06

Autenticando o cliente via Spring Security

Transcrição

[00:00] Agora que já conseguimos recuperar o token do cabeçalho e fazer a validação, podemos finalmente ir para o último passo. Já que o token está ok, vamos pedir para o Spring considerar que o cliente que está disparando essa requisição está autenticado. E caso o token esteja invalido não vamos chamar esse código para autenticar.

[00:42] Agora, eu preciso fazer um if. Se o token estiver válido, tenho que autenticar o usuário. Se estiver inválido, não vai entrar no if, ele vai chamar o filterChain e vai seguir o fluxo da requisição, barrando o usuário. Só para não ficar com essa lógica, vou criar um método privado também autenticarCliente(token). Vou criar esse método. Agora conseguimos implementar a lógica.

[01:35] Diferente do AutenticacaoController, que utilizamos o authentication manager, no filtro não vou usar isso, porque ele é para disparar o processo de autenticação, com usuário e senha. Aqui não quero fazer autenticação via usuário e senha, porque esse processo já foi feito. Só quero falar para o Spring autenticar o usuário. Para fazer isso, o Spring tem uma classe chamada SecurityContextHolder.getContext().setAuthentication. Esse é o método para falar para o Spring considerar que está autenticado. Só que aí preciso passar os dados do usuário. Então, na hora de passar esse método preciso passar o objeto authentication, que tem as informações do usuário que está logado.

[02:51] Ele recebe um objeto authentication, e preciso criar uma variável do tipo authentication, é aquela mesma variável que usamos no

AutenticacaoController. Podemos dar um new, mas aí vamos usar outro construtor. Em um passo os dados do usuário, as credenciais, que vai ser nulo, porque já validei antes, e preciso passar também os perfis de acesso dele.

[03:55] Só vai ter um probleminha, porque não temos a variável usuário. Para dar new no authentication preciso do usuário. De onde ele veio? Precisamos recuperar. A única informação que tenho na mão é o token. E não sei se vocês lembram, mas na classe TokenService, quando escrevemos aquele código para gerar o token, nós setamos uma linha setSubject(logado.getId().toString()). Dentro do token, temos o id do usuário. Se eu tenho o id, consigo carregar o objeto do banco de dados.

[04:41] Eu vou fazer o seguinte. Estou com o token, dentro do token vou ter o id do usuário, preciso recuperar esse id, com algo como Long idUsuario = TokenService.getIdUsuario(token). Vou criar mais esse método no TokenService, em que passo um token e ele me devolve um id que está dentro desse token.

[05:21] Vou pedir para o Eclipse gerar o método para mim. E aí, como faço para recuperar os dados do token? É parecido com o método isTokenValido. Vamos precisar desse código, da classe que faz o parser do token. Vou colocar jwts.parser().setSigningKey(this.secret).parseClaimsJws(token). Esse objeto tem um método na sequência chamado getBody, que devolve o objeto do token em si. Vou chamar esse método e vou pedir para ele guardar isso numa variável local.

[06:08] Ele devolve esse objeto do tipo claims, que é uma classe também da API do JSON web token. Na próxima linha, dentro desse objeto claims, tenho vários métodos get para recuperar as informações do token. Na hora em que geramos o token, setamos o subject. Então aqui consigo chamar o getSubject, para pegar o id de volta.

[06:42] Só que dentro do token tudo é string, mas eu não quero isso como string, quero como long. Mas eu sei que é um long que está lá dentro, que é o id

do usuário, então vou fazer um parse aqui. Com isso eu consigo recuperar o id do usuário que está setado dentro do token.

[07:04] Vou voltar para o meu filter. Agora já tenho o id do usuário. Só que eu preciso do objeto usuário completo. Na sequência, preciso carregar esse usuário do banco de dados. Para carregar do banco de dados tem o repository. Mas não tenho acesso a ele. Vou ter que declarar mais um atributo. Vou chamar a variável de repository. Tem aquele problema de não conseguir injetar. Vou ter que passar esse parâmetro para o construtor, na hora de dar new no nosso filtro. Lembre-se que quem está dando new nesse objeto é a classe security configurations.

[08:00] Eu não tenho essa variável, mas consigo injetar. Vou declarar mais um atributo na classe de configuração, que é o usuarioRepository. Só não posso esquecer o @AutoWired, senão o Spring não vai injetar.

[08:17] Injetei o usuarioRepository, dei o new. No meu filter, quando dei new, eu injetei no construtor o token service e o usuarioRepository. Agora, consigo recuperar o objeto usuário da seguinte maneira: usuário = repository.getOne(idUsuario). Pronto, recuperei o usuário. É ela que passo como parâmetro. Vou salvar, e agora está tudo compilando.

[09:33] Peguei o id do token, recuperei o objeto usuário passando o id, criei o usernameauthenticationtoken passando o usuário, passando nulo na senha, porque não preciso dela, passando os perfis, e aí por fim chamei a classe do Spring que força a autenticação. Essa autenticação é só para esse request. Na próxima requisição ele vai passar no filter de novo, pegar o token e fazer todo o processo. A autenticação é stateless. Em cada requisição eu reautentico o usuário só para executar aquela requisição.

[10:10] No nosso método principal do filter só chamo o autenticar se o token estiver válido. Se não estiver, não vai autenticar, vai seguir o fluxo da requisição e o Spring vai barrar. Agora está tudo implementado. Vamos fazer nosso teste

[10:23] Vou até o Postman e vou testar tudo de novo, do zero. Vou fazer o localhost:8080/auth, método POST. Quero autenticar. Vou primeiro fazer o login. No body vou colocar raw. Vou passar o JSON com e-mail e senha. Vou disparar essa requisição, que é a primeira que o cliente vai fazer, para se autenticar. Disparou, devolveu 200 e voltou o token. Vou copiar o token, criar uma nova requisição do tipo DELETE para o tópico 1. Vou ver o que vai acontecer. Ele me dá o 403.

[11:31] Agora vou na aba authorization e vou colocar que a autorização é via bearer token, vou passar o token, mas vou trocar um caractere, vou passar um token inválido. Ele me dá um 403. Vou passar o token válido e ele me deu inválido, porque não posso usar o getOne, tem que ser o findById. O getOne não carrega o objeto na JPA, só traz o proxy, na hora de carregar os perfis deu erro.

[12:51] Testando de novo, ele me dá o código 200. Consegui deletar o tópico. Finalizamos nossa autenticação utilizando token, com aquela biblioteca jjwt do Java integrada ao Spring security. É um pouco chata essa parte, porque tivemos que criar o filter, e tem todo o processo de pegar o token do cabeçalho, validar, forçar autenticação via Spring security, registrar o filtro no security Configuration, injetar os parâmetros para passar para o filter, porque no filtro não posso receber via injeção de dependências, teve que criar o AutenticacaoController, o token service. É muito código, é chato, mas é aquele código que você faz uma vez só. E é um código de infraestrutura, não estou implementando uma regra do projeto. Você não precisa decorar, saber de cabeça. Você pega um de exemplo e joga no projeto, fazendo adaptações.

[14:24] Com isso, conseguimos fazer nossa autenticação de maneira stateless, sem usar sessão, seguindo a ideia do modelo REST. A API REST não guarda estado de autenticação, então a cada request eu mando o token, reautentico. Fica mais leve, mais fácil de ter escalabilidade na nossa API. Por hoje, esse era o assunto. Na próxima aula vamos ver sobre monitoramento, uma coisa importante que preciso ter em uma API REST, ainda mais se eu estiver trabalhando com aquele modelo de arquitetura de micro serviços.