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: Golang & Clean Architecture

Дисциплина: Языки интернет-программирования

Студент	ИУ6-32Б	14.12.2024	Л.И. Заушников
	(Группа)	(Подпись, дата)	(И.О. Фамилия)
Преподаватель		14.12.2024	В.Д. Шульман
		(Подпись, дата)	(И.О. Фамилия)

Москва, 2024

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

1. Было произведено ознакомление с материалами для подготовки перед выполнением лабораторной работы
2. Был сделан форк репозитория в GitHub (рисунок 1), копия была склонирована локально, была создана от мастера ветка dev и было произведено переключение на неё.

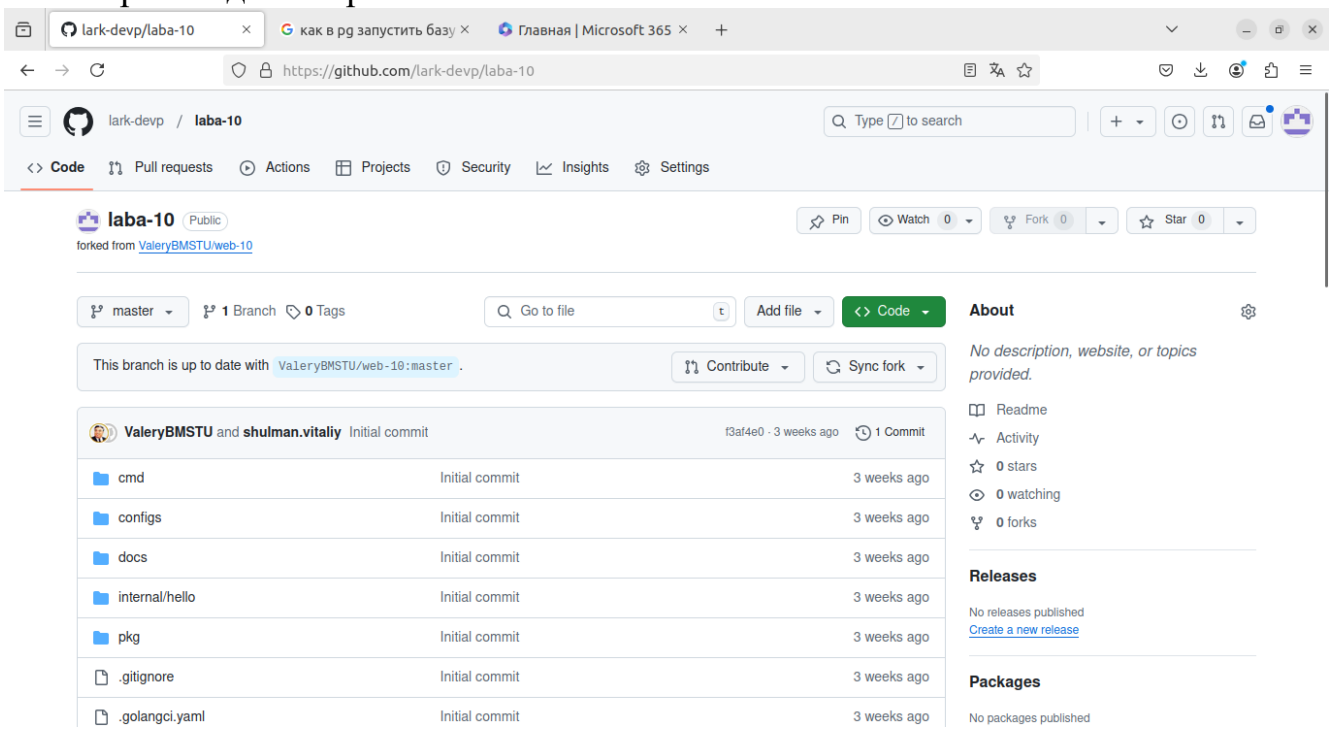

Рисунок 1 - Форкнутый репозиторий

3. Микросервисы, перекочевавшие из 9 лабораторной работы, были модифицированы, следуя гайду по “чистой арихитектуре” (рисунок 2).

