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 № 10

Название: Архитектура микросервисов на Golang

Дисциплина: Языки интернет программирования

Студент	ИУ6-33Б	12.12.24	Пономаренко В.М.
	(Группа)	(Подпись, дата)	(И.О. Фамилия)
Преподаватель		12.12.24	Шульман В.Д.
		(Подпись, дата)	(И.О. Фамилия)

Москва, 2024

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

Задание: Модернизировать код сервисов count и query в соответствии с правилами чистой архитектуры

Ход работы:

/cmd – точка входа в приложение
/configs – статические конфигурации в формате YAML
/internal – модули, не предназначенные для использования вне проекта (работа с БД, API,)
/pkg – библиотеки и пакеты

Задание 1. Query

1. Функционал аналогичен функционалу в предыдущих лабораторных работах
   Ниже приведены листинги

cmd/query/main.go

```go
package main

import (
    "flag"
    "log"

    _ "github.com/lib/pq"
    "github.com/vera2005/lr10/internal/query/api"
    "github.com/vera2005/lr10/internal/query/config"
    "github.com/vera2005/lr10/internal/query/provider"
    "github.com/vera2005/lr10/internal/query/usecase"
)

func main() {
    // Считываем аргументы командной строки
    configPath := flag.String("config-path", "D:\\Go\\lr10\\\\configs\\query.yaml", "путь к файлу конфигурации")
    flag.Parse()

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

    //Инициализация провайдера
    prv := provider.NewProvider(cfg.DB.Host, cfg.DB.Port, cfg.DB.User, cfg.DB.Password, cfg.DB.DBname)
    //Инициализация бизнес-логики
    use := usecase.NewUsecase(cfg.Usecase.DefaultMessage, prv)
    //Инициализация сервера
    srv := api.NewServer(cfg.IP, cfg.Port, cfg.API.MaxMessageSize, use)

    srv.Run()
}
```

configs/query.yaml

```yaml
ip: "127.0.0.1"
port: 8081
api:
  max_message_size: 32
usecase:
  default_message: "User"
db:
  host: "localhost"
  port: 5432
  user: "postgres"
  password: "catjkm8800"
  dbname: "query"
```

internal/query/api/api.go

```go
package api

import (
    "fmt"
```

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

type Server struct {
    maxSize int

    server  *echo.Echo
    address string

    uc Usecase
}

func NewServer(ip string, port int, maxSize int, uc Usecase) *Server {
    api := Server{
        maxSize: maxSize,
        uc:      uc,
    }

    api.server = echo.New()
    api.server.GET("/query", api.GetQuery)
    api.server.POST("/query", api.PostQuery)
    api.server.PUT("/query", api.PutQuery)

    api.address = fmt.Sprintf("%s:%d", ip, port)

    return &api
}

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

internal/query/api/handler.go

```
package api

import (
    "errors"
    "net/http"
    "regexp"

    "github.com/labstack/echo/v4"
    "github.com/vera2005/lr10/pkg/vars"
)

// GetQuery возвращает приветствие случайному пользователю
func (srv *Server) GetQuery(e echo.Context) error {
    msg, err := srv.uc.FetchQuery()
    if err != nil {
        return e.String(http.StatusInternalServerError, err.Error())
    }

    return e.JSON(http.StatusOK, msg)
}

// PostQuery Помещает нового пользователя в БД
func (srv *Server) PostQuery(c echo.Context) error {
    nameInput := c.QueryParam("name") // Получаем Query-параметр
    if nameInput == "" {
        return c.String(http.StatusBadRequest, "Missing 'name' query parameter")
    }
    //проверка на корректность имени
    re := regexp.MustCompile(`[a-zA-Za-яА-Я]`)
    if !re.MatchString(nameInput) {
        return c.String(http.StatusBadRequest, "empty string")
    }
    if len([]rune(nameInput)) > srv.maxSize {
        return c.String(http.StatusBadRequest, "name too large")
    }
    err := srv.uc.SetQuery(nameInput)
    if err != nil {
        if errors.Is(err, vars.ErrAlreadyExist) {
            return c.String(http.StatusConflict, err.Error())
        }
        return c.String(http.StatusInternalServerError, err.Error())
    }
    return c.String(http.StatusCreated, "OK")
```

```
}

func (srv *Server) PutQuery(c echo.Context) error {
    nameInput := c.QueryParam("name") // Получаем Query-параметр
    if nameInput == "" {
        return c.String(http.StatusBadRequest, "Missing 'name' query parameter")
    }
    //проверка на корректность имени
    re := regexp.MustCompile(`[a-zA-Za-яА-Я]`)
    if !re.MatchString(nameInput) {
        return c.String(http.StatusBadRequest, "empty string")
    }
    if len([]rune(nameInput)) > srv.maxSize {
        return c.String(http.StatusBadRequest, "name too large")
    }
    err := srv.uc.ChangeQuery(nameInput)
    if err != nil {
        if errors.Is(err, vars.ErrAlreadyExist) {
            return c.String(http.StatusConflict, err.Error())
        }
        return c.String(http.StatusInternalServerError, err.Error())
    }
    return c.String(http.StatusCreated, "OK")
}
```

internal/query/api/interface.go

```
package api

type Usecase interface {
    FetchQuery() (string, error)
    SetQuery(name string) error
    ChangeQuery(name string) error
}
```

internal/query/config/ config.go

```
package config

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

    API     api     `yaml:"api"`
    Usecase usecase `yaml:"usecase"`
    DB      db      `yaml:"db"`
}

type api struct {
    MaxMessageSize int `yaml:"max_message_size"`
}

type usecase struct {
    DefaultMessage string `yaml:"default_message"`
}

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

internal/query/config/load.go

