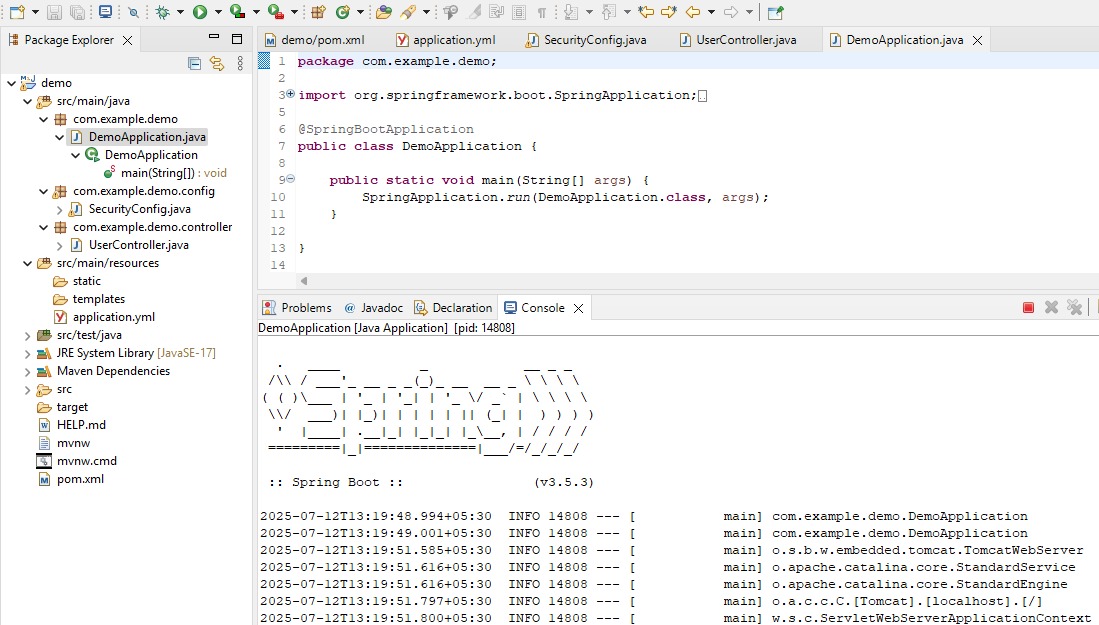
Sample exercises on Centralized Authentication and SSO with Spring Boot 3 and Spring Cloud

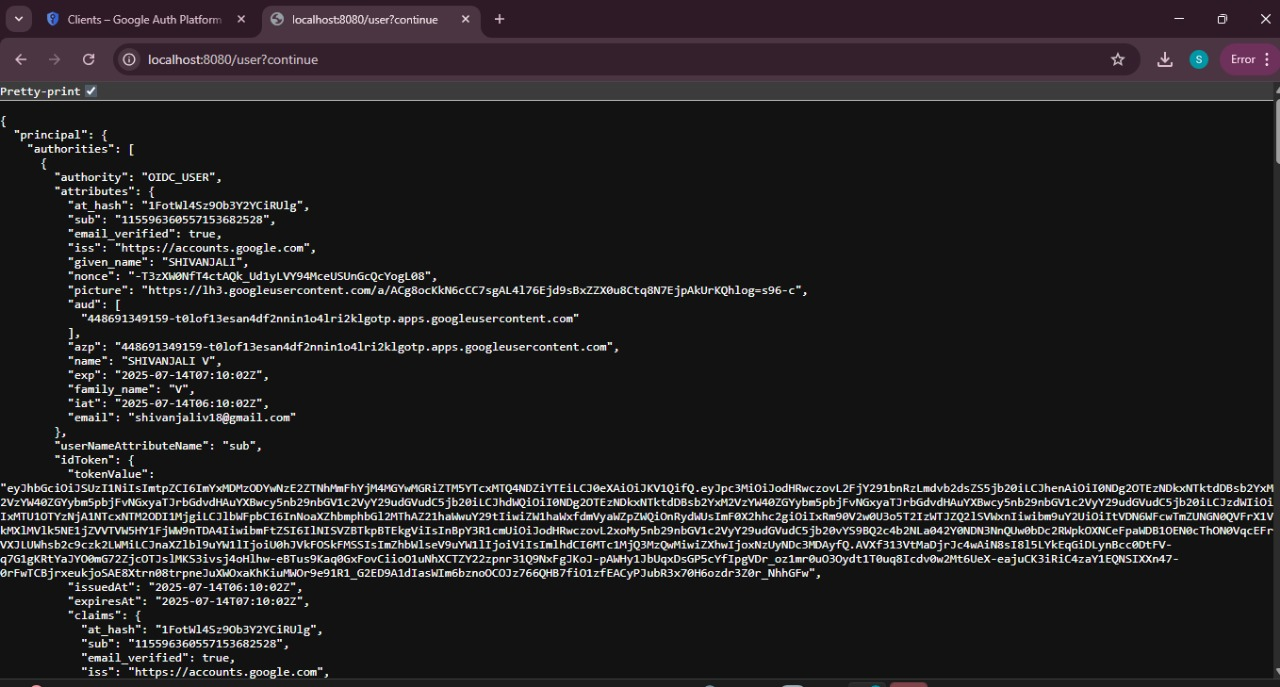
**Exercise 1: Implementing Centralized Authentication with OAuth 2.1/OIDC** Task: Implement centralized authentication using OAuth 2.1/OIDC in a Spring Boot application.

