07

O primeiro Controller

Transcrição

[00:00] Temos a nossa aplicação rodando usando o Spring Boot integrado com o Spring MVC e Thymeleaf. O que nós vamos fazer nesse vídeo é criar a nossa primeira tela, o nosso primeiro “controller” e mandar informações do “controller” para a tela processando com o Thymeleaf.

[00:16] Vou criar aqui uma nova classe chamada de “helloController” e vou colocar dentro de um pacote chamado de “controller” só para separar isso. Nessa classe é onde nós vamos adicionar as nossas “actions”.

[00:34] Por exemplo: eu vou criar um método que retorna uma “String” chamada de “hello()”. Isso é uma “action” que vai processar uma requisição do usuário e vai redirecionar para uma determinada “view”. No caso, para uma “view” chamada de “hello”.

[00:48] A “view” vai ser um HTML que por padrão, já que estamos trabalhando com Thymeleaf, Spring Boot e Spring MVC. Ele já configura para nós uma pasta chamada “templates” dentro de “src/main/resources”. Aqui dentro eu teria que criar uma página chamada de “hello.html”.

[01:07] Como estamos trabalhando com Thymeleaf, ele por padrão, as nossas páginas em Thymeleaf são HTMLs normais. Então, “hello.html”. Nós podemos criar um HTML simples. Vou colocar um “” com ““ para nós não termos problema de acentuação. Eu vou criar um “body” com a mensagem “return=hello”;”. Como eu faço para chegar nessa “view”? Uma requisição que bate no Spring teria que ser direcionada para esse método “hello”.

[01:52] O método “hello”, que é uma “action”, no caso da nossa implementação as “actions” são métodos de classes “controllers” e o retorno dessa “action” é o nome da “view” que deve ser processada, cuja requisição deve ser redirecionada para gerar o HTML que vai ser devolvido para o usuário.

[02:13] Será que isso já está funcionando? Será que se eu fizer uma requisição para a barra “hello”, a requisição bate nesse método e depois vai para a “view” e a “view” mostra a mensagem “hello”? Vamos ver! Vou colocar “localhost:8080/hello”. Pronto!

[02:31] Ainda não está acontecendo nada. Por quê? Porque nós ainda não integramos ainda essa classe “hello” da nossa aplicação com o próprio Spring. Para fazermos isso, nós usamos o quê? Anotações! Então eu vou anotar essa classe com “@Controller”. O “@Controller” é uma anotação do Spring mesmo, então ele reconhece. Isso já é a nossa integração com o Spring. Então você anota a sua classe com “@Controller” e ela passa a ser gerenciado pelo Spring.

[03:06] Cada método é uma “action” que o Spring vai mapear as requisições para bater nos métodos dos nossos “controllers”. Não precisamos ter só um. Podemos ter vários. Por exemplo: o que nós estamos fazendo aqui é um método “get” para “Hello”.

[03:24] Como eu faço para essa requisição “get” chamada “Hello” porque é um path, com o path “hello” que bata nesse método aqui. Como eu aviso para o Spring? Se você receber um “@GetMapping(“/hello”)”. Chame esse método e é esse método que vai processar.

[03:41] É só usar uma anotação chamada “GetMapping” e dentro dele como parâmetro você diz exatamente qual é o “path” que deve ser chamado para bater nesse método. E agora, a única coisa que esse método faz é retornar String “hello”, que é o nome da “view”- que é essa “view” que já criamos. Então, “hello”, pronto! Isso já está certo. Vou só salvar. Não, está tudo salvo. Vou apertar a tecla “F5” e ele já atualizou e já está abrindo o “Oi!”.

[04:11] Isso está acontecendo porque o Spring está o tempo inteiro, a cada mudança que fizemos, ele "reboota", ele faz um "reboot", um "re-deploy". Agora, como fizemos para, a partir das "actions", adicionar informações, valores e variáveis nas "actions" que possam ser lidas pela "view"? Como a "view" recebe informações?

[04:34] Por exemplo: se na "action" eu tivesse chamado um método do tipo "pedidoService.getPedidos()" e aí eu busquei e esse método "getPedidos", busca do banco de dados uma lista de pedidos. Como eu faço para pegar esses pedidos que esse método está retornando e mandar para uma "view"? Como eu passo um valor, uma variável para a "view"?