```
package config

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

```go
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
}
```

internal/query/provider/ provider.go

```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)

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

    return &Provider{conn: conn}
}
```

internal/query/provider/sql.go

```go
package provider

import (
    "database/sql"
    "errors"
)

func (p *Provider) SelectName() (string, error) {
    var msg string
    err := p.conn.QueryRow("SELECT name FROM query ORDER BY id DESC LIMIT 1").Scan(&msg)
    if err != nil {
        if errors.Is(err, sql.ErrNoRows) {
            return "", nil
        }
        return "", err
    }

    return msg, nil
}

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

```
    }

    return nil
}

func (p *Provider) UpdateQuery(n string) error {
    _, err := p.conn.Exec("UPDATE query SET name = $1 WHERE id = (SELECT MAX(id) FROM query)", n)
    if err != nil {
        return err
    }
    return nil
}
```

internal/query/usecase/interface.go

```
package usecase

type Provider interface {
    SelectName() (string, error)
    InsertQuery(string) error
    UpdateQuery(string) error
}
```

internal/query/usecase/query.go

```
package usecase

func (u *Usecase) FetchQuery() (string, error) {
    nam, err := u.p.SelectName()
    msg := "Hello," + nam + "!"
    if err != nil {
        return "", err
    }

    if msg == "" {
        return u.defaultMsg, nil
    }

    return msg, nil
}

func (u *Usecase) SetQuery(name string) error {
    err := u.p.InsertQuery(name)
    if err != nil {
        return err
    }

    return nil
}

func (u *Usecase) ChangeQuery(name string) error {
    err := u.p.UpdateQuery(name)
    if err != nil {
        return err
    }

    return nil
}
```

internal/query/usecase/usecase.go

```
package usecase

type Usecase struct {
    defaultMsg string

    p Provider
}

func NewUsecase(defaultMsg string, p Provider) *Usecase {
    return &Usecase{
        defaultMsg: defaultMsg,
        p:          p,
    }
```

```
}
```

pkg/consts/default.go

```
package consts

const (
    IP = "127.0.0.1"
)
```

pkg/vars/err.go

```
package vars

import "errors"

