JWT in Spring Boot

What is JWT?

A quick and safe method of sending data between parties as a JSON object is JWT (JSON Web Token). Stateless authentication is frequently handled by Spring Boot, particularly in RESTful APIs.

Real-Life Example (Analogy)

Consider JWT to be similar to a movie ticket. Your ticket includes the movie name, time, and seat number when you purchase it. You enter using that ticket; you don't have to present your ID each time. In a similar vein, JWT is used to access restricted routes without repeatedly logging in and contains user information.

How JWT Works in Spring Boot

1. The user enters their username and password to log in.

2. A JWT token is created and returned if the credentials are legitimate.

3. The token is stored by the client (typically in cookies or local storage).

4. The token is transmitted in the Authorization header for each further request as follows: Authorization: <token> Bearer

5. Before handling the request, a JWT filter verifies the token.

6. Spring Security verifies the user and grants access if it is valid.

Why Use JWT?

Stateless: Sessions don't need to be saved on the server.

Scalable: Ideal for distributed systems and microservices.

Secure: Tokens are signed to guard against manipulation.

Flexible: Custom claims, such as user roles or rights, can be included.

How I Implemented JWT in Spring Boot

I utilized the following elements in my project:

A utility class for creating, decoding, and validating JWT tokens is called JwtUtil.

JwtFilter: A filter that uses the token to authenticate requests after intercepting them.

SecurityConfig: Sets up public and protected endpoints, registers the filter, and configures Spring Security.

The Login Controller authenticates users and, upon successful completion, returns a JWT.

Role-based Access: I used annotations such as @PreAuthorize("hasRole('ADMIN')" to restrict endpoints by storing roles in the token.

Result

My application was able to protect sensitive endpoints and authenticate users without retaining session data thanks to this solution.

Manage access depending on roles

Scale across services more readily