API Gateway Role Based Authorization (JWT) - Advann Project

Goal

Implement Role Based Authorization in Spring Cloud API Gateway so that ADMIN and CUSTOMER have different access permissions.

What We Already Implemented

1. JWT Authentication in User-Service (Login/Register/Refresh/Logout)
2. Access Token Blacklisting on Logout (Token stored in DB)
3. API Gateway validates JWT token signature + expiry
4. API Gateway calls User-Service /validate-token endpoint to verify blacklist status
5. JWT Token contains role claim: ROLE_ADMIN or ROLE_CUSTOMER

Industry Flow (Recommended)

In production microservices, API Gateway is the main entry point. It validates the token and enforces authorization rules before forwarding the request to downstream services. User-Service is the source of truth for authentication and blacklisting.

Role Based Rules Implemented

PUBLIC APIs (No Token Required)

- GET /product-service/api/products
- GET /product-service/api/products/**
- /user-service/api/auth/** (login/register/refresh/logout)

ADMIN ONLY APIs

- POST /product-service/api/products
- PUT /product-service/api/products/**
- DELETE /product-service/api/products/**

CUSTOMER / ADMIN APIs (Token Required)

- /cart-service/**
- /order-service/**
- /payment-service/**

Step-by-Step Implementation

Step 1: Ensure JWT contains role claim (User-Service)

Your access token must contain a claim called role. Example payload: { sub: email, role: ROLE_ADMIN }. This role claim will be extracted by API Gateway.

Step 2: Create /validate-token endpoint in User-Service

This endpoint is called by API Gateway to confirm that the access token is not blacklisted and still valid.

```
@GetMapping("/validate-token")
public ResponseEntity<ApiResponse<TokenValidationResponseDto>> validateToken(
        @RequestHeader("Authorization") String authHeader
) {

    String token = authHeader.substring(7);

    boolean isBlacklisted = tokenBlacklistService.isBlacklisted(token);

    if (isBlacklisted) {
        ApiResponse<TokenValidationResponseDto> response = ApiResponse.<TokenValidationResponseDto>bu
                .success(false)
                .message("Token is blacklisted")
                .data(null)
                .build();

        return ResponseEntity.status(HttpStatus.UNAUTHORIZED).body(response);
    }

    String email;
    String role;

    try {
        email = jwtService.extractUsername(token);
        role = jwtService.extractClaim(token, claims -> claims.get("role", String.class));
    } catch (Exception e) {
        ApiResponse<TokenValidationResponseDto> response = ApiResponse.<TokenValidationResponseDto>bu
                .success(false)
                .message("Invalid token")
                .data(null)
                .build();

        return ResponseEntity.status(HttpStatus.UNAUTHORIZED).body(response);
    }

    TokenValidationResponseDto dto = TokenValidationResponseDto.builder()
            .valid(true)
            .email(email)
            .role(role)
            .build();

    ApiResponse<TokenValidationResponseDto> response = ApiResponse.<TokenValidationResponseDto>builde
            .success(true)
            .message("Token is valid")
            .data(dto)
            .build();

    return ResponseEntity.ok(response);
}
```

Step 3: JwtUtil in API Gateway (Extract role)

```
package com.advann.api_gateway.security;

import io.jsonwebtoken.Claims;
import io.jsonwebtoken.Jwts;
import io.jsonwebtoken.security.Keys;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.stereotype.Component;

import java.nio.charset.StandardCharsets;
import java.security.Key;

@Component
public class JwtUtil {
```

```
    @Value("${jwt.secret}")
    private String secretKey;

    private Key getSigningKey() {
        return Keys.hmacShaKeyFor(secretKey.getBytes(StandardCharsets.UTF_8));
    }

    public void validateToken(String token) {
        Jwts.parser()
                .verifyWith((javax.crypto.SecretKey) getSigningKey())
                .build()
                .parseSignedClaims(token);
    }

    public String extractUsername(String token) {
        Claims claims = Jwts.parser()
                .verifyWith((javax.crypto.SecretKey) getSigningKey())
                .build()
                .parseSignedClaims(token)
                .getPayload();

        return claims.getSubject();
    }

    public String extractRole(String token) {
        Claims claims = Jwts.parser()
                .verifyWith((javax.crypto.SecretKey) getSigningKey())
                .build()
                .parseSignedClaims(token)
                .getPayload();

        return claims.get("role", String.class);
    }
}
```

