if err != nil {  
        log.Fatal(err)  
    }  
  
    prv := provider.NewProvider(cfg.DB.Host, cfg.DB.Port, cfg.DB.User,  
        cfg.DB.Password, cfg.DB.DBname)  
    use := usecase.NewUsecase(prv)  
    srv := api.NewServer(cfg.IP, cfg.Port, use)  
  
    log.Printf("Сервер запущен на %s\n", srv.Address)  
    log.Fatal(http.ListenAndServe(srv.Address, srv.Router))  
}
```

api.go

```
package api
```

```
import (  
    "fmt"  
    "net/http"  
  
    "github.com/labstack/echo/v4"  
    "web-10/internal/query/usecase"  
)
```

```

type Server struct {
    Address string
    Router *echo.Echo
    uc *usecase.Usecase
}

func NewServer(ip string, port int, uc *usecase.Usecase) *Server {
    e := echo.New()
    srv := &Server{
        Address: fmt.Sprintf("%s:%d", ip, port),
        Router: e,
        uc: uc,
    }

    srv.Router.GET("/api/user", srv.GetUser)
    srv.Router.POST("/api/user/create", srv.PostUser)

    return srv
}

func (srv *Server) GetUser(c echo.Context) error {
    name := c.QueryParam("name")
    if name == "" {
        return c.JSON(http.StatusBadRequest, map[string]string{"error": "Name
parameter is required"})
    }

    user, err := srv.uc.GetUser(name)
    if err != nil {
        return c.JSON(http.StatusInternalServerError, map[string]string{"error":
err.Error()})
    }

    return c.String(http.StatusOK, "Hello, "+user+"!")
}

func (srv *Server) PostUser(c echo.Context) error {
    var input struct {
        Name string `json:"name"`
    }

    if err := c.Bind(&input); err != nil {
        return c.JSON(http.StatusBadRequest, map[string]string{"error": err.Error()})
    }

```

```
err := srv.uc.CreateUser(input.Name)
if err != nil {
return c.JSON(http.StatusInternalServerError, map[string]string{"error":
err.Error()})
}
```

```
return c.JSON(http.StatusCreated, map[string]string{"message": "Запись
добавлена!"})
}
```

```
usecase.go
package usecase
```

```
import "web-10/internal/query/provider"
```

```
type Usecase struct {
p *provider.Provider
}
```

```
func NewUsecase(p *provider.Provider) *Usecase {
return &Usecase{p: p}
}
```

```
func (u *Usecase) GetUser(name string) (string, error) {
return u.p.SelectUser(name)
}
```

```
func (u *Usecase) CreateUser(name string) error {
return u.p.InsertUser(name)
}
```

```
config.go
package config
```

```
import (
"gopkg.in/yaml.v3"
"io/ioutil"
"path/filepath"
)
```

```
type Config struct {
IP string `yaml:"ip"`
Port int `yaml:"port"`
}
```

```
DB db `yaml:"db"`  
}
```

```
type db struct {  
Host string `yaml:"host"`  
Port int `yaml:"port"`  
User string `yaml:"user"`  
Password string `yaml:"password"`  
DBname string `yaml:"dbname"`  
}
```

```
func LoadConfig(pathToFile string) (*Config, error) {  
filename, err := filepath.Abs(pathToFile)  
if err != nil {  
return nil, err  
}
```

```
yamlFile, err := ioutil.ReadFile(filename)  
if err != nil {  
return nil, err  
}
```

```
var cfg Config
```

```
err = yaml.Unmarshal(yamlFile, &cfg)  
if err != nil {  
return nil, err  
}
```

```
return &cfg, nil  
}
```

```
provider.go  
package provider
```

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

```
type Provider struct {  
conn *sql.DB  
}
```

```

func NewProvider(host string, port int, user, password, dbName string)
*Provider {
    psqlInfo := fmt.Sprintf("host=%s port=%d user=%s password=%s
    dbname=%s sslmode=disable",
    host, port, user, password, dbName)

    conn, err := sql.Open("postgres", psqlInfo)
    if err != nil {
        log.Fatal(err)
    }

    return &Provider{conn: conn}
}

func (p *Provider) SelectUser(name string) (string, error) {
    var user string
    row := p.conn.QueryRow("SELECT name FROM mytable WHERE name =
    $1", name)
    err := row.Scan(&user)
    if err != nil {
        if err == sql.ErrNoRows {
            return "", nil
        }
        return "", err
    }
    return user, nil
}

func (p *Provider) InsertUser(name string) error {
    _, err := p.conn.Exec("INSERT INTO mytable (name) VALUES ($1)", name)
    if err != nil {
        return err
    }
    return nil
}