var (
    ErrAlreadyExist = errors.New("already exist")
)
```

Демонстрация результатов тестирования

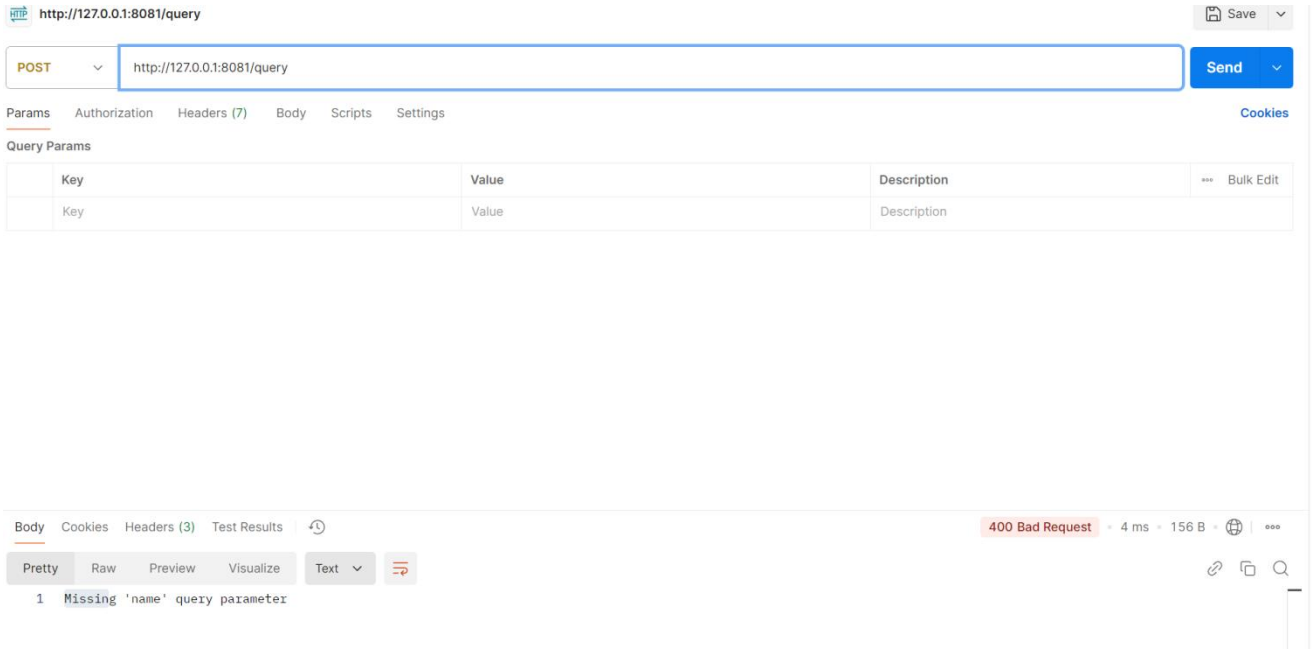


Рисунок 1 –

POST с пропущенным параметром

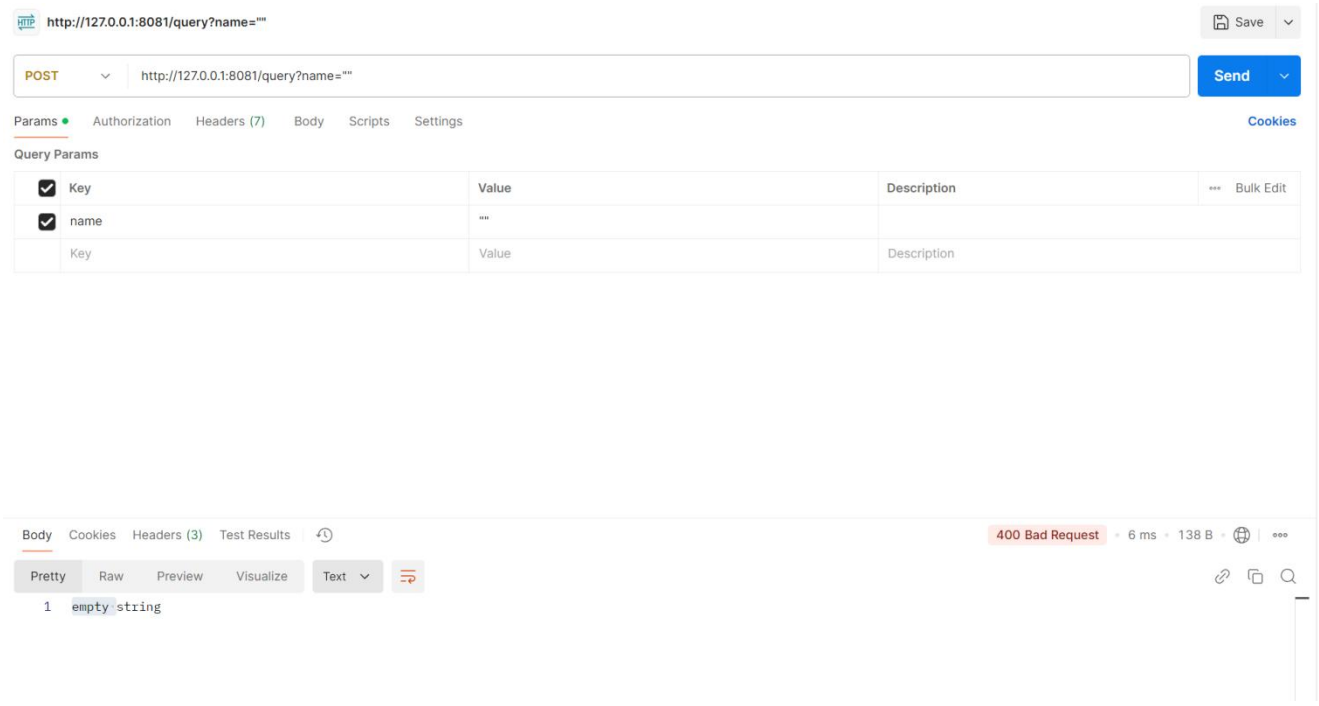


Рисунок 2 -POST с пустой строкой

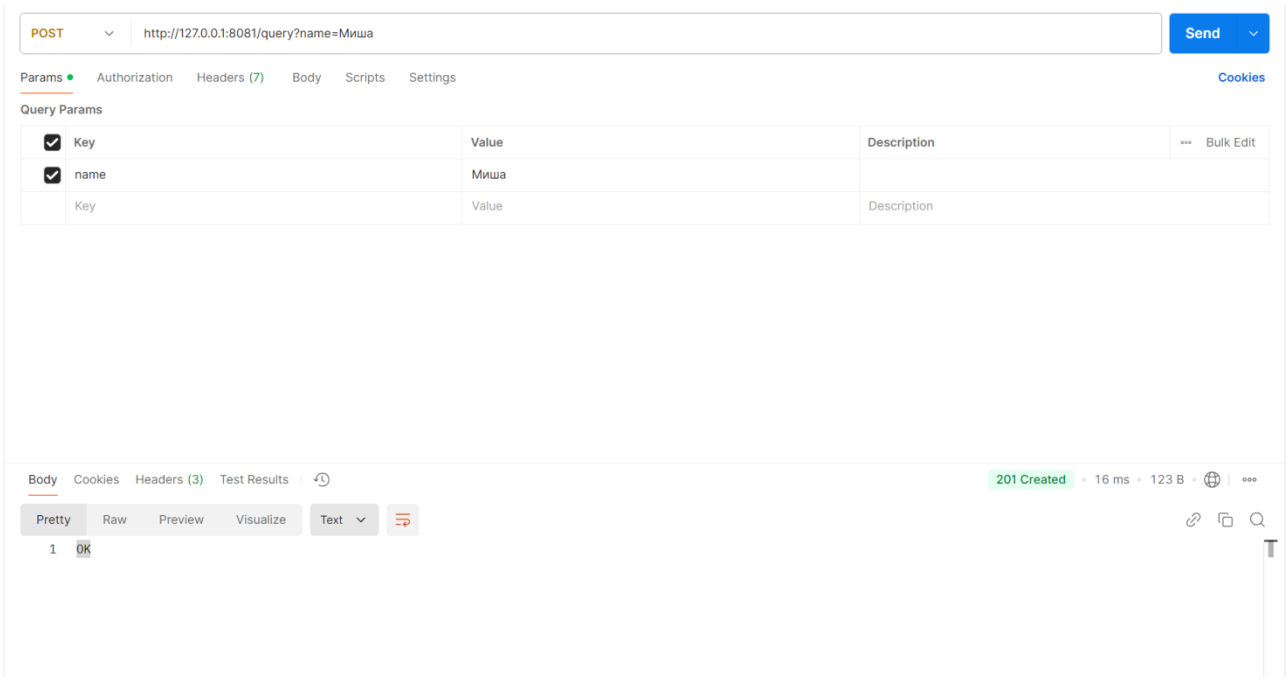


Рисунок 3 – POST

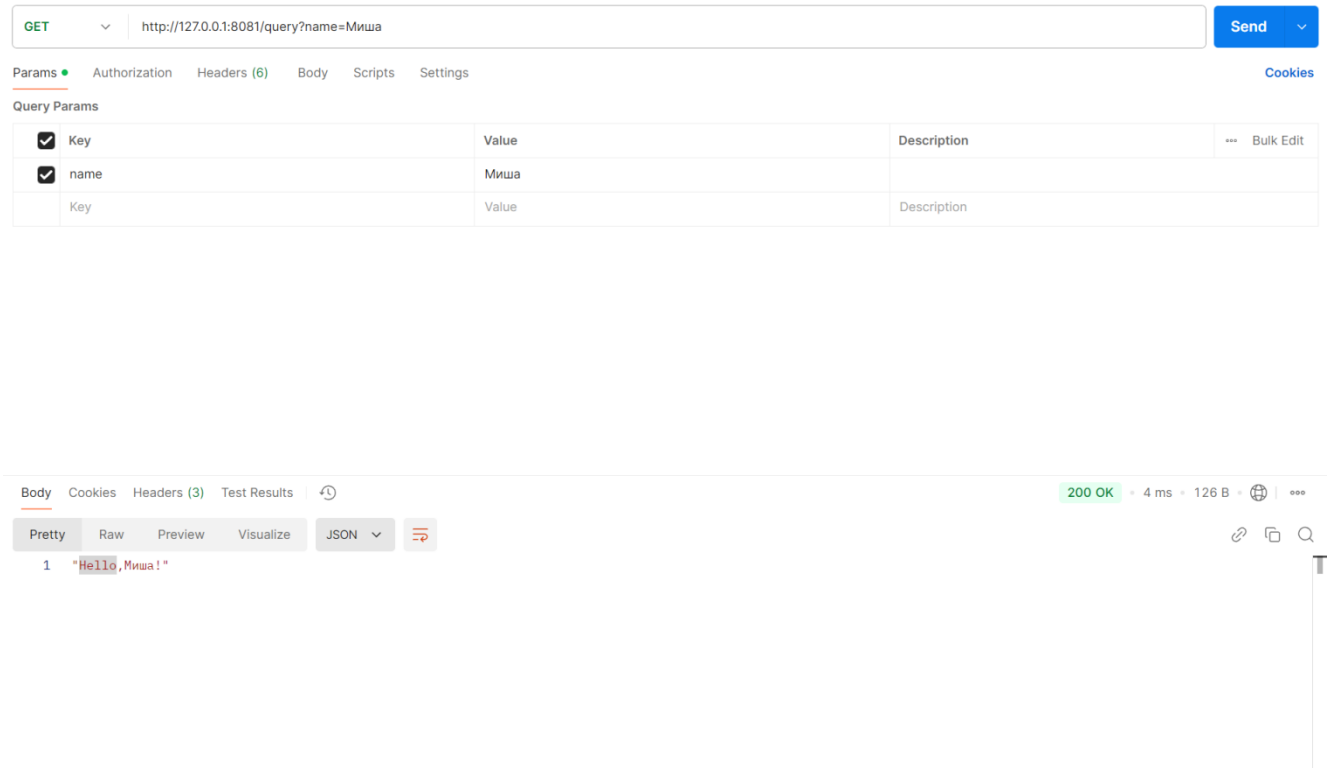