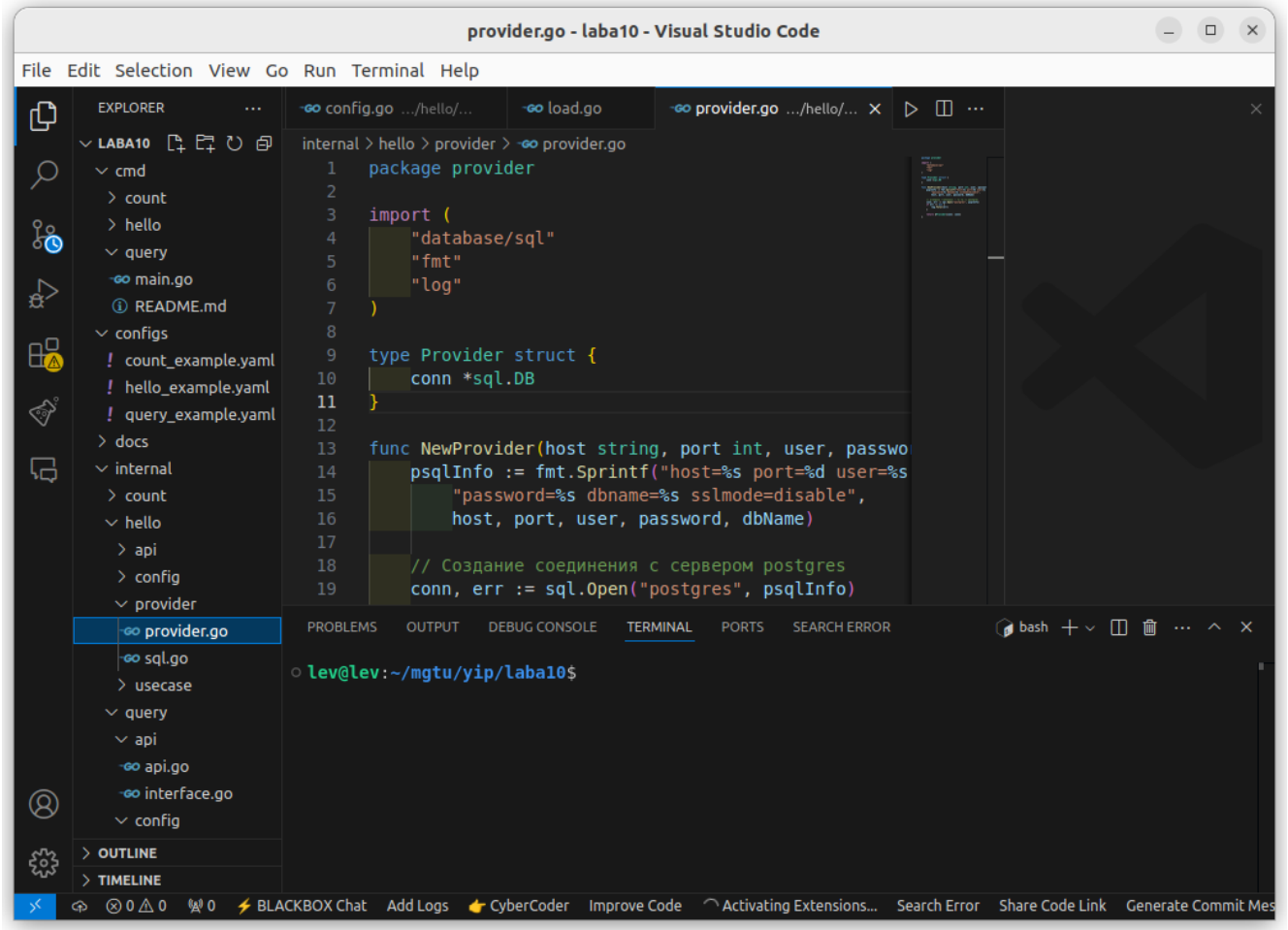

Рисунок 2 - Модификация микросервисов

4. Микросервисы были протестированы, результаты на рисунках ниже.

Задание 1 (вывод строки с приветствием)

Код файлов представлен ниже, тестирование микросервиса представлено на рисунках 4-8.

main.go

package main

import (
"flag"
"log"

"web-10/internal/hello/api"
"web-10/internal/hello/config"
"web-10/internal/hello/provider"
"web-10/internal/hello/usecase"

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

func main() {
// Считываем аргументы командной строки
configPath := flag.String("config-path", "../../configs/hello_example.yaml", "путь к
        файлу конфигурации") //fix пути к конфигу
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(cfg.Usecase.DefaultMessage, prv)
srv := api.NewServer(cfg.IP, cfg.Port, cfg.API.MaxMessageSize, use)

srv.Run()
}
```

api.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}
}
```

