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 Компьютерные системы и сети**

О Т Ч Е Т

по лабораторной работе № 11

Название: Аутентификация пользователей с помощью jwt-токена

Дисциплина: Языки интернет-программирования

Студент	ИУ6-32Б	02.12.2024	Т.А. Гаджиев
	(Группа)	(Подпись, дата)	(И.О. Фамилия)
Преподаватель		02.12.2024	В.Д. Шульман
		(Подпись, дата)	(И.О. Фамилия)

Москва, 2024

Цель работы — получение первичных знаний в области авторизации и аутентификации в контексте веб-приложений.

Ход работы.

1. Сделали форк данного репозитория в GitHub, склонировали получившуюся копию локально, создали от мастера ветку дев и переключились на нее. Скопировали наши микросервисы из 8-ой лабораторной работы:

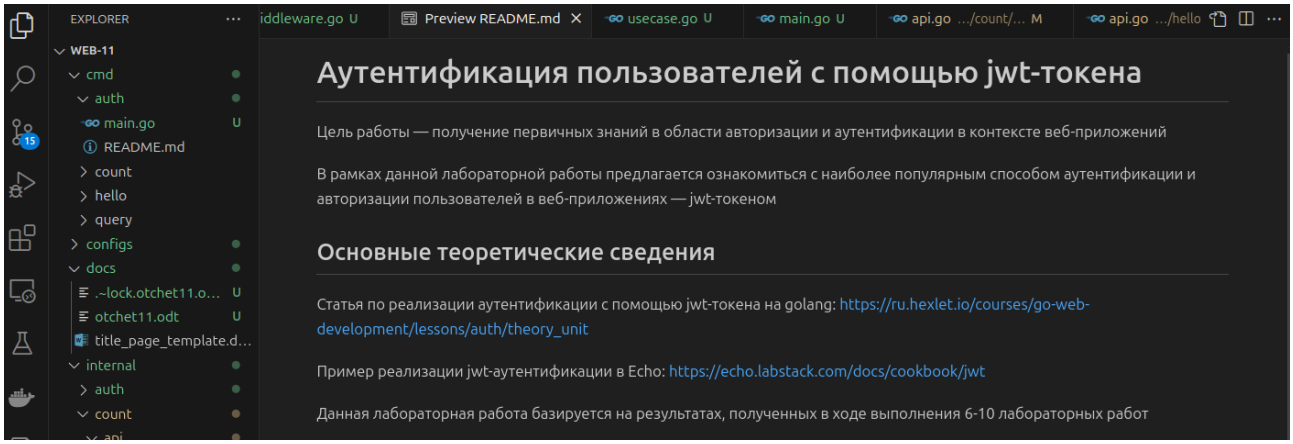

Рисунок 1 — Скопированный репозиторий

2. Реализовали новый микросервис для аутентификации 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"
)

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"
"web-11/internal/auth/usecase"

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

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)

return srv
}

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()})
}

err := srv.uc.Register(input.Username, input.Password)
if err != nil {
return c.JSON(http.StatusInternalServerError, map[string]string{"error": err.Error()})
}

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

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})
}
```

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
}
```

middleware.go

```
package middleware

import (
"net/http"
"strings"
"time"

"github.com/dgrijalva/jwt-go"
"github.com/labstack/echo/v4"
)

var jwtSecret = []byte("your_secret_key")

func GenerateJWT(username string) (string, error) {
token := jwt.NewWithClaims(jwt.SigningMethodHS256, jwt.MapClaims{
"username": username,
"exp": time.Now().Add(time.Hour * 72).Unix(),
})

tokenString, err := token.SignedString(jwtSecret)
if err != nil {
return "", err
}

return tokenString, nil
}

func 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"})
}

tokenString = strings.TrimPrefix(tokenString, "Bearer ")

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)
}
}
```

usecase.go

```
package usecase

import (
"web-11/internal/auth/middleware"
"web-11/internal/auth/provider"
)

type Usecase struct {
provider *provider.Provider
}

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

func (uc *Usecase) Register(username, password string) error {
return uc.provider.CreateUser(username, password)
}

func (uc *Usecase) Login(username, password string) (string, error) {
existingUser, err := uc.provider.GetUser(username)
if err != nil {
return "", err
}
return middleware.GenerateJWT(existingUser)
}
```

