03

Configurando autenticação Stateless

Transcrição

[00:00] Como já discutimos no último vídeo, uma das principais características do modelo REST é que precisamos fazer a comunicação de maneira stateless. Nossa API não está seguindo esse princípio, por conta da autenticação. Estamos usando a autenticação da maneira tradicional com uso de sessão. Toda vez que um usuário se autentica, criamos uma sessão, que nada mais é que um espaço na memória onde estou guardando estado. Isso é ruim, porque se eu tiver zilhões de usuários para cada um vou ter que ter espaço na memória para guardar os dados de autenticação.

[00:37] Na aula de hoje vamos aprender a como não utilizar sessão, e fazer essa parte de autenticação de maneira stateless, seguindo as boas práticas do modelo REST. Para fazer isso, vamos utilizar o JSON web token, uma tecnologia para geração de tokens e autenticação de maneira stateless.

[00:56] Para começar, precisamos usar alguma biblioteca em Java que siga esse modelo do JSON web token. Vamos utilizar uma biblioteca chamada jjwt. O primeiro passo é adicionar essa dependência no nosso projeto.

[01:33] O group Id agora vai ser io.jsonwebtoken, e o artifact é jjwt. Como essa é uma dependência que não vem do Spring Boot, também vamos precisar declarar a versão. No nosso caso, vai ser a 0.9.1. Eu recomendo você a usar essa versão para garantir que não vai dar problemas.

[02:08] Já com a biblioteca no projeto, precisamos configurar a parte de segurança, naquela classe securityConfigurations, para informar para o Spri security que não vamos mais fazer autenticação usando sessão, mas sim

autenticação de maneira stateless. Isso fazemos no método configure, onde estamos configurando as URLs. Embaixo, depois que configuramos as duas URLs e disse que todas as outras precisam estar autenticadas, vamos tirar a parte de and().formLogin(). Não vamos mais usar o formulário de login tradicional, porque senão ele vai criar sessão.

[02:58] Como estamos fazendo autenticação via token, via JSON, precisamos fazer outra configuração. Precisamos chamar o método csrf.disable. Csrf é uma abreviação para cross-site request forgery, que é um tipo de ataque hacker que acontece em aplicações web. Como vamos fazer autenticação via token, automaticamente nossa API está livre desse tipo de ataque. Nós vamos desabilitar isso para o Spring security não fazer a validação do token do csrf.

[03:30] Na sequência, sessionManagement. Vamos chamar aqui um método para dizer que não queremos usar sessão. Nós passamos como parâmetro essionCreationPolicy.STATELESS. Com isso, aviso para o Spring security que no nosso projeto, quando eu fizer autenticação, não é para criar sessão, porque vamos usar token.

[04:08] Mas tem uma desvantagem. Como apagamos aquela linha do formLogin, a partir de agora não temos mais um formulário de login gerado pelo Spring. Isso faz sentido porque as páginas da nossa aplicação não vão ficar na API backend. Vão ficar na aplicação client no frontend. O formulário de login é o cliente que vai fornecer. Ele só vai chamar nossa API passando os dados de login e senha. Só que além de perder o formulário, também perdemos o controller, que fazia a parte de autenticação. Vamos ter que criar um.

[04:50] Vamos criar uma nova classe e chamar de AutenticacaoController. Esse é o controller onde vai estar a lógica de autenticação. Para dizer que é um controller do Spring, coloco o @RestController e @RequestMapping. Precisamos configurar qual vai ser a url que esse controller vai responder. Vou colocar /auth. Se chegar alguma requisição para /auth, o Spring sabe que é pa chamar esse controller, porque é o endereço para autenticação.

[05:41] Preciso ter um método onde vai ter a lógica de autenticação. Vou criar um método public que devolve o ResponseEntity, vou chamar o método de autenticar. Depois vemos que o vamos receber de parâmetro. E esse método vai ser chamado se a requisição for /auth, e via método @PostMapping. Como é autenticação, estou recendo parâmetros de usuário e senha, preciso que seja via método post.

[06:12] Esse método vai ser chamado pelo cliente quando ele solicitar os dados de login e senha. A aplicação cliente tem que configurar para chamar esse endereço /auth, a requisição tem que ser via método post, e no corpo da requisição ele tem que mandar o e-mail e a senha. Precisamos receber então o e-mail e a senha. Também quero validar. Vamos fazer validação via bin validação, para garantir que está vindo e-mail e senha. Mas lembre-se que não vou receber os dados soltos e não vou receber uma classe de domínio. Vou receber um form, seguindo o padrão dto.

[07:27] Seguindo aquele padrão, os dados que chegam do cliente eu recebo numa classe form. Não tem uma classe loginform, preciso criar. Dentro, vou ter só dois atributos, string e-mail e string senha. Quando o cliente fizer a chamada para a nossa API vai ter que mandar dois parâmetros no formato JSON, o e-mail e a senha.

[08:00] Precisa gerar os setters. E está pronta a lógica para gerar o form. Voltando para o meu controller, o cliente vai chamar /auth via método post, vai mandar um JSON com e-mail e senha, que vou receber dentro do form. Nesse primeiro momento, só para não ficar muita coisa, ainda não vou fazer a geração do token. Só vou fazer um system out para saber se está chegando a senha e o e-mail corretos.

[09:12] Preciso retornar alguma coisa. Então, ResponseEntity.ok.build. Só para testar se está ok. A ideia é que no meio eu vou precisar pegar o e-mail e a senha, autenticar no sistema, se estiver tudo ok, gero o token. Mas essa parte vamos fazer depois.

[09:47] Vamos testar se essa parte está funcionando. Vou rodar meu projeto. No Postman, vou mandar o JSON. Disparando a requisição, voltou o 403, porque esqueci de liberar o /auth. Lembre-se que precisamos liberar todas as URLs novas do projeto.

[11:27] Testando novamente, veio ok. Já criei esse novo endpoint para fazer autenticação, e criamos o controller que vai responder esse endpoint. Tudo funcionou certinho. No próximo vídeo temos que fazer a autenticação. Temos que pegar esse e-mail e essa senha, autenticar no sistema, gerar o token e devolver como resposta.