|  |  |
| --- | --- |
| Gerb-BMSTU_01 | **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 № 11**

**Название:** Golang & JWT

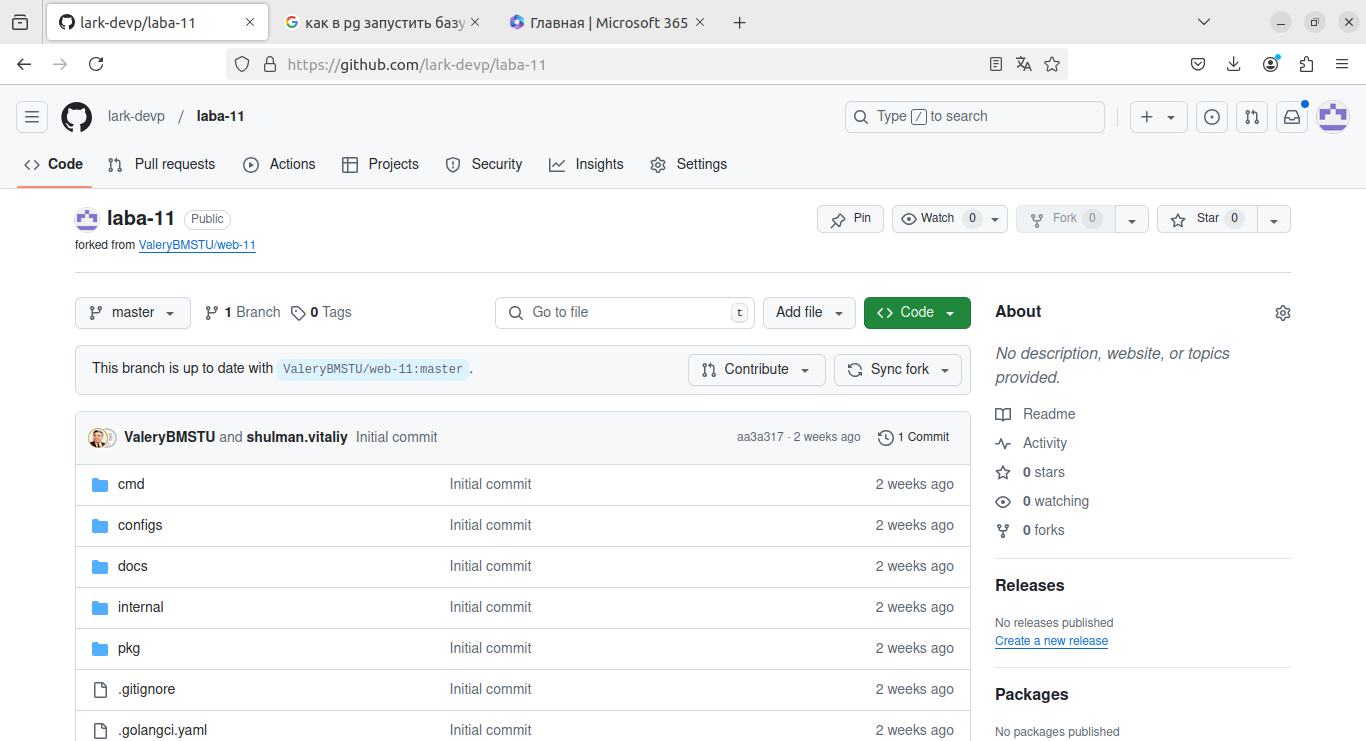
**Дисциплина:** Языки интернет-программирования

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 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 знаний в области авторизации и аутентификации в контексте веб-приложений.

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. Был реализован новый микросервис для аутенфикации пользователей auth с выдачей jwt-токена.

**main.go**

package main

import (

"flag"

"log"

"web-11/internal/auth/api"

"web-11/internal/auth/config"

"web-11/internal/auth/provider"

"web-11/internal/auth/usecase"

\_ "github.com/lib/pq"

)

func main() {

configPath := flag.String("config-path", "../../configs/auth\_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)

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

if err := srv.Router.Start(srv.Address); err != nil {

log.Fatal(err)

}

}

**api.go**

package api

import (

"fmt"

"net/http"

"time"

"web-11/internal/auth/usecase"

"github.com/dgrijalva/jwt-go"

"github.com/labstack/echo/v4"

)

var jwtSecret = []byte("123.456.789")