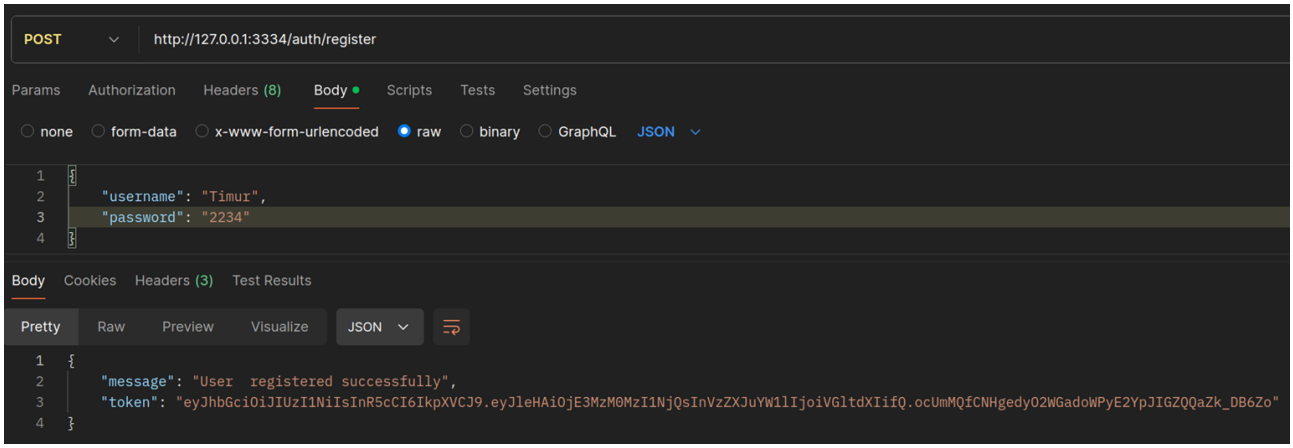

Рисунок 2 — Получение jwt-токена

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJleHAiOjE3MzM0MzI1NjQsInVzZXJuYW1lIjoiVGltdXIifQ.ocUmMQfCNHgedyO2WGadoWPyE2YpJIGZQQaZk_DB6Zo - токен

