02

Usando JDBC Authentication

Transcrição

Segue o script SQL para criação das tabelas `users` e `authorities`:

```
create table users(
    username varchar(50) not null primary key,
    password varchar(200) not null,
    enabled boolean not null
);

create table authorities (
    username varchar(50) not null,
    authority varchar(50) not null,
    constraint fk_authorities_users foreign key(username)
references users(username)
);
create unique index ix_auth_username on authorities
(username,authority);
```

COPIAR CÓDIGO

[00:00] Já temos nossa tela de login aqui, temos um usuário e memória do João. Quando nos logamos aí é que conseguimos acessar os pedidos.

[00:11] O que precisamos fazer agora é salvarmos esses usuários no banco de dados para não ficarmos com eles salvos aqui em memória, porque se eu precisar criar novos usuários eu vou ter que fazer isso programaticamente e não faz o menor sentido. O ideal é ter isso no banco para conseguirmos criar novos usuários no banco e habilitarmos, desabilitarmos e criarmos pedidos associados a usuários e tudo mais. Como fazemos isso?

[00:35] Aqui na documentação já fizemos essa configuração em memória, fizemos esse "Form Login" com "Basic Authentication", com usuários em memória. E agora vamos fazer usando a autenticação com **JDBC**, que é ***Java Database Connectivity***, ou seja, é exatamente a integração com o banco. Aqui ele está falando que o `JdbcDaoImpl` do Spring Security implementa autenticação com usuário e senha buscando usuários do banco de dados.

[01:04] Só que para ele conseguir fazer isso, ele precisa que tenhamos algumas tabelas específicas. Ele tem um "*Default Schema*", que diz como vamos salvar esses usuários no banco, por padrão. E o `JdbcDaoImpl` vai utilizar essas tabelas para buscar os usuários, o status da conta, *authorities* dos usuários, etc.

[01:28] E aqui embaixo, no exemplo 61, ele tem um *Default Schema*, que é um exemplo de como criamos o banco de dados. Eu vou aproveitar, dar um "Ctrl + V" nesse trecho do *Default Schema* da documentação e nós vamos ver, no HeidiSQL, que ele tem dois *scripts*: um para criação da tabela usuário, outro para criação da tabela *authorities*. Vou só adaptar aqui para o MariaDB, removendo esse `ignorecase`, e eu vou aumentar o limite de caracteres, porque aqui é para *username* e aqui é para *password*.

[01:57] Como eu vou criptografar o *password* e o tamanho vai ficar bem grande, eu vou aumentar esse `password varchar(50)` para `password varchar(200)`. Pronto! Removo também o `ignorecase` de `username` e `authority` e executo esse *script* apertando o *play*.

[02:11] Na hora em que eu atualizar o "mudi" vai ter "user" e "authorities", os dois criados; mas estão vazios. Esse "users" da tabela "users" não tem informação nenhuma.

[02:24] O que vamos fazer agora? Existe mais outro método `configure()` que nos dá esse `AuthenticationManagerBuilder`. É exatamente com esse que vamos indicar para o Spring Security que vamos trabalhar com o `jdbcAuthentication()`.

[02:41] Como vamos lidar com `jdbcAuthentication`, o Spring Security vai ter que lidar com isso e ele vai ter que ir no banco de dados. Nós passamos para ele também um `dataSource`, que é onde ele consegue conexões com o banco de dados.

[02:54] Como isso, já está configurado, porque já acessamos banco de dados. Eu vou só injetar aqui o `private DataSource` do javax.sql e pronto, já vamos conseguir fornecer para ele um *dataSource*.

[03:09] A outra coisa que precisamos passar para ele é um `passwordEncoder`, que é quem vai criptografar a senha. E como nós conseguimos esse `passwordEncoder`? Aqui na documentação ele tem um item chamado 10.10.8, que fala sobre `PasswordEncoder`.

[03:31] E aqui ele tem um link para o `PasswordEncoder` acessível neste link (https://docs.spring.io/spring-security/site/docs/5.3.2.RELEASE/reference/html5/#authentication-password-storage). Veja que estamos no item 10.10.9 e o `PasswordEncoder` está no item 5, em "Features". Ele tem uma explicação do que é esse `PasswordEncoder`, como funciona, etc. Aqui ele tem alguns exemplos de como vamos utilizá-lo, como eu pego novas instâncias aqui do *encoder*.

[03:55] E eu vou utilizar esse `BCrypt`, que é um *encoder* bem forte, com senhas bem fortes; vai deixar nossa aplicação com uma boa segurança. Isso já está disponível na nossa aplicação, porque estamos trabalhando com Spring Security. E eu só dou `new` nele e ele passará a funcionar: `new BCryptPasswordEncoder()`. E pronto, passamos o `(encoder)` para o `.passwordEncoder`.

