02

Recuperando o token do header Authorization

Transcrição

[00:00] Na última aula vimos como fazer a geração do token, usando a biblioteca do JSON web token, e criamos nosso controller para fazer a parte de autenticação, que recebe a requisição do cliente com o e-mail e a senha, valida no banco de dados se os dados estão corretos, e se estiver gera o token devolvendo dentro do dto.

[00:34] A primeira coisa que o cliente vai fazer é disparar uma requisição do tipo POST para aquele endereço /auth, que é o endereço de autenticação. E no corpo da requisição ele tem que levar o JSON com e-mail e senha. Eu vou disparar a requisição e no retorno vai voltar como resposta o token e o tipo de autenticação que o cliente vai ter que fazer.

[01:33] A partir dessa requisição, o cliente da API é responsável por pegar o token que voltou na resposta e armazenar em algum lugar. Se for uma aplicação frontend, geralmente ele vai usar um cookie ou um session storage, ou outro lugar em memória. E nas próximas requisições que ele disparar para a nossa API para recursos protegidos ele vai ter que levar esse token. Como ele faz isso?

[02:01] Vou mostrar um exemplo. Imagine que ele acabou de se autenticar e pegou o token. Agora quero chamar aquele endpoint DELETE para /tópicos/2. Quero excluir o tópico de id 2. Configuramos isso para ser restrito. Se eu disparar a requisição sem passar nada, vou receber um 403. Para ele disparar a requisição, ele tem que adicionar mais um cabeçalho, que é o authorization. A ideia é colocar o cabeçalho, com o valor bearer, e o token. Nas próximas

requisições, o cliente tem que sempre mandar o cabeçalho authorization com o token. Se eu disparar a requisição agora, também vai dar 403, porque ainda não implementamos a lógica. Esse vai ser o objetivo dessa aula.

[03:17] Vamos voltar para o nosso projeto. A ideia é que essa lógica tem que rodar antes de cair no meu controller, no meu código. Tenho que interceptar a requisição e executar a lógica. Para fazer isso, vamos criar um filtro. Vou clicar no nosso pacote security, vou criar uma classe, chamar de autenticaçãoViaTokenFilter. No caso, ela vai ser um filter.

[03:55] No Spring, podemos herdar essa classe de outra chamada OncePerRequestFilter, que é um filtro do Spring chamado uma única vez a cada requisição. Dá um erro, porque tem um método abstrato que precisamos implementar. Vou adicionar esse método, chamado doFilterInternal.

[04:15] O que temos que fazer aqui nesse método? Temos que colocar nossa lógica para pegar o token do cabeçalho, verificar se está ok, autenticar no Spring, e a última linha que tem que ter nesse método precisa pegar o terceiro parâmetro, que é o filterChain.doFilter, passando request, response.

[04:40] Essa é a linha para falar para o Spring que já rodamos o que tínhamos que rodar nesse filtro. Só que antes vou precisar recuperar o token no cabeçalho, validá-lo, e se estiver ok autentificar o usuário para o Spring. Eu tenho que autenticar porque como a nossa autenticação é stateless, não existe mais a ideia de usuário logado. Então, quando o cliente disparou a requisição para se autenticar, passando e-mail e senha, eu gerei um token, devolvi para ele, mas eu não guardei esse token no lado do servidor. Minha API não sabe se aquele usuário está logado. Ela tem os tokens que foram gerados, mas ela não sabe quem está ou não está logado. Na próxima requisição que esse cliente disparar, nossa API não tem como saber se esse cliente está autenticado ou não, porque não existe esse conceito de autenticação.

[05:43] A autenticação é feita para cada requisição. Toda requisição que chega para a nossa API, nós pegamos o token, autenticamos o usuário, se estiver ok,

token, executamos a requisição, devolvemos a resposta, acabou o assunto. Na próxima requisição a API nem lembra mais quem é esse cliente. Tenho que pegar o token do cabeçalho, autenticar de novo, rodar o request e devolver a resposta. Por isso é por requisição.

[06:19] Eu preciso pegar o token. O primeiro passo da nossa lógica é recuperar o token do cabeçalho. Para fazer isso, vou criar uma variável string token = recuperarToken. Nesse método passo o request, e ele já me devolve o token. O request tem um método chamado getHeader. Nele, passo o nome do cabeçalho que quero recuperar. No caso, vimos que é authorization.

[07:17] Vou guardar isso em uma variável. Só que aí pode ser que esse cabeçalho não esteja vindo, que ele não esteja nesse formato. Preciso verificar se está correto ou não. Vou fazer um if, se o token for igual a nulo, ou se token está vazio, ou se o token não começa com bearer, eu vou devolver nulo. É só para verificar se está vindo o token, se não é vazio, e se começa com bearer. Se nenhuma dessas condições forem ok, eu devolvo nulo. Senão, devolvo o token. Só que se eu devolver o token assim vai o conteúdo inteiro. Mas eu não quero esse começo. Quero só o token em si, então vai ser token.substring(7, tolen.length). Sete porque vai começar a pegar a partir do espaço até o final da string, que é o conteúdo do token.

[09:00] Devolvi isso. O próximo passo é validar o token, ver se ele está correto. Aí vamos ter que usar nossa biblioteca, o jjwt, para fazer essa validação. Se estiver ok, o terceiro passo é falar para o Spring autenticar o usuário. Mas como são vários passos e é muito código, vamos quebrar isso e fazer aos poucos. Neste vídeo, só vamos pegar o token e imprimir no console para ver se está chegando corretamente.

[09:40] Vamos testar no Postman, mandando o cabeçalho bearer e o token que foi devolvido anteriormente. Não funcionou porque tem mais um detalhe. Eu só criei o filter, mas não tem anotação nenhuma. Embora eu tenha herdado de uma classe, ele não registrou. Precisamos registrar esse filtro para o Spring.

[10:10] Para fazer isso, não é via anotação. Tem que ser na nossa classe Security Configuration. No nosso método configure, que tem as URLs, depois que eu configurei que a autenticação é stateless, vou colocar mais uma sentença, o addFilter. Só que não posso chamar isso, porque o Spring internamente já tem o filtro de autenticação. Ele precisa saber qual a ordem dos filtros, quem vem antes. Por isso, tem que ser o método addFilterBefore. Passo para ele quem é o filtro que quero adicionar e antes de quem esse filtro virá. Depois, damos um new AutenticacaoViaTokenFilter(), UsernamePasswordAuthenticationFilter.class. Esse é o token que já tem no Spring por padrão. Vou falar para o nosso filtro rodar antes dele.

[11:22] Vou salvar. E agora volto no Postman, disparo a requisição. Veio exatamente o token certinho. Vou fazer outro teste mandando o cabeçalho vazio. Disparo a requisição. Ele imprime nulo. Está funcionando certinho.

[12:06] Por hoje, é isso. Na próxima aula vamos continuar nossa implementação, porque agora que já consegui recuperar o token, na próxima aula vamos validá-lo com nossa biblioteca, verificar se o token que está chegando está correto, se é um token válido que foi gerado pela nossa aplicação.