04

Implementando o Login

Transcrição

[00:00] Terminamos a nossa aplicação, ela está rodando aqui. Eu resetei o banco de dados, ele tem esse registro novo de "Victrola Vintage", está no estado "Aguardando" ainda. "Aprovado", "Entregue" e funcionando a busca corretamente, para criar um novo produto com validação. Esse foi o status que terminamos na aula passada.

[00:27] Qual é a ideia? Quando um usuário entra nessa página que estamos chamando de página *home*, a princípio, pelo título pelo menos, o que deveria aparecer para ele são os pedidos do usuário; só que essa página está buscando todos os pedidos.

[00:48] Se viermos no "PedidoController.java", no menu à esquerda, aqui no pedido vai mostrar o formulário, vai criar o novo, o "HomeController" - mesmo que deveria estar mostrando que é essa página aqui dos "Meus Pedidos".

[01:03] O *select* que ele está fazendo no banco de dados, na requisição `get` é o `findAll`, que é a implementação automática de um *repository*. Só para lembrar, nós estendemos "JpaRepository" e já ganhamos vários métodos, incluindo esse aqui.

[01:20] Mas o que nós queremos não é mostrar todos os pedidos, nessa página aqui queremos mostrar todos os pedidos do usuário, os "meus pedidos"; só que ainda não sabemos quem é o usuário. A menos que o usuário se logasse. Só que esse login ainda não está funcionando!

[01:37] O que vamos começar a fazer agora é a implementação da autenticação do usuário usando Spring Security. Vamos começar na versão mais simples dessa implementação e depois vamos evoluindo aos poucos, adicionando vários recursos.

[01:53] A primeira coisa que vamos fazer é irmos no spring.io acessível neste link (https://spring.io/), como sempre viemos fazendo. Eu vou abrir a aba anônima para me facilitar algumas coisas na gravação. O que vamos procurar aqui? No menu "Projects > View all projects" e ele vai listar todos os projetos do Spring. Nós vamos procurar o Spring Security.

[02:14] O que acontece? Você já deve estar vendo que a versão do próprio Spring aqui, do *layout* do site, está diferente.

[02:22] Vieram versões mais novas do curso anterior para cá e as próprias versões aqui, se entrarmos no Spring Security, no próprio Spring Framework, vamos lidar com versões mais recentes. Seria interessante trabalharmos com essas versões.

[02:40] Deixe-me entrar aqui no "Spring Boot", pegar aqui na aba "Learn" a versão "GA", a versão *release* mesmo, a versão 2.3.1.

[02:50] Vamos fazer o seguinte: antes de mais nada, para não termos nenhum problema de pegarmos a última versão de referência da documentação e termos alguma diferença, vamos atualizar nossa aplicação para trabalharmos com essa última versão do Spring.

[03:06] Eu vou voltar no arquivo "pom.xml" do nosso projeto e adicionar lá em cima, onde está `2.2.2.RELEASE`, vamos colocar `2.3.1.RELEASE`. Vou salvar aqui

[03:16] A princípio não é para dar nenhum problema. Às vezes acontece ter algum impacto na atualização do Spring, questão de gestão de módulos, eles podem mudar alguma coisa em relação às dependências; mas eu estou fazendo aqui, ele está reconstruindo o projeto. Vamos ver o que acontece.

[03:36] A princípio ele já construiu. É exatamente aqui no "PedidoController.java" que nós abrimos. Ele está dando um erro de compilação no `import javax.validation.Valid;`, que é exatamente um módulo que ele já não está mais importando automaticamente, ou seja, ele passou a ser um módulo opcional.

[03:58] Esse módulo de validação está separado. Ao invés de de `<artifactId>spring-boot-starter-web</artifactId>`, nós podemos colocar aqui `<artifactId>spring-boot-validation</artifactId>`. Eu copiei aqui a dependência do web, colei aqui no `validation` e salvei. A ideia agora é importarmos essa `spring-boot-starter-validation`. Vamos ver!

[04:21] A princípio a recompilação aqui do projeto está fazendo *build*, tudo de volta ao normal. Eu já derrubei aqui o projeto. Vamos subir de novo e ver se tudo está certo. Ele está subindo, porta 8080, está tudo bem até então. Começou.

[04:45] Vou voltar para a página web dele e dar um *refresh* aqui. Parece que não, mas está tudo certinho, ele já foi no banco de dados, já buscou esses dados, eu consigo navegar para o formulário, a validação está funcionando certinho, a mensagem do *validation*. Está tudo ok.

[05:05] Fizemos a atualização para a versão mais recente do Spring e agora vamos poder evoluir a aplicação, sabendo que vamos lidar com a documentação na última versão. Então também é bem interessante que nós ainda não estamos utilizando muitos recursos do Spring.

