Aula 17: Introdução à Validação de Tokens JWT com Filtros

September 18, 2025

1 Introdução

Na Aula 16, encapsulamos o token JWT em um DTO (`DadosTokenJWT`) e configuramos a chave secreta via `application.properties`. Nesta aula, iniciaremos a proteção dos endpoints da API, exigindo a validação de tokens JWT em requisições. Veremos por que adicionar validação diretamente nos controllers, como o `MedicoController`, é ineficiente e aprenderemos a usar um filtro do Spring Security para centralizar a lógica de validação. As alterações serão feitas no `Visual Studio Code` ou sua IDE preferida, com testes no Insomnia.

2 Revisão do Processo de Autenticação

O processo de autenticação envolve:

- O cliente envia `POST http://localhost:8080/login` com um JSON contendo `login` e `senha`:

```
{
    "login": "usuario@voll.med",
    "senha": "123456"
}
```

- A API retorna um token JWT encapsulado em `DadosTokenJWT`.
- O cliente (ex.: aplicativo mobile) armazena o token e o envia em requisições subsequentes para endpoints protegidos (ex.: `GET /medicos`).

3 Problema da Validação nos Controllers

No `MedicoController` (pacote `med.voll.api.controller`), poderíamos adicionar validação de tokens diretamente no método mapeado com `@GetMapping`, como:

```
@GetMapping
public ResponseEntity listar() {
    String token = // Extrair token da requisição
    if (token == null) {
        return ResponseEntity.status(HttpStatus.UNAUTHORIZED).build();
    } else {
        // Validar token e prosseguir
        return ResponseEntity.ok(...);
    }
}
```

3.1 Problema

Adicionar essa lógica em cada método de cada controller (como `MedicoController`, `PacienteController`, etc.) resulta em código repetitivo e difícil de manter. Uma abordagem mais eficiente é centralizar a validação em um filtro do Spring Security.

4 Filtros no Spring Security

No Spring, as requisições seguem este fluxo:

- **Filtros**: Executados antes do `DispatcherServlet`, decidem se a requisição prossegue ou é interrompida.
- **DispatcherServlet**: Recebe a requisição e identifica o controller apropriado.
- **Handler Interceptors**: Executados após o `DispatcherServlet`, antes do método do controller.

Para validar tokens JWT, usaremos um **filtro** personalizado, que será executado antes de qualquer controller, interceptando todas as requisições para verificar o token.

5 Criando o Filtro de Validação

Na próxima aula, criaremos uma classe de filtro no pacote `med.voll.api.infra.security` para validar tokens JWT. Este filtro:

- Extrairá o token do cabeçalho da requisição (ex.: `Authorization: Bearer <token>`).
- Validará o token usando a biblioteca `java-jwt` e a chave secreta.
- Permitirá ou bloqueará a requisição com base na validade do token.

O filtro será registrado no `SecurityConfigurations` para integrar com o `Spring Security`.

6 Próximos Passos

Compreendemos a necessidade de um filtro para validar tokens JWT de forma centralizada. Na próxima aula, implementaremos o filtro de validação e configuraremos o `Spring Security` para proteger endpoints como `/medicos`. Continue praticando no `Visual Studio Code` ou sua IDE preferida!

7 Dica do Professor

- **Aprofunde-se em Filtros**: Consulte a documentação do Spring Security sobre filtros (`https://docs.spring.io/spring-security/reference/servlet/architecture.html#servlet-filter`) para entender sua integração.
- **Comunidade no X**: Siga perfis como @SpringSecGuru e @APIFilters no X para dicas sobre filtros e segurança em APIs.
- **Pratique**: No Insomnia, crie uma requisição `GET http://localhost:8080/medicos` e adicione o token JWT no cabeçalho `Authorization: Bearer <token>` para simular o comportamento esperado após a implementação do filtro.