05

# Gerando tokens com JWT

## Transcrição

[00:00] Já implementamos o controller com a parte de autenticação, que recebe os dados de autenticação de usuário e senha. O próximo passo é autenticar no Spring e fazer a geração do token. Como não estamos mais usando aquele formulário do Spring, que já tinha um controller que fazia a chamada para a parte de autenticação, vamos ter que fazer a chamada manualmente.

[00:28] Para fazer uma autenticação de maneira programática, manualmente, no Spring security, vamos precisar de uma classe chamada authentication manager. Eu preciso injetar no nosso controller. Vou declarar um atributo que vai ser injetado, do tipo authenticationManager. E vou chamar o atributo de authManager.

[00:55] Porém, tem um pequeno detalhe. Essa classe é do Spring, mas ele não consegue fazer a injeção de dependências dela automaticamente, a não ser que nós configuremos isso. Por algum motivo, ela não vem configurada. Podemos fazer isso na nossa classe SecurityConfiguration. Para a nossa sorte, na classe que estamos herdando já tem um método que sabe criar o authenticationManager. Basta sobrescrevermos esse método.

[01:40] A única coisa que vamos precisar fazer é colocar @Bean em cima do método, porque aí o Spring sabe que esse método devolve o authenticationManager e conseguimos injetar no nosso controller. Vou salvar e voltar para o controller.

[01:57] O próximo passo é pegar no nosso método o authManager e chamar o método authenticate, que vai fazer a autenticação. Para fazer a autenticação,

preciso passar para ele os dados de login. Só que não posso passar o loginform ou a senha soltos. O método authenticate recebe um objeto específico. Na linha de cima vamos precisar criar esse objeto. É um objeto do tipo userNamePasswordAuthenticationToken. Vou chamar de dadosLogin. Tenho que dar new nesse objeto passando para ele o login e a senha. Para não deixar isso no controller, vamos fazer aquele esquema que já tínhamos feito no tópico controller. Vamos pegar o próprio form e criar um método converter.

[02:59] Var dar um erro porque não tem o método converter. Vou pedir para o Eclipse criar para mim. Ele recebe como parâmetro o e-mail e a senha. Essa é a classe que o Spring precisa para o authenticationManager. Vou voltar para o controller, agora o authenticate compila, e esse método devolve um objeto do tipo authentication.

[03:33] Quando o Spring chamar essa linha, do authManager.authenticate, o Spring vai olhar as configurações que fizemos e ele sabe que é para chamar a classe authentication services, que chama o usuário repository para consultar os dados do banco de dados. Se der certo, ele vem para a linha de baixo. Se der errado, vai dar um exception. Se der exception, não quero devolver ok ou erro 500. Quero devolver erro400.

[04:08] Eu vou fazer um tratamento, um try catch, vou mover as duas linhas para dentro do try. Tenta fazer a autenticação. Se deu certo, devolve ok. Só que aí não vai ser exception, vai ser authenticationException. Se der erro, retorne badRequest. Com isso já implementei a parte de autenticação. O cliente vai mandar o usuário e a senha, eu chamo o authenticationManager para ele disparar o processo de autenticação. Se estiver tudo ok, ele vai devolver o ok. Senão, vai cair no catch.

[04:56] Só que eu não quero devolver um ok vazio. Quero devolver o token. Antes de devolver o ok, preciso fazer a geração do token, e vou guardá-lo em uma string. Para gerar o token é que vamos usar a biblioteca JWT, que até então só tínhamos colocado no projeto e não utilizado.

[05:17] Mas para não deixar esse código solto, vamos isolar em uma classe service. Vamos criar uma classe chamada TokenService e vamos ter o método gerar token. Na hora de gerar o token vou precisar identificar quem é o usuário, para qual usuário pertence aquele token. Então, nesse método, vou passar o authentication como parâmetro.

[05:47] Dá erro de compilação porque não existe essa variável. Eu vou criar mais um atributo no meu controller do tipo TokenService. Eu quero que o Spring injete para mim. Só que essa classe não existe, vou pedir para ele criar para mim e vou jogar no pacote configSecurity. Nesse pacote vai ter essa classe, que vai ser um service.

