02

Ajax

Manipulação da quantidade em estoque

Nesse capítulo vamos desenvolver a funcionalidade de manipular a quantidade de produtos que está gravada no banco de dados, então dentro do `ProdutoController` vamos desenvolver uma nova action chamada `DecrementaQtd` que recebe o `id` do produto que será decrementado:

```
public ActionResult DecrementaQtd(int id)
{
}
```

Dentro desse método, precisamos buscar o produto do banco de dados, decrementar sua quantidade, atualizar as informações e por fim redirecionar o usuário para a página de listagem de produtos para que ele veja as informações atualizadas:

```
public ActionResult DecrementaQtd(int id)
{
    ProdutosDAO dao = new ProdutosDAO();
    Produto produto = dao.BuscaPorId(id);
    produto.Quantidade--;
    dao.Atualiza(produto);
    return RedirectToAction("Index");
}
```

Agora na listagem de produtos, podemos adicionar mais uma coluna com um link que chama essa nova action:

```
@foreach(var produto in Model)
{
   <tr>
      <td>@produto.Id</td>
      <td>@Html.ActionLink(produto.Nome, "Visualiza", new {id=produto.Id})</td>
      <td>@produto.Quantidade</td>
      <td><a href="/Produto/DecrementaQtd/@produto.Id">Decrementar</a></td>
   </tr>
}
```

Podemos testar essa nova funcionalidade da aplicação acessando a lista de produtos, url `/produtos` . Quando clicarmos no link de decrementar a quantidade, podemos ver que a quantidade é realmente decrementada no banco de dados, porém a página é recarregada.

Esse recarregamento ocorre pois quando fazemos uma requisição web, o navegador sempre espera a resposta devolvida pelo servidor e depois faz o recarregamento da página, esse é o modo síncrono de requisições web. Para evitarmos o recarregamento das páginas, precisamos utilizar o modo assíncrono de navegação.

Requisições assíncronas com javascript

Para fazermos requisições assíncronas em uma aplicação web, precisamos programar o navegador utilizando a linguagem javascript. Requisições assíncronas feitas por javascript são conhecidas como requisições **ajax**.

Cada navegador implementa as requisições assíncronas de uma forma diferente, e para lidar com essa diferença entre os navegadores do mercado, utilizaremos uma biblioteca javascript chamada jquery (que é colocada pelo visual studio dentro da pasta `Scripts` do projeto).

Para podermos utilizar a biblioteca jquery, precisamos inicialmente importá-la dentro do cshtml da lista de produtos, fazemos isso com o seguinte código:

```
<script type="text/javascript" src="~/Scripts/jquery-1.10.2.js"></script>
```

Agora que importamos a biblioteca jquery, precisamos declarar um bloco de código javascript que realmente fará a requisição ajax:

```
<script type="text/javascript">
   // o script fica aqui
</script>
```

Para fazermos uma requisição ajax do tipo `get` utilizando o jquery, utilizamos a função `$.get` definida na biblioteca e para o `post`, o `$.post`.

Nessas funções, o primeiro argumento que colocamos é a url para onde queremos enviar a requisição e o segundo representa os parâmetros que serão enviados para o servidor:

```
$.post(url, params);
```

No nosso caso, a url é `/Produto/DecrementaQtd`. Já os parâmetros precisam ser enviados através de um dicionário do javascript, onde declaramos o nome do parâmetro e o valor que será associado àquele nome:

```
var url = "/Produto/DecrementaQtd";
var params = { id: 1 };
$.post(url, params);
```

Com esse código, faremos uma requisição ajax para decrementar a quantidade do produto que tem `id` igual a 1. Para podermos reutilizá-lo para os outros produtos cadastrados, vamos isolá-lo dentro de uma função javascript que recebe o id do produto que será decrementado:

```
function decrementa(produtoId){
  var url = "/Produto/DecrementaQtd";
  var params = { id: produtoId };
  $.post(url, params);
}
```

Mas veja que dentro do código da função colocamos manualmente a url para a action `DecrementaQtd` do servidor. O que aconteceria se essa url fosse modificada (pelo `RouteAttribute`, por exemplo)? Nesse caso teríamos que lembrar de modificar também esse código javascript. Para resolvermos esse problema, podemos utilizar um objeto do Asp.Net MVC responsável

por gerar o endereço das actions na camada de visualização da aplicação, o **UrlHelper**, que pode ser acessado através da variável `@Url` dentro do código do cshtml.

Para gerarmos uma url com o `UrlHelper` , utilizamos o método `Action` informando qual é a action e qual é o controller que queremos acessar:

```
@Url.Action("DecrementaQtd", "Produto")
```

A função `decrementa` do javascript utilizando o `UrlHelper` fica da seguinte forma:

```
function decrementa(produtoId){
  var url = "@Url.Action("DecrementaQtd", "Produto")";
  var params = { id: produtoId };
  $.post(url, params);
}
```

Agora precisamos fazer com que o clique no link de decremento chame a função `decrementa` que acabamos de declarar no código javascript. Então vamos inicialmente modificar o link para que quando clicado, o navegador não saia da lista de produtos:

```
<a href="#">Decrementar</a>
```

Para executarmos uma função no clique do link precisamos utilizar o atributo `onclick` dentro da declaração da tag `a` colocando a chamada para o decrementa com o id do produto correto:

```
<a href="#" onclick="decrementa(@produto.Id);">Decrementar</a>
```

Com essa modificação, toda vez que o usuário clicar em um link, o navegador enviará uma requisição ajax para o servidor para decrementar a quantidade de um produto cadastrado.

