06

Testando o repository com outro banco de dados

Transcrição

[00:00] Oi, bem-vindos de volta o curso de Spring Boot na Alura. No último vídeo vimos como testar o Repository. Criamos o teste do `CursoRepository`, fizemos dois cenários utilizando o JUnit e vimos que temos toda a infraestrutura do Spring quando rodamos esse tipo de teste. Então conseguimos fazer uma injeção de dependências e utilizar um componente do Spring.

[00:22] Porém, eu tinha deixado aquela interrogação na cabeça de vocês em relação a como esse teste foi executado. No caso, consideramos que sempre que rodarmos o teste já vão existir alguns registros no banco de dados. E isso tudo vem daquele arquivo "data.sql".

[00:39] Então sempre que rodamos o teste o Spring Boot também carrega esse arquivos e faz os *inserts*, e nós tínhamos o *insert* de dois cursos.

[00:48] Nessa aula vou fazer uma discussão sobre isso e mostrar uma outra abordagem. Eu queria só mostrar um detalhe importante para vocês: abrindo o arquivo `pom.xml` na parte de dependências temos a dependência do H2, que é o banco de dados que estamos utilizando durante o curso. Tem o Swagger, Spring Boot, JSON Web Token, e também o H2.

[01:16] Caso você não esteja utilizando o H2, se você tivesse escolhido usar o MySQL, por exemplo, você não teria essa dependência; você teria a dependência do MySQL no local. E quando você fosse rodar o teste ele não ia

passar, não ia funcionar como no último vídeo. Ele daria um erro dizendo que ele não achou o banco de dados H2, um banco de dados em memória.

[01:43] Essa é uma coisa importante do módulo de teste do Spring Boot. Por padrão o teste do Spring Boot considera que você vai fazer o teste com um banco de dados em memória.

[01:58] Então se na sua dependência você não tivesse o H2 ou outro banco em memória, você só tivesse o MySQL, por exemplo, o Spring Boot daria uma mensagem dizendo "Você até tem um banco de dados, que é o MySQL, mas para fazer os testes eu uso um banco de dados em memória e você não tem ele como dependência no projeto".

[02:16] Então esse é o comportamento padrão do Spring. Ele considera que você fará os testes utilizando um banco em memória.

[02:22] No nosso caso, como já estávamos utilizando o banco de dados H2, que é um banco de dados de memória, o teste funcionou corretamente.

[02:31] E surge essa discussão: eu devo fazer meus testes num banco de dados em memória ou no mesmo banco de dados que eu utilizo em produção, o MySQL, por exemplo? Isso é uma discussão, porque tem vantagens e desvantagens das duas abordagens. Mas o foco não é ficar discutindo isso.

[02:49] Então caso você queira utilizar um banco de dados em memória, você tem que ter a dependência do seu banco de dados em memória. E quando você for rodar o teste, o Spring automaticamente vai rodar os testes no banco de dados em memória.

[03:02] Agora, se você não quiser fazer os testes no banco de dados em memória, quiser utilizar o MySQL, o Oracle ou seja lá qual for o seu banco de dados, você vai ter que fazer algumas mudanças, algumas configurações a mais.

[03:14] Então vamos fazer essa mudança para nessa aula você ver como seria a segunda abordagem. Em cima da classe, além da anotação `@RunWith` e da `@DataJpaTest`, você deveria colocar mais uma anotação, que é essa que eu vou colar. Vou fazer um "Ctrl + Shift + O" para importar.

[03:30] Existe essa outra anotação com um nome meio esquisito: `@AutoConfigureTestDatabase`, que é para você dizer como vai ser a configuração do *test database*, do banco de dados para teste.

[03:42] E entre parênteses você passa esse valor: `(replace = AutoConfigureTestDatabase.Replace.NONE)`. Com isso eu estou falando para o Spring não substituir o banco de dados que eu estou configurando na aplicação pelo banco de dados H2. Porque esse é o comportamento padrão. Se você não colocar essa anotação ele vai substituir as suas configurações e utilizar o banco de dados em memória.

[04:10] Se você não quiser isso, coloca essa anotação e fala para ele não substituir as suas configurações. Colocou essa anotação, na próxima vez que você for rodar o teste ele vai considerar que não é para substituir as configurações do seu banco de dados.

[04:25] E lembra que as configurações estão no arquivo "application.properties". Só que no nosso caso as configurações já são do próprio H2. Então mesmo fazendo essa mudança não mudou nada, na verdade, porque ele já estava usando o H2 mesmo.

[04:42] Vamos imaginar que no seu "application.properties" está o MySQL. Então você teria as configurações do MySQL. Quando você configurasse com a anotação que criamos anteriormente, ele utilizaria o seu banco de dados MySQL.