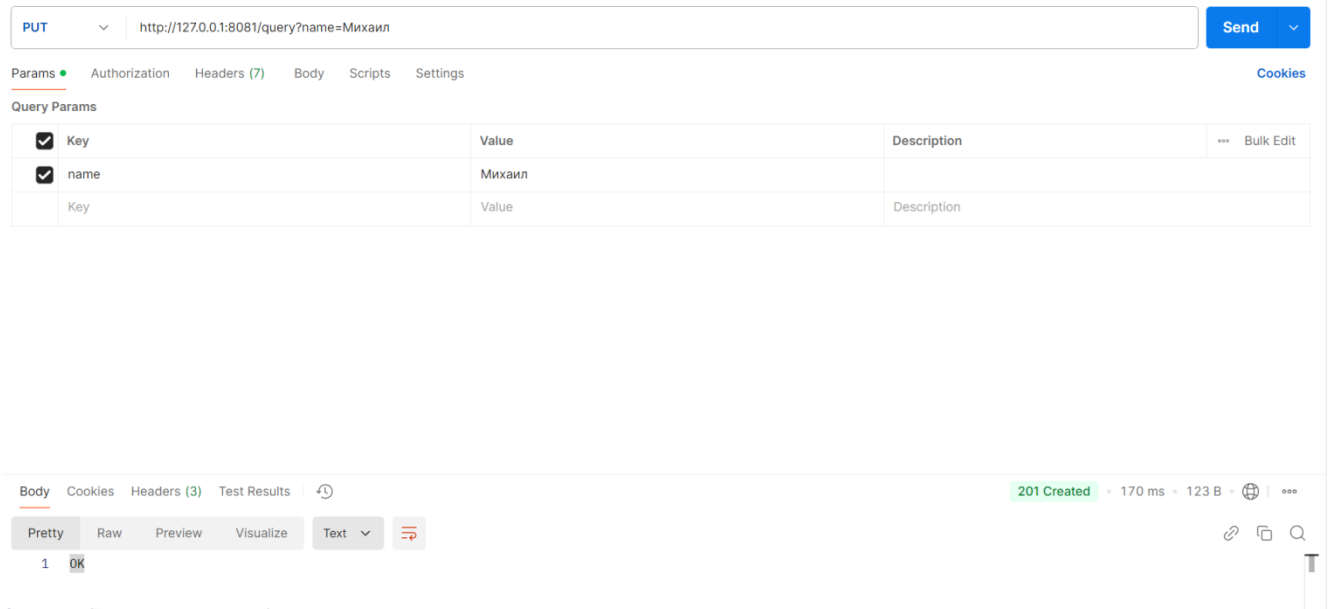


Рисунок 4-GET



20	20	Vera	20	20	Vera
21	21	qwe1	21	21	qwe1
22	22	qwe2	22	22	qwe2
23	23	qwe3	23	23	qwe3
24	24	Vera	24	24	Vera
25	25	"Миша"	25	25	"Миша"
26	26	Михаил	26	26	Миша

Рисунок 5 - PUT с демонстрацией БД

Задаие 2. Count

1. Функционал аналогичен функционалу в предыдущих лабораторных работах Ниже приведены листинги

cmd/count/main.go

