09

Como configurar e usar bancos de dados

Já aprendemos a criar um **Produto** a partir de um formulário, mas ele ainda não é salvo em nenhum lugar. Vamos usar um banco de dados?

O *Play!* vem com um ORM pré-configurado para uso chamado *Ebean*, mas precisamos incluir algumas configurações para ativá-lo. Primeiro, precisamos incluir a seguinte linha de código no arquivo `project/plugins.sbt` para indicar que vamos utilizar o plugin do *Ebean*.

```
addSbtPlugin("com.typesafe.sbt" % "sbt-play-ebean" % "3.0.0")
```

Então, precisamos dizer ao *Play!* que queremos usar o *Ebean*, alterando a linha abaixo do arquivo `build.sbt` que fica na raiz do projeto. Inclua o `PlayEbean` na lista de plugins habilitados.

```
lazy val root = (project in file(".")).enablePlugins(PlayJava, PlayEbean)
```

Ainda no `build.sbt`, precisamos de um banco de dados! Para isso, precisamos incluir o conector do banco *MySQL* nas dependências do projeto.

```
libraryDependencies ++= Seq(
    javaJdbc,
    cache,
    javaWs,
    "mysql" % "mysql-connector-java" % "5.1.36"
)
```

E por fim precisamos configurar a qual banco nosso projeto precisa se conectar! Para isso, altere as configurações do banco de dados em um outro arquivo de configurações, o `conf/application.conf`. As seções de configuração (**play.db** e **db**) já devem existir, bastando descomentar as chaves e alterar o valor quando necessário. Caso não existam, adicione ao final do arquivo.

```
play.db {
    config = "db"
    default = "default"
}
db {
    default.driver = com.mysql.jdbc.Driver
    default.url = "jdbc:mysql://localhost/produtos_api"
    default.username = USERNAME_DO_MYSQL
    default.password = "SENHA_DO_MYSQL"
}
```

O primeiro bloco **play.db** define as chaves de configuração que usaremos para configurar nosso banco. Por exemplo, caso você mude o valor da chave `play.db { default }` para `produtos`, o a configuração total ficaria assim:

```
play.db {
    config = "db"
    default = "produtos"
}
db {
    produtos.driver = com.mysql.jdbc.Driver
    produtos.url = "jdbc:mysql://localhost/produtos_api"
    produtos.username = USERNAME_DO_MYSQL
    produtos.password = "SENHA_DO_MYSQL"
}
```

Isso serve para que você possa configurar diversos bancos de dados para uso na sua aplicação!

A última configuração necessária é indicar ao *Ebean* quais classes ele deve conectar ao banco. No caso, usaremos as classes dentro do pacote **models**. Adicione ao fim do arquivo `conf/application.conf` a seguinte linha.

```
ebean.default = ["models.*"]
```

Se não tiver criado ainda, crie o banco de dados usando o mesmo nome que usou no arquivo de configuração. Conecte-se ao *MySQL* e rode a query `create database produtos_api` .

Agora que temos um banco de dados configurado, precisamos indicar ao *Play!* que nossa classe **Produto** representa uma tabela anotando com `@Entity` e fazer com que o *Ebean* a reconheça, estendendo a classe `Model` . Precisamos também definir um atributo de **id** para ele.

```
import javax.persistence.*;
@Entity
public class Produto extends com.avaje.ebean.Model {
    @Id @GeneratedValue
    private Long id;
    // getter e setter do id
}
```

Neste momento, precisamos acessar alguma página do nosso projeto no browser. Isso serve para que o *Play!* reconheça as mudanças nos modelos e sugira uma evolução, chamada de migração em outros frameworks como *Rails*, para adequar o banco de dados ao novo estado. Aceite que a tabela **produto** seja criada!

Enfim, com toda a configuração pronta e o banco de dados modelado, podemos salvar nosso **Produto** ao recebê-lo no controller. Aproveite para redirecionar o usuário de volta à página de produtos após salvar, utilizando o objeto `routes` presente em todo controller!

```
public Result salvaNovoProduto() {
    Form<Produto> formulario = formularios.form(Produto.class).bindFromRequest();
    Produto produto = formulario.get();
    produto.save();
    return redirect(routes.ProdutoController.formularioDeNovoProduto());
}
```

Para garantir que está tudo ok, cadastre um novo **Produto** e confira se você foi redirecionado e também se os dados foram cadastrados no banco com a query `select * from produto;` !