МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение
высшего образования
«МИРЭА – Российский технологический университет»

РТУ МИРЭА

Институт информационных технологий (ИИТ)

Кафедра практической и прикладной информатики(ППИ)

ОТЧЕТ ПО ПРАКТИЧЕСКОЙ РАБОТЕ

по дисциплине «Микросервисная архитектура»

Практическое задание № 9

Студент группы *ИКБО-25-20, Валиков Кирилл Юрьевич* _____ подпись)

Преподаватель *Лаптев И.А.*

_____ подпись)

Отчет представлен _____202__г.

1 Keycloak

```yaml
server:
  port: 9000
spring:
  application:
    name: api_gateway
  cloud:
    gateway:
      discovery:
        locator:
          enabled: true
          lowerCaseServiceId: true
      routes:
        - id: users
          uri: lb://USERSERVICE
          predicates:
            - Path=/users/**
        - id: appeals
          uri: lb://APPEALSERVICE
          predicates:
            - Path=/appeals/**
  security:
    oauth2:
      resourceserver:
        jwt:
          issuer-uri: http://localhost:8080/realms/spring-microservice-realm
          jwk-set-uri: http://localhost:8080/realms/spring-microservice-realm/protocol/openid-connect/certs
eureka:
  client:
    enabled: true
    service-url:
      defaultZone: http://localhost:8761/eureka/

logging:
  level:
    org.springframework.security: TRACE
```

Рисунок 1. Изменения в application.yml

```java
package com.example.apigateway.config;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.security.config.Customizer;
import org.springframework.security.config.annotation.web.reactive.EnableWebFluxSecurity;
import org.springframework.security.config.web.server.ServerHttpSecurity;
import org.springframework.security.web.server.SecurityWebFilterChain;


no usages
@Configuration
@EnableWebFluxSecurity
public class SecurityConfig {

    no usages
    @Bean
    public SecurityWebFilterChain securityWebFilterChain(ServerHttpSecurity serverHttpSecurity) {
        return serverHttpSecurity.csrf(ServerHttpSecurity.CsrfSpec::disable)
                .authorizeExchange(exchange -> exchange.pathMatchers( ...antPatterns: "/eureka/**") Access
                        .permitAll() AuthorizeExchangeSpec
                        .anyExchange().authenticated()
                ).oauth2ResourceServer((oauth) -> oauth
                        .jwt(Customizer.withDefaults()))
                .build();
    }
}
```

Рисунок 2. SecurityConfig

```yaml
api-gateway:
  container_name: api-gateway
  image: windflaw8/api-gateway
  build:
    context: ./api-gateway
    dockerfile: Dockerfile
  depends_on:
    - eureka-server
  ports:
    - 9000:9000
  environment:
    - SPRING_APPLICATION_NAME=api_gateway
    - SPRING_SECURITY_OAUTH2_RESOURCESERVER_JWT_ISSUER-URI=http://keycloak:8080/realms/spring-microservice-realm
    - SPRING_SECURITY_OAUTH2_RESOURCESERVER_JWT_JWK-SET-URI=http://keycloak:8080/realms/spring-microservice-realm/protocol/openid-connect/certs
    - eureka.client.serviceUrl.defaultZone=http://eureka-server:8761/eureka/
keycloak:
  container_name: keycloak
  image: quay.io/keycloak/keycloak:latest
  hostname: keycloak
  ports:
    - 8080:8080
  environment:
    - KEYCLOAK_ADMIN=admin
    - KEYCLOAK_ADMIN_PASSWORD=admin
  command:
    - start-dev
    - --import-realm
  volumes:
    - ./keycloak-realm/microservice-auth.json:/opt/keycloak/data/import/microservice-auth.json

