02

Usando JPA puro

Transcrição

Segue as dependências e propriedades para copiar:

```
<dependency>
	<groupId>org.springframework.boot</groupId>
	<artifactId>spring-boot-starter-data-jpa</artifactId>
</dependency>
<dependency>
	<groupId>mysql</groupId>
	<artifactId>mysql-connector-java</artifactId>
	<scope>runtime</scope>
</dependency>
```

COPIAR CÓDIGO

```
spring.jpa.hibernate.ddl-auto=update
spring.datasource.url=jdbc:mysql://localhost:3306/mudi
spring.datasource.username=root
spring.datasource.password=root
```

COPIAR CÓDIGO

[00:00] Se olharmos o “Controller” que está fazendo aquele redirecionamento para a página que estamos mostrando o pedido, conseguimos perceber que o pedido está “hardcoded”.

[00:10] O objetivo nesse vídeo é pegar o pedido que estamos mostrando na interface e trazer do banco de dados de forma não “hardcoded”, ou seja, nós

temos que adicionar uma conexão com o banco de dados antes e depois fazerrmos o mapeamento via JPA, que é a tecnologia que vamos utilizar aqui.

[00:31] Como fazemos para adicionar uma conexão para o banco de dados integrada via JPA usando o Spring Boot? Nós já criamos o nosso projeto e não adicionamos as dependências necessárias para fazer isso.

[00:48] O que vamos fazer para simplificar? Nós vamos abrir o “start.spring.io” e nas “Dependencies” vamos adicionar as que precisamos agora, que é o JPA e o driver do MySQL. Vamos utilizar o MariaDB, mas o MariaDB é uma ramificação do MySQL e ele utiliza o mesmo drive.

[01:12] Clica em “Explore”, nesse link, e ele vai nos mostrar as dependências que teríamos que ter no nosso projeto se tivéssemos adicionado eles antes. Eu vou copiar o “spring-boot-starter-dat-jpa” e o “mysql-onnector-java” e vou adicionar antes do DevTools. Vou apertar as teclas “Ctrl + Shift + F” e pronto! Eu adicionei o “starter-data-jpa” e o “mysql-connector-java”. Já consigo me conectar com o banco de dados.

[01:45] Só que, do que nós precisamos a mais? Precisamos das propriedades de como preencher e quais são as propriedades que nós precisamos adicionar no projeto para ensinarmos o Spring a se conectar com o banco. Vamos fazer o seguinte: “spring guides jpa com mysql”. Você coloca isso no Google e vai aparecer, “Acessing data with MyWQL”.

[02:10] É um guia do próprio site do Spring mostrando como fazer essa configuração. No início ele já abre o “start.spring.io”, também selecionando aquelas dependências que nós acabamos de adicionar, que é o JPA e o driver. Estão aqui.

[02:27] Esse aqui e o Data JPA. Essas são as dependências que acabamos de adicionar. Mais embaixo tem as propriedades que precisamos configurar. Eu já colei no projeto, no “application.properties” e coloquei “spring.jpa.hibernate.ddl-auto=update”.

[02:44] Então ele automaticamente vai atualizar com os novos bancos de dados que formos adicionando ao projeto. A URL eu coloquei “localhost:3306/mudi” que é o banco de dados que vamos usar aqui. Esse é o nome do banco. O usuário e senha é “root”.

[02:59] Quando você instala o MariaDB na sua máquina, ele instala um “client” chamado de HeidiSQL. Eu já configurei um acesso ao “root” e nós vamos criar um banco de dados. Eu tinha criado, mas deletei. Banco de dados do “mudi”.

[03:17] Pronto! Ele não tem tabela nenhuma, mas não precisamos criar na mão. Por quê? Porque a nossa aplicação vai subir com o “ddl-auto-update”. Então isso vai fazer com que ele crie a própria tabela que precisamos. Qual é a tabela que precisamos criar no banco de dados?

[03:34] A tabela que vai ter os dados do pedido. Como fazemos isso? Essa configuração com o JPA. Nós abrimos a classe “pedido”, dizemos que ele é uma “Endity” do pacote “Javax.persistence”, adicionamos esse atributo “id” que você não tem. Isso eu fiz agora.

