08

Ajax com jQuery

Transcrição

O método `UpdadeQuantidade()` do `PedidoController` está pronto para receber requisições http POST, mas essas requisições irão enviar dados no corpo da requisição e não na url. Precisamos fazer uma adaptação nos parâmetros do método: ao invés de dois parâmetros, passaremos apenas um que será um objeto que conterá as duas informações id e quantidade.

Inseriremos uma instância da classe `ItemPedido` .

```
namespace CasaDoCodigo.Controllers
{

<****!****>

public IActionResult Resumo()
        {
            return View(pedidoRepository.GetPedido());
        }

                [HttpPost]
                public void UpdateQuantidade(itemPedido itemPedido)`
                {
                }

    }
}
```

`ItemPedido` contém tanto o id, que está em `BaseModel` como `Quantidade` , uma propriedade do `ItemPedido` .

```
public class ItemPedido: BaseModel
{
    [Required]
    public Pedido Pedido { get; private set; }
    [Required]
    public Produto Produto { get; private set; }
    [Required]
    public int Quantidade {get; private set; }
    [Required]
    public decimal PrecoUnitario { get; private set; }

<****!****>
```

Como iremos passar `ItemPedido` a partir do corpo da requisição, devemos anotar esse parâmetro com um atributo que indica sua proveniência, portanto adicionaremos `FromBody` .

```
namespace CasaDoCodigo.Controllers
{

<****!****>
```

```
public IActionResult Resumo()
        {
            return View(pedidoRepository.GetPedido());
        }

                [HttpPost]
                public void UpdateQuantidade([FromBody]itemPedido itemPedido)`
                {
                }


    }
}
```

Com isso, podemos voltar ao nosso código JavaScript e programar a requisição. Esse tipo de requisição utiliza uma técnica chamada `ajax` do jQuery, uma função que exige parâmetros.

```
@{
    ViewData["Title"] = "Carrinho";
}

<****!****>

@section Scrips
{
    <script type="text/javascript">
        function clickIncremento(btn) {
            var linhaDoItem = $(btn).parents('[item-id]');
            var itemId = $(linhaDoItem).attr('idem-id');
            var novaQtde = $(linhaDoItem).find('input').val();

                        $.ajax({

                        });

                        debugger;
        }
    </script>
```

Passaremos para `ajax` as propriedades do objeto JavaScript, mas antes vamos compreender melhor esta função. Ajax é uma sigla: **A***synchronous* ***J**avascript **X**ml. Inicialmente tratava-se de uma técnica que utilizava o JavaScript e XML para fazer requisições a sincronizar o servidor, o que significa que o seu código não vai aguardar a resposta da requisição para continuar rodando. O XML caiu em desuso para o Ajax, e ultimamente se usa o JSON para formatação de objetos.

Para fazermos uma requisição ajax utilizando o jQuery precisaremos quatro parâmetros:

1. **url**: endereço do nosso método do controller(pedido/updatequantidade)
2. **type**: o tipo de requisição http que escolhemos (POST)
3. **contentType**: o formado dos dados (JSON)
4. **data** : o objeto que levará os dados do cliente para o servidor

Isso esclarecido, iremos inserir esses quatro parâmetros em nossa função `ajax` :

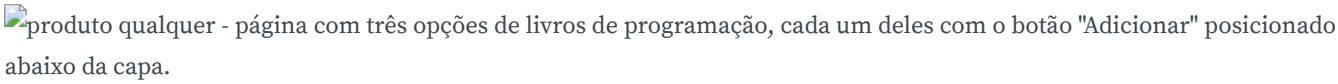
```
@{
    ViewData["Title"] = "Carrinho";
}

<****!****>

@section Scrips
{
    <script type="text/javascript">
        function clickIncremento(btn) {
            var linhaDoItem = $(btn).parents('[item-id]');
            var itemId = $(linhaDoItem).attr('idem-id');
            var novaQtde = $(linhaDoItem).find('input').val();

                        $.ajax({
                            url:'/pedido/updatequantidade',
                            type: 'POST',
                            contentType: 'application/json',
                            data:

                        });

                        debugger;
        }
    </script>
```