[05:21] Vamos colocar o Spring Security na nossa aplicação. Como sempre, aqui do lado, à esquerda, dá para vermos, no menu do site spring.io, o "Spring Security". Eu busco entrar na documentação, vou pegar aqui a versão "GA" (a versão *release*) e ver como ele nos ensina a trabalharmos com os próprios módulos, os próprios projetos do Spring.

[05:46] Estou o esperando carregar o menu aqui. Eu vou em "Samples", no menu à esquerda, aqui tem "Spring Security Community", "What's New in Spring Security 5.3", "Getting Spring Security" - essa é uma parte importante, principalmente porque você tem que adicionar a dependência no Spring Security na sua aplicação, olhe o que ele explica aqui.

[06:05] Eu vou no "pom.xml" e vou adicionar, assim como colocamos o `validation`, eu vou colocar o `spring-boot-starter-security`. Então pronto! Isso aqui já é o primeiro passo para adicionarmos o Spring Security na nossa aplicação.

[06:20] E agora eu vou em "Samples", procurar alguma ajuda da própria documentação para nós fazermos a configuração mais simples do Spring Security. Eu vou procurar aqui "Java Configuration".

[06:38] Tem "Hello Web Security", aqui no item 10 tem a parte de "Authentication", que é o primeiro que temos fazer. Vamos ver o que conseguimos aqui no menu, "Username/Password Authentication" e tem o "Basic Authentication".

[06:58] O "Form Login" vai abrir uma tela de formulário, mas eu quero instalar o Spring Security e já conseguir me logar com usuário e senha, colocando segurança na aplicação fazendo o mínimo possível. O "Basic Authentication" é exatamente isso.

[07:13] A documentação é bem legal para entendermos quais são as classes, os *interceptors* e tudo mais - coisas que vamos ver mais para frente, o que são *interceptors*, os filtros. Vale a pena dar uma lida aqui.

[07:24] E eu vou procurar algum exemplo de código, assim como eu tinha o exemplo para adicionar na dependência. Aqui ele está exibindo o código de um `protected void configure(HttpSecurity http)`, mas ele não está dizendo exatamente onde vamos colocar isso, ele não está sendo tão explícito, não está sendo tão legal aqui conosco.

[07:46] Se você ficar um pouco perdido e pela documentação do spring.io não conseguir evoluir, uma boa coisa a se fazer é procurar por "spring guide basic authentication". Então vamos procurar no próprio site do Spring. "Getting Started | Securing a Web Application" acessível neste link (https://spring.io/guides/gs/securing-web/), acho que esse é exatamente o que queremos!

[08:08] Isso aqui é todo um tutorial, ele mostra tudo o que você precisa fazer, fala até a IDE em que você pode trabalhar - e é do zero, então você vê que está adicionando até o Thymeleaf, web. Está fazendo a configuração um pouquinho diferente que a nossa, você vê o mapeamento MVC dele é diferente.

[08:29] Mas o que eu estou procurando é a configuração do Spring Security. Aqui ele já começou a adicionar a dependência...

[08:35] E pronto, aqui mais abaixo já tem um exemplo de código.

[08:38] Vamos voltar à documentação. Olhe esse método `configure(HttpSecurity)`, na verdade ele é um `@Override` do método `configure()` que recebe `HttpSecurity` de `WebSecurityConfigurerAdapter`, ou seja, é um adaptador.

[08:57] Se procurarmos aqui na documentação, `WebSecurityConfigurerAdapter`, ele está dentro do próprio Spring Security, ou seja, é uma classe que vai nos ajudar a integrar nossa aplicação com Spring Security e ele também nos oferece uma anotação aqui para fazermos inicialização junto da camada de segurança da nossa aplicação.

[09:15] O que eu vou fazer é copiar mesmo. Nós vamos no `WebSecurityConfig`, na nossa pasta principal, "br.com.alura.mvc.mudi", aqui no pacote principal da aplicação. Eu vou criar uma nova classe chamada `WebSecurityConfig`, que estende `WebSecurityConfigurerAdapter`. Então aqui, `extends WebSecurityConfigurerAdapter {` e deixamos ele importar aqui.

[09:43] Vou adicionar essas duas dependências: `@Configuration` e `@EnableWebSecurity`. Esse `@Configuration` é do próprio Spring, atualizando para o Spring que essa é a classe de configuração. Aperto as teclas "Ctrl + Shift" e ele nos importa aqui. Eu vou reimplementar o método `configure()` que recebe `HttpSecurity`.