```go
package main

import (
    "flag"
    "log"

    _ "github.com/lib/pq"
    "github.com/vera2005/lr10/internal/count/api"
    "github.com/vera2005/lr10/internal/count/config"
    "github.com/vera2005/lr10/internal/count/provider"
    "github.com/vera2005/lr10/internal/count/usecase"
)

func main() {
    // Считываем аргументы командной строки
    configPath := flag.String("config-path", "D:\\Go\\lr10\\\\configs\\count.yaml", "путь к файлу конфигурации")
    flag.Parse()

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

    //Инициализация провайдера
    prv := provider.NewProvider(cfg.DB.Host, cfg.DB.Port, cfg.DB.User, cfg.DB.Password, cfg.DB.DBname)
    //Инициализация бизнес-логики
    use := usecase.NewUsecase(cfg.Usecase.DefaultCount, prv)
    //Инициализация сервера
    srv := api.NewServer(cfg.IP, cfg.Port, cfg.API.MaxNum, use)

    srv.Run()
}
```

configs/ count.yaml

```yaml
ip: "127.0.0.1"
port: 8081
api:
  max_number: 100000
usecase:
  default_count: "0"
db:
  host: "localhost"
  port: 5432
  user: "postgres"
  password: "catjkm8800"
  dbname: "count"
```

internal/ count /api/api.go

```go
package api

import (
    "fmt"

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

type Server struct {
    maxNum int

    server  *echo.Echo
    address string

    uc Usecase
}

func NewServer(ip string, port int, maxNum int, uc Usecase) *Server {
    api := Server{
        maxNum: maxNum,
        uc:     uc,
    }

    api.server = echo.New()
    api.server.GET("/count", api.GetCount)
    api.server.POST("/count", api.PostCount)
    api.server.PUT("/count", api.PutCount)

    api.address = fmt.Sprintf("%s:%d", ip, port)

    return &api
}

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

internal/ count /api/handler.go

```go
package api

import (
    "errors"
    "net/http"

    "github.com/labstack/echo/v4"
    "github.com/vera2005/lr10/pkg/vars"
)

type CountInput struct {
    Val float32 `json:"val"` // Используем float32 для автоматической проверки числового значения
}

func (srv *Server) GetCount(e echo.Context) error {
    msg, err := srv.uc.FetchCount()
    if err != nil {
        return e.String(http.StatusInternalServerError, err.Error())
    }

    return e.JSON(http.StatusOK, msg)
}

func (srv *Server) PostCount(c echo.Context) error {
    input := CountInput{}
    // Привязка входных данных и проверка на ошибки
    if err := c.Bind(&input); err != nil {
        return c.String(http.StatusBadRequest, "Неправильный формат JSON")
    }
    err := srv.uc.SetCount(input.Val)
    if err != nil {
        if errors.Is(err, vars.ErrAlreadyExist) {
            return c.String(http.StatusConflict, err.Error())
        }
        return c.String(http.StatusInternalServerError, err.Error())
    }
    return c.String(http.StatusCreated, "OK")
}

func (srv *Server) PutCount(c echo.Context) error {
    input := CountInput{}
    if err := c.Bind(&input); err != nil {
        return c.String(http.StatusBadRequest, "Неправильный формат JSON")
    }
    err := srv.uc.ChangeCount(input.Val)
    if err != nil {
        if errors.Is(err, vars.ErrAlreadyExist) {
            return c.String(http.StatusConflict, err.Error())
        }
        return c.String(http.StatusInternalServerError, err.Error())
    }
    return c.String(http.StatusCreated, "OK")
}
```

internal/ count /api/interface.go

```go
package api

type Usecase interface {
    FetchCount() (string, error)
    SetCount(float32) error
    ChangeCount(float32) error
}
```

internal/ count /config/ config.go

```go
package config

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

    API     api     `yaml:"api"`
    Usecase usecase `yaml:"usecase"`
    DB      db      `yaml:"db"`
}

type api struct {
    MaxNum int `yaml:"max_number"`
}

type usecase struct {
    DefaultCount string `yaml:"default_count"`
}

type db struct {
    Host     string `yaml:"host"`
    Port     int    `yaml:"port"`
    User     string `yaml:"user"`
```

```
    Password string `yaml:"password"`
    DBname   string `yaml:"dbname"`
}
```

internal/ count /config/load.go

```
package config

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

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
}
```

internal/ count /provider/ 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)

