05

Autorização baseada em Roles

Transcrição

[00:00] Olá, bem-vindos de volta ao treinamento de Spring Boot. Na última aula aprendemos a atualizar a versão do Spring Boot do nosso projeto e atualizamos da versão 2.1.4 para a 2.3.1. E agora o projeto está atualizado, está tudo funcionando, compilando corretamente.

[00:18] Nessa aula vamos aprender um pouco mais sobre segurança, sobre o conceito de autorização que nós não vimos no último curso. Não sei se vocês lembram, mas no segundo curso de Spring Boot aprendemos como colocar segurança na aplicação, para cuidar da parte de autenticação e autorização. No `pom.xml`, se descermos um pouco podemos ver que tem a dependência do Spring Security.

[00:41] Nós tivemos que colocar essa dependência do Spring Security, também tivemos que criar a classe `SecurityConfigurations`, onde fizemos todo o processo de configuração de como funcionaria nossa autenticação, e a parte de autorização que acabamos deixando de lado.

[00:58] Tivemos que criar a classe `Usuario` e a classe `Perfil`, para representar o perfil do usuário. Fizemos todo aquele mecanismo de autenticação, no começo fazendo autenticação tradicional, com sessão, que era aquela autenticação utilizada em aplicações web tradicionais.

[01:13] Mas depois vimos que não era uma boa prática essa autenticação tradicional com sessão em uma API Rest. É justamente o que desenvolvemos no curso, uma API Rest do fórum da Alura.

[01:23] Então migramos todo o código para utilizar aquele esquema de autenticação *stateless*. Então usamos a biblioteca JJWT para fazer a geração de Tokens.

[01:35] Quando o usuário quer acessar o fórum da Alura, quer se autenticar, primeiro ele entra no endereço/auth, aquele *endpoint* para se autenticar; passa o e-mail e a senha. Se tiver os dados corretos nós devolvemos um Token e a aplicação *frontend* dessa pessoa tem que guardar esse Token e nas próximas aquisições sempre enviar esse Token como um cabeçalho.

[01:59] E tivemos que criar aquela classe que faz toda essa lógica de recuperar esse Token do cabeçalho, validar, forçar uma autenticação durante essa requisição e liberar a requisição para o usuário.

[02:13] Então fizemos todo esse trabalho utilizando Spring Security e a biblioteca para gerar Tokens JJWT.

[02:19] Só que em todo esse processo nós focamos basicamente na parte de autenticação. Eu queria restringir algumas URL's. Na Security Configurations, tem algumas URL's que estão públicas para listar os tópicos, para detalhar um tópico, para se autenticar. Mas todo o resto está restrito. Só que a minha preocupação era toda nessa parte de autenticação.

[02:41] Então antes de executar a lógica, a regra de negócio, eu precisava verificar se o usuário estava logado, se ele tinha o Token, se estava autentificado. E deixamos de fora a parte de autorização.

[02:53] E nem sempre numa aplicação basta a pessoa estar logada. Às vezes têm regras de negócio que envolvem perfil de acesso. Você está autentificado, mas você tem permissão para executar essa ação no sistema? Então foi isso que ficou faltando e é justamente o que veremos na aula de hoje.

[03:10] Então o que vamos fazer aqui no caso do nosso projeto? Eu vou abrir a classe TopicosController, que é onde estão todas funcionalidades referentes a

tópicos. E a última que fizemos é aquela parte de excluir, de remover.

[03:24] Então uma das funcionalidades é que eu queria permitir que o usuário pudesse excluir um determinado tópico. Só que do modo como está implemetado nós não fazemos nenhuma verificação de autorização. Basta que a pessoa esteja autenticada no sistema para ela conseguir excluir um tópico.

[03:41] Vamos imaginar que a regra não é desse jeito, vamos pensar que a regra vai funcionar da seguinte maneira: só pode excluir um tópico se a pessoa tiver o perfil de moderador.