[06:21] Voltando para o meu controller, dá erro porque não existe esse método. Vou pedir para ele gerar para mim. Ele vai criar. E aí vou colocar a API do JWT para fazer a geração do token. Ele tem um método que cria um objeto builder onde posso setar informações para ele construir o token.

[06:52] Precisamos acertar alguma coisas. A primeira coisa vai ser o issuer. Quem é que está gerando o token. Vou colocar que foi a API do fórum da Alura, porque aí o cliente consegue identificar quem foi que fez a geração.

[07:17] Outra coisa que preciso setar é o usuário, quem é o dono desse token, quem é o usuário autenticado a quem esse token pertence. E tenho que passar uma string que identifique unicamente meu usuário.

[07:32] Lembra que esse método eu passei como parâmetro authentication? Esse authentication tem um método chamado getPrincipal para conseguirmos recuperar o usuário que está logado. Eu vou jogá-lo em uma variável usuário, que vou chamar de logado. Vai dar um erro de compilação, porque o getPrincipal devolve um object, então tenho que fazer um cast para usuário.

[07:59] No subject, vou colocar logado.getId e vou passar o id. Mas o id precisa que seja string. Também preciso dizer qual foi a data de geração do token. Quando ele foi concedido. E ele trabalha usando a API de datas antigas do Jav…

então ele está esperando um date. Eu vou criar uma variável ali em cima de date e vou importar do Java.

[08:45] O token também tem uma data de validação, onde ele vai expirar, igual a sessão tradicional, para não ficar infinito, porque isso seria um risco de segurança. E tenho que passar uma data. Eu poderia pegar a data hoje, somar com trinta minutos, mas esse tempo, para não ficar no código, vou injetar em uma propriedade do application.properties chamada fórum.jwt.expiration= e passei um tempo em milissegundos. Coloquei um dia, só para ficar mais fácil no teste. Na prática, o ideal é que o tempo seja menor.

[10:12] Tem uma senha também que depois vou mostrar onde vamos utilizar. Voltando ao TokenService. Tendo essa propriedade, como injetar? Vou declarar um atributo. Para injetar não vai ser um Autowired, porque ele procura um bean que está configurado, e no caso é uma string. Para injetar coisas, parâmetros do Application.properties, usamos a anotação @Value, e ela recebe como parâmetro o nome da propriedade.

[10:56] O Spring vai no application.properties, vai ver quem é a propriedade, pegar o valor dela e injetar aqui. Já vou aproveitar e injetar aquela senha. Vou jogar em um atributo chamado secret.

[11:15] A data de expiração não é o expiration, porque o expiration é o tempo em milissegundos. Tenho que pegar aquele tempo e somar com a data atual. Então, vou criar outro date. Posso passar como parâmetro hoje.getTime, para pegar os milissegundos da variável hoje, e somar com a variável expiration, porque aí ele vai fazer a soma dos dois milissegundos e criar a nova data. É justamente esse data expiração que vou passar como parâmetro.

[11:57] A última coisa que precisamos setar é: tem um método chamado signwith. Pela especificação JSON webtoken, o token tem que ser criptografado. Preciso dizer para ele quem é o algoritmo de criptografia e a senha da minha aplicação, que é usada para fazer a assinatura e gerar o REST da criptografia do token. É aqui que vou injetar o secret, que está no

application.properties. A ideia é pegar um programa que gera uma string aleatória gigantesca e usar como senha. Para não ficar isso no código, podemos colocar no application.properties.

[12:45] Só preciso dizer qual é o algoritmo, e aí existe a classe signatureAlgorithm.HS256. No final, um compact para ele compactar e transformar isso em uma string. Ele dá um erro porque está fazendo operações com o date, da API de datas do Java antigo, mas esse é o jeito mais simples, não vai ter nenhum problema.

[13:27] Terminei. Consegui fazer a lógica para gerar o token. Para testar, vamos no controller. Já chamei o service e mandei ele criar um token. Vou dar um system out só para ver se está gerando o token certo. Para testar, vou usar o Postman. Vou disparar a requisição POST. Deu algum erro no nosso projeto na hora de pegar a data.

[14:27] O expiration estava como string, tem que ser um long. Agora voltou 200, deu o system out. No próximo exercício, vamos ver como no controller podemos fazer para devolver esse token, porque aqui eu só fiz o system out para ver se está tudo ok.