## Step-by-Step Explanation:

1. Add dependencies for Spring Security and OAuth2 Client in your `pom.xml`.
2. Configure OAuth2 client properties in `application.yml`.
3. Create a security configuration class to set up OAuth2 login.
4. Implement a controller to handle login and display user information.

**Output:**





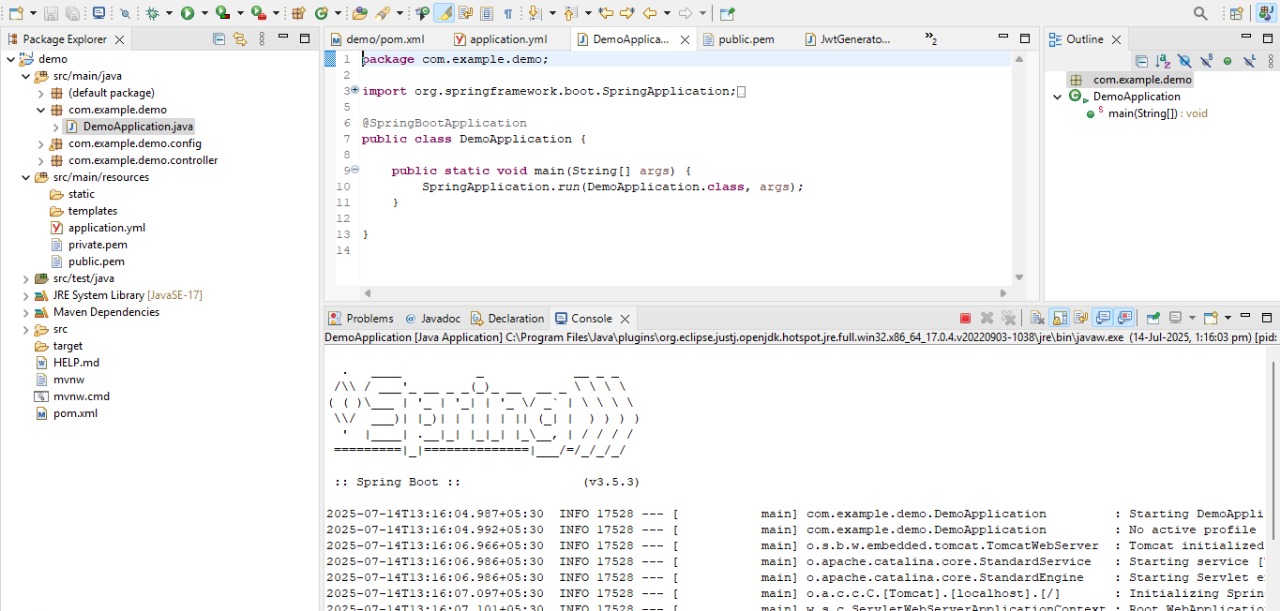
# Exercise 2: Configuring Authorization Servers and Resource Servers

