11

Optativa - Usando o Hibernate

Vimos como configurar e utilizar o *Ebean* para acessar o banco de dados, mas é possível também usar o *Hibernate*! Que tal aprender a configurá-lo?

Para começo de conversa, se já criou, remova todas as tabelas do banco de dados do projeto. Também volte o projeto para a versão 2.2, antes de configurarmos o Ebean. Precisaremos fazer algumas das mesmas configurações, mas é mais fácil começar do zero do que alterar o projeto já configurado. Então vamos lá, passo a passo:

1. Altere o arquivo `build.sbt`, incluindo três novas dependências: o suporte a JPA, o Hibernate e o MySQL.

```
libraryDependencies ++= Seq(
 ...
 javaJpa,
 "mysql" % "mysql-connector-java" % "5.1.36",
 "org.hibernate" % "hibernate-entitymanager" % "5.1.0.Final"
)
```

2. Altere o arquivo `conf/application.conf`, descomentando e adicionando onde necessário ou adicionando tudo 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 = USUARIO
 default.password = "SENHA"
}
db.default.jndiName = DefaultDS
jpa.default = produtos
```

3. Crie o arquivo `conf/META-INF/persistence.xml` com o seguinte conteúdo:

```
<persistence xmlns="http://xmlns.jcp.org/xml/ns/persistence"
          xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/persistence http://xmlns.jcp.org/x
          version="2.1">

 <persistence-unit name="produtos" transaction-type="RESOURCE_LOCAL">
     <provider>org.hibernate.jpa.HibernatePersistenceProvider</provider>
     <non-jta-data-source>DefaultDS</non-jta-data-source>
     <properties>
         <property name="hibernate.dialect" value="org.hibernate.dialect.MySQL5Dialect"/>
         <property name="hibernate.hbm2ddl.auto" value="update" />
     </properties>
```

```
 </persistence-unit>
</persistence>
```

4. Adicione um **id** e as configurações do banco de dados no modelo de **Produto**.

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

Com isso temos o banco configurado, mas ainda não conseguimos acessá-lo: salvar e carregar os produtos. Para isso precisaremos utilizar o gerenciador de entidades, o `EntityManager`. Recarregue as configurações do seu projeto no console com o comando `reload` e atualize as dependências com `clean; eclipse`, subindo em seguida com `~run`. Quando terminar de subir o projeto, atualize o projeto no *Eclipse* dando um `F5` no projeto. Enfim podemos acessar o banco de dados.

Vamos então salvar nosso produto! Para isso usaremos uma classe que gera o `EntityManager` pra gente, a `JPAApi`, que pode ser injetada!

```
public class ProdutoController extends Controller {
    @Inject
    private JPAApi jpa;
    public Result salvaNovoProduto() {
        Form<Produto> formulario = formularios.form(Produto.class).bindFromRequest();
        Produto produto = formulario.get();
        jpa.em().persist(produto);
        return redirect(routes.ProdutoController.formularioDeNovoProduto());
    }
}
```

E pronto! Confira no seu banco de dados que existe uma nova tabela chamada **Produto** e que ela contém seu novo produto cadastrado!