Atualizando a página com o jQuery

Fizemos com que os links da tabela enviem requisições ajax para o servidor para decrementar a quantidade de um produto cadastrado, porém a quantidade que está sendo mostrada na tabela não está sendo atualizada.

Quando o servidor terminar de responder a requisição ajax, caso ela seja bem sucedida, queremos atualizar o valor da quantidade que está sendo exibida. Para executarmos uma função quando uma requisição ajax é bem sucedida, precisamos utilizar o terceiro argumento do `$.post` . Esse argumento recebe a função que será executada caso a requisição seja bem sucedida:

```
function decrementa(produtoId){
  var url = "/Produto/DecrementaQtd";
  var params = { id: produtoId };
  $.post(url, params, atualiza);
}

function atualiza(){
```

```
  // código que será executado depois do ajax.
}
```

Dentro do `atualiza`, podemos, por exemplo, mostrar uma caixa de mensagens se a requisição foi bem sucedida:

```
function atualiza(){
  alert("Sucesso");
}
```

Agora quando clicarmos em um dos links de decremento, o navegador fará uma requisição ajax e o usuário verá a mensagem `"Sucesso"`. Mas ao invés de mostrar uma mensagem, queremos atualizar a quantidade atual exibida na tabela. Para isso, utilizaremos o jQuery.

Para conseguirmos manipular o conteúdo de uma página utilizando o jquery, precisamos inicialmente buscar o elemento que queremos manipular, no caso o `td` que mostra a quantidade atual do produto. Para buscarmos o elemento mais facilmente, colocaremos um `id` na declaração do `td` da quantidade:

```
<td id="quantidade">@produto.Quantidade</td>
```

Mas com esse código, o mesmo `id` será colocado para todos os produtos. Para conseguirmos um `id` único para cada produto, precisamos, por exemplo, colocar o `@produto.Id`:

```
<td id="quantidade@produto.Id">@produto.Quantidade</td>
```

Mas se executarmos o código, veremos que no html gerado, o id de todos os `td`s ficou com o código: `quantidade@produto.Id`. Isso aconteceu porque `quantidade@produto.Id` se parece com um endereço de e-mail válido e por isso o Razor não interpreta o código para ler o id do produto. Para forçarmos o Razor a interpretar o código, precisamos colocar parêntesis em volta da expressão que deve ser executada:

```
<td id="quantidade@(produto.Id)">@produto.Quantidade</td>
```

Agora dentro do código do atualiza, precisamos do id e da quantidade do produto que foi modificado para conseguirmos atualizar a página. Para podermos utilizar mais facilmente as informações de um objeto devolvido pelo servidor, nós faremos com que o servidor devolva uma resposta em um formato que pode ser utilizado facilmente pelo javascript, o **Json** (**Javascript Object Notation**).

Para devolvermos o Json do produto do servidor, utilizamos mais um método herdado da classe `Controller` chamado `Json` passando qual é o objeto que queremos devolver como resposta:

```
public ActionResult DecrementaQtd(int id)
{
    ProdutosDAO dao = new ProdutosDAO();
    Produto produto = dao.BuscaPorId(id);
    produto.Quantidade--;
    dao.Atualiza(produto);
    return Json(produto);
}
```

Para podermos utilizar a resposta do servidor dentro da função `atualiza`, precisamos apenas colocar um argumento na declaração da função:

```
function atualiza(resposta) {
  // usa a resposta devolvida
}
```

Dentro da variável `resposta` temos o Json do produto que foi devolvido pelo servidor. Como a resposta está no formato Json, podemos ler as propriedades do produto diretamente do Json. Para lermos, por exemplo, a `Quantidade` do produto, utilizamos o código: `resposta.Quantidade`. Podemos, por exemplo, mostrar a quantidade atualizada com o seguinte código:

```
function atualiza(resposta) {
  alert(resposta.Quantidade);
}
```

Agora que já sabemos como utilizar o Json, aprenderemos como utilizar o jquery para atualizar a página. O primeiro passo é buscarmos o elemento que queremos atualizar dentro do html da página. Para isso precisamos utilizar a função `$` definida pelo jquery passando o id do elemento que queremos buscar com o prefixo `#`:

```
function atualiza(resposta) {
  var elemento = $("#quantidade" + resposta.Id);
}
```

Agora que temos o elemento que será atualizado, precisamos utilizar a função `html` do jquery para atualizar o conteúdo do html do elemento:

```
function atualiza(resposta){
  var elemento = $("#quantidade" + resposta.Id);
  elemento.html(resposta.Quantidade);
}
```

Agora nossa página consegue decrementar a quantidade utilizando requisições ajax para o servidor e também atualizar o conteúdo da página utilizando a resposta que foi devolvida. O código final do `Views/Produto/Index.cshtml`:

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

<table>
  <thead>
    <tr>
      <th>Id</th>
      <th>Nome</th>
      <th>Quantidade</th>
      <th></th>
    </tr>
  </thead>
  <tbody>
    @foreach(var produto in Model)
    {
      <tr>
```

```
            <td>@produto.Id</td>
            <td>@produto.Nome</td>
            <td id="quantidade@(produto.Id)">@produto.Quantidade</td>
            <td><a href="#" onclick="decrementa(@produto.Id);">Decrementar</a></td>
          </tr>
        }
      </tbody>
    </table>

    <script type="text/javascript" src="~/Scripts/jquery-1.10.2.js"></script>
    <script type="text/javascript">
      function decrementa(produtoId){
        var url = "@Url.Action("DecrementaQtd", "Produto")";
        var params = {id : produtoId};
        $.post(url, params, atualiza);
      }

      function atualiza(resposta){
        var elemento = $("#quantidade" + resposta.Id);
        elemento.html(resposta.Quantidade);
      }
    </script>
```