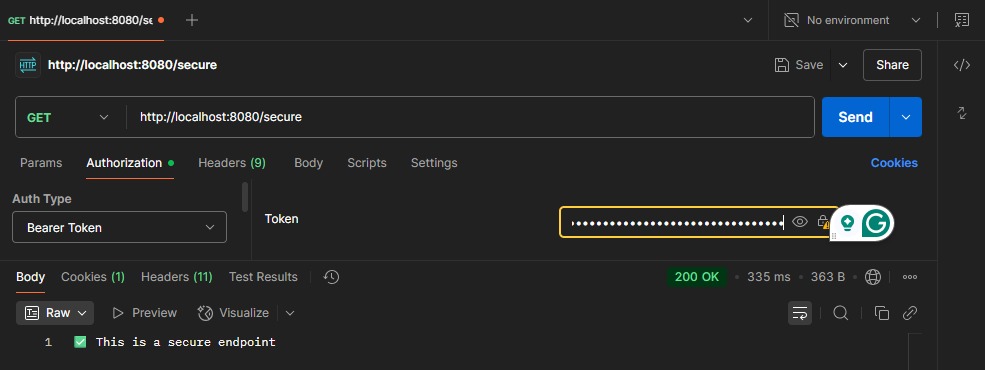
Task: Configure Authorization Servers and Resource Servers in a Spring Boot application.

## Step-by-Step Explanation:

1. Add dependencies for Spring Security and OAuth2 Resource Server in your `pom.xml`.
2. Configure the Authorization Server properties in `application.yml`.
3. Create a security configuration class for the Resource Server.
4. Implement a controller to secure endpoints.

# Output:





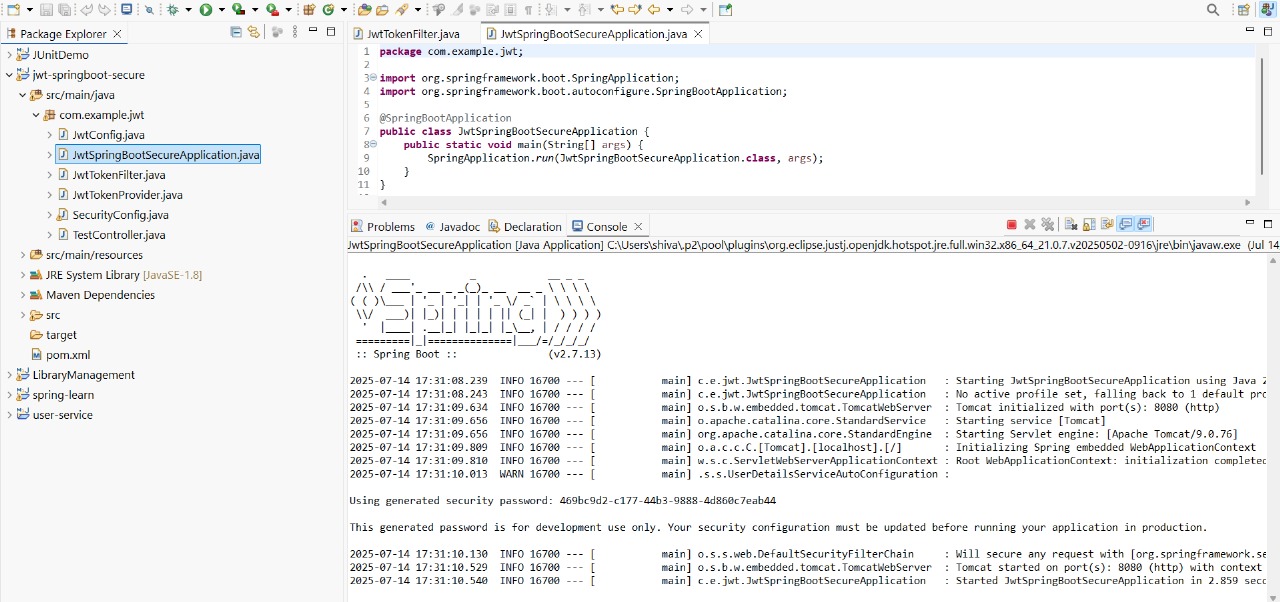
# Exercise 3: Using JSON Web Tokens (JWT) for Secure Communication