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"`
}
```

usercase.go

```
package usecase

type Usecase struct {
defaultMsg string

p Provider
}

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

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}
}
```

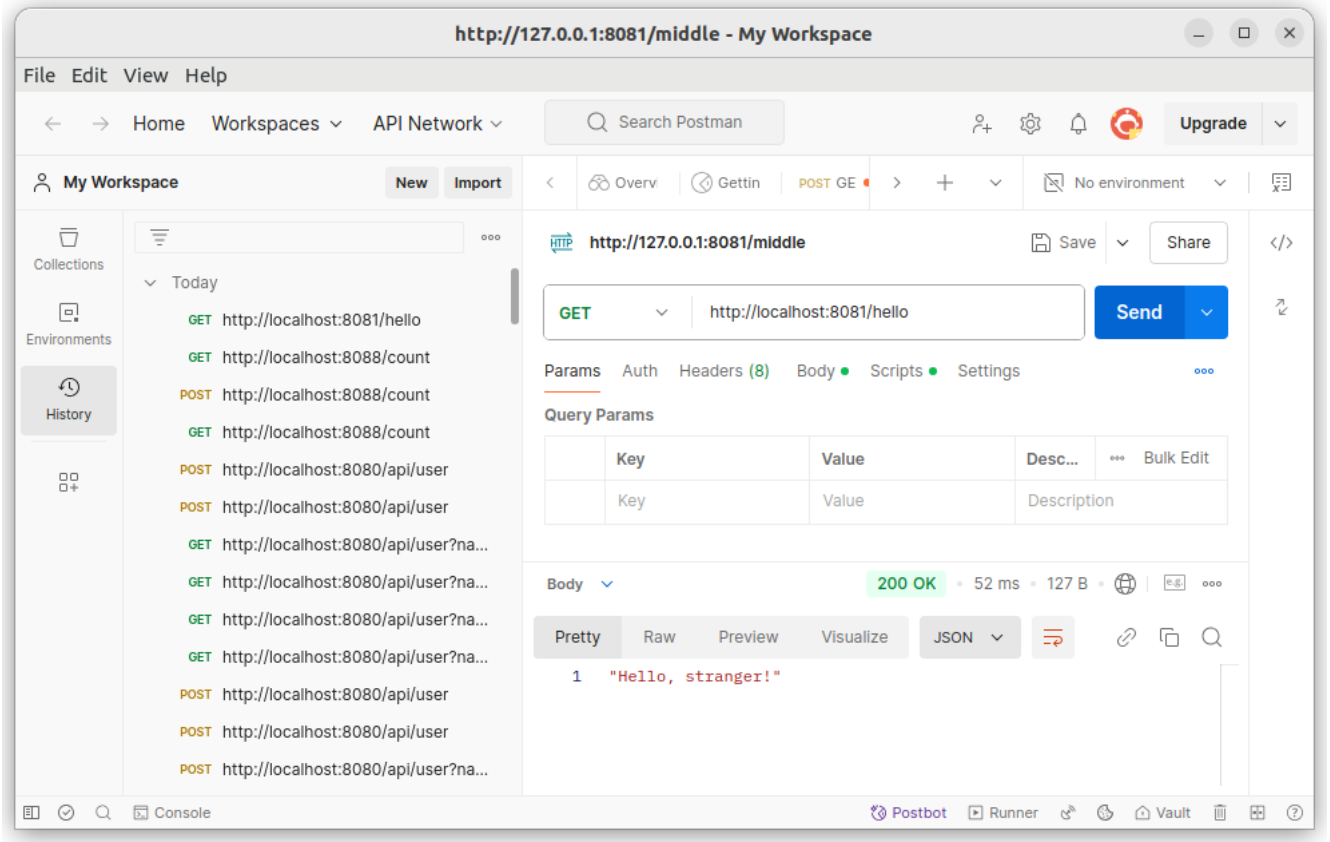

Рисунок 4 - Тестирование микросервиса hello

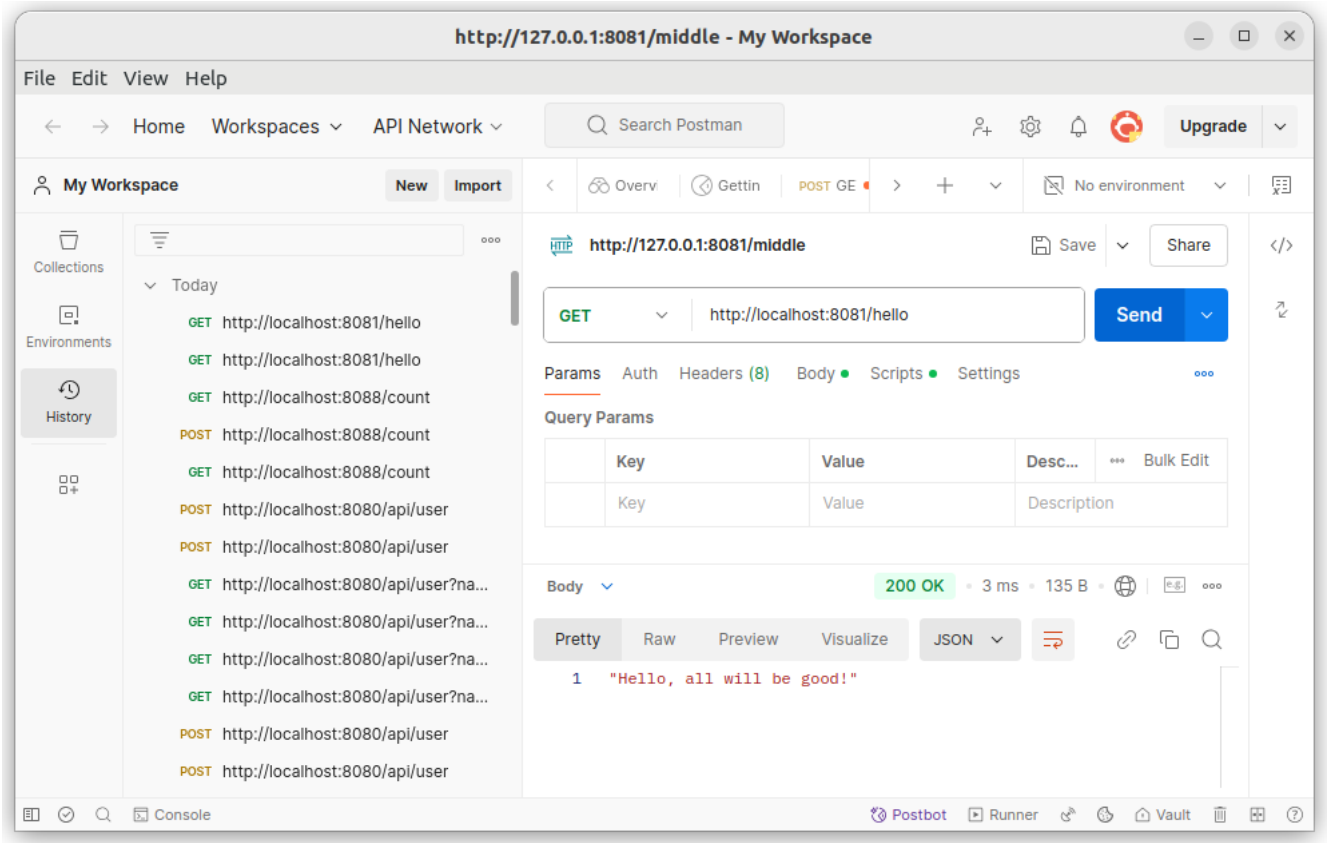

Рисунок 5 - Тестирование микросервиса hello

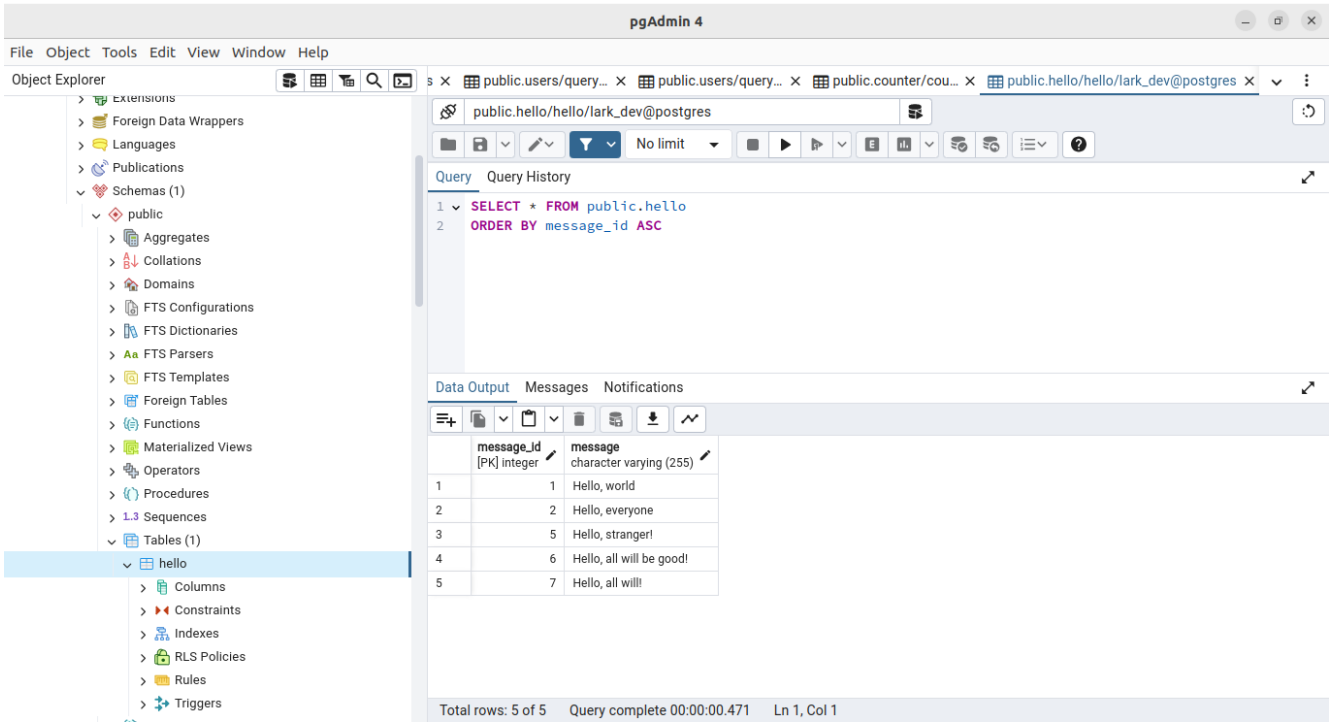

Рисунок 6 - Тестирование микросервиса hello

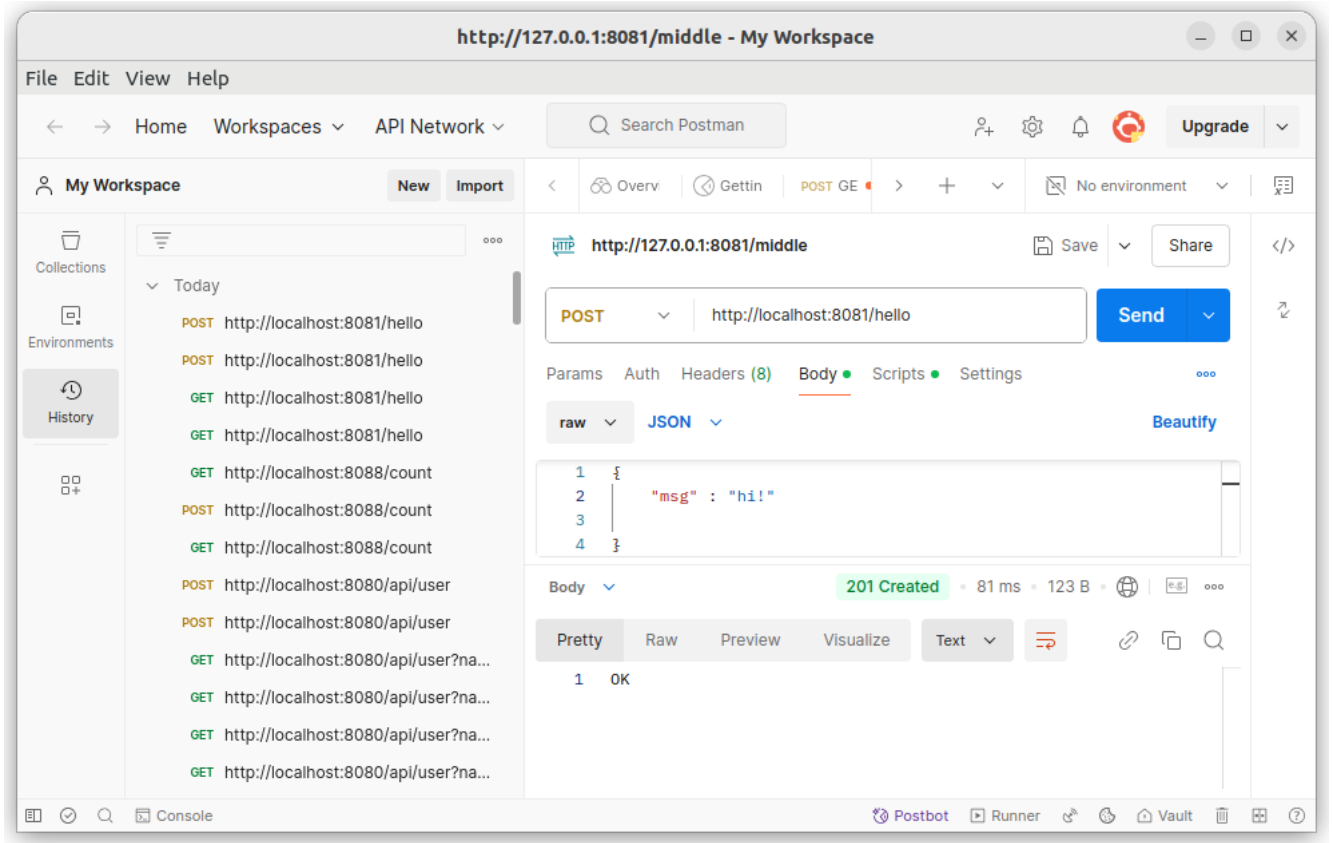

Рисунок 7 - Тестирование микросервиса hello

