**Setting Up Token-Based Authentication with Spring Cloud Gateway and Keycloak**

**Prerequisites**

* Java Development Kit (JDK) installed
* Maven installed
* Keycloak server installed and running
* Basic knowledge of Spring Boot and Keycloak

**Step 1: Add Dependencies**

Add the following dependencies to your pom.xml:

**XML**

<dependency>

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

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

</dependency>

<dependency>

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

<artifactId>spring-boot-starter-oauth2-client</artifactId>

</dependency>

<dependency>

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

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

</dependency>

Door AI gegenereerde code. Controleer en gebruik zorgvuldig. [Meer informatie over veelgestelde vragen](https://www.bing.com/new" \l "faq" \t "_blank).

**Step 2: Configure Security**

Create a security configuration class to handle OAuth2 login and token relay:

**Java**

@Configuration

public class SecurityConfig extends WebSecurityConfigurerAdapter {

@Override

protected void configure(HttpSecurity http) throws Exception {

http

.authorizeRequests(authorizeRequests ->

authorizeRequests

.anyRequest().authenticated()

)

.oauth2Login();

}

}

Door AI gegenereerde code. Controleer en gebruik zorgvuldig. [Meer informatie over veelgestelde vragen](https://www.bing.com/new" \l "faq" \t "_blank).

**Step 3: Configure Gateway**

Update your application.yml to configure the routes and enable token relay:

spring:

cloud:

gateway:

routes:

- id: user-api

uri: http://localhost:8081

predicates:

- Path=/user/\*\*

filters:

- TokenRelay

**Step 4: Enable Token Relay**

Ensure the TokenRelay filter is applied to forward the OAuth2 access token to downstream services:

**Java**

@Configuration

public class GatewayConfig {

@Bean

public RouteLocator customRouteLocator(RouteLocatorBuilder builder) {

return builder.routes()

.route("user-api", r -> r.path("/user/\*\*")

.filters(f -> f.tokenRelay())

.uri("http://localhost:8081"))

.build();

}

}

Door AI gegenereerde code. Controleer en gebruik zorgvuldig. [Meer informatie over veelgestelde vragen](https://www.bing.com/new" \l "faq" \t "_blank).

**Step 5: Secure User API**

Ensure your user API is configured to validate the JWT token. Add the following dependencies to your user API’s pom.xml:

**XML**

<dependency>

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

<artifactId>spring-boot-starter-oauth2-resource-server</artifactId>

</dependency>

Door AI gegenereerde code. Controleer en gebruik zorgvuldig. [Meer informatie over veelgestelde vragen](https://www.bing.com/new" \l "faq" \t "_blank).

Then, configure the resource server in your user API:

**Java**

@Configuration

public class ResourceServerConfig extends WebSecurityConfigurerAdapter {

@Override

protected void configure(HttpSecurity http) throws Exception {

http

.authorizeRequests(authorizeRequests ->

authorizeRequests

.anyRequest().authenticated()

)

.oauth2ResourceServer(oauth2 -> oauth2.jwt());

}

}

Door AI gegenereerde code. Controleer en gebruik zorgvuldig. [Meer informatie over veelgestelde vragen](https://www.bing.com/new" \l "faq" \t "_blank).

**Step 6: Test the Setup**

1. Start your Keycloak server and ensure the realm and clients are configured correctly.
2. Run your Spring Cloud Gateway and user API applications.
3. Use a tool like Postman to obtain an access token from Keycloak and make requests to your API Gateway, ensuring the token is properly relayed and validated.