```

Рисунок 3. KeyCloak в docker-compose

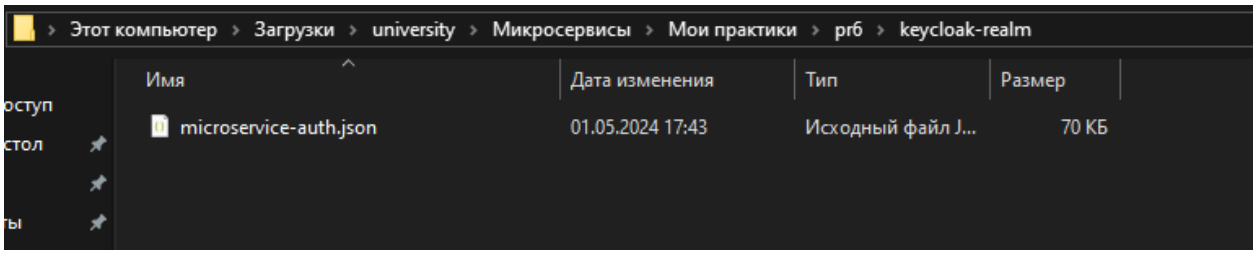

Рисунок 4. Настроенный reaml для импорта

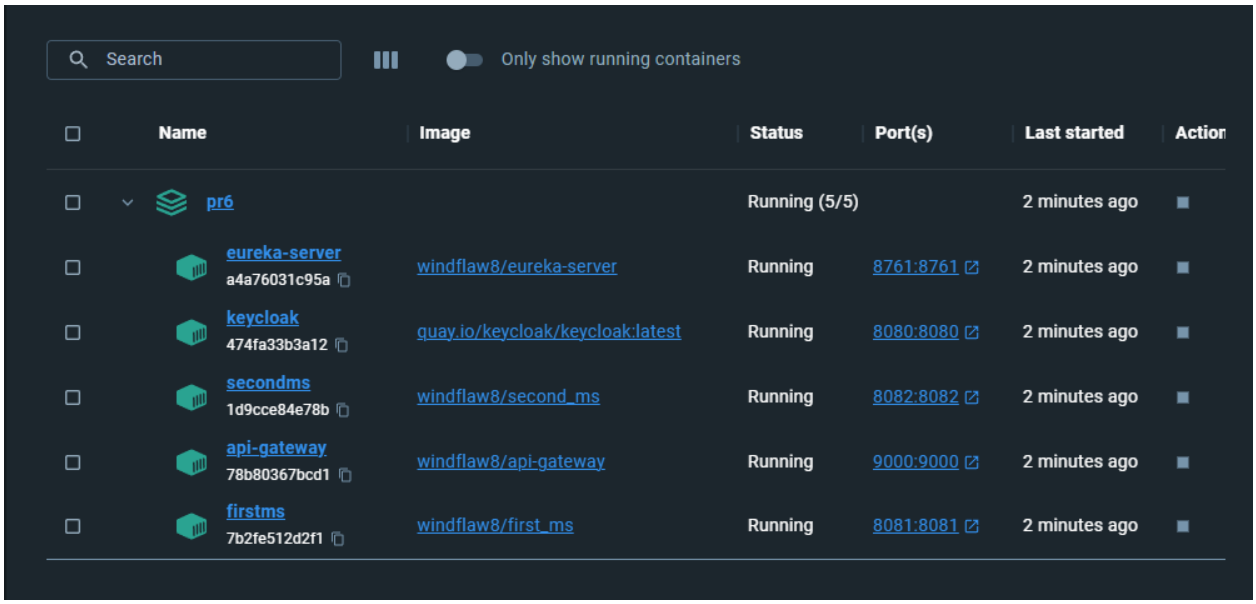

Рисунок 5. Запущенные контейнеры

