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 Название направления**

О Т Ч Е Т

по рубежному контролю №2

Название: Разработка Restful-сервиса на Golang

Дисциплина: Языки интернет программирования

Студент ИУ6-32Б (Группа) 17.12.2024 (Подпись, дата) Л.И. Заушников (И.О. Фамилия)

Преподаватель 17.12.2024 (Подпись, дата) В.Д. Шульман (И.О. Фамилия)

Москва, 2024

Цель работы

Разработать микросервис на golang, эмулирующий механику электронной почты

Сервис должен предоставлять следующие возможности:
- отправить письмо от имени пользователя X пользователям Y1,Y2,...
- получить список писем в ящике пользователя Y
- удалить письмо в ящике пользователя Y

Сущность "письмо" должно иметь как минимум 4 поля: тема, текст, отправитель и получатели (получателей может быть несколько)

Через файл конфигурации должен настраиваться максимальный размер текста писем, отправляемых пользователями

Ход работы

1) Для реализации микросервиса создали такую архитектуру проекта:

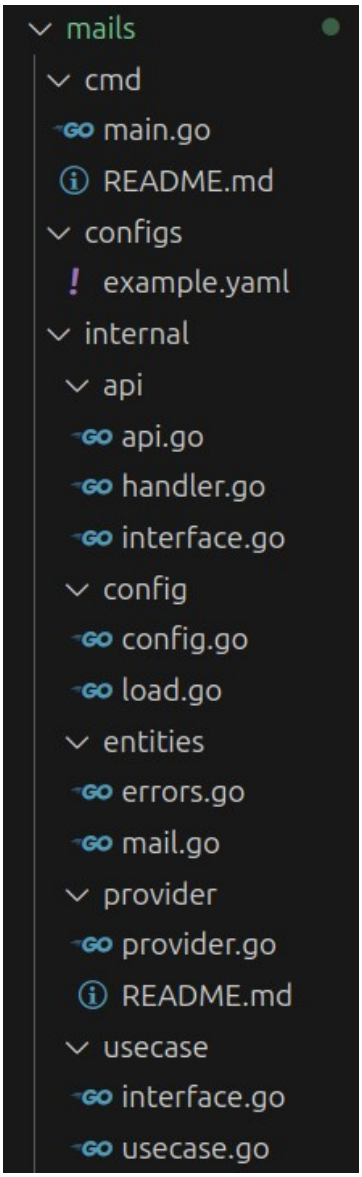

Рисунок 1 — Структура проекта

2) Создали базу данных PostgreSQL для работы сервиса:

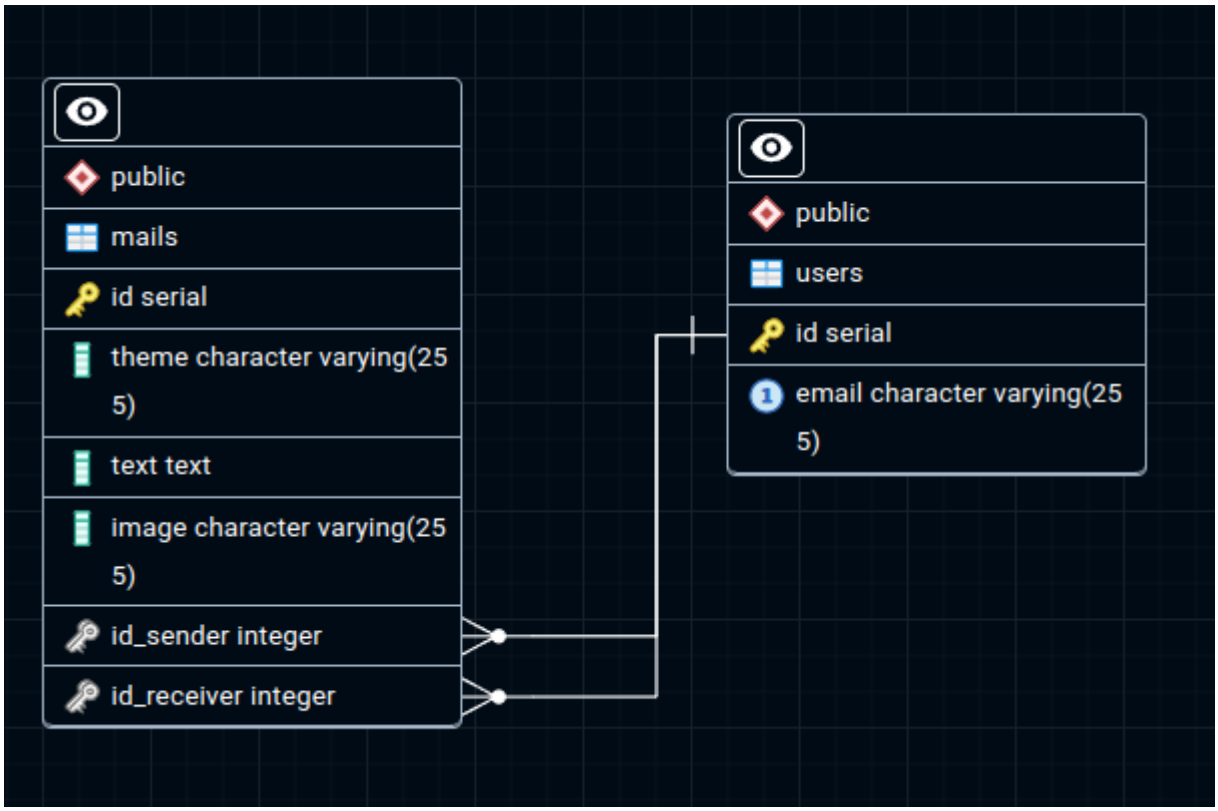

Рисунок 2 — ERD базы данных Mail

3) Реализовали наш сервис:

provider.go

```go
package provider

import (
"database/sql"
"fmt"
"log"
"mail/internal/entities"
)

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) SendMail(mail entities.Mail) error {
for _, receiverID := range mail.Receivers {
_, err := p.conn.Exec(`INSERT INTO mails (theme, text, image, id_sender,
id_receiver) VALUES ($1, $2, $3, $4, $5)`,
mail.Theme, mail.Text, mail.Image, mail.SenderID, receiverID)
if err != nil {
return err
}
}
return nil
}

func (p *Provider) GetMailsByUserID(userID int) ([]entities.Mail, error) {
mails := []entities.Mail{}
rows, err := p.conn.Query(`SELECT id, theme, text, image, id_sender FROM mails
WHERE id_receiver = $1`, userID)
if err != nil {
return nil, err
}
defer rows.Close()

for rows.Next() {
var mail entities.Mail
if err := rows.Scan(&mail.ID, &mail.Theme, &mail.Text, &mail.Image,
&mail.SenderID); err != nil {
return nil, err
}
mails = append(mails, mail)
}
return mails, nil
}

func (p *Provider) DeleteMail(mailID int) error {
_, err := p.conn.Exec(`DELETE FROM mails WHERE id = $1`, mailID)
return err
}

func (p *Provider) UserExist(userID int) bool {
var exists bool
err := p.conn.QueryRow(`SELECT EXISTS(SELECT 1 FROM users WHERE id = $1)`,
userID).Scan(&exists)
if err != nil {
log.Println("Error checking user existence:", err)
return false
}
return exists
}

func (p *Provider) MailExist(mailID int) bool {
var exists bool
```

```
err := p.conn.QueryRow(`SELECT EXISTS(SELECT 1 FROM mails WHERE id = $1)`,
mailID).Scan(&exists)
if err != nil {
log.Println("Error checking mail existence:", err)
return false
}
return exists
}
```

interface.go(для провайдера)

```
package usecase

import "mail/internal/entities"

type Provider interface {
SendMail(mail entities.Mail) error
GetMailsByUserID(userID int) ([]entities.Mail, error)
DeleteMail(mailID int) error
UserExist(userID int) bool
MailExist(mailID int) bool
}
```