Task: Use JSON Web Tokens (JWT) for secure communication in a Spring Boot application.

## Step-by-Step Explanation:

1. Add dependencies for Spring Security and JWT in your `pom.xml`.
2. Configure JWT properties in `application.yml`.
3. Create a security configuration class to set up JWT authentication.
4. Implement a controller to secure endpoints using JWT.

# Output:



**Sample Hands-on Exercises on Edge Services and API Gateway with Spring Boot 3 and Spring Cloud**

# **Exercise 1: Implementing Edge Services for Routing and Filtering**

\*\*Task:\*\* Implement an edge service for routing and filtering requests in a microservices architecture using Spring Boot 3 and Spring Cloud.

# **Step-by-Step Explanation:**

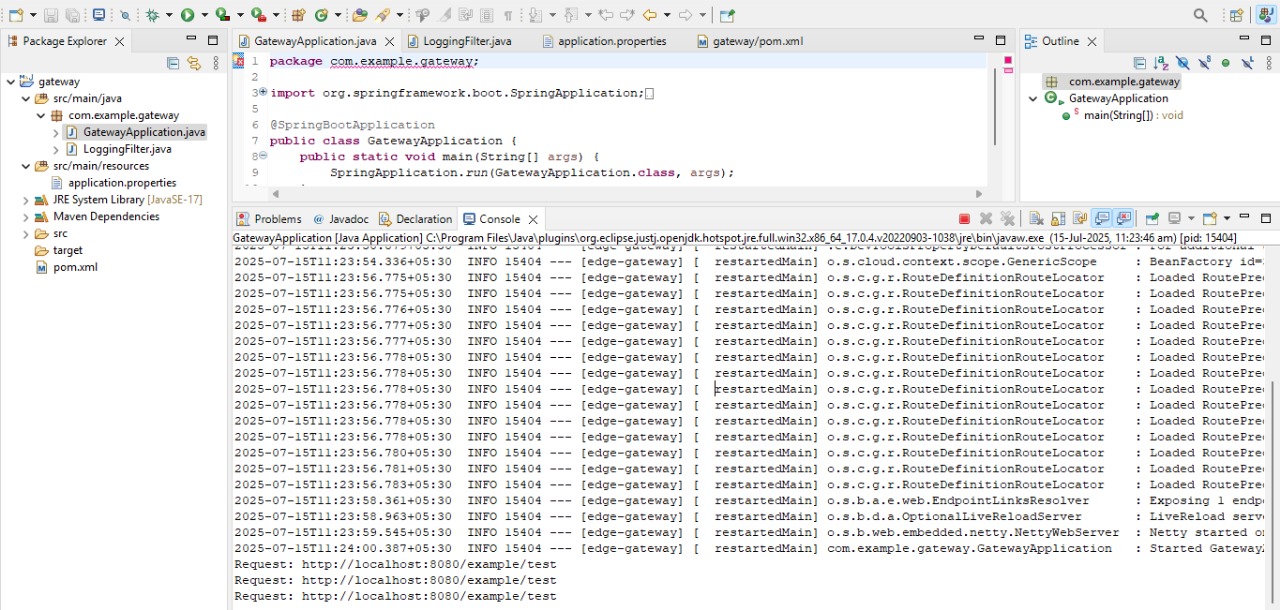
1. Create a new Spring Boot project with the necessary dependencies for Spring Cloud Gateway.