[03:53] Então imagina que tem dois perfis no fórum da Alura: tem o perfil de aluno, que é qualquer aluno que vai cadastrar os tópicos, responder e ajudar as outras pessoas; e tem o perfil de moderador, que é aquela pessoa que controla, que organiza e ajuda os alunos que estão com dificuldades na plataforma.

[04:10] Então vamos considerar que nem o próprio aluno pode excluir o tópico. Ele pode marcar como solucionado, ou ele pode fechar o tópico para não receber mais perguntas. Mas vamos considerar que ele não tem permissão para excluir. Só quem tem perfil de moderador pode fazer isso.

[04:27] Para implementar essa regra, na verdade não vamos mexer no Controller, não precisamos fazer um `if else` . Nós fazemos isso direto na classe `SecurityConfigurations` , no mesmo local onde configuramos as URL's do projeto.

[04:47] Então vamos adicionar mais uma URL. Acima estão todas as URL's que estão públicas. Abaixo de `.anyRequest().authenticated()` tem que estar autenticado. Mas não apenas isso tem que ter o perfil de moderador no caso da exclusão.

```
//Configuracoes de autorizacao
@Override
protected void configure(HttpSecurity http) throws Exception
```

```
{
    http.authorizeRequests()
    .antMatchers(HttpMethod.GET, "/topicos").permitAll()
    .antMatchers(HttpMethod.GET, "/topicos/*").permitAll()
    .antMatchers(HttpMethod.POST, "/auth").permitAll()
    .antMatchers(HttpMethod.GET, "/actuator/**").permitAll()
    .anyRequest().authenticated()
    .and().csrf().disable()

.sessionManagement().sessionCreationPolicy(SessionCreationPoli

    .and().addFilterBefore(new
AutenticacaoViaTokenFilter(tokenService, usuarioRepository),
UsernamePasswordAuthenticationFilter.class);
}
```

COPIAR CÓDIGO

[05:01] Eu vou copiar a última linha de URL, vou colar logo abaixo e modificar para: `.antMatchers(HttpMethod.DELETE, "/topicos/*")` . A requisição é `DELETE` , que configuramos no Controller e é `/tópicos/` seguida de alguma coisa, que no caso é o id.

```
//Configuracoes de autorizacao
@Override
protected void configure(HttpSecurity http) throws Exception
{
    http.authorizeRequests()
    .antMatchers(HttpMethod.GET, "/topicos").permitAll()
    .antMatchers(HttpMethod.GET, "/topicos/*").permitAll()
    .antMatchers(HttpMethod.POST, "/auth").permitAll()
    .antMatchers(HttpMethod.GET, "/actuator/**").permitAll()
    .antMatchers(HttpMethod.DELETE,
"/topicos/*").hasRole("MODERADOR")
    .anyRequest().authenticated()
    .and().csrf().disable()
```

```
.sessionManagement().sessionCreationPolicy(SessionCreationPoli

    .and().addFilterBefore(new
AutenticacaoViaTokenFilter(tokenService, usuarioRepository),
UsernamePasswordAuthenticationFilter.class);
}
```

COPIAR CÓDIGO

[05:15] Se olharmos no TopicosController é `@DeleteMapping("/{id}")`. Então eu quero configurar isso.

[05:22] Só que na verdade não é `permitAll`, senão eu estou liberando. Para restringir o perfil existe outro método, chamado `hasRole`. E dentro eu passo qual é o Role, qual é o perfil do usuário.

[05:35] Então só pode executar uma requisição do tipo de DELETE para o endereço `/topicos/*` se tiver o perfil que eu vou colocar o nome. Você pode chamar de qualquer coisa, eu vou chamar de moderador. E pronto: `.antMatchers(HttpMethod.DELETE, "/topicos/*").hasRole("MODERADOR")`.

[05:48] Só isso, não precisa fazer mais nada. Então o Controller nem vai ver que tem essa regra. Na verdade, o Spring Security faz esse controle para nós.