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

    return &Provider{conn: conn}
}
```

internal/ count /provider/sql.go

```
package provider

import (
    "database/sql"
    "errors"
)

func (p *Provider) SelectCount() (string, error) {
    var msg string
    err := p.conn.QueryRow("SELECT summa FROM count ORDER BY id DESC LIMIT 1").Scan(&msg)
    if err != nil {
        if errors.Is(err, sql.ErrNoRows) {
            return "", nil
        }
        return "", err
    }

    return msg, nil
}
```

```go
func (p *Provider) InsertCount(v float32) error {
    _, err := p.conn.Exec("INSERT INTO count (val, summa) VALUES ($1, $1 + (SELECT COALESCE(summa, 0) FROM count ORDER BY id DESC LIMIT 1))", v)
    if err != nil {
        return err
    }

    return nil
}

func (p *Provider) UpdateCount(v float32) error {
    _, err := p.conn.Exec("UPDATE count SET val = $1, summa = (val + (SELECT summa FROM count WHERE id = ((SELECT MAX(id) FROM count) - 1))) WHERE id
= (SELECT MAX(id) FROM count)", v)
    if err != nil {
        return err
    }
    return nil
}
```

internal/ count /usecase/interface.go

```go
package usecase

type Provider interface {
    SelectCount() (string, error)
    InsertCount(float32) error
    UpdateCount(float32) error
}
```

internal/ count /usecase/count.go

```go
package usecase

func (u *Usecase) FetchCount() (string, error) {
    msg, err := u.p.SelectCount()
    if err != nil {
        return "", err
    }
    return msg, nil
}

func (u *Usecase) SetCount(v float32) error {
    err := u.p.InsertCount(v)
    if err != nil {
        return err
    }

    return nil
}

func (u *Usecase) ChangeCount(v float32) error {
    err := u.p.UpdateCount(v)
    if err != nil {
        return err
    }

    return nil
}
```

internal/ count /usecase/usecase.go

```go
package usecase

type Usecase struct {
    defaultMsg string

    p Provider
}

func NewUsecase(defaultMsg string, p Provider) *Usecase {
    return &Usecase{
        defaultMsg: defaultMsg,
        p:          p,
    }
}
```

pkg/consts/default.go

```go
package consts

const (
```

```
    IP = "127.0.0.1"
)
```

pkg/vars/err.go

```
package vars

import "errors"