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.POST("/auth/register", srv.Register)

srv.Router.POST("/auth/login", srv.Login)

srv.Router.GET("/protected-route", srv.JWTMiddleware(srv.ProtectedRoute))

return srv

}

// GenerateJWT создает новый JWT-токен для указанного пользователя

func GenerateJWT(username string) (string, error) {

token := jwt.NewWithClaims(jwt.SigningMethodHS256, jwt.MapClaims{

"username": username,

"exp": time.Now().Add(time.Hour \* 72).Unix(), // Токен будет действовать 72 часа

})

tokenString, err := token.SignedString(jwtSecret)

if err != nil {

return "", err

}

return tokenString, nil

}

func (srv \*Server) JWTMiddleware(next echo.HandlerFunc) echo.HandlerFunc {

return func(c echo.Context) error {

tokenString := c.Request().Header.Get("Authorization")

if tokenString == "" {

return c.JSON(http.StatusUnauthorized, map[string]string{"error": "Token is required"})

}

token, err := jwt.Parse(tokenString, func(token \*jwt.Token) (interface{}, error) {

if \_, ok := token.Method.(\*jwt.SigningMethodHMAC); !ok {

return nil, echo.ErrUnauthorized

}

return jwtSecret, nil

})

if err != nil || !token.Valid {

return c.JSON(http.StatusUnauthorized, map[string]string{"error": "Invalid token"})

}

return next(c)

}

}

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

var input struct {

Username string `json:"username"`

Password string `json:"password"`

}

if err := c.Bind(&input); err != nil {

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

}

// Регистрируем пользователя

err := srv.uc.Register(input.Username, input.Password)

if err != nil {

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

}

// Генерируем JWT-токен для нового пользователя

token, err := GenerateJWT(input.Username) // Генерация токена

if err != nil {

return c.JSON(http.StatusInternalServerError, map[string]string{"error": "Failed to generate token"})

}

return c.JSON(http.StatusCreated, map[string]string{"message": "User registered successfully", "token": token}) // Возвращаем токен

}

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

var input struct {

Username string `json:"username"`

Password string `json:"password"`

}

if err := c.Bind(&input); err != nil {

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

}

token, err := srv.uc.Login(input.Username, input.Password)

if err != nil {

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

}

return c.JSON(http.StatusOK, map[string]string{"token": token})

}

// Пример защищенного маршрута

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

return c.JSON(http.StatusOK, map[string]string{"message": "This is a protected route!"})

}

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) GetUser(username string) (string, error) {

var existingUser string

err := p.db.QueryRow("SELECT username FROM users WHERE username = $1", username).Scan(&existingUser)

if err != nil {

return "", err

}

return existingUser, nil

}

func (p \*Provider) CreateUser(username, password string) error {

\_, err := p.db.Exec("INSERT INTO users (username, password) VALUES ($1, $2)", username, password)

return err

}

**usecase.go**

package usecase

import (

"time"

"github.com/dgrijalva/jwt-go"

)

type Usecase struct {

provider Provider

}

var jwtSecret = []byte("123.456.789")

func NewUsecase(prv Provider) \*Usecase {

return &Usecase{

provider: prv,

}

}

func GenerateJWT(username string) (string, error) {

token := jwt.NewWithClaims(jwt.SigningMethodHS256, jwt.MapClaims{

"username": username,

"exp": time.Now().Add(time.Hour \* 72).Unix(), // Токен будет действовать 72 часа

})

tokenString, err := token.SignedString(jwtSecret)

if err != nil {

return "", err

}

return tokenString, nil

}

func (uc \*Usecase) Register(username, password string) error {

return uc.provider.CreateUser(username, password)

}

func (uc \*Usecase) Login(username, password string) (string, error) {

username, err := uc.provider.GetUser(username)