[05:56] Quando chamarmos o endereço `@DeleteMapping("/{id}")`, o Spring verifica o perfil da pessoa que está logada. Se for moderador, então ele entra e executa linha por linha. Se não for moderador ele nem entra, ele devolve um 403, um *forbidden* como resposta. Então isso deixa o nosso código do Controller bem limpo.

[06:15] E agora já podemos testar. Porém, temos um problema: nós não tínhamos essa questão de perfil no projeto. Não sei se vocês lembram, eu vou abrir aquele arquivo "data.sql", onde tem alguns *inserts*; toda vez que rodamos o projeto, o Hibernate faz esses *inserts* no nosso banco de dados, o H2.

[06:36] Veja que só temos um usuário, e não tem esse negócio de perfil. Então para fazer esse teste eu vou copiar a primeira linha e vou criar outro usuário, mas com o nome de Moderador, o e-mail vai ser `moderador@email.com e a senha eu vou deixar a mesma do aluno, que é 123456`.

[06:57] Só que não é só isso, eu estou criando o usuário, mas e o perfil? Qual é o perfil do aluno e qual é o perfil desse segundo usuário? Lembra que além da classe Usuario nós criamos uma classe chamada Perfil, que é uma entidade da JPA. Então tem uma tabela de perfil, nós só não estávamos utilizando; essa tabela estava vazia, sem nenhum registro.

[07:17] Agora precisamos utilizar essa tabela. Então vou colocar mais um *insert* abaixo do que acabamos de fazer: `INSERT INTO PERFIL` , vou inserir um registro na tabela de perfil; só tem duas colunas, que é o id e o nome, então `INSERT INTO PERFIL(id, nome) VALUES()` .

[07:31] O primeiro perfil vai ser o aluno, e o Spring Security tem um padrão, em que o nome do perfil tem que ter o prefixo `ROLE_` . Então `INSERT INTO PERFIL(id, nome) VALUES(1, 'ROLE_ALUNO');` .

[07:42] E além do aluno eu vou ter outro perfil que vai ser o Moderador, então `INSERT INTO PERFIL(id, nome) VALUES(2, 'ROLE_MODERADOR');` .

[07:51] Pronto, agora tenho dois perfis no sistema, o `ROLE_ALUNO` e o `ROLE_MODERADOR` .

[07:57] Porém, agora eu preciso associar com esses dois usuários. Essa associação é feita numa tabela de JOIN. Então além da tabela usuário e da tabela perfil, tem uma tabela chamada `USUARIO_PERFIS` . É "perfis" porque esse é o nome do atributo na classe Usuário.

[08:16] Essa tabela tem duas colunas, a `usuario_id` e a `perfis_id` , então fica `INSERT INTO USUARIO_PERFIS(usuario_id, perfis_id);` .

[08:25] E depois colocamos os valores: eu quero jogar no usuário de id 1, que é o aluno, o perfil 1, então `INSERT INTO USUARIO_PERFIS(usuario_id, perfis_id) VALUES(1, 1);` .

[08:32] Vou duplicar essa linha e agora eu quero jogar no usuário de id 2, que é o moderador, o perfil de moderador, então `INSERT INTO USUARIO_PERFIS(usuario_id, perfis_id) VALUES(2, 2);` .

[08:39] Pronto, com isso eu já consigo configurar dois usuários no sistema, sendo um aluno e um moderador, cada um com seu respectivo perfil.

[08:49] Salvei, vou rodar o projeto. Ele vai inicializar, vai carregar. A parte de código já está pronta, era só isso; era só colocar aquele `hasRole` no SecurityConfigurations, e o `data.sql` era só para popular o banco de dados.

[09:05] Não precisa mexer em mais nada, é bem simples fazer o controle de autorização: é a URL, `hasRole` e qual é o perfil que você quer restringir para essa URL.

[09:14] Para testar vamos utilizar o Postman. Não sei se vocês lembram, mas estávamos utilizando o Postman no outro curso para simular requisições HTTP para nossa API.

[09:23] Então eu vou fazer uma requisição GET: `http://locahost:8080/topicos/1` . Eu quero detalhar o tópico de id 1, só para recuperarmos os dados e verificar quais são as informações desse tópico. Ele está inicializando.

