# API Gateway + JWT Integration (Advann Microservices)

## Complete Step-by-Step Guide with Full Working Code

This document explains how to integrate **Spring Cloud API Gateway** with **JWT Authentication** in a **Spring Boot Microservices** architecture (Advann project). It includes: Eureka Service Discovery integration Routing configuration JWT validation in Gateway Public vs Protected endpoints Token Blacklisting (Logout) integration Calling USER-SERVICE from API Gateway using LoadBalancer WebClient End-to-End testing steps using Postman

# 1. Project Architecture

Your microservices architecture includes: **eureka-service-registry** (Port: 8761) **api-gateway** (Port: 8080) **product-service** (Port: 8081) **cart-service** (Port: 8082) **user-service** (Port: 8083) **order-service** (Port: 8084) **payment-service** (Port: 8085) All client requests should go through **API Gateway** instead of directly calling services.

## Why API Gateway is Required

API Gateway is the single entry point of your application. It provides: Centralized Authentication & Authorization Routing and Load Balancing using Eureka Request filtering and logging Rate limiting (optional) CORS handling Centralized security policies

# 2. API Gateway Setup (Dependencies)

## 2.1 api-gateway/pom.xml

```xml
<dependencies>

    <!-- Spring Cloud Gateway -->
    <dependency>
        <groupId>org.springframework.cloud</groupId>
        <artifactId>spring-cloud-starter-gateway</artifactId>
    </dependency>

    <!-- Eureka Client -->
    <dependency>
        <groupId>org.springframework.cloud</groupId>
        <artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>
    </dependency>

    <!-- WebFlux (Required for Gateway + WebClient) -->
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-webflux</artifactId>
    </dependency>

    <!-- Devtools -->
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-devtools</artifactId>
        <scope>runtime</scope>
        <optional>true</optional>
    </dependency>

    <!-- Testing -->
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-test</artifactId>
        <scope>test</scope>
    </dependency>

</dependencies>
```

**Note:** Spring Cloud Gateway works only with WebFlux, not Spring MVC.

# 3. API Gateway Configuration (application.yml)

## 3.1 api-gateway/src/main/resources/application.yml

```
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/
```

## Explanation:

**lb://SERVICE-NAME** means load-balanced routing using Eureka discovery. **StripPrefix=1** removes the first part of the path. Example: /user-service/api/auth/login becomes /api/auth/login inside user-service.

## 4. JWT Authentication in API Gateway

We want: Public endpoints to work without token (Register/Login, Product listing) Protected endpoints to require JWT token (Cart/Order/Payment) JWT validation (signature + expiry) in gateway Blacklist validation by calling USER-SERVICE

# 5. Create WebClient LoadBalanced Bean

## 5.1 WebClientConfig.java

```java
package com.advann.api_gateway.config;

import org.springframework.cloud.client.loadbalancer.LoadBalanced;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.web.reactive.function.client.WebClient;

@Configuration
public class WebClientConfig {

    @Bean
    @LoadBalanced
    public WebClient.Builder webClientBuilder() {
        return WebClient.builder();
    }
}
```

## 6. Create JwtUtil for Token Validation

### 6.1 JwtUtil.java

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

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

import java.util.Date;

@Component
public class JwtUtil {

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

    public void validateToken(String token) {
        Claims claims = Jwts.parser()
                .setSigningKey(jwtSecret)
                .parseClaimsJws(token)
                .getBody();

        Date expiration = claims.getExpiration();
        if (expiration.before(new Date())) {
            throw new RuntimeException("Token expired");
        }
    }
}
```

### 6.2 Add jwt.secret in api-gateway application.yml

```
jwt:
  secret: YOUR_SECRET_KEY_HERE
```

# 7. Create JwtAuthFilter (GlobalFilter)

## 7.1 JwtAuthFilter.java (Final Working Code)

```
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.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();

        // PUBLIC ENDPOINTS
        if (path.startsWith("/api/auth/")
                || path.startsWith("/api/products/")
                || path.equals("/api/products")) {
            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 JWT signature + expiry
        try {
            jwtUtil.validateToken(token);
        } catch (Exception e) {
            exchange.getResponse().setStatusCode(HttpStatus.UNAUTHORIZED);
            return exchange.getResponse().setComplete();
        }

        // Call USER-SERVICE validate-token API (Blacklist validation)
        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();
                });
    }
}
```

## 8. USER-SERVICE validate-token API (Blacklist Check)

### 8.1 Add validate-token endpoint in AuthController.java

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

    String token = authHeader.substring(7);

    boolean isBlacklisted = tokenBlacklistService.isBlacklisted(token);

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

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

    ApiResponse<String> response = ApiResponse.<String>builder()
            .success(true)
            .message("Token is valid")
            .data("VALID")
            .build();

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

## 9. Logout Flow (Blacklisting)

```
POST http://localhost:8080/user-service/api/auth/logout

Headers:
Authorization: Bearer <ACCESS_TOKEN>

Body:
{
  "refreshToken": "<REFRESH_TOKEN>"
}
```

## 10. End-to-End Testing Steps

Register/Login using Gateway Call Protected APIs using JWT token Logout (blacklist token) Try calling protected APIs again using old token Gateway should return 401 Unauthorized

## 11. Common Mistakes (Important)

**Calling validate-token using localhost:8080** inside gateway causes infinite recursion. Always call service via Eureka name: **http://USER-SERVICE** Always restart services after changing gateway filters. StripPrefix changes the path. So public endpoints must match the stripped path.