2. Configure the application properties for routing.

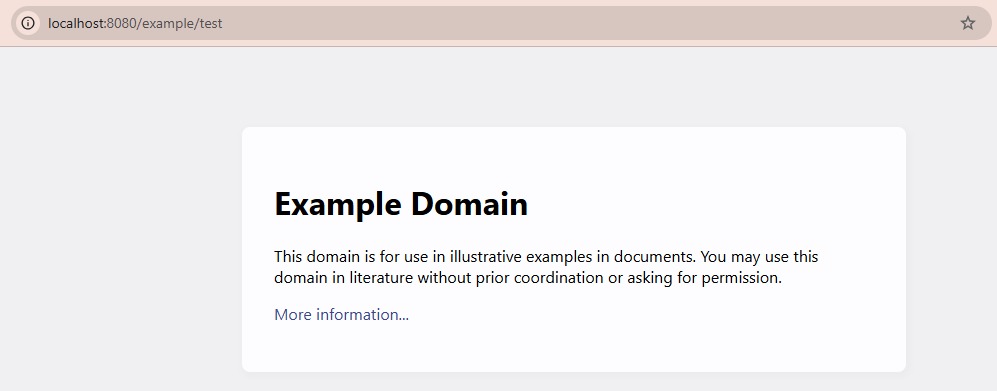
3. Implement a custom filter for logging requests.

4. Test the routing and filtering functionality.

# **Output:**



This proves your custom GlobalFilter is working correctly.



# **Exercise 2: Load Balancing in an API Gateway**

\*\*Task:\*\* Implement load balancing in an API Gateway using Spring Boot 3 and Spring Cloud.

# **Step-by-Step Explanation:**

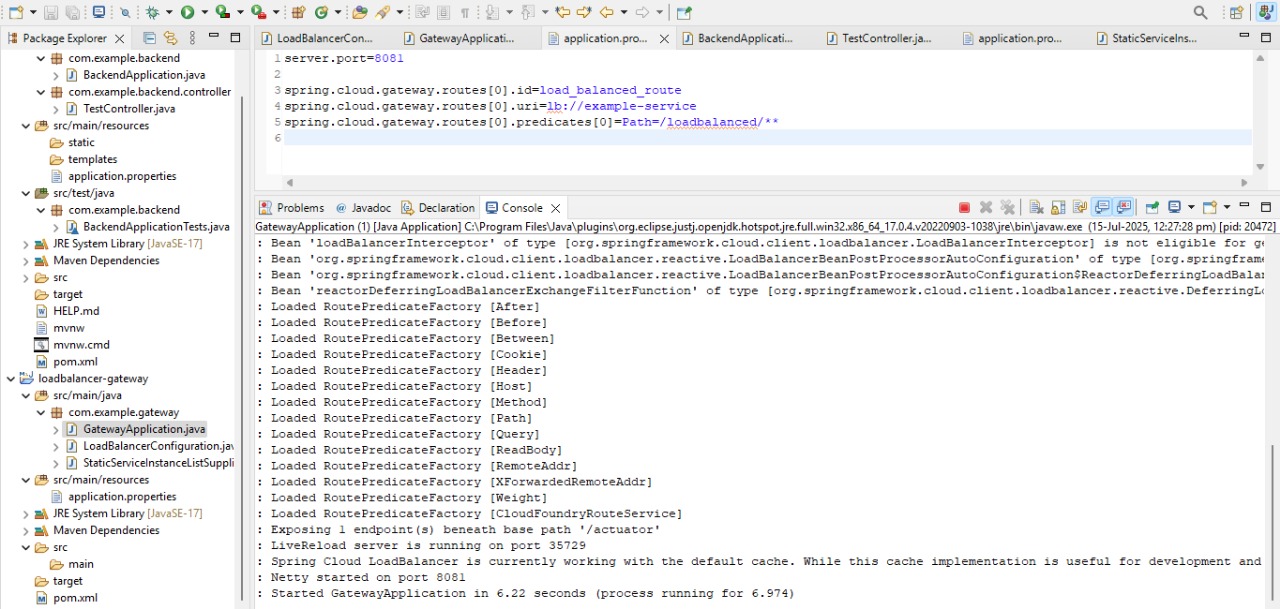
1. Create a new Spring Boot project with the necessary dependencies for Spring Cloud Gateway and Spring Cloud LoadBalancer.

