Aula 22: Configurando a Ordem dos Filtros no Spring Security

September 18, 2025

1 Introdução

Na Aula 21, configuramos o Spring Security para liberar o endpoint /login e exigir autenticação para outros endpoints, usando o SecurityContextHolder no SecurityFilter para registrar o usuário autenticado. Nesta aula, corrigiremos a ordem de execução dos filtros, garantindo que o SecurityFilter seja executado antes do filtro padrão do Spring (UsernamePasswordAuthenticationFi Testaremos as requisições no Insomnia para confirmar o comportamento correto. As alterações serão feitas no Visual Studio Code ou sua IDE preferida.

2 Testando as Requisições no Insomnia

Testamos as seguintes requisições no Insomnia:

- **Efetuar Login** (POST http://localhost:8080/login):

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

  Retorna código 200 (OK) com o token JWT encapsulado, pois o endpoint está liberado (permitAll()).

- **Detalhar Médico** (GET http://localhost:8080/medicos/id):
  - **Sem token**: Retorna erro 403 (Forbidden), pois o endpoint exige autenticação.
  - **Com token**: Configuramos o token na aba "Auth" (tipo "Bearer Token"), mas ainda recebemos erro 403 (Forbidden).

O erro 403 persiste porque o filtro SecurityFilter não está sendo executado antes do filtro padrão do Spring, que bloqueia requisições não autenticadas.

3 Corrigindo a Ordem dos Filtros

Para garantir que o SecurityFilter seja executado primeiro, atualizamos a classe SecurityConfigurations (pacote med.voll.api.infra.security):

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

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.http.HttpMethod;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import
    org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;
```

```java
import org.springframework.security.config.http.SessionCreationPolicy;
import org.springframework.security.web.SecurityFilterChain;
import
    org.springframework.security.web.authentication.UsernamePasswordAuthenticationFilter;

@Configuration
@EnableWebSecurity
public class SecurityConfigurations {

    @Autowired
    private SecurityFilter securityFilter;

    @Bean
    public SecurityFilterChain securityFilterChain(HttpSecurity http) throws
        Exception {
        return http.csrf().disable()
                .sessionManagement().sessionCreationPolicy(SessionCreationPolicy.STATELESS)
                .and().authorizeHttpRequests()
                .requestMatchers(HttpMethod.POST, "/login").permitAll()
                .anyRequest().authenticated()
                .and().addFilterBefore(securityFilter,
                    UsernamePasswordAuthenticationFilter.class)
                .build();
    }
}
```

3.1 Explicação do Código

- `@Autowired private SecurityFilter securityFilter`: Injeta o filtro personalizado.
- `.authorizeHttpRequests()`: Usa a nova API do Spring Security (substituindo `authorizeRequests`, que está obsoleta).
- `.requestMatchers(HttpMethod.POST, "/login").permitAll()`: Libera o endpoint `POST /login`.
- `.anyRequest().authenticated()`: Exige autenticação para outros endpoints.
- `.addFilterBefore(securityFilter, UsernamePasswordAuthenticationFilter.class)`: Garante que o `SecurityFilter` seja executado antes do filtro padrão do Spring.

4 Testando Após a Correção

Após salvar as alterações e reiniciar o servidor, testamos novamente no Insomnia:

- **Efetuar Login** (`POST /login`): Retorna código 200 (OK) com o token JWT.
- **Detalhar Médico** (`GET /medicos/id`):
  - **Sem token**: Retorna erro 403 (Forbidden).
  - **Com token**: Configuramos o token na aba "Auth" (tipo "Bearer Token"), e agora retorna código 200 (OK) com o JSON esperado:

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

- **Listagem de Médicos** (`GET /medicos`): Com o token configurado, retorna código 200 (OK) e o JSON:

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

5 Próximos Passos

Corrigimos a ordem dos filtros, garantindo que o `SecurityFilter` valide o token JWT antes do filtro padrão do Spring. Com isso, a autenticação e autorização estão funcionando corretamente. Na próxima aula, poderemos explorar melhorias, como tratamento de erros mais robusto ou adição de novos endpoints protegidos. Continue praticando no `Visual Studio Code` ou sua IDE preferida!

6 Dica do Professor

- **Aprofunde-se em Filtros**: Consulte a documentação do Spring Security sobre ordem de filtros (`https://docs.spring.io/spring-security/reference/servlet/architecture.html#servlet-filterchain`) para entender a configuração de cadeias de filtros.
- **Comunidade no X**: Siga perfis como @SpringSecPro e @FilterChainGuru no X para dicas sobre filtros e segurança em APIs.
- **Pratique**: No Insomnia, teste outras requisições protegidas (ex.: `POST /medicos`) com e sem o token no cabeçalho `Authorization` e observe as respostas (200 OK ou 403 Forbidden).