**Spring 2025 – SWER007**

**Lab 11: Extract Authentication into a Standalone Auth Service**

**✅ Objective**

Extract the existing authentication logic from the monolithic codebase into a standalone **auth-service**. Secure employee-service and department-service by verifying JWT tokens passed through the **API Gateway**.

**🔧 Prerequisites**

You should now have:

* employee-service, department-service, and api-gateway all running
* Eureka-based service discovery working
* Gateway routing to services based on path

**1️ Create Auth Service Project**

Use Spring Initializr to generate a new Spring Boot project:

**🔹 auth-service**

spring init --dependencies=web,security,eureka-client auth-service

Add these dependencies manually if needed:

<dependency>

<groupId>io.jsonwebtoken</groupId>

<artifactId>jjwt-api</artifactId>

<version>0.11.2</version>

</dependency>

<dependency>

<groupId>io.jsonwebtoken</groupId>

<artifactId>jjwt-impl</artifactId>

<version>0.11.2</version>

<scope>runtime</scope>

</dependency>

<dependency>

<groupId>io.jsonwebtoken</groupId>

<artifactId>jjwt-jackson</artifactId>

<version>0.11.2</version>

<scope>runtime</scope>

</dependency>

**3️ Register Auth Service with Eureka**

In application.yml:

server:

port: 8083

spring:

application:

name: auth-service

eureka:

client:

service-url:

defaultZone: http://localhost:8761/eureka

Register it with Eureka and confirm on http://localhost:8761

**4️ Secure Services with JWT via Gateway**

**1. Create a JWT Utility Class in api-gateway-service**

This class will be used to **validate the JWT** from incoming requests.

import io.jsonwebtoken.Claims;

import io.jsonwebtoken.Jwts;

import org.springframework.stereotype.Component;

import java.util.Date;

import java.util.function.Function;

@Component

public class JwtUtil {

    private final String SECRET\_KEY = "5dab7a23fbbd1f3eccb5ef839ac89af5fb6f0f0f5cc832209d8d459b15e4e4d5"; // Change this to a strong secret

    public String extractUsername(String token) {

        return extractClaim(token, Claims::getSubject);

    }

    public Date extractExpiration(String token) {

        return extractClaim(token, Claims::getExpiration);

    }

    public <T> T extractClaim(String token, Function<Claims, T> claimsResolver) {

        final Claims claims = Jwts.parser()

                .setSigningKey(SECRET\_KEY)

                .parseClaimsJws(token)

                .getBody();

        return claimsResolver.apply(claims);

    }

    public boolean isTokenValid(String token) {

        try {

            extractUsername(token);

        } catch (Exception e) {

            return false;

        }

        return !isTokenExpired(token);

    }

    private boolean isTokenExpired(String token) {

        return extractExpiration(token).before(new Date());

    }

}

**2. Create an Authentication Filter (AuthenticationFilter.java)**

This filter will **intercept all incoming requests** and check for a valid JWT in the Authorization header.

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.http.HttpStatus;

import org.springframework.stereotype.Component;

import org.springframework.web.server.ResponseStatusException;

import org.springframework.web.server.ServerWebExchange;

import org.springframework.cloud.gateway.filter.GatewayFilterChain;

import org.springframework.cloud.gateway.filter.GlobalFilter;

import org.springframework.core.Ordered;

import reactor.core.publisher.Mono;

@Component

public class AuthenticationFilter implements GlobalFilter, Ordered {

@Autowired

private JwtUtil jwtUtil;

@Override

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

String path = exchange.getRequest().getURI().getPath();

// Skip auth check for login endpoint

if (path.contains("/auth/login") || path.contains("/auth/register") ) {

return chain.filter(exchange);

}

// Check for Authorization header

if (!exchange.getRequest().getHeaders().containsKey("Authorization")) {

throw new ResponseStatusException(HttpStatus.UNAUTHORIZED, "Missing Authorization Header");

}

String authHeader = exchange.getRequest().getHeaders().getFirst("Authorization");

if (authHeader == null || !authHeader.startsWith("Bearer ")) {

throw new ResponseStatusException(HttpStatus.UNAUTHORIZED, "Invalid Authorization Header");

}

String token = authHeader.substring(7);

if (!jwtUtil.isTokenValid(token)) {

throw new ResponseStatusException(HttpStatus.UNAUTHORIZED, "Invalid or Expired Token");

}

return chain.filter(exchange);

}

@Override

public int getOrder() {

return -1; // Execute early in the filter chain

}

}

**3. Secure Routes in application.yml**

Only secure the routes you want protected (e.g., /employees, /departments):

spring:

cloud:

gateway:

routes:

- id: employee-service

uri: lb://employee-service

predicates:

- Path=/employees/\*\*

- id: department-service

uri: lb://department-service

predicates:

- Path=/departments/\*\*

- id: auth-service

uri: lb://auth-service

predicates:

- Path=/auth/\*\*

This ensures the auth-service endpoint /auth/login is open, while everything else goes through the JWT filter.

**5️ Test Auth Integration**

* Request a token from POST /auth/login
* Call /employees or /departments with:

Authorization: Bearer <token>

* You should get access only with a valid token

**📝 Lab Tasks**

1. Create a new project: auth-service
2. Implement POST /auth/login to return a JWT
3. Register it with Eureka
4. Add a JWT filter in API Gateway
5. Protect employee-service and department-service routes
6. Test end-to-end authentication

**🎯 Expected Outcome**

* Standalone auth-service returning JWT tokens
* API Gateway secures downstream services using token validation
* Only authenticated users can access employee/department APIs