usecase.go

```
package usecase

import (
"mail/internal/entities"
)

type Usecase struct {
p Provider
}

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

func (u *Usecase) SendMail(mail entities.Mail) error {
if !u.UserExists(mail.SenderID) {
return entities.ErrUserNotFound
}

for _, receiverID := range mail.Receivers {
if !u.UserExists(receiverID) {
return entities.ErrUserNotFound
}
}

return u.p.SendMail(mail)
}

func (u *Usecase) GetMailsByUserID(userID int) ([]entities.Mail, error) {
```

```
if !u.UserExists(userID) {
return nil, entities.ErrUserNotFound
}
return u.p.GetMailsByUserID(userID)
}

func (u *Usecase) DeleteMail(mailID int) error {
if !u.MailExists(mailID) {
return entities.ErrMailNotFound
}
return u.p.DeleteMail(mailID)
}

func (u *Usecase) UserExists(userID int) bool {
return u.p.UserExist(userID)
}

func (u *Usecase) MailExists(mailID int) bool {
return u.p.MailExist(mailID)
}
```

handler.go

```
package api

import (
"mail/internal/entities"
"net/http"
"strconv"

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

func (s *Server) SendMail(c echo.Context) error {
var mail entities.Mail
if err := c.Bind(&mail); err != nil {
return c.String(http.StatusBadRequest, "Invalid input")
}

if err := mail.Validate(s.cfg); err != nil {
return c.String(http.StatusBadRequest, err.Error())
}

if !s.uc.UserExists(mail.SenderID) {
return c.String(http.StatusNotFound, entities.ErrUserNotFound.Error())
}

for _, receiverID := range mail.Receivers {
if !s.uc.UserExists(receiverID) {
return c.String(http.StatusNotFound, entities.ErrUserNotFound.Error())
}
}
```

```
return c.String(http.StatusCreated, "Mail sent successfully")
}

func (s *Server) GetMails(c echo.Context) error {
userID, err := strconv.Atoi(c.Param("user_id"))
if err != nil {
return c.String(http.StatusBadRequest, "Invalid user ID")
}

if !s.uc.UserExists(userID) {
return c.String(http.StatusNotFound, entities.ErrUserNotFound.Error())
}

mails, err := s.uc.GetMailsByUserID(userID)
if err != nil {
return c.String(http.StatusInternalServerError, err.Error())
}

return c.JSON(http.StatusOK, mails)
}

func (s *Server) DeleteMail(c echo.Context) error {
mailID, err := strconv.Atoi(c.Param("id"))
if err != nil {
return c.String(http.StatusBadRequest, "Invalid mail ID")
}

if !s.uc.MailExists(mailID) {
return c.String(http.StatusNotFound, entities.ErrMailNotFound.Error())
}

if err := s.uc.DeleteMail(mailID); err != nil {
return c.String(http.StatusInternalServerError, err.Error())
}

return c.String(http.StatusOK, "Mail deleted successfully")
}
```

interface.go(для usecase)

```
package api

import "mail/internal/entities"

type Usecase interface {
SendMail(mail entities.Mail) error
GetMailsByUserID(userID int) ([]entities.Mail, error)
DeleteMail(mailID int) error
UserExists(userID int) bool
MailExists(mailID int) bool
}
```

api.go

```
package api

import (
"mail/internal/config"
"strconv"

"github.com/go-playground/validator/v10"
"github.com/labstack/echo/v4"
)

type Server struct {
server *echo.Echo
address string
uc Usecase
validate *validator.Validate
cfg *config.Config
}

func NewServer(ip string, port int, userUc Usecase, cfg *config.Config)
*Server {
api := Server{
uc: userUc,
validate: validator.New(),
cfg: cfg,
}

api.server = echo.New()
api.server.POST("/mails", api.SendMail)
api.server.GET("/mails/:user_id", api.GetMails)
api.server.DELETE("/mails/:id", api.DeleteMail)

api.address = ip + ":" + strconv.Itoa(port)

return &api
}

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

Прописали конфигурацию для настройки ограничений для полей(максимальная и минимальная длина строки)

example.yaml