[09:54] Isso seria só para vermos que todos os tópicos que eu inseri no `data.sql` estão cadastrados, têm 3 registros. Até vou mudar o expoente, vou chamar o `http://locahost:8080/topicos` , porque eu quero ver todos os registros.

[10:12] Ele deu um problema por causa do *actuator*. Eu não subi o projeto do *actuator*, não vamos mexer nele.

[10:34] Vou ver se não digitei nada errado. Digitei errado, o correto fica `http://localhost:8080/topicos`. E estão lá os 3 registros.

[10:54] E agora eu quero testar o excluir, foi no excluir que fizemos a regra. Vou abrir uma nova aba no Postman, vou copiar o endereço `http://localhost:8080/topicos/1` e vou mudar o método de GET para DELETE.

[11:13] Vamos tentar simular dessa maneira, vou mandar um DELETE para esse endereço. E apareceu "forbidden", porque lembrem que o endereço de listar os tópicos está público, mas o de excluir está privado. Você precisa estar autentificado no sistema.

[11:28] Eu vou abrir uma nova aba e colocar o endereço como `http://localhost:8080/auth`, que é o endereço para autenticarmos. Mas para autenticar, a requisição é do tipo POST. Na aba "Headers", tem que selecionar "Content-Type" no campo KEY e "application/json" no campo VALUE, para dizermos para a API que estamos mandando dados no formato JSON.

[11:51] E no menu "Body" vou selecionar raw e vou passar direto o JSON com os dados. Para fazer autenticação tínhamos que mandar o campo e-mail, e para testar eu vou logar com `aluno@email.com`, então fica "email":"aluno@email.com", "senha":"123456"`.

[12:14] Vou disparar essa requisição. Ele deveria processar com sucesso e me devolver aquele Bearer Token, que é gerado pelo JJWT. E ele gerou.

[12:27] Vou copiar o Token de autenticação. Agora eu vou voltar na requisição de deletar e para mandar o Token lembra que ele é um cabeçalho, ele vem na aba "Headers".

[12:37] Ou você pode vir na aba “Authorization”, que é mais fácil, escolher o Bearer Token e colar no campo ao lado. Vou disparar a requisição de excluir agora e deu “forbidden”, porque eu estou logado, mas eu loguei como aluno, e o aluno não tem permissão. Então funcionou, ele não me deixou excluir o registro.

[12:57] Agora na aba “Body” eu vou trocar para moderador: `”email”:”moderador@email.com”` . Vou me autenticar como moderador. Vai disparar a requisição e vai vir outro Token, com o final diferente.

[13:08] Vou copiar esse Token, na aba “Authorization” vou apagar o que já tinha e substituir. Agora é para dar 200, é para dar que a requisição foi processada com sucesso. E voltou o HTTP status para 200.

[13:19] Só para garantir vou listar de novo os registros e veio o registro de id 2 o registro de id 3. Então ele excluiu de fato o registro de id 1.

[13:29] E para fazer a parte de autorização baseado em perfil, é basicamente você vir no Security Configuration, configurar qual é a URL e qual é o perfil que tem permissão para disparar. Não basta apenas estar autenticado, tem que também ter esse perfil.

[13:46] Um detalhe importante é que aqui no SecurityConfiguration nós não colocamos aquele `ROLE_` . O Spring já coloca isso automaticamente.

[13:53] Então no banco de dados o perfil tem que estar `ROLE_ALUNO` , `ROLE_MODERADOR` . Mas no `hasRole` você só passa o nome do Role, o Spring já coloca automaticamente o prefixo `ROLE_` .

[14:05] E com isso fechamos a aula de hoje. Nós vimos que é bem simples e tranquilo fazer um controle de autorização baseado em perfil no Spring. Lembrando de popular o seu banco de dados, a tabela de perfis e a tabela de JOIN entre o usuário e o perfil para o Spring saber quem tem cada tipo de perfil.