if err != nil {

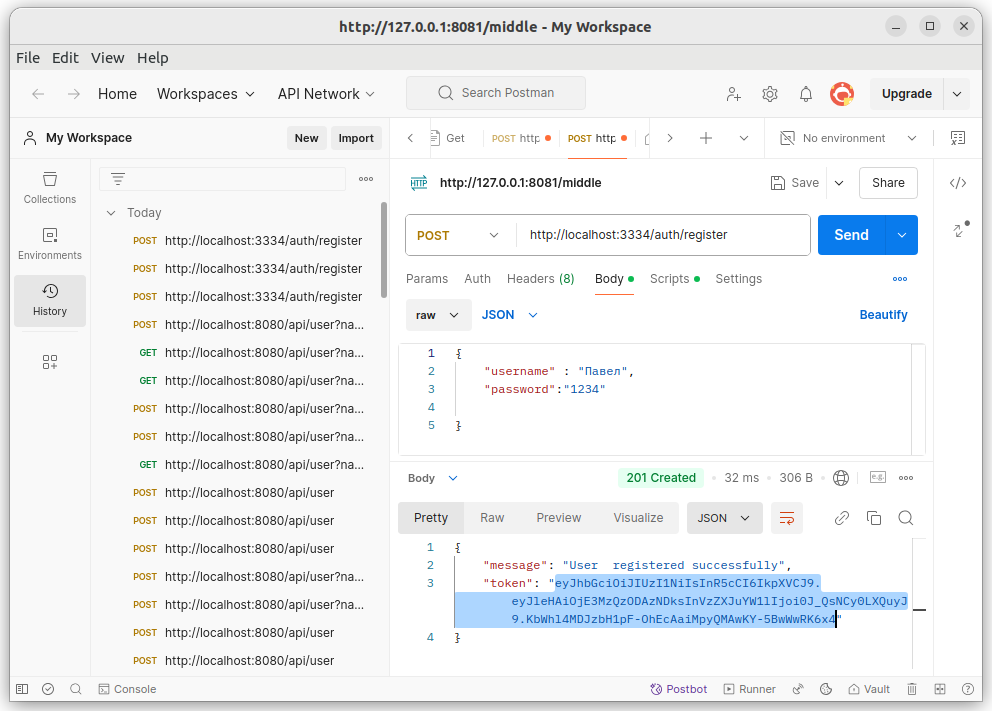
return "", err // Возвращаем ошибку, если пользователь не найден