```
ip: "localhost"
port: 8089
db:
host: "localhost"
port: 5432
user: "postgres"
password: "postgres"
```

```
dbname: "mail"
mail_theme_min_len: 3
mail_theme_max_len: 1000
mail_text_min_len: 4
mail_text_max_len: 1000
```

errors.go

```
package entities

import (
"errors"
"mail/internal/config"
"strconv"
)

var (
ErrMessageLengthInvalid = errors.New("message length must be between 5 and
1000 characters")
ErrInvalidEmailFormat = errors.New("invalid email format")
ErrMailNotFound = errors.New("mail not found")
ErrUserNotFound = errors.New("user not found")
)

func (m *Mail) Validate(cfg *config.Config) error {
if len(m.Theme) < cfg.MailThemeMinLen || len(m.Theme) > cfg.MailThemeMaxLen {
return errors.New("theme length must be between " +
strconv.Itoa(cfg.MailThemeMinLen) + " and " +
strconv.Itoa(cfg.MailThemeMaxLen) + " characters")
}
if len(m.Text) < cfg.MailTextMinLen || len(m.Text) > cfg.MailTextMaxLen {
return errors.New("text length must be between " +
strconv.Itoa(cfg.MailTextMinLen) + " and " + strconv.Itoa(cfg.MailTextMaxLen)
+ " characters")
}
if m.SenderID <= 0 {
return ErrUserNotFound
}
if len(m.Receivers) == 0 {
return ErrUserNotFound
}
return nil
}
```

mail.go

```
package entities

type Mail struct {
ID int `json:"id,omitempty"`
Theme string `json:"theme"`
Text string `json:"text"`
Image string `json:"image,omitempty"`
SenderID int `json:"sender_id"`
```

```
Receivers []int `json:"receivers"`
}
```

main.go

```
package main

import (
"flag"
"log"
"mail/internal/api"
"mail/internal/config"
"mail/internal/provider"
"mail/internal/usecase"

_ "github.com/lib/pq"
)

func main() {
configPath := flag.String("config-path", "../configs/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)
uc := usecase.NewUsecase(prv)
srv := api.NewServer(cfg.IP, cfg.Port, uc, cfg)
srv.Run()