[10:02] O que ele está fazendo aqui? Não precisamos de todos esses passos, mas é legal entendermos o que ele está fazendo. Ele está dizendo que ele está utilizando esse "http" e está chamando esse `.authorizeRequests()`, e que tudo o que for `("/", "/home")` ele está permitindo tudo e qualquer outro *request* a pessoa tem que estar autenticada - ou seja, ele está dizendo que nesse exemplo o "/home" é público, a pessoa não precisa estar autenticada.

[10:32] Nós vamos fazer algo mais simples, vamos colocar aqui o `http`. Eu vou colocar o `.authorizeRequests()` também. Vou apagar isso aqui, `// TODO Auto-generated method stub`, `super.configure(http);` e inserir `http.authorizeRequests`. Vou colocar qualquer *request*. Então já vou procurar aqui por `.anyRequest().authenticated()`.

[10:53] No final o que ele termina aqui? Ele usa `.formLogin()`. Então é `.authorizeRequests()`, ele ajusta porque essas páginas têm que ser públicas, todas as outras têm que estar autenticadas e está usando o `formLogin()`. Nós vamos fazer a versão mais simplificada, que é usando o `httpBasic()`. Tem que usar o `and` aqui, `.and().httpBasic()`.

[11:19] Então vamos fazer esse `authorizeRequests()`, `anyRequest()` e `httpBasic()`. Podemos até quebrar aqui para entendermos essa configuração dos *requests* todos. Estamos usando `httpBasic()`!

[11:34] Só que é o seguinte: já habilitamos, então agora o usuário só vai conseguir acessar nossa aplicação se ele estiver autenticado. Só que para se autenticar, ele precisa de um usuário e senha.

[11:44] Outra coisa que precisamos fazer é copiarmos esse outro método aqui, `UserDetailsService()`, que ele está criando através dessa classe `UserDetails` e da classe `User` - que é um *builder*, que ele está vindo com `.build();`. É um *builder* de usuário, que nesse caso é um usuário criado em memória mesmo.

[12:01] Eu vou copiar esse método todo e vou criar o usuário João lá. Vou usar as teclas "Ctrl + Shift + O" e ele vai pedir uma classe que ele não sabe importar, que é a classe *user*, provavelmente é do pacote Security mesmo, do *core*.

[12:18] Vou colocar o `.username("joao")`, o `.password("joao")` e vou deixar `.roles("ADM")`. Nós vamos entender posteriormente para o quê uma *role* serve.

[12:29] Só que, o que está acontecendo? A IDE está dizendo que esse método é *deprecated* e isso acontece porque não é seguro usar isso em produção. Se entrarmos aqui, eu segurei a tecla "Ctrl", aparece a mãozinha. Clico em `.withDefaultPasswordEncoder()`, entro na implementação do método.

[12:46] Tem todo esse Javadoc aqui em cima, essa documentação. Aqui em cima tem o motivo do de ele ser *deprecated*.

[12:53] Não é considerado seguro usar isso daqui em produção, apenas para demonstrações e configurações iniciais, só para conhecermos o Sprint Security - que por enquanto é o nosso caso. Depois vamos deixar de usá-lo e fazer uma configuração melhor de segurança. Mas é o que eu falei: nós vamos pelo menos inicializar a configuração do Spring Security.

[13:17] Vou salvar isso e subir a aplicação de novo. Ela não está rodando. Vamos subir. Vamos ver se isso está funcionando, vamos ver se o Spring já vai plugar essa classe e se já vai utilizar essa configuração `@Override` aqui para inicializar a camada de autenticação com esse usuário e de autorização. Essa aqui é a configuração de autorização.

[13:40] Estou dizendo que para todas as requisições o usuário tem que estar autenticado. Estamos utilizando a configuração mais simples, o *basic*.

[13:50] Vamos tentar acessar nosso site de novo, voltar para o "/home", não deveríamos conseguir sem logar. Nesse `httpBasic` o que nos aparece já é uma janelinha do próprio navegador para conseguirmos nos logar. Eu vou colocar o nome de usuário "maria" e senha "maria". Vamos ver se nos logamos. Não permite. Então: "joao", "joao" e aperto a tecla "Enter". Ok!

[14:16] Ou seja, realmente está funcionando, aquele usuário realmente está sendo usado pelo Spring e nós conseguimos fazer a configuração mais básica aqui de segurança de autorização na nossa aplicação.

[14:28] Voltando, vamos fazer isso posteriormente, mas é possível dizermos que, por exemplo, na página *home* todo mundo é permitido, ou seja, o usuário não precisa necessariamente estar autenticado. Então tem esse `.antMatchers` na documentação para fazer essa configuração. Não estamos fazendo isso, estamos dizendo que para todas as requisições o usuário tem que estar autenticado.

[14:48] Posteriormente nós vamos evoluir isso. Até o próximo vídeo!