[04:18] Outra coisa que eu vou fazer é a seguinte: como eu já quero começar aqui com algum usuário, com um *password* criptografado eu vou me aproveitar do próprio recurso que tem aqui; para ao invés de fazermos esse

`InMemoryUserDetailsManager`, que isso não vai mais funcionar, eu vou arrancar isso daqui e vou criar um usuário aqui.

[04:38] Só que não vai ser `withDefaultPasswordEncoder()`, isso não faz sentido, mas sim com um `encoder` que eu vou utilizar aqui, que nós mesmos criamos. Só que eu já vou fazer `(encoder.encode()` e vou criptografar essa senha do `("joao")`. Pronto, já criptografo logo!

[05:00] E no final, o que eu faço? Uso o método `.withUser()` e passo esse usuário aqui. Quando eu salvo isso, ele vai reiniciar a aplicação e já vai criar lá no banco de dados o usuário. Então ele já está integrado com o banco e ele criou para mim um usuário inicial. Se eu atualizar essa tabela, já tenho aqui o "joao" com o *password* dele criptografado.

[05:24] Eu vou aproveitar também e criar "maria", outro usuário. Vou salvar esse aqui e ele vai restartar a aplicação, agora criando outro usuário, que é o usuário Maria. Se eu atualizar a tabela, terá dois usuários no banco de dados.

[05:41] Vamos ver se está funcionando. No navegador, vou clicar em "logout" aqui em cima, à direita, voltar para a tela de login, vou colocar o "joao", que agora vamos nos autenticar com o João que está no banco de dados, com esse registro aqui e com esse *password* - e ele está *enabled*, está habilitado.

[06:01] Se eu clicar em "login", ele se loga. Parece que não, mas o que acontece é que você logou. Se eu tentar acessar "/home", ele me permite e antes eu não conseguia. Se eu der um "logout" e tentar acessar "home", eu não consigo, ele volta para cá.

[06:20] Agora vamos fazer o seguinte: está vendo esse "/login"? Eu vou apertar a tecla "Enter" aqui e pronto, a última requisição que eu fiz foi para "/login".

[06:29] Agora eu vou me autenticar como Maria, pode ser tanto Maria quanto João, não importa. Quando eu der o "Login", olhe o que acontece. Ele veio para a *home*. Mas olhe o que eu vou fazer: vou dar um "logout" e ele deu um *redirect*

aqui para login. Se eu tentar me logar de novo, ele volta para essa própria tela. Por quê? Porque a requisição anterior foi feita para essa tela.

[06:49] Então o que o Spring Security faz? Como ele me jogou para a tela aqui de login, porque eu não estava autenticado, na hora que eu me loguei ele vai me redirecionar para a tela anterior. Só que a tela anterior é essa própria tela de login.

[07:05] O que nós podemos fazer para melhorarmos isso? Porque depois que eu me logo, eu não quero estar na tela de login, o ideal é ir logo para a *home*. O que eu posso fazer é forçar, pedir para o Spring Security para toda vez que o usuário se autenticar, ele vir para "/home". Como fazemos isso?

[07:21] Aqui no "WebSecurityConfig.java", no `formLogin`, temos o `.loginPage`, `.permitAll` e temos outro, que é o `defaultSuccessUrl`. Então posso colocar aqui: `.defaultSuccessUrl`, `alwaysUse`, eu vou colocar como verdadeiro, vou colocar `("/home", true)`.

[07:46] Deixe ele subir de novo a aplicação. Olhe o erro que ele está dando: "Duplicate entry 'maria' for key 'PRIMARY'". O que aconteceu? Como eu mantive esse `UserDetails user =`, na hora em que subiu de novo a aplicação, ele tentou criar esse usuário de novo. Então vou tirar isso daqui, e também o `.withUser(user);`. Vou subir novamente a aplicação, sem tentar criar o usuário "Maria" ou qualquer outro usuário de novo. Agora ele subiu.

[08:13] Voltando para o login, vou tentar entrar no login. Ele já está logado, vou dar "logout". Vou entrar agora com o João. Dou o "Login" e vai para "/home", ou seja, ele sempre vai para "/home". Se eu der "logout" e tentar "maria", ele também vai para "/home". Então ele sempre vai me redirecionar para "/home" - exatamente porque eu coloquei esse `defaultSuccessUrl`.

[08:44] Finalizando esse vídeo, nós temos os usuários salvos no banco de dados e o que precisamos fazer agora é associarmos os pedidos e as ofertas que criarmos a esses usuários.

[08:54] Quando eu criar um pedido, é o pedido desse usuário; quando eu listar pedidos do usuário, são os pedidos desse usuário mesmo. Se eu estou logado com o João, eu não vou ver todos os pedidos que estão no banco de dados, mas só os que o João criou. Então é isso, até o próximo vídeo!