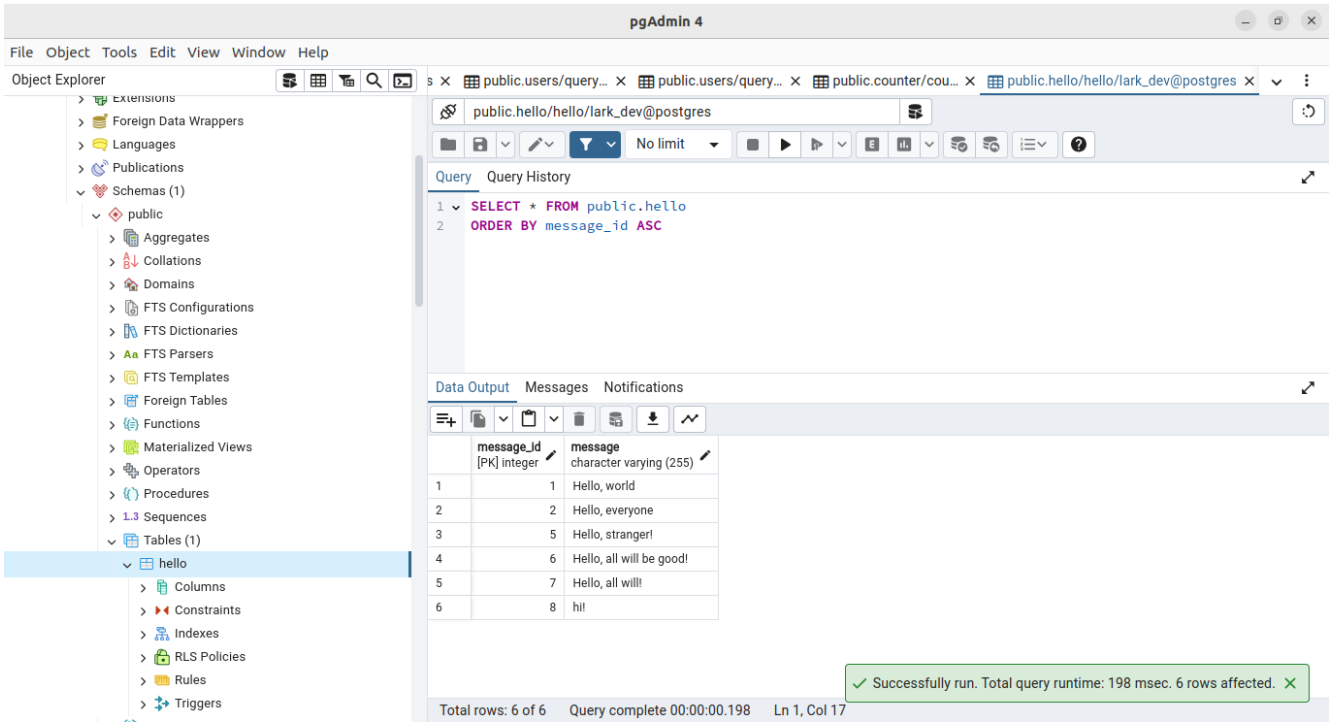

Рисунок 8 - Тестирование микросервиса hello

Задание 2 (вывод строки с ключом)

Код файлов представлен ниже, тестирование микросервиса представлено на рисунках 9-11.

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"
)

type Server struct {
Address string
Router  *echo.Echo
```

```
uc      Usecase
}

func NewServer(ip string, port int, uc 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", 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 (
"errors"
)

type Usecase struct {
defaultMsg string
p       Provider
}

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

