



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

**ОТЧЕТ**

**по лабораторной работе № 10**

Название: Архитектура микросервисов на Golang

**Дисциплина:** Языки интернет-программирования

Студент

ИУ6-33Б  
(Группа)

06.09.2024  
(Подпись, дата)  
14.09.2024

Б.Ш Истамилов  
(И.О. Фамилия)

Преподаватель

14.09.2024  
(Подпись, дата)

В.Д. Шульман  
(И.О. Фамилия)

Москва, 2024

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

Ход работы:

Код программы:

main.go

```
package main
```

```
import (  
    "flag"  
    "log"
```

```
    "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", "Path to  
configuration file")  
    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)  
    uc := usecase.NewUsecase(prv)  
    srv := api.NewServer(cfg.IP, cfg.Port, uc)  
    srv.Run()  
}
```

api.go

```
package api
```

```
import (  
    "fmt"
```

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

```
type Server struct {  
    server *echo.Echo  
    address string  
    uc      Usecase  
}
```

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

```
    srv.server.GET("/api/user", srv.GetUser)
    srv.server.POST("/api/user", srv.AddUser)
```

```
    return srv
}
```

```
func (srv *Server) Run() {
    srv.server.Logger.Fatal(srv.server.Start(srv.address))
}
```

handler.go

```
package api
```

```
import (
    "net/http"
```

```
    "web-10/internal/query/model"
```

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

```
)
```

```
func (srv *Server) GetUser(c echo.Context) error {
    name := c.QueryParam("name")
    if name == "" {
        return c.String(http.StatusBadRequest, "Parameter 'name' is required")
    }
}
```

```
    user, err := srv.uc.GetUser(name)
    if err != nil {
        return c.String(http.StatusInternalServerError, err.Error())
    }
    if user == nil {
        return c.String(http.StatusNotFound, "User not found")
    }
}
```

```
    return c.JSON(http.StatusOK, user)
}
```

```
func (srv *Server) AddUser(c echo.Context) error {
    var user model.User
    if err := c.Bind(&user); err != nil {
        return c.String(http.StatusBadRequest, "Invalid JSON format")
    }
}
```

```

    if err := srv.uc.AddUser(user.Name); err != nil {
        return c.String(http.StatusInternalServerError, err.Error())
    }

```

```

    return c.String(http.StatusCreated, "User added successfully")
}

```

interface.go

```
package api
```

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

```

type Usecase interface {
    GetUser(name string) (*model.User, error)
    AddUser(name string) error
}

```

config.go

```
package config
```

```

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"`
}

```

load.go

```
package config
```

```

import (
    "io/ioutil"

    "gopkg.in/yaml.v3"
)

```

```

func LoadConfig(path string) (*Config, error) {
    data, err := ioutil.ReadFile(path)
    if err != nil {
        return nil, err
    }
    var cfg Config
    err = yaml.Unmarshal(data, &cfg)

```

```

    if err != nil {
        return nil, err
    }
    return &cfg, nil
}

```

model.go

```
package model
```

```

type User struct {
    ID    int    `json:"id"`
    Name  string `json:"name"`
}

```

provider.go

```
package provider
```

```

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

```

```
    "web-10/internal/query/model"
```

```

    _ "github.com/lib/pq"
)

```

```

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)
    }

```

```

    if err = conn.Ping(); err != nil {
        log.Fatal(err)
    }

```

```

    // Создаем таблицу users, если ее нет
    _, err = conn.Exec(`
        CREATE TABLE IF NOT EXISTS users (
            id SERIAL PRIMARY KEY,
            name TEXT NOT NULL
        );

```

```

    })
    if err != nil {
        log.Fatal(err)
    }

    return &Provider{conn: conn}
}

func (p *Provider) GetUser(name string) (*model.User, error) {
    var user model.User
    err := p.conn.QueryRow(
        "SELECT id, name FROM users WHERE name = $1", name).Scan(&user.ID, &user.Name)
    if err == sql.ErrNoRows {
        return nil, nil
    } else if err != nil {
        return nil, err
    }
    return &user, nil
}

func (p *Provider) AddUser(name string) error {
    _, err := p.conn.Exec("INSERT INTO users (name) VALUES ($1)", name)
    return err
}

```

usecase.go

```
package usecase
```

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

```

type Provider interface {
    GetUser(name string) (*model.User, error)
    AddUser(name string) error
}

```

```

type Usecase struct {
    provider Provider
}

```

```

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

```

```

func (u *Usecase) GetUser(name string) (*model.User, error) {
    return u.provider.GetUser(name)
}

```

```

func (u *Usecase) AddUser(name string) error {
    return u.provider.AddUser(name)
}

```

query\_example.yaml

```
ip: "127.0.0.1"
port: 3333
db:
  host: "localhost"
  port: 5432
  user: "postgres"
  password: "postgres"
  dbname: "sandbox"
```

2)Перепишем сервис count под новую архитектуру.