}
```

4) Протестировали программу(рис. 3-6). Проверили работу исключений(рис. 7-9)

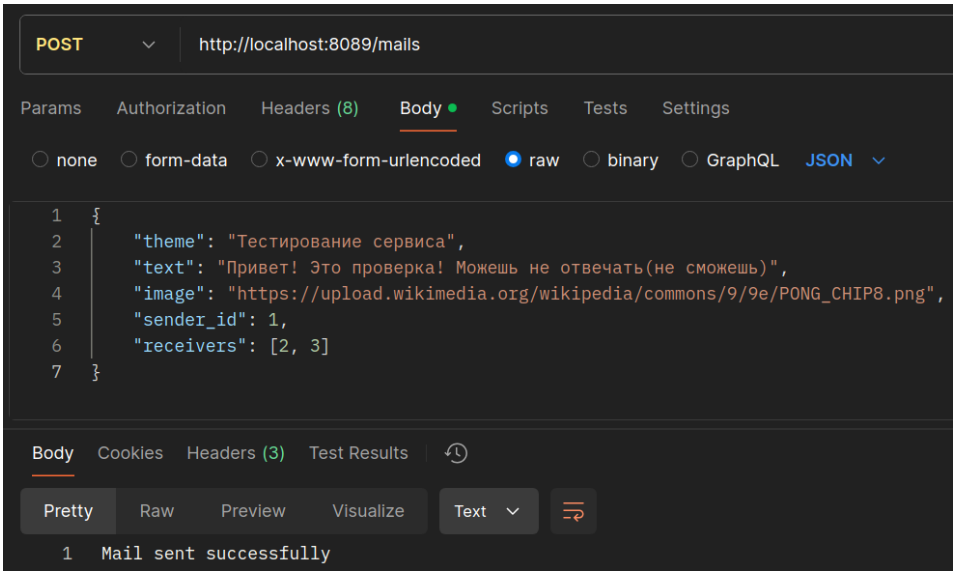

Рисунок 3 — Отправка писем(Post запрос)

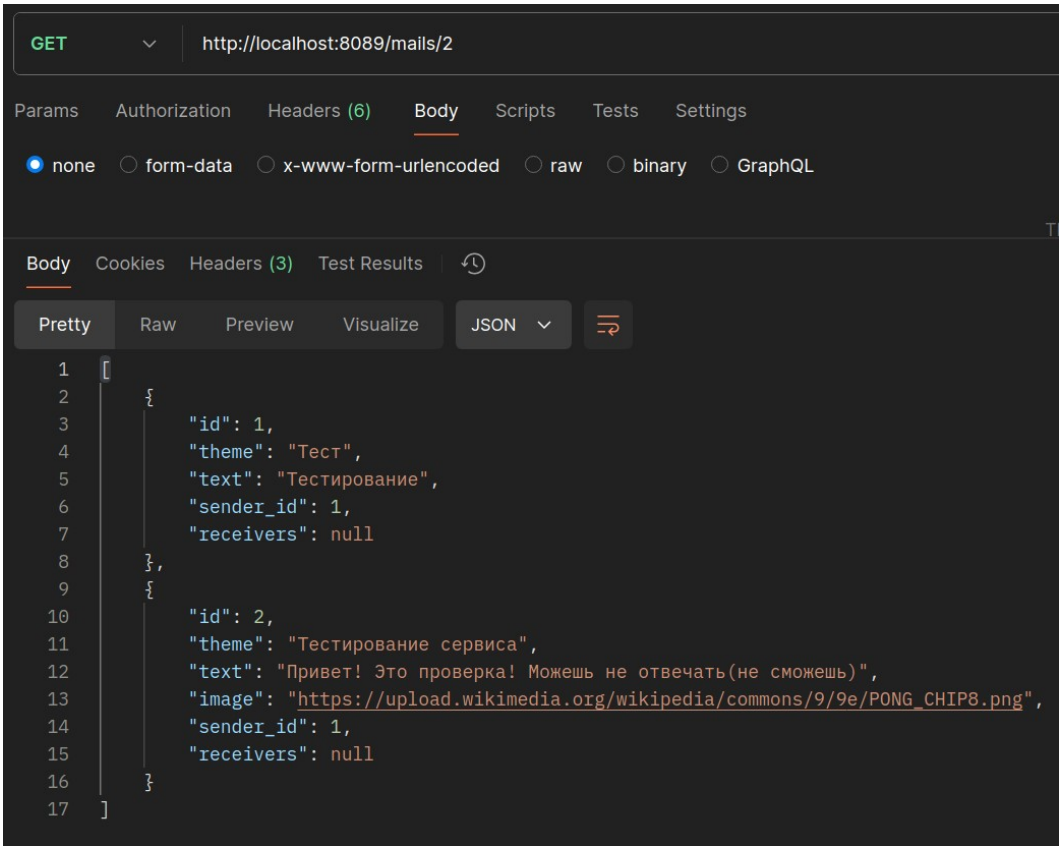

Рисунок 4 — Проверка почтового ящика(Get запрос)

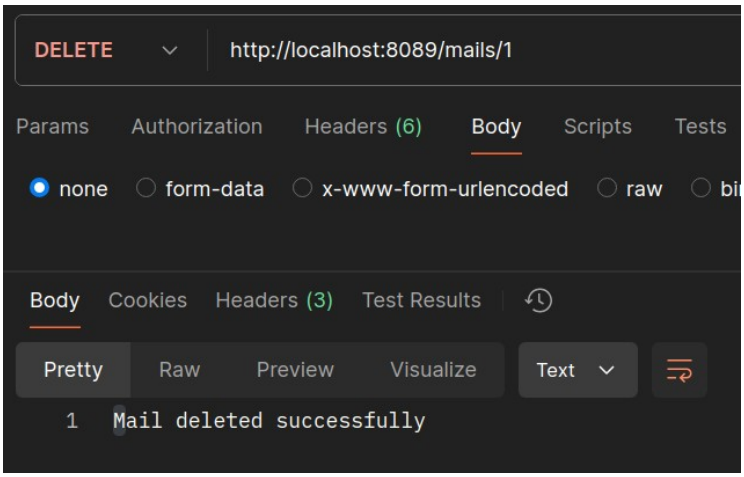

