08

Autenticando o usuário

Transcrição

[00:00] Nós já implementamos a parte de autorização, configurando as URLs que são públicas e que são restritas. Já pegamos a classe usuário, perfil, ensinamos para o Spring, que é a classe de usuário de perfil, colocamos para ele gerar o formulário, testamos, ele está bloqueando, jogando para o formulário, mas ainda não consigo me logar no sistema. Ainda não ensinei para o Spring como faço isso. Esse é o objetivo dessa aula.

[00:44] Vamos voltar para o nosso código. Na classe securityConfigurations, até então estávamos mexendo nesse configure, que é o configure para parte de autorização das URLs, mas lembre-se que eu tinha comentado que tem outro configure que é da parte de autenticação. É aqui que ensinamos para o Spring como vai ser o processo de autenticação do nosso projeto.

[01:09] Esse método recebe um parâmetro chamado authenticationManagerBuilder. Se usarmos essa variável, auth., você vai ver que tem vários métodos. Um deles é o UserDetailsService, que é para dizer para o Spring qual a classe, a service que tem a lógica de autenticação. Temos que passar para ele uma classe que está implementando uma interface que é a classe onde estará a lógica de autenticação.

[01:42] Eu vou passar como parâmetro autenticaçãoService, que vai ser nossa classe com a lógica de autenticação. Essa classe não existe, por isso está dando erro de compilação. Antes disso, precisamos criar essa classe. Vou colocar ela no pacote security, porque tem a ver com segurança.

[02:18] Essa classe vai ser uma classe gerenciada pelo Spring, só que não é um controller, não é um repository. Vai ser um service mesmo. Em cima da classe, vou colocar o @Service e importar do Spring.

[02:38] Para dizermos para o Spring que essa service é a service que tem a lógica de autenticação, também vamos usar uma interface. Vamos implementar a interface UserDetailsService. Lembra que a classe usuário é UserDetails? Já vai dar erro de compilação, porque precisa ter um método, o método loadUserByUsername. Quando entrarmos no formulário de login, digitar o e-mail e clicarmos no botão de login, o Spring vai saber que essa é a classe autenticação e vai chamar esse método. Por isso precisamos da interface. Ele vai procurar esse método, que recebe como parâmetro o e-mail que digitamos na tela de login.

[03:40] E a senha? Para o Spring, ele só passa o e-mail e nós temos que fazer a consulta no banco de dados filtrando só por isso. A senha ele faz em memória. No geral, em uma aplicação já fazemos a consulta no banco filtrando pelo usuário e pela senha. No Spring security é diferente. Nós buscamos pelo e-mail e o Spring faz a checagem da senha em memória.

[04:05] Preciso implementar a lógica de fazer a consulta no banco de dados, passando como parâmetro esse e-mail. Como estou numa service, lembra que não é a service que faz a lógica de acesso ao banco de dados. Estamos usando o padrão repository. Então, no nosso pacote repository vamos precisar de um repository para o usuário. Só temos o do tópico e do curso, se não me engano.

[04:27] Vamos criar uma nova interface no pacote repository, e ela vai ser a usuarioRepository. Lembre-se que todo repository tem que dar um extends JpaRepository<usuário, Long>.

[04:49] Vamos ter que ter um método que faz a busca pelo e-mail. Ele vai devolver um usuário e o nome do método seguindo aquele padrão de nomenclatura. Ele recebe como parâmetro um string e-mail.

[05:13] Voltando para minha service, nesse método tenho que chamar o repository. Vou precisar declarar um atributo do tipo usuarioRepository, vou chamar a variável de repository e vou pedir para o Spring injetar o parâmetro @AutoWired. Para fazer a consulta, repository.findbyemail. E passamos como parâmetro username, que foi o Spring que passou de acordo com o que o usuário digitou na tela de login.