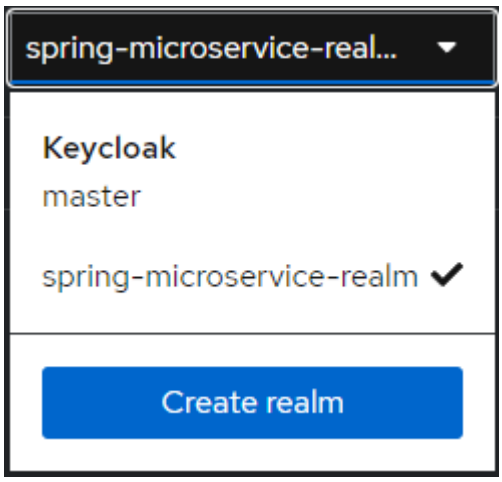

Рисунок 6. Созданный reaml

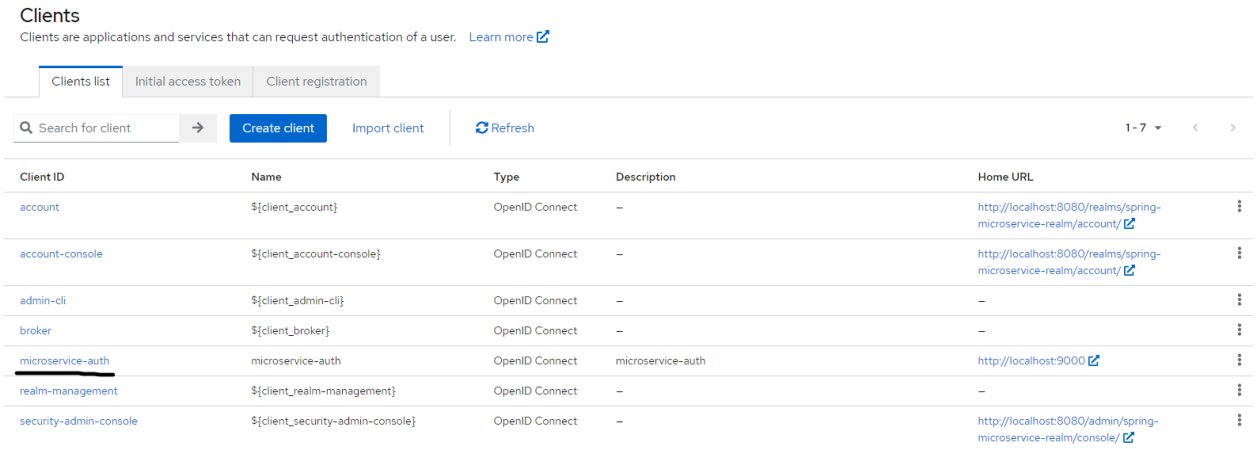

Рисунок 7. Созданный client

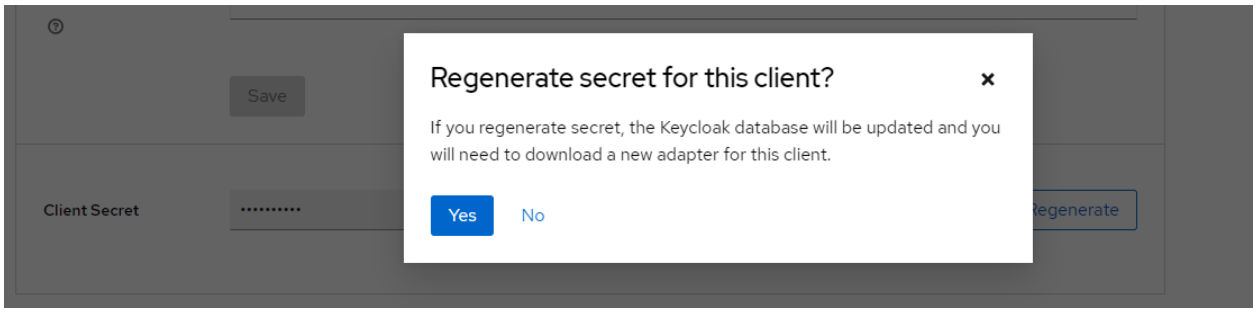

Рисунок 8. Генерируем client secret

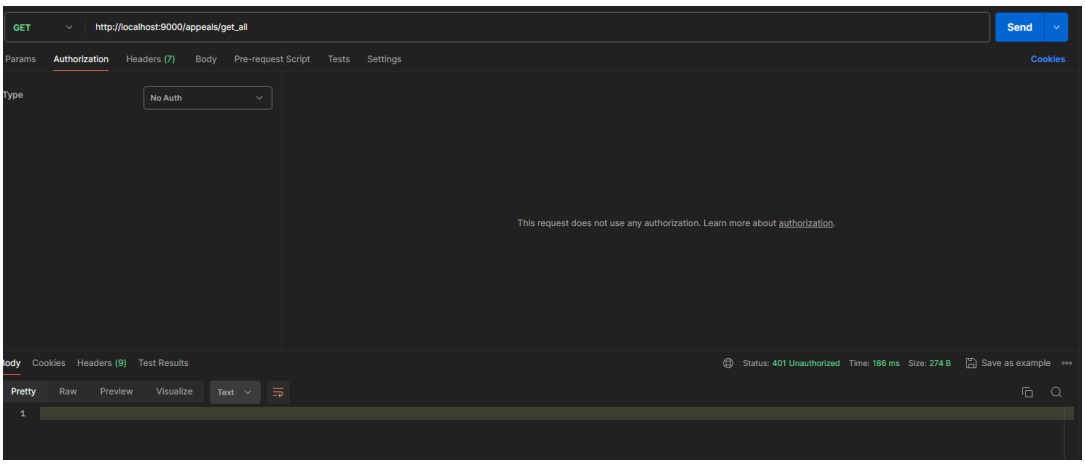