```
p:          p,
}
}

func (u *Usecase) GetUser(name string) (string, error) {
user, err := u.p.SelectUser(name)
if err != nil {
return "", err
}
if user == "" {
return u.defaultMsg, nil
}
return user, nil
}

func (u *Usecase) CreateUser(name string) error {
err := u.p.InsertUser(name)
if err != nil {
return err
}
if name == "" {
return errors.New("имя пользователя не может быть пустым")
}
return nil

}
```

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 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 users 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 users (name) VALUES ($1)", name)
if err != nil {
return err
}
return nil
}

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

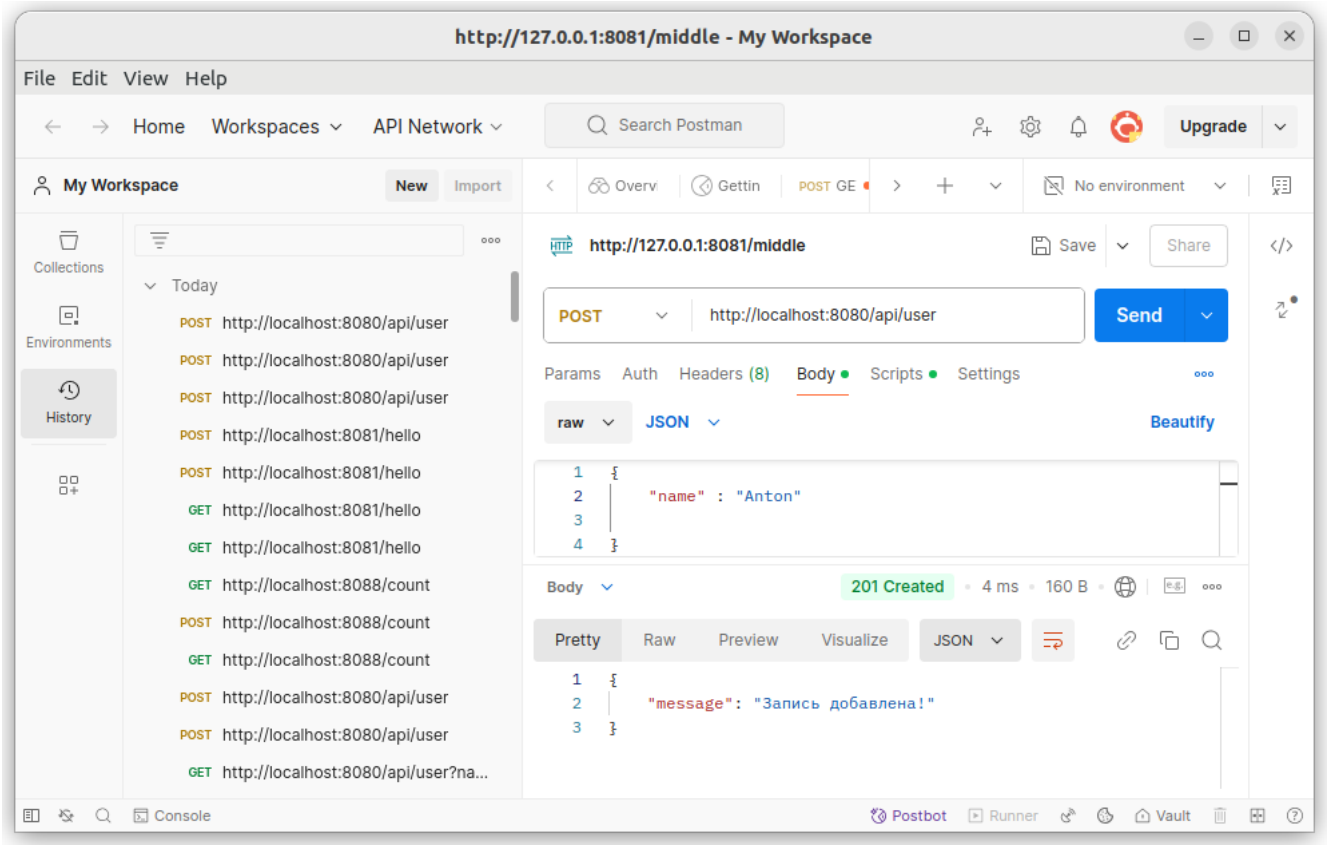

Рисунок 9 - Тестирование микросервиса query

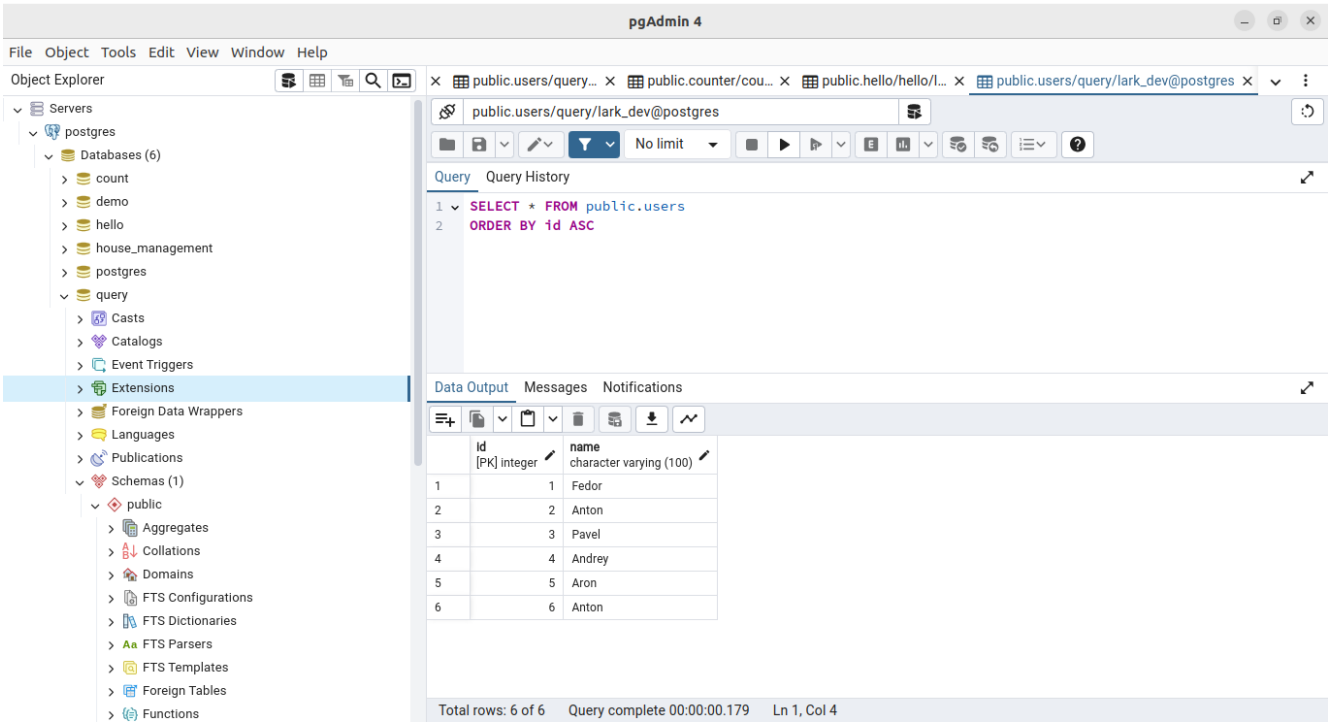

Рисунок 10 - Тестирование микросервиса query

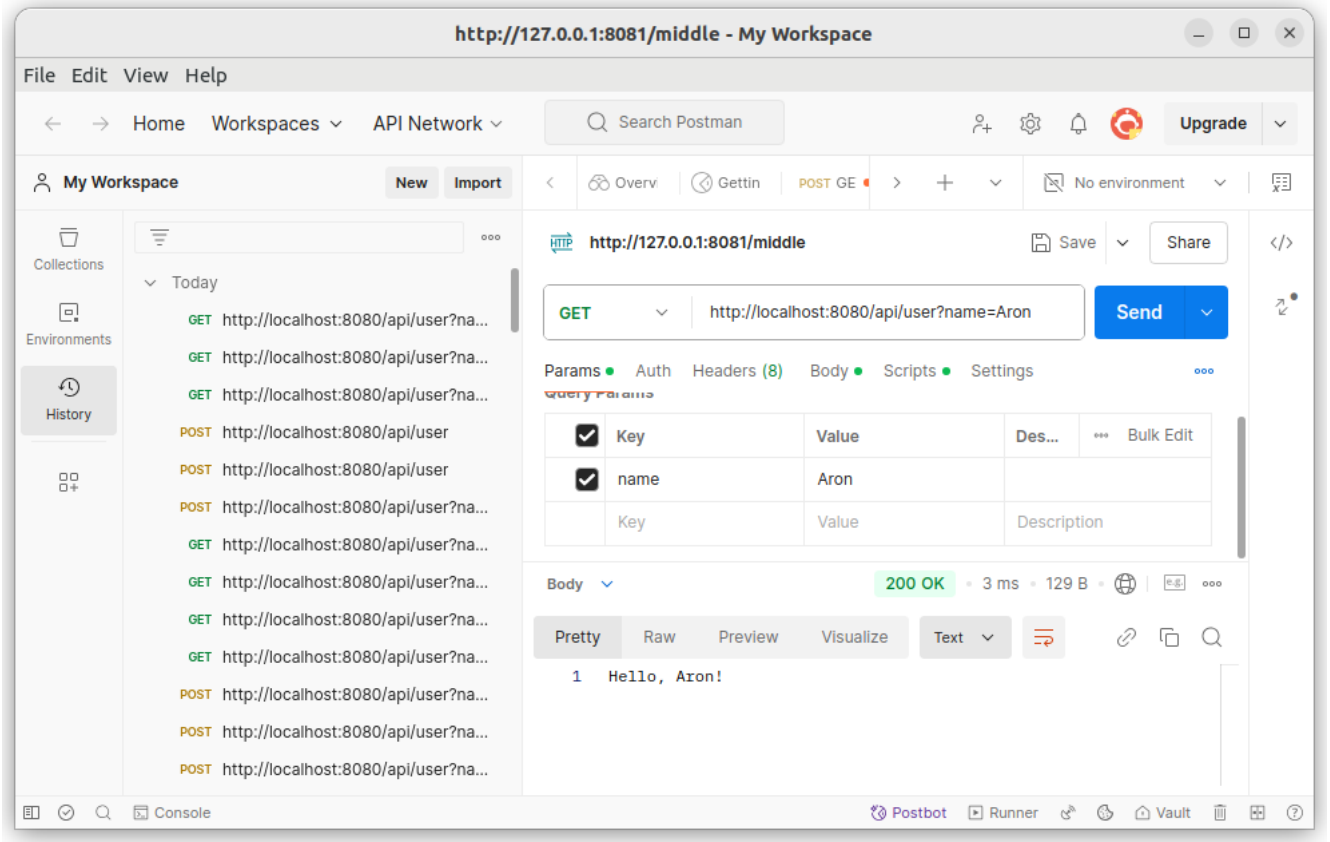

Рисунок 11 - Тестирование микросервиса query

Задание 3 (вывод и обновление счётчика)

Код файлов представлен ниже, тестирование микросервиса представлено на рисунках 12-15.

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"
"net/http"

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

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