[05:49] Vou guardar o retorno dessa chamada numa variável. Ela devolve um usuário. Vai fazer a consulta e vai devolver o usuário. Na verdade, lembre-se que ele vai fazer a consulta e se o método find não encontrar ele retorna vazio. Vimos isso no TopicosController. Quando fizemos o detalhar, passávamos o id, mas se não existisse o id, ele jogava exception. Começamos a usar o findById, que devolve um optional, e o optional pode ser que tenha resultado ou não. Então, vou precisar fazer o mesmo esquema, porque estou passando o e-mail, mas vai que não existe um usuário com esse e-mail. Eu vou alterar o findbyemail no repository para devolver o optional. Pode ser que ele encontre o usuário, pode ser que não.

[06:59] Vou trocar o retorno para optional de usuário. Se veio um usuário, é porque está certo o e-mail que você digitou. Simplesmente devolvo o usuário. Se não vier o usuário, preciso dizer para o Spring que o usuário não existe. Lembre-se que dá para fazer um if desse optional. Se estiver presente, devolve. Senão, jogo uma exception para avisar o Spring que o usuário não foi encontrado.

[07:48] Se chegou ali embaixo, ele digitou um usuário que não existe no banco, throw new usernameNotFoundException. E passo uma string com texto. Essa é a lógica do nosso serviço de autenticação. Quando eu entrar na tela de login e clicar em sign in, o Spring sozinho vai chamar a service.

[08:36] Agora, voltando para o security Configuration, preciso passar o autenticaçãoService para ele. Só que aqui vou precisar injetar nosso autenticaçãoService. Nessa classe de configuração você consegue também fazer injeções de dependência.

[09:12] Passei para ele a lógica de autenticação. E compila, porque essa classe nossa implementa a interface do Spring. A princípio, está tudo ok, já ensinei para o Spring como faço a lógica de autenticação. para testar, preciso ter um usuário cadastrado no banco de dados.

[09:40] Lembra daquele arquivo data.sql? Naquele script de inicialização do banco de dados eu criei um registro para o usuário. O e-mail dele é alura@email,com, a senha é 123456. Porém, em qualquer aplicação deixar a senha em aberto no banco de dados não é uma boa prática. Geralmente o pessoal utiliza algum algoritmo para gerar um HASH dessa senha e salva no banco de dados a senha com o HASH.

[10:12] Também vamos fazer isso na classe de autenticação. Nós chamamos o auth, o UserDetailsService, só que antes do ponto e vírgula colocamos um passwordEncoder. Conseguimos dizer para ele qual o algoritmo de HASH que vamos utilizar para a senha. Também não adianta nada usar um algoritmo que não é seguro. O Spring suporta alguns algoritmos que são seguros hoje em dia. Um deles é o tal do BCrypt. Posso instanciar um objeto BCryptPasswordEncoder. É uma classe do próprio Spring security que já implementa a geração do algoritmo do HASH, da senha, usando o algoritmo específico.

[11:12] Só que no nosso data.sql, quando ele vai criar um usuário no banco, ele está jogando a senha em texto aberto. Não posso fazer isso. Tenho que passar um valor que é o HASH no formato do BCrypt. Para facilitar, vou na minha classe SecurityConfigurations e vou dar um system out, vou chamar o método encode e vou passar a senha 123456. Se eu rodar esse main, só estou falando para ele pegar o BCrypt e gerar o HASH da senha.

[12:18] No arquivo SQL vou trocar a senha pela senha criptografada. Vou voltar no SecurityConfiguration e vou apagar o main, só para termos a tenha gerada. Agora, na teoria, já consigo simular. Vou tentar me autenticar com aquele usuário. Digitando a senha errada, tento logar, ele não permite. Com a senha

certa, funciona. Consigo me autenticar no sistema, e a partir de agora consigo chamar os endpoints que estão restritos.

[13:30] Com isso, fechamos a parte de autenticação usando o Spring security, configurando a parte de usuário, senha e serviço que tem a lógica de autenticação.