Рисунок 5 — Удаление письма

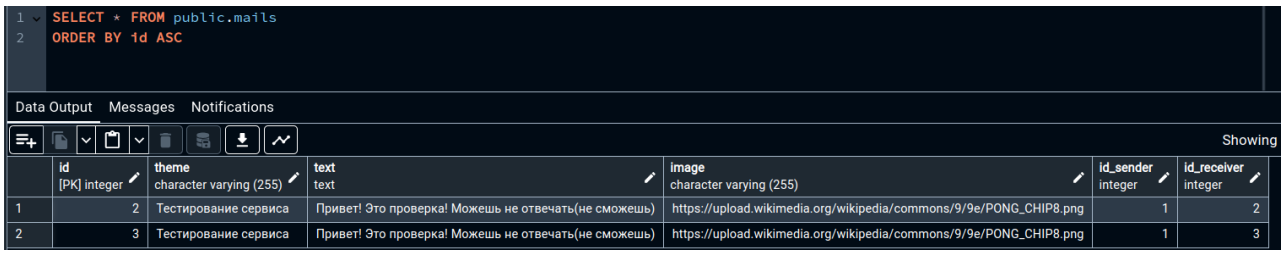

Рисунок 6 — Итоговое состояние таблицы

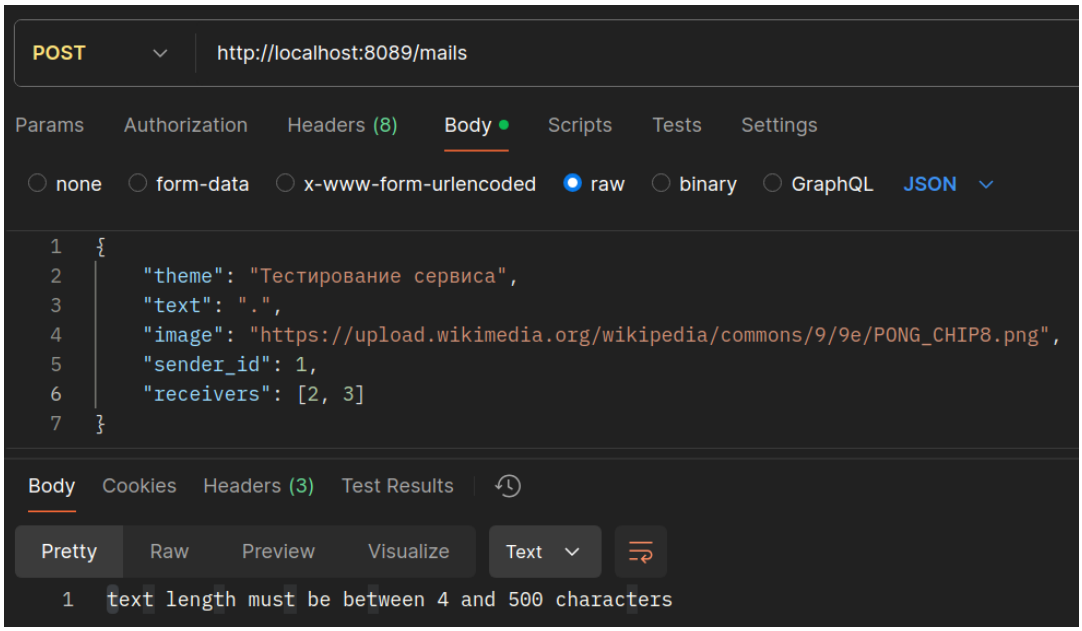

Рисунок 7 — Проверка конфигурации(минимальное сообщение)

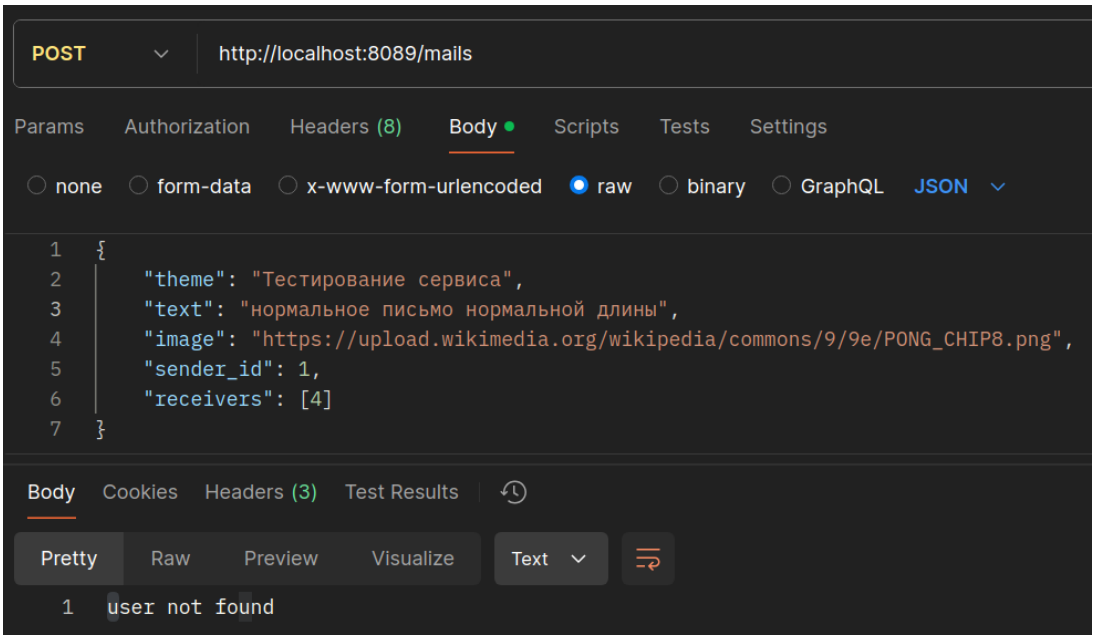

Рисунок 8 — Проверка пользователя на существование

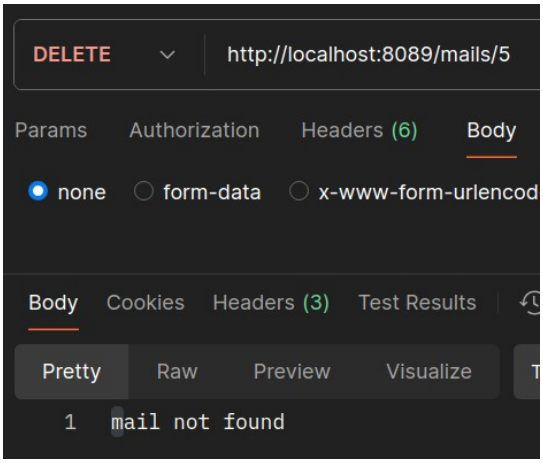

Рисунок 9 — Проверка существования письма

Вывод: В ходе выполнения рубежного контроля по разработке RESTful-сервиса на языке Golang была успешно реализовали функциональность микросервиса для хранения и управления письмами.