Aula 18: Criando um Filtro de Segurança no Spring Security

September 18, 2025

1 Introdução

Na Aula 17, discutimos a necessidade de validar tokens JWT de forma centralizada usando um filtro no Spring Security, evitando a repetição de código nos controllers. Nesta aula, criaremos a classe `SecurityFilter` no pacote `med.voll.api.infra.security` para interceptar requisições antes de chegarem aos controllers. Configuraremos o filtro para permitir a continuação do fluxo de requisições e testaremos seu funcionamento no Insomnia. As alterações serão feitas no `Visual Studio Code` ou sua IDE preferida.

2 Criando o Filtro `SecurityFilter`

No pacote `med.voll.api.infra.security`, criamos a classe `SecurityFilter`, que estende `OncePerRequestFilter` para garantir que o filtro seja executado uma vez por requisição:

```java
package med.voll.api.infra.security;

import jakarta.servlet.FilterChain;
import jakarta.servlet.ServletException;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;
import org.springframework.stereotype.Component;
import org.springframework.web.filter.OncePerRequestFilter;
import java.io.IOException;

@Component
public class SecurityFilter extends OncePerRequestFilter {

    @Override
    protected void doFilterInternal(HttpServletRequest request,
        HttpServletResponse response, FilterChain filterChain) throws
        ServletException, IOException {
        filterChain.doFilter(request, response);
    }
}
```

2.1 Explicação do Código

- `@Component`: Registra o filtro como um componente do Spring, permitindo que seja carregado automaticamente.
- `extends OncePerRequestFilter`: Garante que o filtro seja executado uma vez por requisição, mesmo em cadeias de filtros complexas.
- `doFilterInternal`: Método que processa a requisição. A chamada `filterChain.doFilter(request, response)` permite que a requisição continue para o próximo filtro ou controller.

3 Testando o Filtro

Para verificar o funcionamento do filtro, inicialmente adicionamos um log temporário (`System.out.println("CHAMADO")`) no método `doFilterInternal`. No Insomnia, disparamos a requisição `GET http://localhost:8`

```
[]
```

3.1 Problema Inicial

O retorno foi código 200 (OK), mas sem o JSON esperado (lista de médicos). No console da IDE, o log "FILTRO CHAMADO" apareceu, confirmando que o filtro foi executado. O problema ocorreu porque não chamamos `filterChain.doFilter`, interrompendo o fluxo da requisição.

3.2 Correção

Removemos o log e adicionamos `filterChain.doFilter(request, response)` ao método `doFilterInternal`. Após reiniciar o servidor e disparar a requisição novamente no Insomnia, recebemos o JSON esperado:

```
[
    {
        "id": 1,
        "nome": "Nome do Médico",
        "email": "medico1@voll.med",
        "crm": "123456",
        "especialidade": "ORTOPEDIA"
    },
    ...
]
```

O código 200 (OK) e o JSON confirmam que o filtro está funcionando corretamente, permitindo a continuação do fluxo da requisição.

4 Próximos Passos

Criamos e testamos o filtro `SecurityFilter`, que atualmente permite todas as requisições. Na próxima aula, implementaremos a lógica de validação de tokens JWT no método `doFilterInternal`, extraindo o token do cabeçalho `Authorization` e verificando sua validade com a biblioteca `java-jwt`. Continue praticando no `Visual Studio Code` ou sua IDE preferida!

5 Dica do Professor

- **Aprofunde-se em Filtros**: Consulte a documentação do Spring Security sobre `OncePerRequestFilter` (`https://docs.spring.io/spring-security/reference/servlet/architecture.html#servlet-once-per-request-filter`) para entender sua integração com a cadeia de filtros.
- **Comunidade no X**: Siga perfis como @SpringFilters e @SecurityGuru no X para dicas sobre filtros e segurança em APIs.
- **Pratique**: No Insomnia, teste outras requisições (ex.: `POST /medicos`) e verifique no console da IDE se o filtro é chamado, confirmando que ele intercepta todas as requisições.