Keycloak Integration for Next.js and Spring Microservices

Table of Contents

Status	REVIEW
Version	1.0.0
Author(s)	@Daniel Steinhauf
Date of creation	Dec 18, 2025
Last updated	Dec 18, 2025
Approved on	
<name of the person(s) in charge to approve the document >	Date of approval: yyyy-mm-dd

Overview

This guideline defines the authentication and authorization architecture for applications built with **Next.js** and **Spring-based microservices**, using **Keycloak** as the identity and access management solution. It ensures secure end-user authentication, backend API protection, and fine-grained access control between microservices.

Components

- **Keycloak**: Central identity provider, issuing tokens (ID, Access, Refresh).
- **Next.js Fullstack App**: Provides frontend UI and backend API routes.
- **Spring Microservices**: Backend services registered as confidential clients.
- **OAuth2 / OpenID Connect**: Protocols used for authentication and authorization.

Authentication Flows

End User Authentication (Next.js Frontend)

- **Client Type**: Public client in Keycloak.

- **Flow**: Authorization Code Flow with PKCE.
- **Steps**:
  a. User requests access to a protected page.
  b. Next.js redirects to Keycloak login.
  c. User authenticates; Keycloak returns authorization code.
  d. Next.js backend exchanges code for tokens (ID, Access, Refresh).
  e. Tokens are stored in **secure HTTP-only cookies**.
  f. Frontend uses session context for SSR/ISR rendering.
  g. Backend API routes validate the session and tokens before processing requests.

Backend Authentication / Microservice-to-Microservice Authentication

- **Client Type**: Confidential clients in Keycloak.
- **Flow**: Client Credentials Grant.
- **Steps**:
  a. Microservice requests a token from Keycloak using client ID + secret.
  b. Keycloak issues an Access Token with defined scopes/roles.
  c. A microservice attaches a token to the request headers.
  d. The microservice validates the token (signature, audience, scopes).
  e. Access is granted or denied based on token claims.

Fine-Grained Access Control Between Microservices

- **Scopes**: Define resource-level permissions (e.g., `order.read`, `payment.write`).
- **Roles**: Assign service-level identities (e.g., `order-service-client`).
- **Audience Claims**: Ensure tokens are intended for the correct service.
- **Policies**: Use Keycloak Authorization Services to define which clients can access which services.
- **Principle of Least Privilege**: Tokens only include minimal scopes required.

Example:

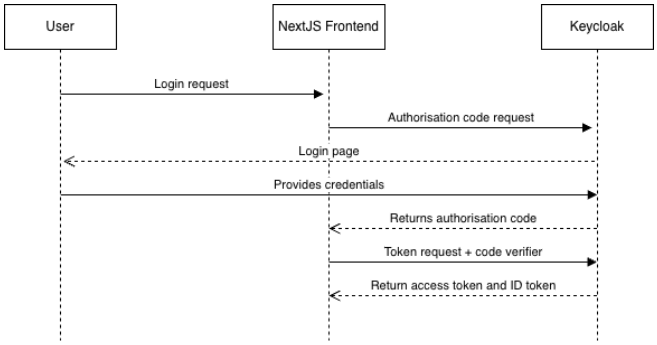
- Order Service only accepts tokens with `order.read` or `order.write`.
- Payment Service only accepts tokens with `payment.process`.
- Microservice A cannot call the payment Service unless explicitly granted `payment.process`.

Security Considerations

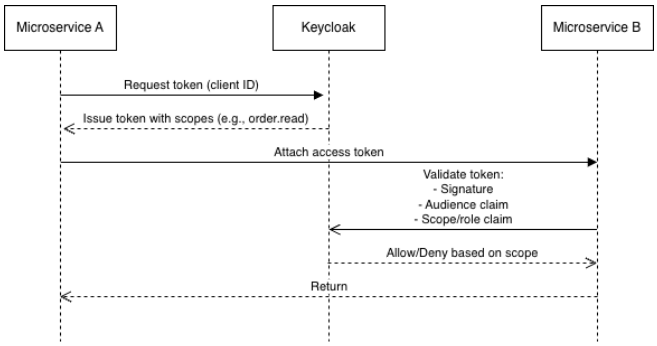
- Always use **HTTPS** for token exchange and API calls.
- Store tokens in **HTTP-only cookies** (avoid localStorage).
- Regularly rotate client secrets for microservices.
- Validate **signature, expiration, audience, and scopes** for every incoming token.
- Apply **least privilege** when assigning scopes and roles.

Architecture Diagrams

Next.js Fullstack Authentication



Backend Authentication / Microservice-to-Microservice Restricted Access



Summary

This architecture guideline ensures:

- **Secure end-user authentication** via Next.js + Keycloak.
- **Protected backend API routes** in Next.js.
- **Controlled microservice communication** using client credentials and scopes.
- **Fine-grained access control** enforced by Keycloak policies.