Step 4: JwtAuthFilter in API Gateway (Role Based Authorization)

```
package com.advann.api_gateway.security;

import lombok.RequiredArgsConstructor;
import org.springframework.cloud.gateway.filter.GatewayFilterChain;
import org.springframework.cloud.gateway.filter.GlobalFilter;
import org.springframework.http.HttpHeaders;
import org.springframework.http.HttpMethod;
import org.springframework.http.HttpStatus;
import org.springframework.stereotype.Component;
import org.springframework.web.reactive.function.client.WebClient;
import org.springframework.web.server.ServerWebExchange;
import reactor.core.publisher.Mono;

@Component
@RequiredArgsConstructor
public class JwtAuthFilter implements GlobalFilter {

    private final JwtUtil jwtUtil;
    private final WebClient.Builder webClientBuilder;

    @Override
    public Mono<Void> filter(ServerWebExchange exchange, GatewayFilterChain chain) {

        String path = exchange.getRequest().getURI().getPath();
        HttpMethod method = exchange.getRequest().getMethod();

        // PUBLIC ENDPOINTS
        if (path.startsWith("/api/auth/")
                || (path.startsWith("/api/products") && method == HttpMethod.GET)) {
            return chain.filter(exchange);
        }
```

```
        // Authorization Header check
        if (!exchange.getRequest().getHeaders().containsKey(HttpHeaders.AUTHORIZATION)) {
            exchange.getResponse().setStatusCode(HttpStatus.UNAUTHORIZED);
            return exchange.getResponse().setComplete();
        }

        String authHeader = exchange.getRequest().getHeaders().getFirst(HttpHeaders.AUTHORIZATION);

        if (authHeader == null || !authHeader.startsWith("Bearer ")) {
            exchange.getResponse().setStatusCode(HttpStatus.UNAUTHORIZED);
            return exchange.getResponse().setComplete();
        }

        String token = authHeader.substring(7);

        // Validate token signature + expiry
        try {
            jwtUtil.validateToken(token);
        } catch (Exception e) {
            exchange.getResponse().setStatusCode(HttpStatus.UNAUTHORIZED);
            return exchange.getResponse().setComplete();
        }

        // Extract role from JWT
        String role;
        try {
            role = jwtUtil.extractRole(token);
        } catch (Exception e) {
            exchange.getResponse().setStatusCode(HttpStatus.UNAUTHORIZED);
            return exchange.getResponse().setComplete();
        }

        // ADMIN ONLY: Product Add/Update/Delete
        if (path.startsWith("/api/products") && method != HttpMethod.GET) {
            if (role == null || !role.equals("ROLE_ADMIN")) {
                exchange.getResponse().setStatusCode(HttpStatus.FORBIDDEN);
                return exchange.getResponse().setComplete();
            }
        }

        // CUSTOMER / ADMIN allowed
        if (path.startsWith("/api/cart")
                || path.startsWith("/api/orders")
                || path.startsWith("/api/payments")) {

            if (role == null || !(role.equals("ROLE_CUSTOMER") || role.equals("ROLE_ADMIN"))) {
                exchange.getResponse().setStatusCode(HttpStatus.FORBIDDEN);
                return exchange.getResponse().setComplete();
            }
        }

        // Call USER-SERVICE validate-token endpoint (Blacklist check)
        return webClientBuilder.build()
                .get()
                .uri("http://USER-SERVICE/api/auth/validate-token")
                .header(HttpHeaders.AUTHORIZATION, "Bearer " + token)
                .retrieve()
                .bodyToMono(String.class)
                .flatMap(res -> chain.filter(exchange))
                .onErrorResume(e -> {
                    exchange.getResponse().setStatusCode(HttpStatus.UNAUTHORIZED);
                    return exchange.getResponse().setComplete();
                });
    }
}
```

Step 5: API Gateway application.yml routes

```
server:
  port: 8080

spring:
  application:
    name: api-gateway

  cloud:
    gateway:
      routes:
        - id: product-service
          uri: lb://product-service
          predicates:
            - Path=/product-service/**
          filters:
            - StripPrefix=1

        - id: user-service
          uri: lb://user-service
          predicates:
            - Path=/user-service/**
          filters:
            - StripPrefix=1

        - id: cart-service
          uri: lb://cart-service
          predicates:
            - Path=/cart-service/**
          filters:
            - StripPrefix=1

        - id: order-service
          uri: lb://order-service
          predicates:
            - Path=/order-service/**
          filters:
            - StripPrefix=1

        - id: payment-service
          uri: lb://payment-service
          predicates:
            - Path=/payment-service/**
          filters:
            - StripPrefix=1

eureka:
  client:
    register-with-eureka: true
    fetch-registry: true
    service-url:
      defaultZone: http://localhost:8761/eureka/
```

Testing Steps (Postman)

```
1) Login through Gateway:
POST http://localhost:8080/user-service/api/auth/login

2) Copy accessToken from response.

3) PUBLIC API (No token needed):
GET http://localhost:8080/product-service/api/products?page=0&size=2

4) ADMIN ONLY API test:
POST http://localhost:8080/product-service/api/products
Header: Authorization: Bearer <ADMIN_TOKEN>

If token has ROLE_ADMIN => 200/201 OK
```

```
If token has ROLE_CUSTOMER => 403 Forbidden

5) Logout (Blacklist token):
POST http://localhost:8080/user-service/api/auth/logout

6) After logout test secured API again:
GET http://localhost:8080/order-service/api/orders/1
Expected: 401 Unauthorized (Token blacklisted)

7) Difference between errors:
401 Unauthorized = invalid/expired/blacklisted token
403 Forbidden = valid token but not enough role permission
```

Important Notes (Interview + Production)

1. JWT role claim is used for authorization in gateway.
2. Blacklisting is checked by calling user-service validate-token API.
3. API Gateway enforces security before routing to microservices.
4. GET product APIs are public, but modifying product requires ADMIN.
5. API Gateway should always be the main entry point for clients.