02

Usando anotações e consultas no banco

Precisamos garantir que nosso **Produto** salvo no banco esteja adequado: que ele tenha um nome, um tipo, um código e um preço! Faremos então uma validação dos dados do formulário utilizando um sistema embutido no *Play!*.

O código a seguir confere se o formulário tem erros e em caso positivo renderiza novamente o formulário, já preenchido com os dados anteriores.

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

Porém isso não basta, pois precisamos dizer quais as restrições que cada campo tem. Para isso utilizaremos a anotação `@Required` nos campos que são obrigatórios, passando como parâmetro uma mensagem a ser exibida caso o campo seja deixado em branco.

```
import play.data.validation.Constraints;

@Required(message = "Produto precisa ter um título")
private String titulo;
@Required(message = "Produto precisa ter um código")
private String codigo;
@Required(message = "Produto precisa ter um tipo")
private String tipo;
@Required(message = "Produto precisa ter um preço")
private Double preco;
```

A restrição do modelo é automaticamente inclusa também no na view com o atributo HTML `required`, então na maior parte das vezes o usuário vai se deparar com a validação HTML e não verá a mensagem de erro. Mas um espertinho que altere o formulário e remova o atributo de algum campo dará de cara com um campo vermelho e a mensagem!

Existem outras validações que precisam ser feitas. Temos que conferir se já existe um **Produto** cadastrado no banco de dados com o código utilizado ou também se o preço do **Produto** é maior ou igual a zero, pois não podemos vender nada com preço negativo. No caso do preço, fazemos a seguinte lógica:

```
public Result salvaNovoProduto() {
    Form<Produto> formulario = formularios.form(Produto.class).bindFromRequest();
    Produto produto = formulario.get();
    if (produto.getPreco() < 0.0) {
        formulario.reject(new ValidationError("preco", "Preço tem que ser maior ou igual a zero'
    }
    if (formulario.hasErrors()) {
```

```
        return badRequest(formularioDeNovoProduto.render(formulario));
    }
```

Caso o preço seja negativo, rejeitamos o campo do formulário com um erro de validação, um *ValidationError*, que recebe o nome do campo e uma mensagem de erro.

Já para conferir se o código é realmente único, precisamos fazer consultas ao banco de dados, então vamos criar um objeto de acesso a dados, ou *Data Access Object*, um **DAO**, com um método que retorne um produto com dado código.

```
package daos;
public class ProdutoDAO {
    private Finder<Long, Produto> produtos = new Finder<>(Produto.class);
    public Produto comCodigo(String codigo) {
        return produtos
            .where()
            .eq("codigo", codigo)
            .findUnique();
    }
}
```

Fazemos então a validação do código. Caso o *DAO* retorne algum **Produto** com o código inserido, há repetição e portanto o formulário deve ser rejeitado.

```
@Inject
private ProdutoDAO produtoDAO;
public Result salvaNovoProduto() {
    Form<Produto> formulario = formularios.form(Produto.class).bindFromRequest();
    Produto produto = formulario.get();
    if (produtoDAO.comCodigo(produto.getCodigo()) != null) {
        formulario.reject(new ValidationError("codigo", "Já existe um produto com este código."));
    }
    if (produto.getPreco() <= 0.0) {
        formulario.reject(new ValidationError("preco", "Preço tem que ser positivo."));
    }
     if (formulario.hasErrors()) {
    // ...
}
```

E pronto, temos uma validação funcional para nosso **Produto**!