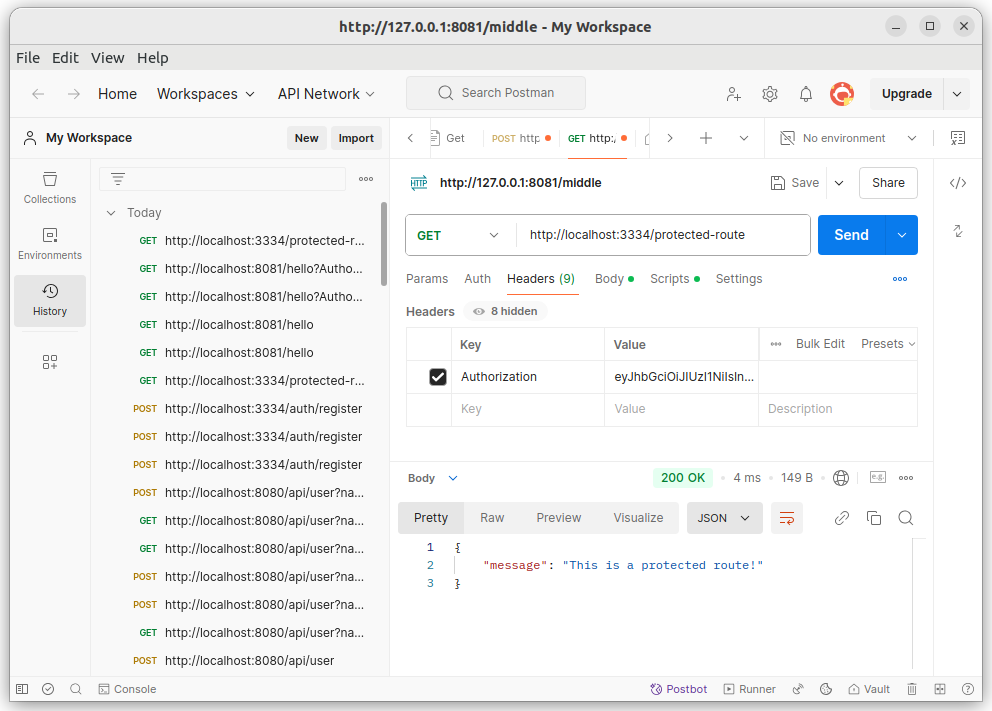
}

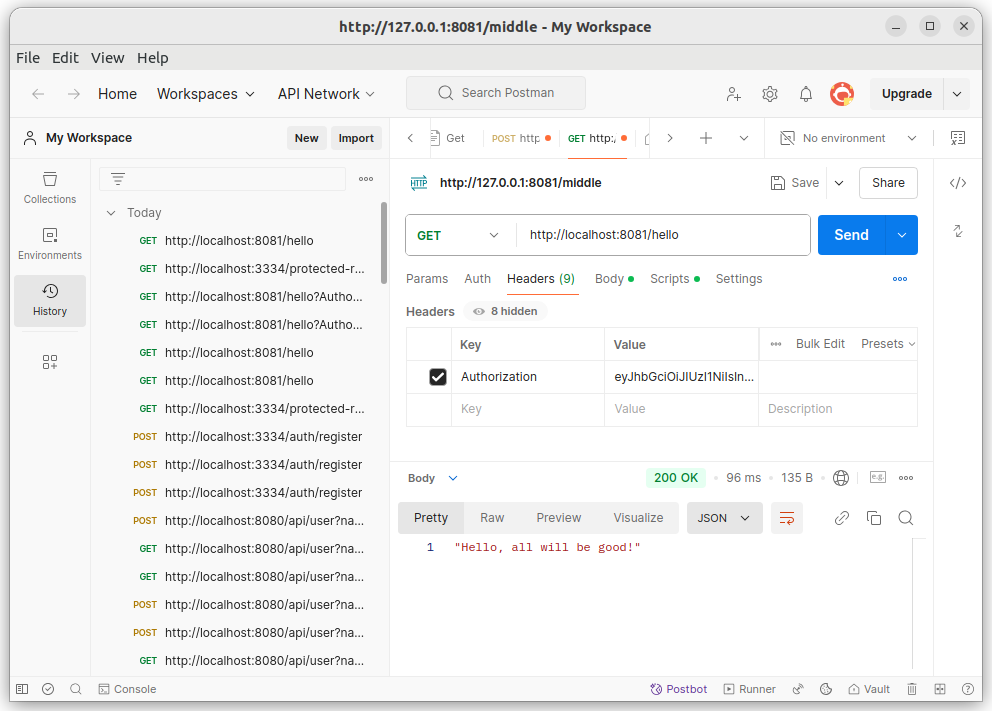
return GenerateJWT(username) // Генерируем и возвращаем JWT