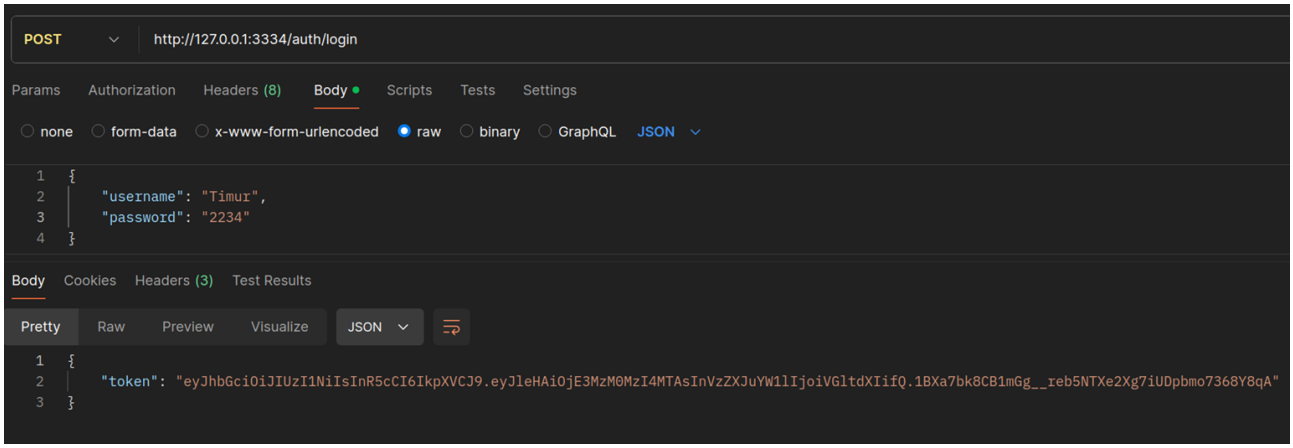

Рисунок 3 — Получение jwt-токена при входе

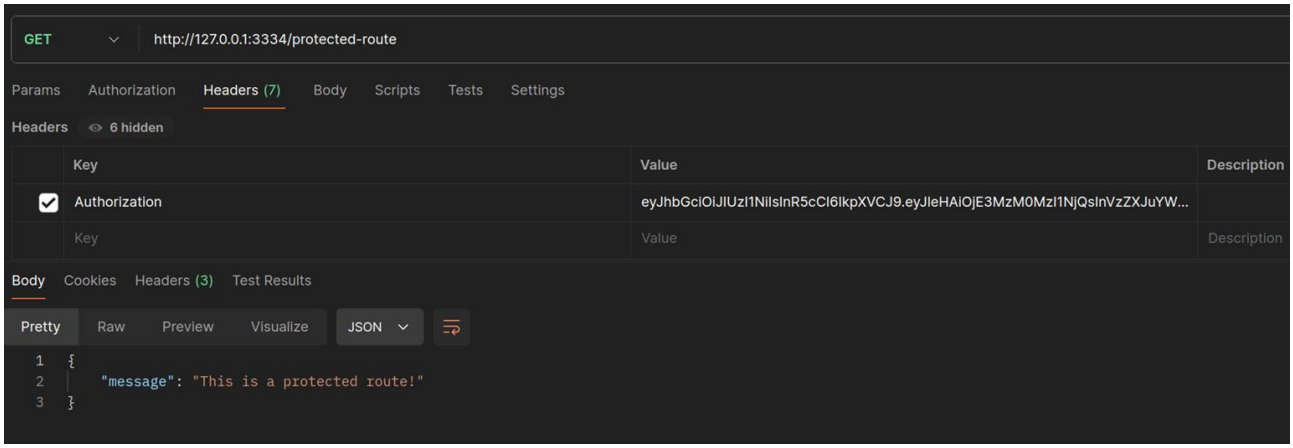

Рисунок 4 — Подключение к маршруту

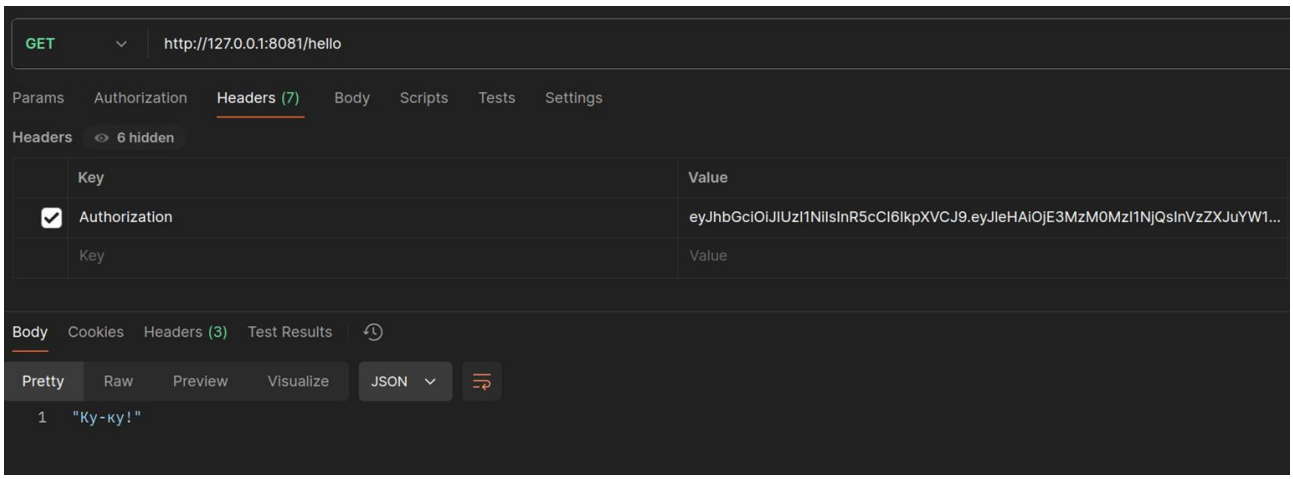

Рисунок 5 — Проверка сервиса hello

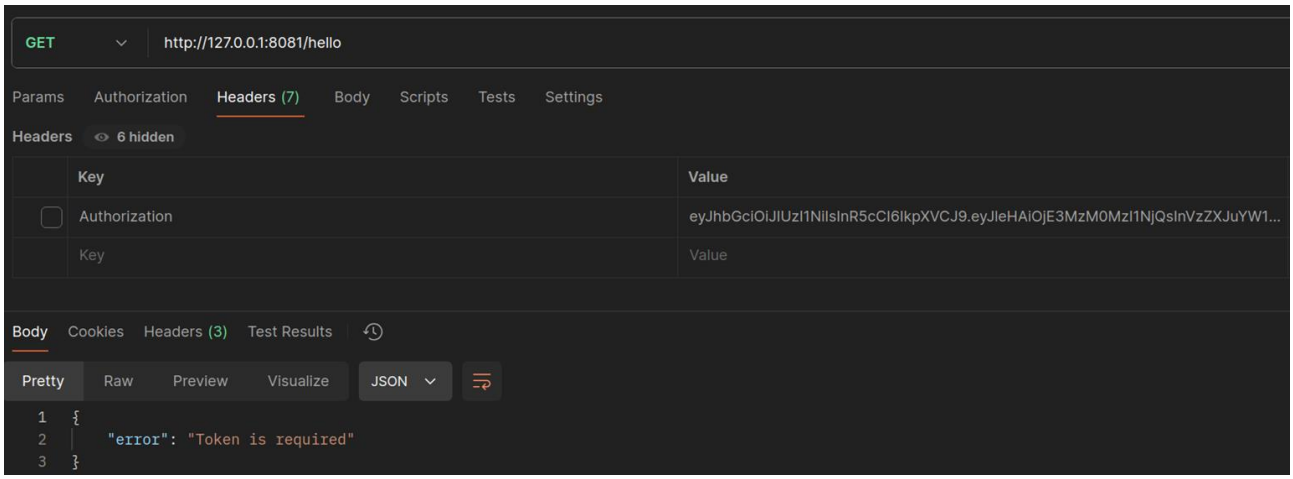

Рисунок 6 — Проверка сервиса hello без токена

3. Зафиксировали изменения, сделали коммит и отправили полученное состояние ветки дев в удаленный репозиторий GitHub. Через интерфейс GitHub создали Pull Request dev --> master

Заключение: в ходе лабораторной работы получили первичные навыки в области авторизации и аутентификации в контексте веб-приложений.