02

Trabalhando Melhor com as views

Nos capítulos anteriores desenvolvemos uma página que mostra a lista de todos os produtos cadastrados no banco de dados, porém nessa página, para não colocarmos muitas informações, listamos o `Id` , `Nome` e `Quantidade` dos produtos. Vamos agora desenvolver uma nova página onde podemos visualizar todas as informações de um produto cadastrado no banco de dados.

Vamos então adicionar um novo método no `ProdutoController` que recebe um id, busca no banco o produto com `Id` igual ao recebido e envia essa informação para ser exibida na view. Vamos chamar essa action de `Visualiza` .

```
public ActionResult Visualiza(int produtoId)
{
    ProdutosDAO dao = new ProdutosDAO();
    Produto produto = dao.BuscaPorId(produtoId);
    ViewBag.Produto = produto;
    return View(produto);
}
```

Vamos agora implementar a view para a action `Visualiza` .

```
Id: @ViewBag.Produto.Id<br />
Nome: @ViewBag.Produto.Nome<br />
Categoria: @ViewBag.Produto.Categoria.Nome<br />
Preço: @ViewBag.Produto.Preco<br />
Descrição: @ViewBag.Produto.Descricao<br />
Quantidade: @ViewBag.Produto.Quantidade<br />

<a href="/Produto">Voltar para a lista de produtos</a>
```

Repare que codificamos o link para a lista de produtos diretamente na view. O problema disso é que, se um dia o link mudar, precisaremos mudar em todos os HTMLs que apontem para o mesmo endereço! Muito trabalhoso!

Podemos utilizar um novo método do `HtmlHelper` , o `ActionLink` . Esse método cria o código html de um link de acordo com os parâmetros passados. Queremos definir que o texto do link deve ser `Voltar para a lista de produtos` e que esse link deve nos redirecionar para a action `Index` da classe `ProdutoController` , para isso vamos utilizar a versão sobrecarregada de `ActionLink` que recebe o texto do link, o nome da action e o nome do controller.

```
@Html.ActionLink(texto, nomeDaAction, nomeDoController)
```

Em nosso caso, queremos inserir a seguinte chamada:

```
@Html.ActionLink("Voltar para a lista de produtos", "Index", "Produto")
```

Vamos agora inserir o link para a página de visualização para cada item da lista de produtos. Queremos criar links que enviem o usuário para a action `Visualiza` do `ProdutoController` passando `produtoId` como argumento. Como muitas vezes

uma action deve receber um identificador e realizar sua tarefa de acordo com o id recebido, o ASP.NET MVC configura a rota padrão com um id genérico como argumento opcional. Dessa forma podemos criar urls da forma `/Produto/Visualiza/1` .

Para que a action receba o valor do id capturado pelo ASP.NET MVC, devemos fazer com que ela receba um argumento chamado `id` de qualquer tipo. Em nosso caso, queremos receber um id inteiro, logo vamos substituir `produtoId` por `id` na action `Visualiza` :

```
public ActionResult Visualiza(int id)
{
    ProdutosDAO dao = new ProdutosDAO();
    Produto produto = dao.BuscaPorId(id);
    ViewBag.Produto = produto;
    return View(produto);
}
```

Vamos modificar a lista de produtos para que ela tenha links para a visualização. Abra o arquivo `Views/Produto/Index.cshtml` e localize o laço que percorre a lista de produtos:

```
@foreach (var produto in ViewBag.Produtos)
{
    <tr>
        <td>@produto.Id</td>
        <td>@produto.Nome</td>
        <td>@produto.Quantidade</td>
    </tr>
}
```

Queremos que produto.Nome seja um link para a página de visualização. Para realizar essa tarefa, podemos criar o link diretamente:

```
<td><a href="/Produto/Visualiza/@produto.Id">@produto.Nome</a></td>
```

ou utilizar o `ActionLink` do `HtmlHelper` . O `ActionLink` possui uma versão sobrecarregada que recebe o texto do link, o nome da action e um objeto cujos atributos serão utilizados para preencher os parâmetros da url. Nesse caso, queremos enviar um objeto que tenha um atributo chamado id contendo o valor do id do produto:

```
new { id = produto.Id }
```

Logo o código do link fica da seguinte forma:

```
<td>@Html.ActionLink(produto.Nome, "Visualiza", new { id = produto.Id })</td>
```

Porém temos um problema, o código da view não compila! O problema é que o compilador não sabe qual é o tipo da variável produto, logo não consegue descobrir que a expressão `produto.Nome` gera uma `String` e `produto.Id` gera um `int` . Para resolver esse problema devemos fazer com que a variável `produto` seja do tipo `Produto` :

```
@foreach(CaelumEstoque.Models.Produto produto in ViewBag.Produtos)
```

Como agora o tipo de `produto` é conhecido, o código do `ActionLink` funciona!

Views fortemente tipadas

Ao contrário do que acontece no código do controller, nas views não temos nenhum auxílio da IDE, o Visual Web Developer. Isso ocorre pois as variáveis enviadas para a view através da `ViewBag` são dinâmicas, ou seja, o uso da variável só é verificado na execução do código e não na compilação. No ASP.NET MVC esse problema é resolvido através de **Views Fortemente Tipadas** (Strongly Typed View).

Para utilizarmos uma view fortemente tipada, precisamos enviar a a variável para a view de uma forma diferente. Ela precisa ser enviada como argumento do método `View`. Por exemplo, na action `Index` do `ProdutoController`, a lista de produtos pode ser enviada através do seguinte código:

```
public ActionResult Index()
{
    ProdutosDAO dao = new ProdutosDAO();
    IList<Produto> produtos = dao.Lista();
    return View(produtos);
}
```

A variável passada como argumento do método `View` pode ser acessada pela view através de uma variável chamada `Model`, essa variável é considerada pelo Asp.Net MVC como a variável principal da view. Agora precisamos modificar a lista de produtos (`Views/Produto/Index.cshtml`) para que ela utilize a `Model` ao invés de `ViewBag.Produtos` :

```
@foreach(CaelumEstoque.Models.Produto produto in Model)
```

Mas apenas essa modificação não é suficiente para o Asp.Net MVC saber o tipo da variável `Model`, no início do código da view, precisamos indicar qual é o tipo da variável `Model` através do comando `@model` do Razor:

```
@model IList<CaelumEstoque.Models.Produto>
```

Com isso estamos declarando qual é o tipo da variável `Model` que está sendo utilizada na view e por isso podemos utilizar livremente a inferência de tipos no código do `foreach` :

```
@foreach(var produto in Model)
```

Como agora o Razor sabe que a variável `Model` é do tipo `IList<CaelumEstoque.Models.Produto>`, a inferência de tipos deduzirá que a variável `produto` declarada no `foreach` é do tipo `CaelumEstoque.Models.Produto` ao invés do tipo dinâmico.