**Step 1: Set Up the API Gateway**

1. **Create a Spring Boot project** for the API Gateway.
   * Use **Spring Cloud Gateway** and **Spring Security** dependencies.
   * Include the dependencies in pom.xml:

xml

Copy code

<dependency>

<groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-starter-gateway</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-security</artifactId>

</dependency>

<dependency>

<groupId>io.jsonwebtoken</groupId>

<artifactId>jjwt</artifactId>

<version>0.9.1</version>

</dependency>

1. **Configure routing for microservices** in application.yml:

spring:

cloud:

gateway:

routes:

- id: product-service

uri: http://localhost:8081

predicates:

- Path=/products/\*\*

- id: customer-service

uri: http://localhost:8082

predicates:

- Path=/customers/\*\*

- id: order-service

uri: http://localhost:8083

predicates:

- Path=/orders/\*\*

**Step 2: JWT Token Creation in Customer Service**

1. **Generate a JWT Token during login** in the Customer Service:
   * Add a dependency for JWT (io.jsonwebtoken).
   * Implement the token generation logic:

java

Copy code

public String generateToken(String username, List<String> roles) {

return Jwts.builder()

.setSubject(username)

.claim("roles", roles)

.setExpiration(new Date(System.currentTimeMillis() + 1000 \* 60 \* 15)) // 15 mins

.signWith(SignatureAlgorithm.HS256, SECRET\_KEY)

.compact();

}

1. **Return the token** in the login API response:

java

Copy code

@PostMapping("/customers/login")

public ResponseEntity<?> login(@RequestBody LoginRequest request) {

// Authenticate user logic

String token = generateToken(user.getUsername(), user.getRoles());

return ResponseEntity.ok(new LoginResponse(token));

}

**Step 3: Validate JWT in API Gateway**

1. **Create a JWT Authentication Filter**:
   * Write a filter to validate JWT tokens in the API Gateway:

java

Copy code

@Component

public class JwtAuthenticationFilter implements GlobalFilter {

private static final String SECRET\_KEY = "your-secret-key";

@Override

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

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

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

exchange.getResponse().setStatusCode(HttpStatus.UNAUTHORIZED);

return exchange.getResponse().setComplete();

}

try {

String jwt = token.substring(7);

Claims claims = Jwts.parser()

.setSigningKey(SECRET\_KEY)

.parseClaimsJws(jwt)

.getBody();

// Add roles to headers for downstream services

exchange.getRequest().mutate()

.header("roles", String.join(",", claims.get("roles", List.class)))

.build();

} catch (Exception e) {

exchange.getResponse().setStatusCode(HttpStatus.FORBIDDEN);

return exchange.getResponse().setComplete();

}

return chain.filter(exchange);

}

}

1. **Register the filter in the gateway**:
   * Add the filter globally in your application.yml:

yaml

Copy code

spring:

cloud:

gateway:

default-filters:

- name: JwtAuthenticationFilter

**Step 4: Implement Role-Based Authorization**

1. **Create a Role-Based Authorization Filter**:
   * Enforce role-based access for specific routes:

java

Copy code

@Component

public class RoleBasedAuthorizationFilter implements GlobalFilter {

@Override

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

String rolesHeader = exchange.getRequest().getHeaders().getFirst("roles");

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

if (path.startsWith("/products/manage") && !rolesHeader.contains("ROLE\_MANAGER")) {

exchange.getResponse().setStatusCode(HttpStatus.FORBIDDEN);

return exchange.getResponse().setComplete();

}

if (path.startsWith("/orders") && !rolesHeader.contains("ROLE\_CUSTOMER")) {

exchange.getResponse().setStatusCode(HttpStatus.FORBIDDEN);

return exchange.getResponse().setComplete();

}

return chain.filter(exchange);

}

}

1. **Add the filter in the routing configuration**:

yaml

Copy code

spring:

cloud:

gateway:

routes:

- id: product-service

uri: http://localhost:8081

predicates:

- Path=/products/\*\*

filters:

- name: RoleBasedAuthorizationFilter

**Step 5: Secure Individual Microservices**

1. **Validate JWT Tokens in Each Service**:
   * Add a dependency for io.jsonwebtoken in all microservices.
   * Create a filter to validate JWT tokens in individual services:

java

Copy code

public boolean validateToken(String token) {

try {

Jwts.parser().setSigningKey(SECRET\_KEY).parseClaimsJws(token);

return true;

} catch (Exception e) {

return false;

}

}

1. **Use Role-Based Annotations in Controllers**:
   * Secure endpoints using Spring Security annotations:

java

Copy code

@RestController

@RequestMapping("/products")

public class ProductController {

@PreAuthorize("hasRole('ROLE\_MANAGER')")

@PostMapping("/manage")

public ResponseEntity<Product> addProduct(@RequestBody Product product) {

return ResponseEntity.ok(product);

}

}