[04:56] Porém, ele usaria o mesmo banco de dados que é usado pela aplicação. E no seu banco de dados provavelmente você já tem um monte de registros. Pode ser que o seu banco esteja vazio e isso vai influenciar nos testes.

[05:09] Então algo comum de se fazer e que é uma boa prática de testes é ter uma separação dos dados da aplicação, do ambiente de desenvolvimento de maneira isolada dos dados que eu uso para os testes.

[05:24] Costuma-se fazer isso criando dois bancos de dados distintos. Um para usar durante o desenvolvimento e um para usar durante os testes. E esse banco de teste geralmente estará vazio, ele sempre estará num estado que não vai interferir nos meus testes automatizados. Então vamos simular isso.

[05:41] E perceba que eu falei “ambiente”. Lembra que numa aula anterior falamos sobre ambientes, do *profile*? Então o teste é outra situação em que podemos precisar utilizar aquele esquema de *profile*. Vamos precisar fazer isso.

[05:58] No caso, como estamos falando de ambiente em relação a configurações, como fazemos isso? Nós temos o arquivo “application.properties”, mas imaginando que tem as configurações do meu banco de dados de desenvolvimento.

[06:13] Como que eu falo para o Spring que não é para usar essas propriedades, e sim as do teste? Eu vou ter que fazer algo do tipo `ifElse`. Se if estiver no ambiente de desenvolvimento, use essas propriedades. Se não (else), use essas outras propriedades.

[06:27] Só que na verdade não vai ser um `ifElse`, esse é um arquivo “properties”, não dá para fazer isso.

[06:31] A ideia é criar outro arquivo “application.properties”. Então vou dar um “Ctrl + C” e “Ctrl + V” nesse arquivo. Só que vou renomear como “application-test.properties”, para dizer que é o “application.properties” do ambiente do *profile* teste. E nesse arquivo você muda, você aponta para o seu banco de dados de teste.

[06:51] Então eu vou apagar tudo o que não tem a ver com o banco de dados, coisas do Actuator, de segurança. Vou deixar só as configurações em relação ao banco de dados.

[07:06] Como no nosso caso do curso é o mesmo banco de dados, o H2, eu só vou trocar o *database*. Eu vou falar que no "application-test" você vai usar um banco chamado `alura-forum-test`. Assim conseguimos simular, ver que ele está carregando de outro banco de dados.

[07:24] E pronto, eu tenho dois "application.properties". Um para um ambiente que eu não falei nada, e outro para um ambiente de teste. Mais para frente quando formos falar de Deploy veremos que vamos precisar ter um arquivo "properties" para cada ambiente.

[07:41] Agora vamos voltar para o teste. Eu falei que é para o Spring não substituir as minhas configurações. Só que por padrão ele vai carregar o "application.properties", porque quando ele rodar meu teste, como ele vai saber qual é o ambiente atual, qual é o *profile* ativo do momento?

[08:00] Lembra que eu mostrei para vocês como mudar o *profile*, no VM arguments? Nós vamos ver uma segunda estratégia de mudar o *profile*. Em cima da classe vamos colocar mais uma anotação, que é o `@ActiveProfiles`. Vou dar um "Ctrl + Shift + O". É uma anotação que vem do pacote do Spring, o "org.springframework.test".

[08:24] Isso é para dizer ao Spring que quando ele rodar a classe de teste, é para forçar esse *profile* como sendo ativo no momento. E nas aspas eu passo o *profile* de teste: `@ActiveProfiles("test")`.

[08:36] Então nos testes, como eu não estou rodando o projeto pela classe `ForumApplication`, por aquele Main, o Spring não vai ler aquele VM arguments. Então ele não vai saber qual é o *profile* ativo do momento. Ele vai carregar aquele "default".

[08:51] Então para eu dizer para o Spring forçar o *profile* ativo como sendo o de teste, eu coloco essa anotação `@ActiveProfiles("test")`. Quando rodar essa classe de teste ele vai considerar que o ambiente que está ativo no momento é o ambiente de teste.

[09:05] Vamos rodar esse teste só para ver o que vai acontecer. Clico com o botão direito, escolho "Run As > JUnit Test". A ideia é que agora ele considere que o *profile* ativo do momento seja o *profile* de teste. Ele vai ler o "application-teste.properties". E ele vai ver que eu estou usando outro banco de dados, que é o de teste.

[09:28] Só que como eu tenho o arquivo "data.sql", não importa se eu tenho dois *profiles*, ele vai carregar o "data.sql". Mas vamos considerar que eu quero que o banco de teste não carregue o "data.sql", porque eu vou considerar que sempre o meu banco de dados de teste está vazio.