Код программы:

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", "Path to
configuration file")
    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)
    uc := usecase.NewUsecase(prv)
    srv := api.NewServer(cfg.IP, cfg.Port, uc)
    srv.Run()
}
```

api.go

```
package api
```

```
import (
    "fmt"
```

```

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

type Usecase interface {
    FetchCount() (int, error)
    IncreaseCount(int) error
}

type Server struct {
    server *echo.Echo
    address string
    uc      Usecase
}

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

    srv.server.GET("/count", srv.GetCount)
    srv.server.POST("/count", srv.IncreaseCount)

    return srv
}

func (srv *Server) Run() {
    srv.server.Logger.Fatal(srv.server.Start(srv.address))
}

```

handler.go

```

package api

import (
    "fmt"
    "net/http"
    "strconv"

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

func (srv *Server) GetCount(c echo.Context) error {
    count, err := srv.uc.FetchCount()
    if err != nil {
        return c.String(http.StatusInternalServerError, err.Error())
    }
    return c.JSON(http.StatusOK, map[string]int{"count": count})
}

func (srv *Server) IncreaseCount(c echo.Context) error {

```



```

var input struct {
    Value int `json:"value"`
}

if err := c.Bind(&input); err != nil || input.Value == 0 {
    // Если не удалось привязать JSON, пробуем получить из формы
    countStr := c.FormValue("count")
    if countStr == "" {
        return c.String(http.StatusBadRequest, "Invalid input")
    }
    value, err := strconv.Atoi(countStr)
    if err != nil {
        return c.String(http.StatusBadRequest, "Invalid input value")
    }
    input.Value = value
}

if err := srv.uc.IncreaseCount(input.Value); err != nil {
    return c.String(http.StatusBadRequest, err.Error())
}

return c.String(http.StatusOK, fmt.Sprintf("Counter increased by %d",
input.Value))
}

```

config.go

```
package config
```

```

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"`
}

```

load.go

```
package config
```

```

import (
    "io/ioutil"

    "gopkg.in/yaml.v3"
)

```

```

func LoadConfig(path string) (*Config, error) {
    data, err := ioutil.ReadFile(path)
    if err != nil {
        return nil, err
    }
    var cfg Config
    err = yaml.Unmarshal(data, &cfg)
    if err != nil {
        return nil, err
    }
    return &cfg, nil
}

```

provider.go

```

package provider

```

```

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

```

```

    _ "github.com/lib/pq"
)

```

```

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", psqInfo)
    if err != nil {
        log.Fatal(err)
    }

```

```

    if err = conn.Ping(); err != nil {
        log.Fatal(err)
    }

```

```

    // Создаем таблицу counter, если ее нет
    _, err = conn.Exec(`
        CREATE TABLE IF NOT EXISTS counter (
            id SERIAL PRIMARY KEY,
            value INTEGER NOT NULL
        );
    `)
    if err != nil {
        log.Fatal(err)
    }

```

```

    // Инициализируем счетчик, если записи нет
    var exists bool
    err = conn.QueryRow("SELECT EXISTS(SELECT 1 FROM counter)").Scan(&exists)
    if err != nil {
        log.Fatal(err)
    }
    if !exists {
        _, err = conn.Exec("INSERT INTO counter (value) VALUES (0)")
        if err != nil {
            log.Fatal(err)
        }
    }
}

return &Provider{conn: conn}
}

```

sql.go

```
package provider
```

```

func (p *Provider) FetchCount() (int, error) {
    var count int
    err := p.conn.QueryRow("SELECT value FROM counter LIMIT 1").Scan(&count)
    if err != nil {
        return 0, err
    }
    return count, nil
}

```

```

func (p *Provider) IncreaseCount(value int) error {
    _, err := p.conn.Exec("UPDATE counter SET value = value + $1", value)
    return err
}

```

usecase.go

```
package usecase
```

```
import "web-10/pkg/vars"
```

```

type Provider interface {
    FetchCount() (int, error)
    IncreaseCount(int) error
}

```

```

type Usecase struct {
    provider Provider
}

```

```

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

```

```
func (u *Usecase) FetchCount() (int, error) {  
    return u.provider.FetchCount()  
}
```

```
func (u *Usecase) IncreaseCount(value int) error {  
    if value <= 0 {  
        return vars.ErrInvalidValue  
    }  
    return u.provider.IncreaseCount(value)  
}
```

count\_example.yaml

```
ip: "127.0.0.1"  
port: 3333  
db:  
    host: "localhost"  
    port: 5432  
    user: "postgres"  
    password: "postgres"  
    dbname: "sandbox"
```

Вывод: Я научился организовывать код проекта на Go для поддержки и расширяемости проекта.