var (
    ErrAlreadyExist = errors.New("already exist")
)
```

Демонстрация результатов тестирования

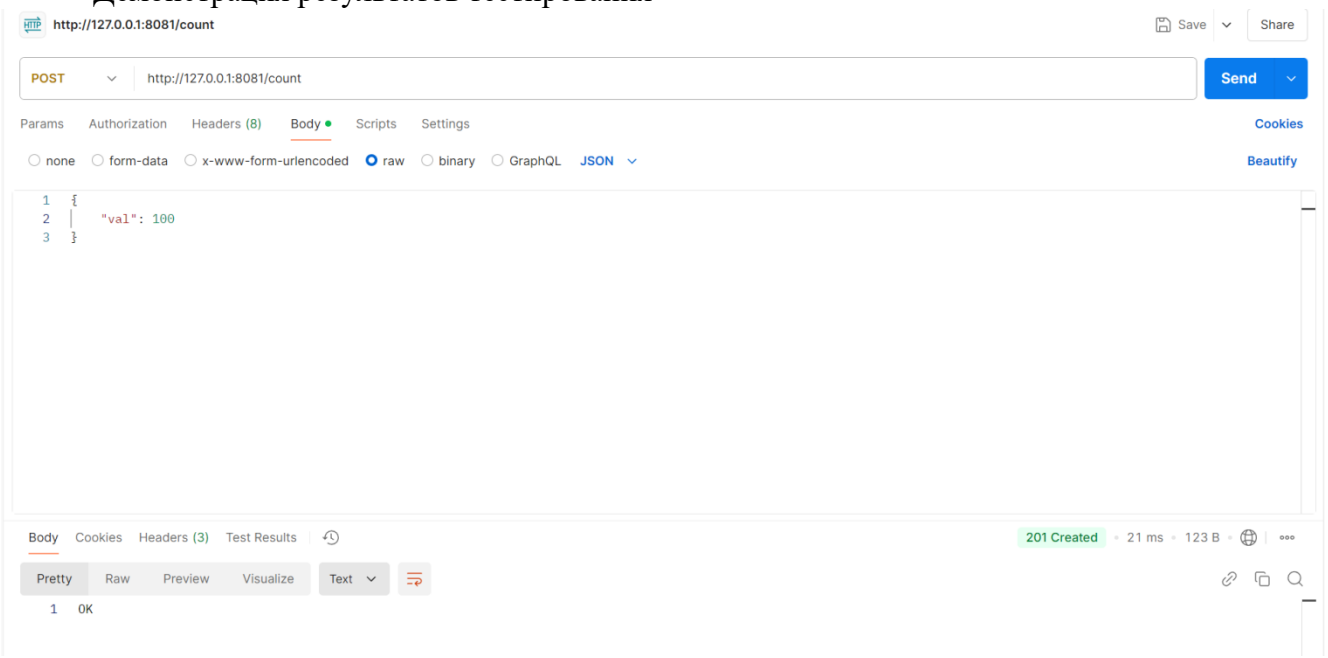
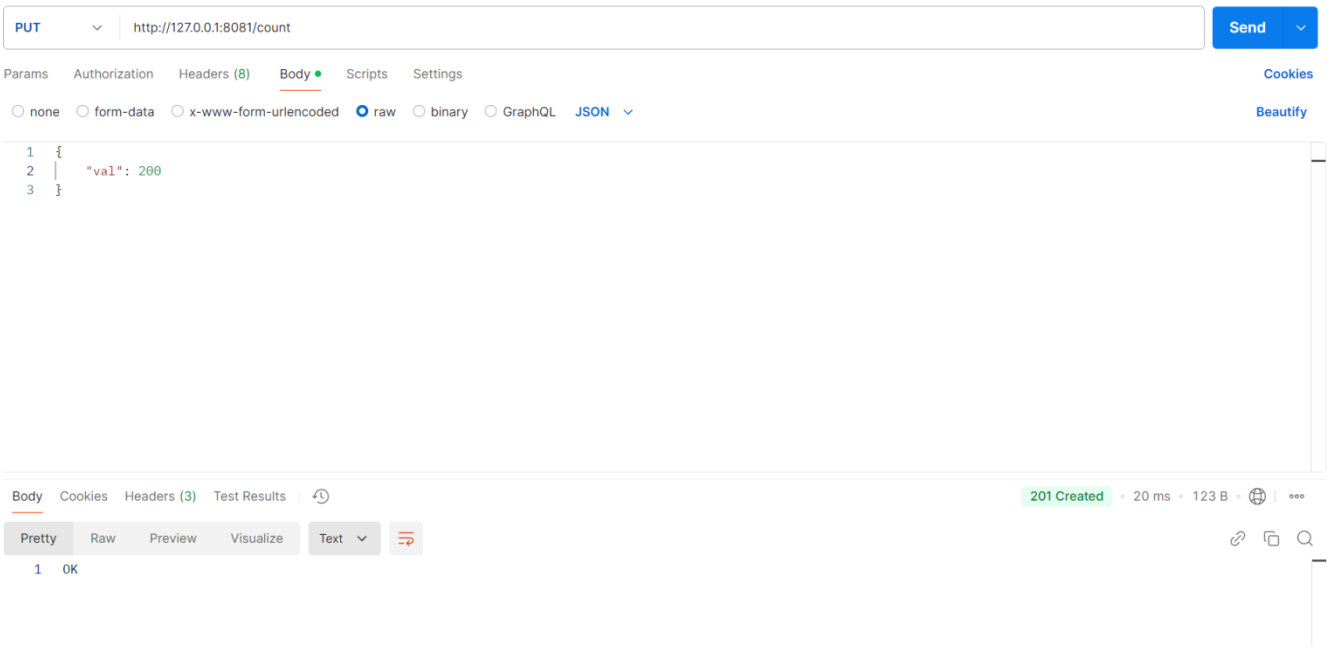


Рисунок 6 – POST



	id [PK] integer	val numeric (10,2)	summa numeric (10,2)
1	1	0.00	0.00
2	2	1.00	1.00
3	3	3.00	4.00
4	4	5.00	9.00
5	5	24.00	33.00
6	6	2.00	35.00
7	7	10.00	45.00
8	8	80.00	125.00
9	9	100.00	225.00

	id [PK] integer	val numeric (10,2)	summa numeric (10,2)
1	1	0.00	0.00
2	2	1.00	1.00
3	3	3.00	4.00
4	4	5.00	9.00
5	5	24.00	33.00
6	6	2.00	35.00
7	7	10.00	45.00
8	8	80.00	125.00
9	9	200.00	325.00

Рисунок 7 – PUT

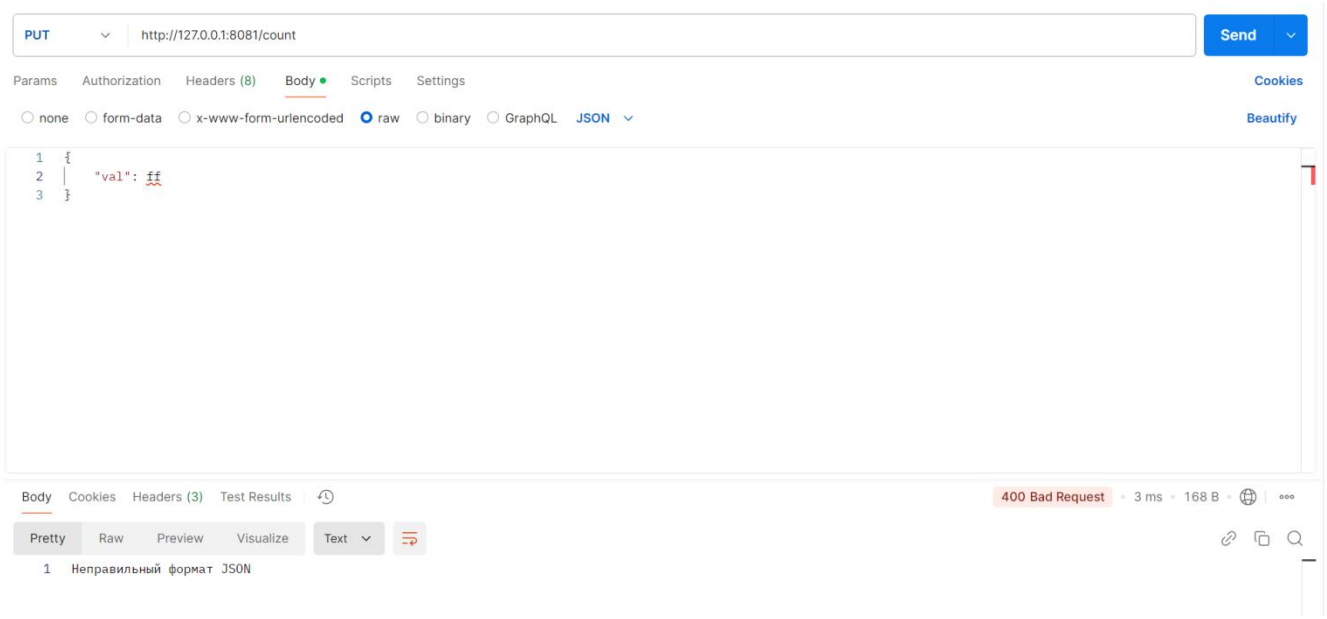
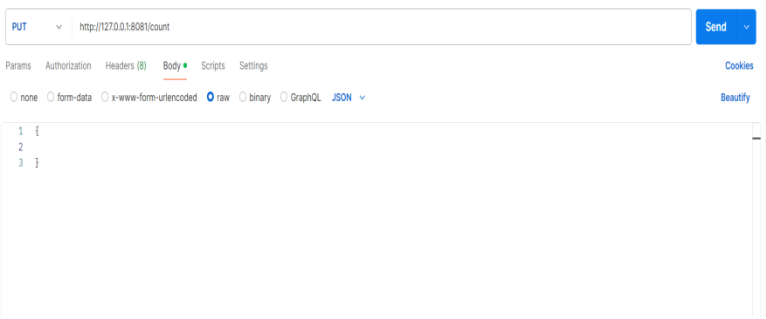