```
}

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

s.Router.GET("/count", s.GetCounter)
s.Router.POST("/count", s.UpdateCounter)

return s
}

func (s *Server) GetCounter(c echo.Context) error {
count, err := s.Usecase.HandleGetCount()
if err != nil {
return c.JSON(http.StatusInternalServerError, map[string]string{"error": err.Error()})
}
return c.String(http.StatusOK, fmt.Sprintf("%d", count))
}

func (s *Server) UpdateCounter(c echo.Context) error {
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 := s.Usecase.HandlePostCount(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"})
}

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"

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

```
type usecase struct {
provider Provider
}

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

func (u *usecase) HandleGetCount() (int, error) {
counter, err := u.provider.GetCounter()
if err != nil {
return 0, err
}
return counter, nil
}

func (u *usecase) HandlePostCount(count int) error {
return u.provider.UpdateCounter(count)
}

func (u *usecase) HandleGetCountHTTP(c echo.Context) error {
counter, err := u.HandleGetCount()
if err != nil {
return c.JSON(http.StatusInternalServerError, map[string]string{"error": err.Error()})
}
return c.String(http.StatusOK, fmt.Sprintf("%d", counter))
}

func (u *usecase) HandlePostCountHTTP(c echo.Context) error {
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.HandlePostCount(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"})
}
```

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
        sslmode=disable",
host, port, user, password, dbName)

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

```
}

return &provider{db: conn}
}

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

func (p *provider) UpdateCounter(value int) error {
_, err := p.db.Exec("UPDATE counter SET value = value + $1", value)
return err
}
```