func (p *Provider) Close() error {
    return p.conn.Close()
}
}

```

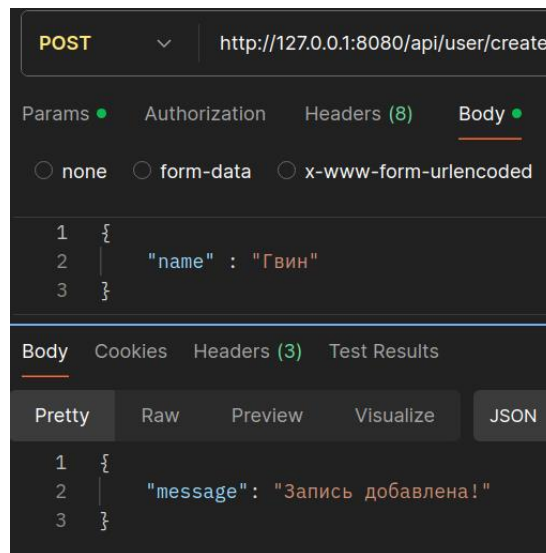



Рисунок 5 — Post запрос в postman

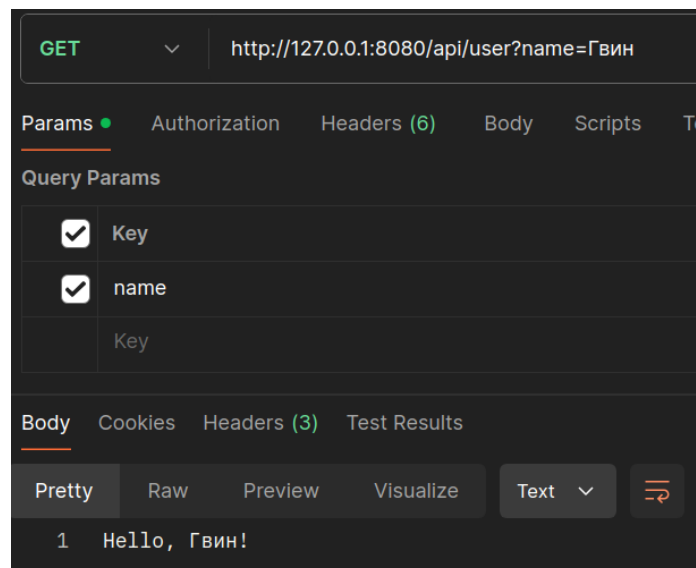


Рисунок 6 — Get запрос в postman

	Id [PK] integer	name character varying (255)
1	1	John Doe
2	2	КрутоеИмя
3	3	ЛучшееИмя
4	4	CoolName
5	5	Аноним
6	6	Добромысл
7	7	Любава
8	8	Гвин

Рисунок 7 — Содержимое таблицы имён

Задача 3(Count):

```
main.go
package main

import (
    "flag"
    "log"
    "web-10/internal/count/api"
    "web-10/internal/count/config"
    "web-10/internal/count/provider"
    "web-10/internal/count/usecase"

    _ "github.com/lib/pq"
)

func main() {
    configPath := flag.String("config-path", "../configs/count_example.yaml",
        "путь к файлу конфигурации")
    flag.Parse()

    cfg, err := config.LoadConfig(*configPath)
    if err != nil {
        log.Fatal(err)
    }

    prv := provider.NewProvider(cfg.DB.Host, cfg.DB.Port, cfg.DB.User,
        cfg.DB.Password, cfg.DB.DBname)
    use := usecase.NewUsecase(prv)
    srv := api.NewServer(cfg.IP, cfg.Port, use)

    log.Printf("Сервер запущен на %s\n", srv.Address)
    srv.Run()
}

api.go
package api

import (
    "fmt"
    "web-10/internal/count/usecase"

    "github.com/labstack/echo/v4"
)
```

```

type Server struct {
    Address string
    Router *echo.Echo
    Usecase *usecase.Usecase
}

func NewServer(ip string, port int, use *usecase.Usecase) *Server {
    s := &Server{
        Address: fmt.Sprintf("%s:%d", ip, port),
        Router: echo.New(),
        Usecase: use,
    }

    s.Router.GET("/count", s.HandleCount)
    s.Router.POST("/count", s.HandleCount)

    return s
}

func (s *Server) HandleCount(c echo.Context) error {
    return s.Usecase.HandleCount(c)
}

func (s *Server) Run() {
    s.Router.Logger.Fatal(s.Router.Start(s.Address))
}

```

config.go