Рисунок 8 - тест с некорректными данными



	id [PK] integer	val numeric (10,2)	summa numeric (10,2)
1	1	0.00	0.00
2	2	1.00	1.00
3	3	3.00	4.00
4	4	5.00	9.00
5	5	24.00	33.00
6	6	2.00	35.00
7	7	10.00	45.00
8	8	80.00	125.00
9	9	0.00	325.00

Рисунок 9 - тест без данных (отработка по дефолтному значению)

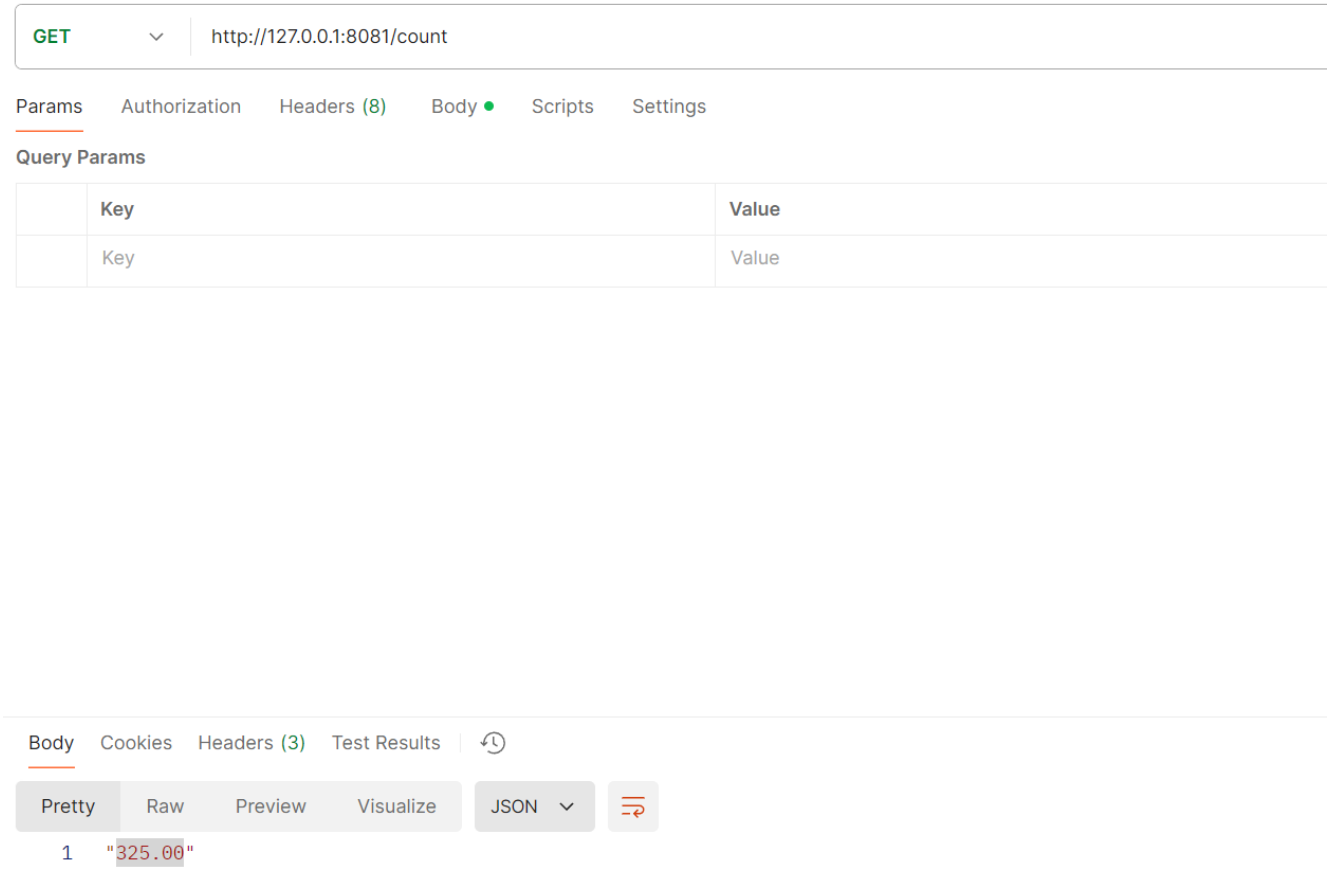


Рисунок 10-GET

Выводы: В ходе выполнения данной лабораторной работы был изучен и применен на практике концепт чистой архитектуры при создании микросервисов на языке программирования GoLang.

Список использованных источников:

1) https://github.com/golang-standards/project-layout?tab=readme-ov-file
2) https://youtu.be/V6lQG6d5LgU?si=17sjfwTYCWZMSHlw