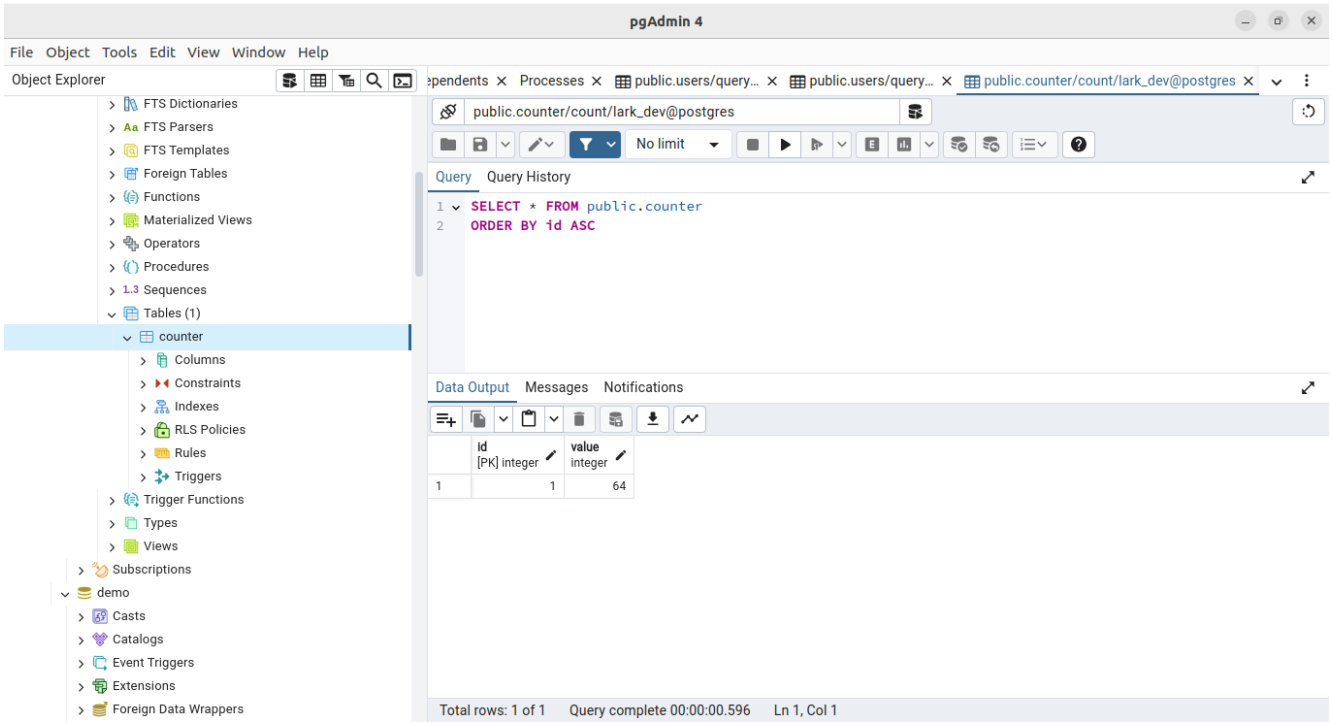

Рисунок 12 - Тестирование микросервиса count

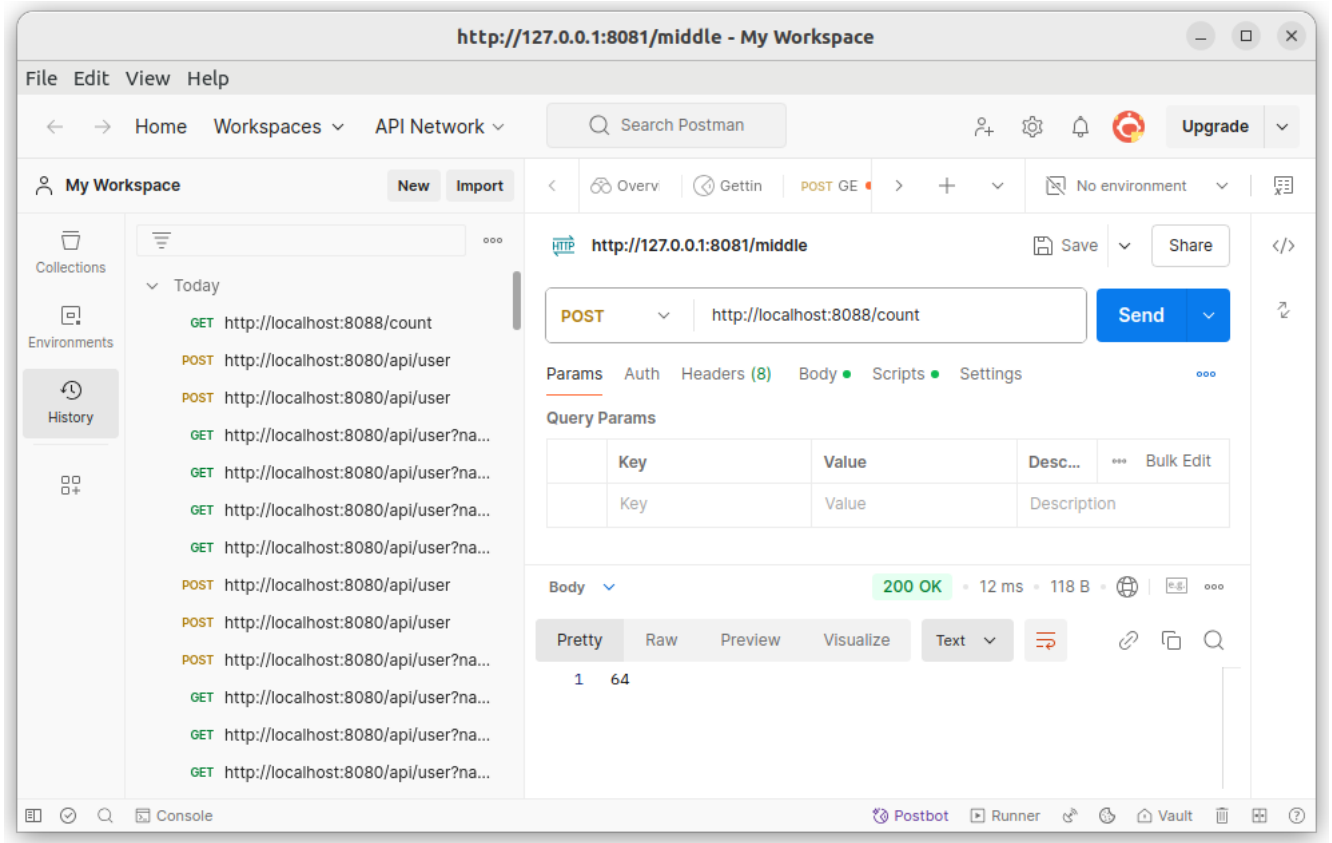

Рисунок 13 - Тестирование микросервиса count

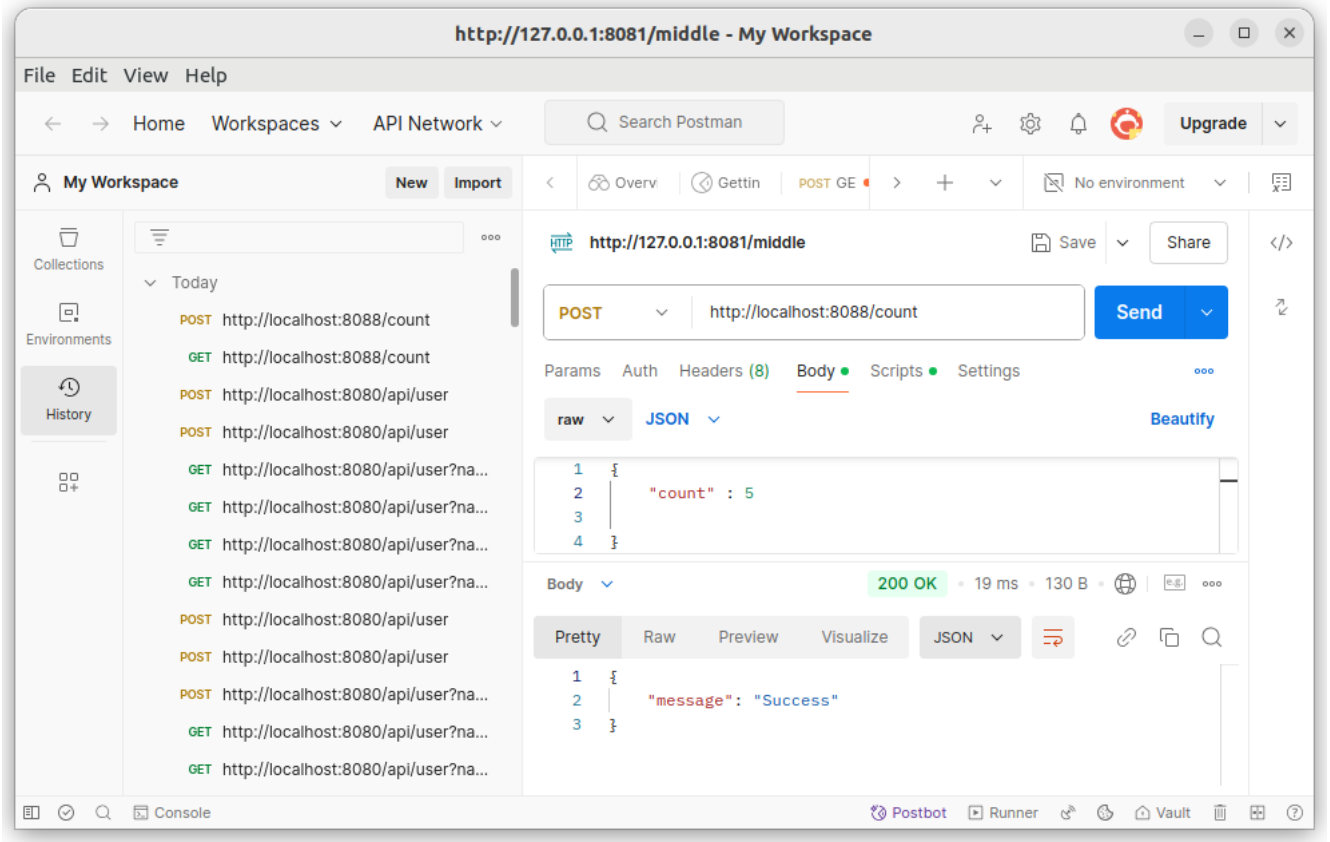

Рисунок 14 - Тестирование микросервиса count

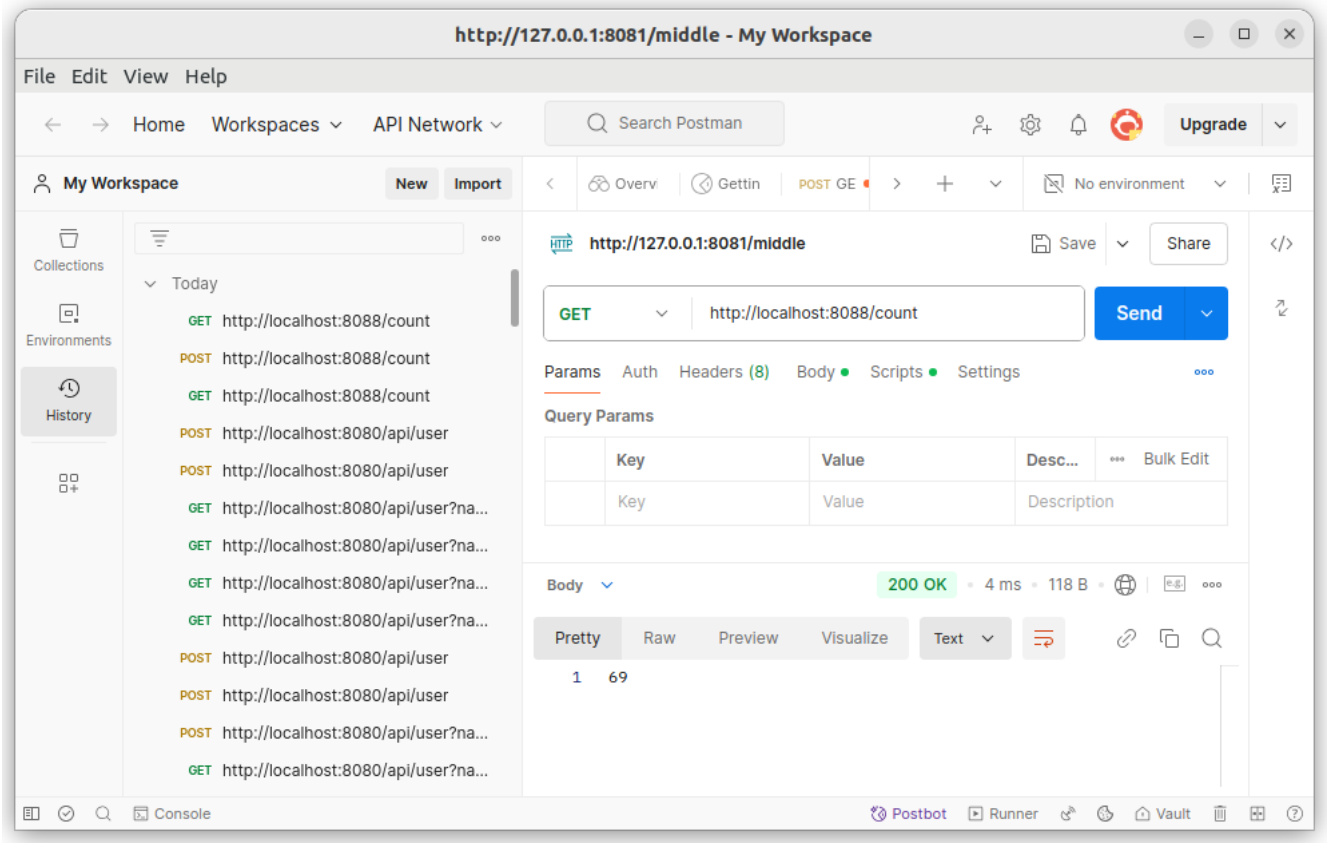

Рисунок 15 - Тестирование микросервиса count

Все изменения были зафиксированы, был сделан commit и произошла отправка в удалённый репозиторий GitHub. Через интерфейс GitHub был создан Pull Request dev --> master.

Заключение: были получены первичные навыки организации кодовой базы проекта на Golang.

Список использованных источников:

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