[04:55] Nós vimos no curso, vocês devem ter visto o curso de "Servlets", que usando "HTTP ServletRequest", você consegue adicionar atributos neles que já vão ser acessados pela "view". Então dê um "request.Attribute()". Vou colocar "nome" e esse é o nome do atributo e esse é o valor do atributo, "Mundo".

[05:24] Então estou adicionando esse método com esse valor "Mundo" no "request". Os valores que estão no "request" podem ser acessados na camada de visão usando o Thymeleaf. Por exemplo: imagine que eu tivesse um "span" e fosse só "João", porque o Thymeleaf vai substituir esse valor depois. Se eu apertar a tecla "F5", aparece o "João". Como eu faço para substituir esse valor pelo valor contido no atributo "nome"? Usando o Thymeleaf!

[05:55] Nós adicionamos um atributo, que é o "th:text" e você passa o nome da variável. Como você passa e qual é o nome da variável que o Thymeleaf tem que pegar o valor e substituir ele pelo "João"? Usando o símbolo de dólar e o abre e fecha chaves. Essa é a Expression Language com o Thymeleaf.

[06:17] Aqui dentro eu coloco o nome do atributo, cujo nome é "nome". O identificador do atributo que adicionamos é "nome". Pronto! Se eu salvar, será que já vai aparecer "Oi Mundo" em vez de "João"? Vamos ver! Vou apertar a tecla "F5" e beleza, apareceu "Oi Mundo!"

[06:36] Então o Thymeleaf processou esse HTML e substituiu o valor contido dentro do “span” pelo valor que ele pegou no atributo “nome”. O atributo do “request”, “nome”. Parece estranho, mas por que isso é interessante? Porque aqui nós conseguimos abrir essa página porque estamos com o servidor rodando.

[06:57] Mas veja, essa é uma página simples, não deveria ser tão difícil abrir uma página dessa e poder alterar o HTML sem ter que subir o servidor. E realmente, não precisamos subir o servidor. Se você quiser abrir e visualizar essa página, é só colocar “Open With”, “Web Browser” e você conseguirá ver.

[07:16] Só que no caso, vai aparecer “João” porque aquele “th:text” não vai ser processado pelo Browser. Eu vou apertar as teclas “Ctrl + X” e abrir no Browser mesmo. Então, fazendo uma requisição para o Spring, para o servidor, para a nossa aplicação; a “view” vai ser processada e vai aparecer “Oi Mundo!”, mas se você abrir e passar o endereço inteiro do HTML, você também conseguirá visualizar.

[07:42] Ou seja, para alterar o HTML, a visão, CSS, para o designer, para o [FRONT END] que está mexendo só com HTML puro, ele não precisa subir o servidor, ele pode vir e alterar a “view” diretamente no HTML. Ele consegue visualizar o estilo visual todo sendo aplicado aqui.

[08:01] Então se você exibir o código-fonte dessa página estática, ela vai aparecer como “Oi span th:text=“${nome}”>João“. Só que esse atributo não é renderizado, não afeta em nada a visualização, então vai aparecer “João”.

[08:16] No caso do Spring mesmo, chamando a aplicação e rodando o Spring, se você visualizar o código-fonte vai aparecer “Oi Mundo” e acabou. Então essa é a nossa primeira aplicação. Nós vimos nesse vídeo como nós criamos um “controller” integrado com Spring, que tem as nossas “actions”, que é exatamente esse “HelloController” e como nós mandamos informação para a visão utilizando o “HttpServletRequest”. Só que nós não precisamos utilizar o “HttpServletRequest”.

[08:48] O Spring nos dá uma interface chamada “model” onde podemos adicionar valores nele. Em vez de usar o “request”, use o “model” chamando o método “addAttribute”. Ele funciona do mesmo jeito que o “HttpServletRequest” para esse caso, só que sem ter que ficar utilizando uma camada mais abaixo, uma camada de “Servlets”. Nós estamos só no nível do Spring. Então não precisamos mexer no “HttpServletRequest”.

[09:14] Se atualizarmos a página de novo... Eu vou fechar essas estáticas, e apertar a tecla “F5”. Tudo continua funcionando. Até o próximo vídeo!