05

Tela de carrinho e método Include

Transcrição

Já que estamos salvando os itens de pedido, começaremos a mostrá-los na *view* de carrinho, que modificaremos, pois atualmente ela está estática e com o HTML fixo no arquivo. Abriremos `Carrinho.cshtml` e começaremos definindo o modelo a ser passado à *view* no início do arquivo, por meio da diretiva `@model`, seguido de seu tipo.

Ao voltarmos ao `PedidoController.cs`, veremos que chamamos uma *view* na *action* de `Carrinho()` passando `pedido.Itens`, e que os itens de pedido são uma lista de `ItemPedido`. Assim, modificaremos `Carrinho.cshtml` para passarmos como modelo um `IList` de `ItemPedido`.

```
@{
    ViewData["Title"] = "Carrinho";
}
@model IList<ItemPedido>;
```

Faremos alterações para o mecanismo do Razor, que é o motor de renderização do ASP.NET Core MVC para a introdução das informações do modelo no HTML. Faremos isto identificando as partes que queremos trocar nesta *view*. Na `div` de classe `panel-body`, temos as linhas dos produtos, que iremos colapsar:

```
<div class="panel-body">
    <div class"row row-center linha-produto">...</div>
    <div class"row row-center linha-produto">...</div>
    <div class"row row-center linha-produto">...</div>
</div>
```

Isolaremos o trecho acima e, tendo as três linhas identificadas, deixaremos somente a primeira e deletaremos as restantes. Criaremos um laço em torno deste trecho que contém somente uma linha de item de pedido para conseguirmos renderizá-la quantas vezes for necessário e montar os itens do carrinho.

Como vamos inserir um código C#, usaremos `@`, e uma estrutura de repetição `foreach`, com a `div`. Formataremos o código com "Ctrl + K + D".

```
<div class="panel-body">

    @foreach (var item in Model)
    {
        <div class"row row-center linha-produto">...</div>
    }
</div>
```

Precisaremos identificar as partes que estão fixas e precisam ser substituídas, como o nome da imagem de um produto, por exemplo, `/images/produtos/large_047.jpg`, em que `047` indica o código. O `src` ficará da seguinte forma: `/images/produtos/large_@(item.Produto.Codigo).jpg`.

Mais para baixo, no código, temos o nome e preço do produto:

```
<div class="col-md-3">Arduino Pr&#xE1;tico</div>
<div class="col-md-2 text-center">R$ 69,90</div>
```

Que alteraremos para:

```
<div class="col-md-3">@(item.Produto.Nome)</div>
<div class="col-md-2 text-center">R$ @(item.PrecoUnitario)</div>
```

Mais adiante, encontramos a quantidade dos itens, no caso, `2`:

```
<input type="text" value="2"
             class="form-control text center" />
<span class="input-group-btn">
    <button class="btn btn-default">
        <span class="glyphicon-plus"></span>
    </button>
</span>
```

Que será substituído por `@(item.Quantidade)`. Da mesma forma, mais para baixo há:

```
<div class="col-md-2">
    R$ <span class="pull-right" subtotal>
            139,80
    </span>
</div>
```

Substituiremos o valor `139,80` por `@(item.Quantidade * item.PrecoUnitario)`, com uma operação matemática em C# porque não temos `SubTotal`. Adiante, temos o somatório, os totalizadores deste carrinho:

```
<div class="col-md-10">
    <span numero-itens>
        Total: 3
        itens
    </span>
</div>
<div class="col-md-2">
    Total: R$ <span class="pull-right" total>
            419,40
    </span>
</div>
```

Em que substituiremos `3` por `@(Model.Count())`, e `419,40`, o valor total do carrinho, por `@(Model.Sum(i => i.Quantidade * i.PrecoUnitario))`, que é a soma dos subtotais dos elementos do modelo. Com isso, temos a *view* pronta para ser renderizada com os dados do modelo, portanto salvaremos o arquivo e rodaremos a aplicação no navegador.

Vamos adicionar um produto qualquer pressionando o botão "Adicionar" e seremos direcionados à tela de carrinho. Então, clicaremos em "Adicionar Produtos" para voltarmos à tela anterior e adicionarmos outro produto. Faremos isso mais uma vez, com outro livro. As telas de carrinho mostram exatamente o produto escolhido, um de cada vez, e isto é um problema...

Os produtos não são acumulados no carrinho, por conta de como a consulta obtém o pedido a ser exibido na tela de carrinho. Vamos pausar a aplicação e voltar ao `PedidoRepository.cs` em que encontraremos `GetPedido()` para a obtenção do pedido a partir do nosso DbSet, cujo código faz uma consulta em cima de uma entidade, que na verdade é convertido para uma consulta SQL do SQL Server.

Isso não é o que esperamos, pois além da consulta ao pedido, é necessário incluir as entidades que fazem parte dela. Isto é, se escrevêssemos uma consulta SQL Server, teríamos que colocar também as junções, a cláusula `join` para podermos juntar as tabelas.

Junto ao pedido, usaremos `Include()`, e resolveremos a referência com "Ctrl + .", passando uma expressão lambda, indicando a entidade a ser incluída. Além disso, acrescentaremos os produtos que o item utiliza, desta vez, não com `Include()`, e sim `ThenInclude()`, algo como "inclua em seguida de". O código, portanto, ficará da seguinte maneira:

```
var pedidoId = GetPedidoId();
var pedido = dbSet
        .Include(p => p.Itens)
            .ThenInclude(i => i.Produto)
        .Where(p => p.Id == pedidoId)
        .SingleOrDefault();
```

Vamos rodar a aplicação mais uma vez. Chegaremos à tela de carrinho, passando primeiro pela tela do carrossel. Adicionaremos um produto qualquer ao carrinho, voltaremos à tela anterior para adicionar outro, e mais um. Todos os produtos selecionados são listados na tela do carrinho, como gostaríamos.

Aprendemos como fazer uma consulta utilizando o `Include()` e o `ThenInclude()`, bem como renderizar a nossa tela de carrinho.