[03:52] Anote ele como “@Id @GeneratedValue” e coloque como “(strategy = GenerationType.IDENTITY)”. Porque aí ele vai criar essa tabela “pedido” com uma coluna chamada “[ID AUTO INCREMENTO]”. Nós vamos ver isso no HeidiSQL.

[04:11] Se formos para o “client” do Heidi e atualizarmos, não mudará nada porque a nossa aplicação não está rodando. Ela está aqui, está derrubada. Então eu vou subir o “MudiApplication.java”, dar um “Run As”, “Java Application” nele.

[04:25] Ele vai subir e já vai criar a tabela de banco de dados para o pedido. Ele não listou quem fez essa criação, mas se dermos um “refresh”, vamos ver que já tem o pedido lá. O que vamos fazer? Nós vamos criar na tabela pedido, um registro com as informações que queremos adicionar.

[04:51] Vou clicar em “+”, vou adicionar uma descrição, vou pegar aqueles dados que já criamos no “Controller”. Então, “nome_produto” vou colocar aquele celular, o Redmi Note 8. O “url_imagem”, está aqui. O “urk_produto”, colo lá também. “data_da_entrega” vai ficar nulo. “descricao” nós podemos adicionar essa descrição inútil, fake. E “valor_negociado” não vamos criar.

[05:25] Então aqui em cima tem esse botão verde que vai comitar essa alteração que fizemos aqui. Não sei se ele fez. Vamos clicar no “pedido” de novo e listar os dados. Beleza. Ele já fez sim. Pronto!

[05:43] Como eu faço para fazer uma consulta? Nós já nos conectamos com o banco. Tanto que ele criou a tabela lá. Agora só falta nós lermos esse banco de dados. Eu vou primeiro apagar essa lista. Então esse aqui não tem nenhuma lista de pedidos.

[06:02] Como eu faço para poder me comunicar com o banco de dados usando o JPA? Nós utilizamos uma classe chamada de “endityManager” e vamos pedir para o “hibernate” configurar esse “endityManager” para nós usando um “@PersistenceContext”.

[06:23] Então é só fazer “endityManager”. Como eu faço uma consulta? Vou dar um “endityMagaer createNameNamedQuery()”, vou fazer um “select p from Pedido p”. Estou buscando todos os pedidos do banco. No caso, só tem um. Não é o ideal, mas é isso.

[06:43] Vou adicionar a variável chamada de “Query”. Se eu tivesse que passar um parâmetro, é utilizando esse aqui. Mas por enquanto nós não vamos utilizar parâmetro nenhum, ou seja, nós só queremos dar um “getResultList()”. Aí eu vou criar aqui o resultado.

[07:00] Essa lista vai ser um resultado que eu vou chamar de “pedidos”. É uma lista do tipo “Pedido”. Só que podemos adicionar nesse “createNamedQuery” a classe de “resultClass” que queremos. Então eu vou colocar “Pedido.class” e

esse “getResultList” vai realmente nos retornar uma lista de pedidos que eu vou adicionar aqui.

[07:25] Beleza! Vamos ver se isso está funcionando. Vou na página novamente e aperto a tecla “F5”. Deu erro. Por que ele deu erro? “select p from Pedido p”. Ele está achando ruim esse “select” aqui. “No query defined for that name [select p from Pedido p”. É porque eu coloquei “NamedQuery” e é só “Query” mesmo.

[07:46] “createQuery”. “createdmedQuery” é como se já existisse uma “query” mapeada em outro lugar utilizando anotações. Nós podemos fazer isso mais para frente, mas o objetivo aqui é sermos mais práticos. Beleza, já conseguiu recuperar! Esses dados que ele está mostrando aqui são do banco de dados. Quer ver? Eu vou fazer uma atualização. Vou colocar o “Xaomi Redmi Note 10”.

[08:08] Vou apertar no verde para comitar a alteração que eu fiz e em “Atualizar página”. Olhe só, ele já mudou aqui! Então é isso que precisamos para fazer a integração da nossa aplicação com o banco de dados usando Spring Boot. Ele já vem com a dependência “started” que adicionamos e já nos configura muita coisa.

[08:27] Até o próximo vídeo!