Antes de adicionarmos o objeto em `data` , iremos definir esse objeto em uma variável: ele terá a informação `Id` e `Quantidade` , essa segunda terá o conteúdo da variável `novaQtdade` . No entanto, não podemos utilizar o objeto sem antes transforma-lo em uma string, portanto adicionaremos `JSON.stringfy` para realizar essa conversão.

```
@{
    ViewData["Title"] = "Carrinho";
}

<****!****>

@section Scrips
{
    <script type="text/javascript">
        function clickIncremento(btn) {
            var linhaDoItem = $(btn).parents('[item-id]');
            var itemId = $(linhaDoItem).attr('idem-id');
            var novaQtde = $(linhaDoItem).find('input').val();

                        var data = {
                            Id: itemId,
                            Quantidade: novaQtde
                        };

                        $.ajax({
                            url:'/pedido/updatequantidade',
```

```
                type: 'POST',
                contentType: 'application/json',
                data: JSON.stringify(data)
            });

            debugger;
        }
    </script>
```

Executaremos a aplicação para averiguar se tudo opera como o esperado. Entraremos na página de compras e escolheremos um produto qualquer e clicaremos no botão "Adicionar".

 produto qualquer - página com três opções de livros de programação, cada um deles com o botão "Adicionar" posicionado abaixo da capa.

Voltaremos para a página do carrinho e clicaremos sobre o botão "+", como de costume. Seremos redirecionados para o código de `PedidoController` e poderemos investigar o parâmetro que foi recebido pelo método `UpdateQuantidade()`. O resultado foi :

```
id: 7009
Pedido: null
PrecoUnitario: 0
Produto: null
Quantidade: 0
```

A `Quantidade` não foi passada, precisamos descobrir a causa. Iremos até a classe `itemPedido` averiguar o que pode ter acontecido. O que foi passaod para o servidor foi apenas o `Id`, mas essa informação não se encontra na classe, e sim em `BaseModel`.

```
public class ItemPedido: BaseModel
{
    [Required]
    public Pedido Pedido { get; private set; }
    [Required]
    public Produto Produto { get; private set; }
    [Required]
    public int Quantidade {get; private set; }
    [Required]
    public decimal PrecoUnitario { get; private set; }

<****!****>
```

Ao investigarmos a classe `BaseModel`, perceberemos que ela possui o Id anotado com o atributo `DataMember`. Além disso, há também o atributo `DataContract`. Nós só conseguimos passar o `Id` porque tanto a classe `BaseModel` como a propriedade `Id` estão marcadas com atributos que permitem a sua seralização. Como recebemos uma string a partir do corpo da requisição, tal string pôde ser convertida corretamente em objeto. Em `ItemPedido` não conseguimos realizar o mesmo procedimento, precisamos anotar tanto a classe quanto as propriedades `ItemPedido` com os atributos.

```
namespace CasaDoCodigo.Models
{
    [DataContract]
```

```
    public abstract class BaseModel
    {
        [DataMember]
        pubic int Id { get; protected set; }

    }
```

<****!****>

Navegaremos até a classe `ItemPedido` e coemçaremos a anotar com o atributo `DataContract`, depois, para cada propriedade do pedido marcaremos como `DataMember`.

```
[DataContract]
public class ItemPedido: BaseModel
{
    [Required]
    [DataMember]
    public Pedido Pedido { get; private set; }
    [Required]
    [DataMember]
    public Produto Produto { get; private set; }
    [Required]
    [DataMember]
    public int Quantidade {get; private set; }
    [Required]
    [DataMember]
    public decimal PrecoUnitario { get; private set; }
```

<****!****>

Executaremos a aplicação, e na página carrossel escolheremos um novo produto, seremos direcionados para a página de carrinho e clicaremos sobre o botão "+" para aumentar a quantidade de itens. Feito isso, conseguimos ver as informações passadas para `ItemPedido`:

```
Id : 8009
Pedido : null
PrecoUnitario : 0
Produto : null
Quantidade : 1
```

Temos a `Quantidade` com valor `1`, obtido por meio do código JavaScript. Continuaremos nosso projeto e veremos como essas informações serão utilizadas na aplicação.