Рисунок 9. Запрос без токена к api-gateway

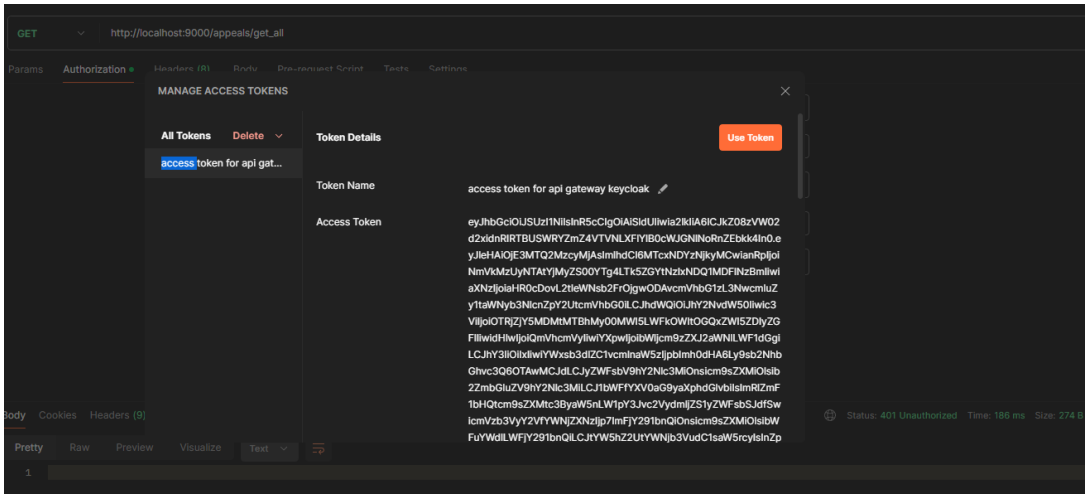

Рисунок 10. Созданный token в postman

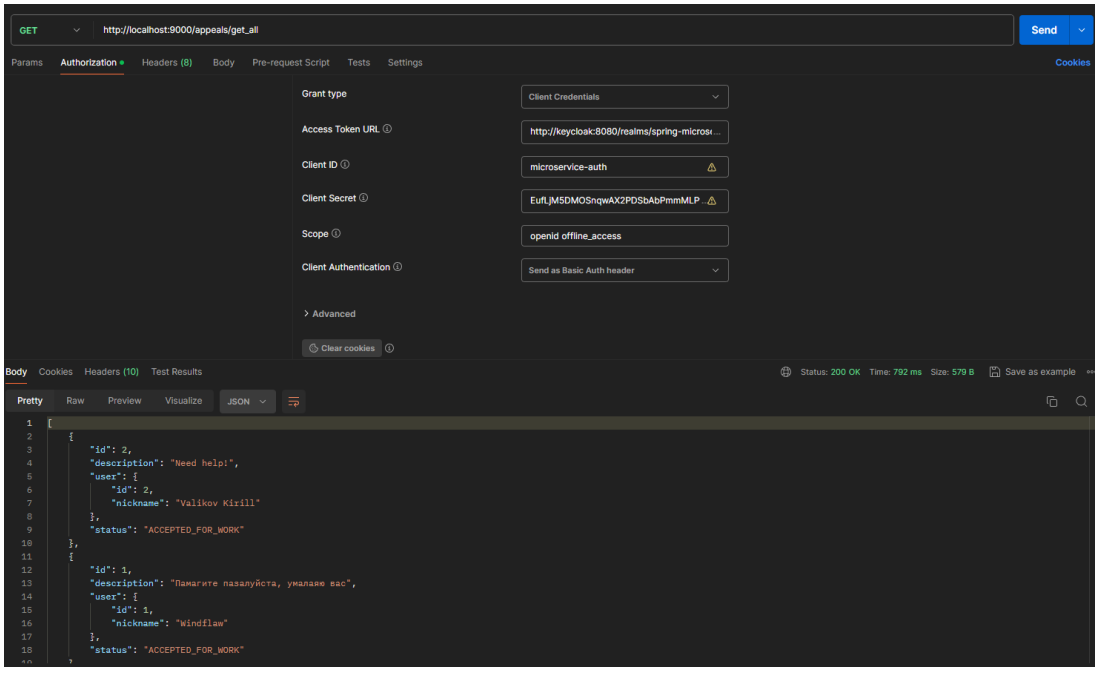

Рисунок 11. Успешный запрос с токеном