```
package config
```

```
import (
    "io/ioutil"
    "path/filepath"

```

```

    "gopkg.in/yaml.v3"
)

```

```

type Config struct {
    IP string `yaml:"ip"`
    Port int `yaml:"port"`
    DB DBConfig `yaml:"db"`
}

```

```

type DBConfig struct {
    Host string `yaml:"host"`
    Port int `yaml:"port"`
    User string `yaml:"user"`
    Password string `yaml:"password"`
    DBname string `yaml:"dbname"`
}

```

```
func LoadConfig(pathToFile string) (*Config, error) {
    filename, err := filepath.Abs(pathToFile)
    if err != nil {
        return nil, err
    }
```

```
    yamlFile, err := ioutil.ReadFile(filename)
    if err != nil {
        return nil, err
    }
```

```
    var cfg Config
```

```
    err = yaml.Unmarshal(yamlFile, &cfg)
    if err != nil {
        return nil, err
    }
```

```
    return &cfg, nil
}
```

```
usecase.go
package usecase
```

```
import (
    "fmt"
    "net/http"
    "web-10/internal/count/provider"

    "github.com/labstack/echo/v4"
)
```

```
type Usecase struct {
    provider *provider.Provider
}
```

```
func NewUsecase(prv *provider.Provider) *Usecase {
    return &Usecase{provider: prv}
}
```

```
func (u *Usecase) HandleCount(c echo.Context) error {
    switch c.Request().Method {
    case http.MethodGet:
        counter, err := u.provider.GetCounter()
        if err != nil {
            return c.JSON(http.StatusInternalServerError, map[string]string{"error": err.Error()})
        }
```

```
        err = u.provider.UpdateCounter(1)
        if err != nil {
```

```

return c.JSON(http.StatusInternalServerError, map[string]string{"error": err.Error()})
}

return c.String(http.StatusOK, fmt.Sprintf("%d", counter+1))

case http.MethodPost:
var requestBody struct {
Count int `json:"count"`
}

if err := c.Bind(&requestBody); err != nil {
return c.JSON(http.StatusBadRequest, map[string]string{"error": "это не число"})
}

err := u.provider.UpdateCounter(requestBody.Count)
if err != nil {
return c.JSON(http.StatusInternalServerError, map[string]string{"error": err.Error()})
}
return c.JSON(http.StatusOK, map[string]string{"message": "Success"})

default:
return c.JSON(http.StatusMethodNotAllowed, map[string]string{"error": "Неизвестный метод"})
}
}

```

provider.go
package provider

```

import (
"database/sql"
"fmt"
"log"
)

type Provider struct {
db *sql.DB
}

func NewProvider(host string, port int, user, password, dbName string) *Provider {
psqlInfo := fmt.Sprintf("host=%s port=%d user=%s password=%s dbname=%s\nsslmode=disable",
host, port, user, password, dbName)

conn, err := sql.Open("postgres", psqlInfo)
if err != nil {
log.Fatal(err)
}

return &Provider{db: conn}
}

```

```

}

func (dp *Provider) GetCounter() (int, error) {
var counter int
row := dp.db.QueryRow("SELECT value FROM counter_table LIMIT 1")
err := row.Scan(&counter)
if err != nil {
return 0, err
}
return counter, nil
}

func (dp *Provider) UpdateCounter(value int) error {
_, err := dp.db.Exec("UPDATE counter_table SET value = value + $1", value)
return err
}

```

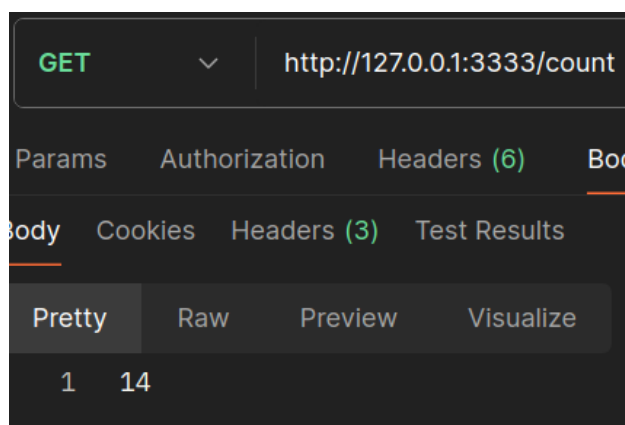


Рисунок 6 — Get запрос в Postman


	value 
1	14

Рисунок 7 — Содержимое таблицы counter_table

3. Зафиксировали изменения, сделали коммит и отправили полученное состояние ветки dev в удаленный репозиторий GitHub. Через интерфейс GitHub создали Pull Request dev --> master

Заключение: в ходе лабораторной работы получили первичные навыки организации кодовой базы проекта на Golang.