2. Configure the application properties for load balancing.

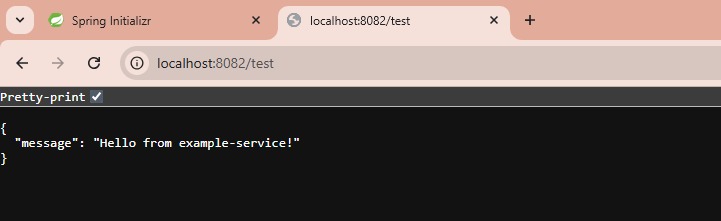
3. Implement a custom load balancing configuration.

4. Test the load balancing functionality.

# **Output:**



Netty started on port 8081.



# **Exercise 3: Resilience Patterns in an API Gateway**

\*\*Task:\*\* Implement resilience patterns in an API Gateway using Spring Boot 3 and Spring Cloud.

# **Step-by-Step Explanation:**

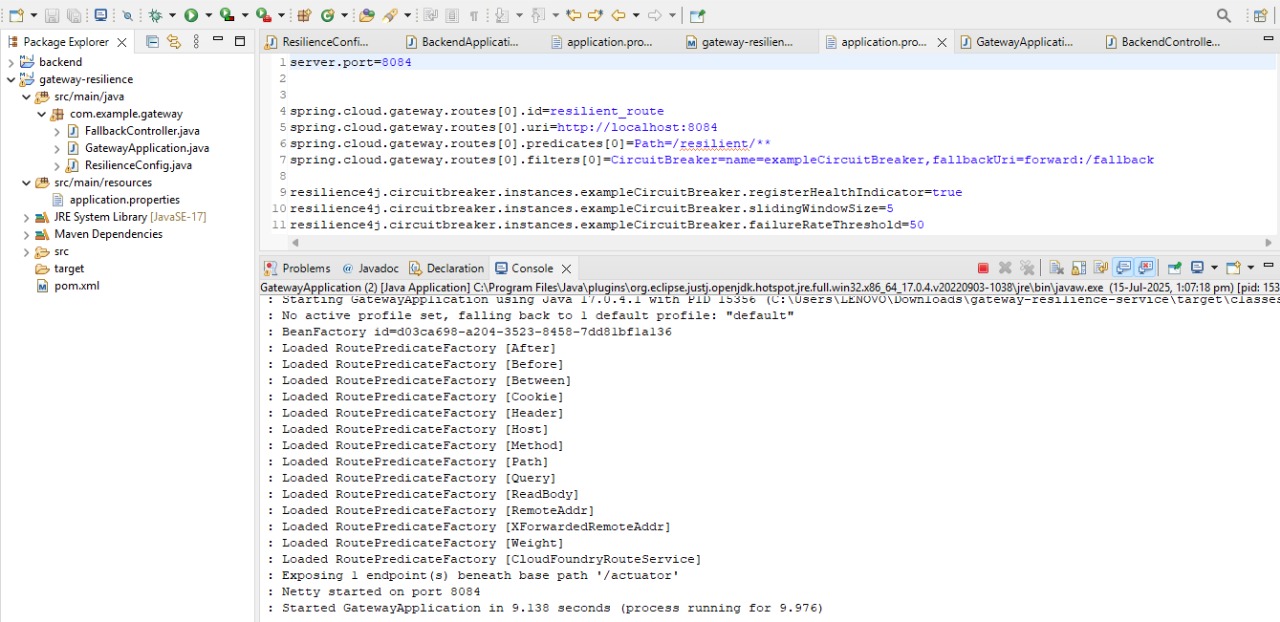
1. Create a new Spring Boot project with the necessary dependencies for Spring Cloud Gateway and Resilience4j.

2. Configure the application properties for resilience patterns.

3. Implement a custom resilience configuration.

4. Test the resilience functionality.

# **Output:**



Netty started on port 8084.

