**Central API Gateway with Security Across All Services**

This guide provides detailed instructions to implement a **Central API Gateway** and security for all services in a microservices architecture. The gateway will handle authentication and authorization using **JWT-based security** while routing requests to the respective services. Security policies will enforce restricted access to specific endpoints based on user roles (CUSTOMER, MANAGER, or ADMIN).

**Key Requirements**

1. **Login Required**:
   * Customers must log in to:
     + Create a cart.
     + Place orders.
     + Manage their profile.
   * Customers can **view products** without logging in.
2. **Restricted Access**:
   * Only MANAGER or ADMIN roles can access endpoints to:
     + Manage products.
     + Access administrative functionalities.

**Step 1: Design the API Gateway**

1. **Purpose**:
   * The API Gateway will serve as the central entry point for all client requests.
   * It will handle:
     + Authentication (validating JWT tokens).
     + Authorization (enforcing role-based access control).
     + Request routing to respective microservices.
2. **Technologies**:
   * Use **Spring Cloud Gateway** or **Zuul** for the API Gateway.
   * Use **Spring Security** for authentication and authorization.
3. **Components**:
   * **Authentication Filter**:
     + Validate JWT tokens included in the Authorization header.
   * **Role-Based Authorization Filter**:
     + Extract roles from the JWT token and enforce access policies.
   * **Route Configuration**:
     + Define routing rules to forward requests to specific microservices.

**Step 2: Configure the API Gateway**

1. **JWT Validation**:
   * Extract the token from the Authorization header.
   * Decode and validate the token signature using the shared secret key.
   * Extract claims (e.g., username and roles) from the token payload.
2. **Route Configuration**:
   * Define routes for each microservice in the application.yml or configuration class.
   * Example:

yaml

Copy code

spring:

cloud:

gateway:

routes:

- id: product-service

uri: http://localhost:8081

predicates:

- Path=/products/\*\*

- id: cart-service

uri: http://localhost:8082

predicates:

- Path=/carts/\*\*

- id: order-service

uri: http://localhost:8083

predicates:

- Path=/orders/\*\*

- id: customer-service

uri: http://localhost:8084

predicates:

- Path=/customers/\*\*

1. **Global Filters**:
   * Apply filters globally to:
     + Enforce authentication for all secured routes.
     + Allow specific endpoints (e.g., /products) to bypass authentication.

**Step 3: Secure Endpoints with Role-Based Access Control**

1. **Public Endpoints**:
   * Allow unauthenticated access to specific endpoints such as:
     + GET /products: View product details.
2. **Customer-Specific Endpoints**:
   * Restrict access to customers for actions such as:
     + POST /carts: Create a cart.
     + POST /orders: Place an order.
     + PUT /customers/profile: Update profile.
   * Require the ROLE\_CUSTOMER for these endpoints.
3. **Manager/Admin Endpoints**:
   * Restrict access to managers or admins for actions such as:
     + POST /products: Add a new product.
     + PUT /products/{id}: Update a product.
     + DELETE /products/{id}: Remove a product.
   * Require the ROLE\_MANAGER or ROLE\_ADMIN.
4. **Gateway Authorization Policy**:
   * Define role-based access rules in the API Gateway using filters or interceptors.

**Step 4: Implement JWT-Based Security**

1. **Token Generation**:
   * Generate JWT tokens during login in the **Customer Service**.
   * Include claims for:
     + username
     + roles (e.g., ROLE\_CUSTOMER, ROLE\_MANAGER).
2. **Token Validation**:
   * Validate JWT tokens in the API Gateway.
   * Verify:
     + Token expiration.
     + Token signature using a secret key.
3. **Shared Secret Key**:
   * Use a shared secret key between the API Gateway and the authentication service to sign and validate tokens.
4. **Token Format**:
   * Header: alg (algorithm), typ (token type)
   * Payload: username, roles, exp (expiration timestamp)
   * Signature: Encrypted hash of the header and payload.

**Step 5: Apply Security Filters in the API Gateway**

1. **Authentication Filter**:
   * Check for the presence of the Authorization header.
   * Validate the JWT token.
   * Reject requests with invalid or missing tokens for secured endpoints.
2. **Authorization Filter**:
   * Extract roles from the token.
   * Match roles against the required roles for the requested endpoint.
   * Reject requests from users without sufficient permissions.
3. **Public Endpoint Filter**:
   * Allow requests to endpoints like GET /products without requiring authentication.

**Step 6: Secure Microservices**

1. **Microservice Security**:
   * Each microservice should validate user roles for actions it handles.
   * Use security annotations or filters to enforce access control.
2. **Role Verification**:
   * Example: In the Product Service, allow only ROLE\_MANAGER or ROLE\_ADMIN to add, update, or delete products.
3. **Fallback Handling**:
   * If the API Gateway fails to validate the token, microservices should handle the fallback security by returning unauthorized errors.

**Step 7: Define Authentication and Authorization Rules**

1. **Routing Rules**:
   * **Public Access**:
     + Routes like GET /products should allow unauthenticated access.
   * **Customer Access**:
     + Routes like POST /carts, POST /orders, and PUT /customers/profile should require the ROLE\_CUSTOMER.
   * **Manager/Admin Access**:
     + Routes like POST /products and PUT /products/{id} should require ROLE\_MANAGER or ROLE\_ADMIN.
2. **Role-Based Filters**:
   * Define custom filters for role-based access control in the API Gateway.