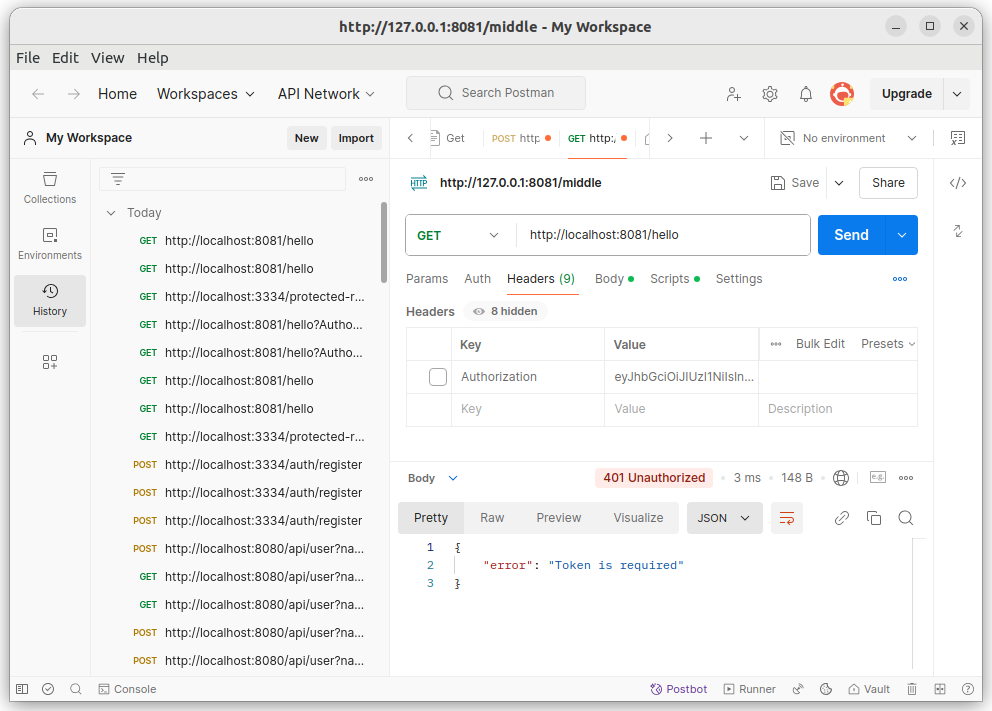
}

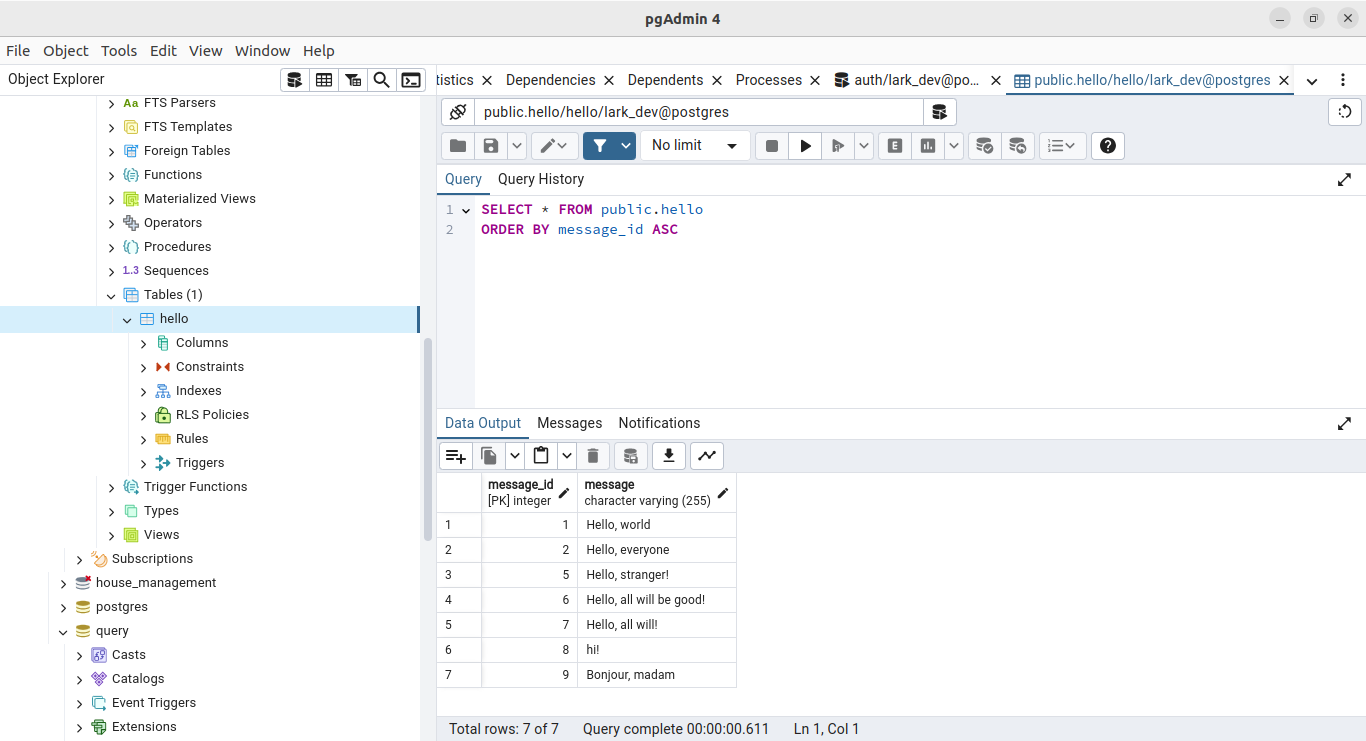
Микросервисы были протестированы с учётом добавления аутентификации, результаты тестирования представлены на рисунках 2-6.

Рисунок 2 - Тестирование микросервисов с аутентификацией

Рисунок 3 - Тестирование микросервисов с аутентификацией

Рисунок 4 - Тестирование микросервисов с аутентификацией

Рисунок 5 - Тестирование микросервисов с аутентификацией

Рисунок 6 - Тестирование микросервисов с аутентификацией

1. 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 знаний в области авторизации и аутентификации в контексте веб-приложений.

**Список использованных источников:**

1. <https://ru.hexlet.io/courses/go-web-development/lessons/auth/theory_unit>

2. https://echo.labstack.com/docs/cookbook/jwt