[09:46] Então existe uma outra propriedade: `spring.datasource.initialization-mode=` e em seguida você fala qual é o modo de inicialização. Vou colocar *never*, então nunca inicialize meu banco: `spring.datasource.initialization-mode=never`.

[10:14] Então ele vai rodar com o banco de dados vazio, ele não vai ler o "data.sql". E no ambiente de testes é isso que eu quero, eu quero que ele não leia o "data.sql", eu quero que o banco esteja vazio.

[10:24] Agora se rodarmos novamente os testes, ele vai ler essa propriedade, não vai substituir as minhas configurações. Então se você tivesse o banco MySQL ele não ia sobrescrever pelo H2, ia continuar usando o MySQL.

[10:43] Como eu falei para ele usar o *profile* ativo `test`, se tiver no seu projeto um "application-test.properties", ele ia carregar esse "application.properties". E no caso eu falei qual é o banco de dados, para não ser o mesmo do ambiente de desenvolvimento, para ser o do ambiente de testes.

[11:02] E eu também disse para ele não inicializar o banco de dados. Então ele ia começar dessa maneira, os testes iam rodar em outro banco de dados, no ambiente de teste, numa base separada. E esse teste no meu caso deveria falhar, porque agora eu estou considerando que tem o curso HTML 5 no banco, mas na verdade não tem porque ele está vazio.

[11:22] Já terminou meu teste e falhou. Ele deu `AssertionError`. O `assertNotNull` falhou. Ele falou que está nula essa variável, ou seja, não encontrou um curso com nome HTML 5.

[11:34] E no caso desse cenário do teste com banco vazio, você deveria antes de rodar o teste inicializar o banco de dados conforme o cenário que você está testando.

[11:46] E para fazer isso você pode injetar uma outra variável: `private TestEntityManager em;`. A JPA tem o EntityManager, mas você pode usar esse teste EntityManager que é fornecido pelo próprio Spring Boot no módulo de teste. E ele já tem alguns métodos a mais para você.

[12:08] No caso, você deveria antes de persistir, criar um curso, só para simular: `Curso html5 = new Curso();`, `html5.setNome(nomeCurso)`. E na classe curso tem também a categoria, que é programação, `html5.setCategoria("Programacao");`.

[12:33] E então você pega o EntityManager e passa esse curso HTML 5: `em.persist(html5);`.

[12:40] Nessa segunda abordagem em que você está usando outro banco de dados e ele está vazio, antes de cada cenário você tem que popular de acordo com o cenário. Se nesse cenário eu quero buscar um curso que já existe, então primeiro eu tenho que criar esse curso, porque eu considero que o banco está vazio. Essa seria a ideia.

[12:55] Vamos rodar novamente e ver o que vai acontecer. No outro cenário de teste eu não preciso mexer, porque é o cenário que não existe um curso. Então eu não criei nada, não populei o banco de dados antes, mandei ele pesquisar um curso chamado JPA, ele vai devolver nulo. E nesse caso o `assertNull` vai dar certo, deveria passar esse segundo cenário de teste.

[13:19] Então essa é a segunda abordagem. Se você não quer usar o banco de dados em memória, quer testar usando o próprio banco de dados da aplicação, e quer fazer configurações para apontar para outra base de dados, você vai ter que complementar com a anotação `@AutoConfigureTestDatabase` e fazer as configurações usando o *profile* de teste, criando um arquivo "application-test.properties".

[13:40] E no "application-test.properties" você pode ter outras configurações distintas das que você utiliza no ambiente de desenvolvimento.

[13:46] O teste já está em execução, agora os dois testes deveriam passar. E realmente, os dois testes passaram normalmente.

[13:56] Então essa é uma segunda abordagem para você fazer testes de Controller. Vai depender do que você e o seu time de desenvolvimento achar mais apropriado. Vamos usar um banco em memória para os testes ou não? Vamos usar o mesmo banco de dados que usamos no ambiente de desenvolvimento, no ambiente de produção?

[14:11] Caso seja a segunda opção, você vai ter que fazer essas configurações a mais e os seus testes vão ter que lidar com esse cenário de o banco de dados estar vazio.

[14:19] Mas como o foco não é em estratégia de testes, não vou aprofundar muito nisso, vamos focar só na parte do Spring Boot.

[14:25] Espero que vocês tenham gostado desse vídeo. Continuamos no próximo vídeo, onde vamos fazer mais testes no nosso projeto. Vejo vocês nc

próximo vídeo. Um abraço e até lá.