01

# Criando uma ViewModel

### Transcrição

Estamos visualizando a view de `Carrinho.cshtml` que exibe corretamente as informações e tratando dos eventos de click do botão de "+" e de "-", assim como da digitação de quantidade diretamente na caixa de texto, que será gravada no banco de dados.

Nesta view fazemos o cálculo do total geral de itens do carrinho utilizando uma expressão em C#, A `@(Model.Sum(i => i.Quantidade * i.PrecoUnitario))` , que utiliza a quantidade do item e multiplica por seu preço unitário, ao final é realizada a soma desses valores, dessa forma é obtida o total no carrinho. Essa é uma regra de negócio, e essa informação não deveria estar contida dentro da view, que por definição só abriga regras de apresentação.

```
</div>
    <div class="panel-footer">
        <div class="row">
            <div class="col-md-10">
                <span numero-itens>
                    Total: @(Model.Count())
                    itens
                </span>
            </div>
            <div class="col-md-2">
                Total: R$ <span class="pull-right" total>
                    @(Model.Sum(i => i.Quantidade * i.PrecoUnitario))
                </span>
            </div>
        </div>
    </div>
</div>
```

O ideal seria mover essa regra de negócio para um local mais adequeado, para isso criaremos uma classe nova que fornecerá os dados para a view. No começo da nossa view teremos a declaração do modelo com diretiva `@Model` , que recebe uma lista `ItemPedido` . Trocaremos essa informação por uma nova classe que fornecerá um modelo novo e específico para esta view, isto é, uma *view model*.

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

<h3>Meu Carrinho</h3>
```

Criaremos a *view model* dentro da pasta `Models` do nosso projeto. Criaremos um novo diretório chamado `ViewModels` e dentro dele adicionaremos numa nova classe cujo nome será `CarrinhoViewModel.cs` .

A vantagem de uma view model é que ela não precisa ser associada ao Entity Framework, pois diferentes das classes do modelo, não será gravada no banco de dados como tabela.

```
namespace CasaDoCodigo.Models.ViewModels
{
    public class CarrinhoViewModel
    {
    }
}
```

`CarrinhoViewModel`, portanto, irá fornecer uma série de dados para view, como uma lista de itens de pedido. Vamos declarar essa informação como uma propriedade pública, uma lista de pedidos que chamaremos de `Itens` e será somente de leitura, por isso a declararemos como `get`.

```
namespace CasaDoCodigo.Models.ViewModels
{
    public class CarrinhoViewModel
    {
        public IList<ItemPedido> Itens { get; }
    }
}
```

A próxima etapa é gerar o construtor para que essa classe receba a lista de itens. Selecionaremos a linha `public IList<ItemPedido> Itens { get; }`, acionaremos o atalho "Ctrl + ." e escolheremos a opção "Gerar Construtor".

```
namespace CasaDoCodigo.Models.ViewModels
{
    public class CarrinhoViewModel
    {
        public CarrinhoViewModel(IList<ItemPedido> itens)
        {
            itens = itens;
        }

        public IList<ItemPedido> Itens { get; }
    }
}
```

Forneceremos para a view uma propriedade que calculará o valor total disponível no carrinho, para isso declararemos uma propriedade pública chamada `Total` que retornará um valor que contém casas decimais, isto é, a soma do subtotal de itens. Usaremos o método `Sum()` e como parâmetro passaremos uma expressão lambda.

```
namespace CasaDoCodigo.Models.ViewModels
{
    public class CarrinhoViewModel
    {
        public CarrinhoViewModel(IList<ItemPedido> itens)
        {
            Itens = itens;
        }

        public IList<ItemPedido> Itens { get; }

        public decimal Total => Itens.Sum(i => i.Quantidade * i.PrecoUnitario);
    }
```

```
}
```

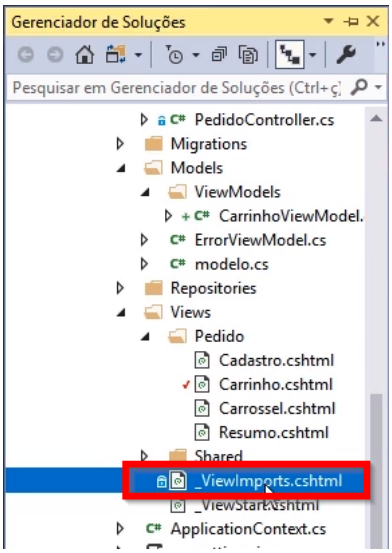
Utilizaremos essa nova classe como modelo da view `Carrinho.cshtml` .

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

<h3>Meu Carrinho</h3>
```

No entanto, veremos uma marcação de erro do Visual Studio em `CarrinhoViewModel` , isso se deve porque colocamos essa classe em outro namespace que não foi reconhecido. Temos duas alternativas: inserir antes do nome da classe seu respectivo namespace, ou incluir uma nova diretiva using no arquivo `_ViewImports.cshtml` . Recorreremos a segunda opção.

Na área "Gerenciador de Soluções" abriremos o arquivo `_ViewImports.cshtml` .




Dentro desse arquivo teremos algumas diretivas using.

```
@using CasaDoCodigo
@using CasaDoCodgio.Models
@addTagHelper *, Microsoft.AspNetCore.Mvc.TagHelpers
```

Adicionaremos uma nova diretiva referente a `ViewModels` .

```
@using CasaDoCodigo
@using CasaDoCodgio.Models
@using CasaDoCodigo.Models.ViewModels
@addTagHelper *, Microsoft.AspNetCore.Mvc.TagHelpers
```

Com isso, não teremos mais uma mensagem de erro em `CarrinhoViewModel` na view `Carrinho.cshtml` .

O próximo passo é modificar algumas informações da view de modo que ela se adeque à `CarrinhoViewModel` , com por exemplo o laço `foreach` que obtém os itens a partir de `Model` , que costumava ser uma lista, mas atualmente trata-se de um objeto que contém uma lista.

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

        @foreach (var item in Model)
        {

            <div class="row row-center linha-produto" item-id="@item.Id">
```

Substituiremos `Model` por `Model.Itens`.

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

        @foreach (var item in Model.Itens)
        {

            <div class="row row-center linha-produto" item-id="@item.Id">
```

Mais abaixo no código da view, teremos a quantidade total de itens `Total`, que era fornecido por `Model.Count`. Substituiremos `Model.Count` por `Model.Itens.Count`.

```
</div>
    <div class="panel-footer">
        <div class="row">
            <div class="col-md-10">
                <span numero-itens>
                    Total: @(Model.Itens.Count())
                    itens
                </span>
            </div>
```

Mais abaixo, teremos o cálculo do valor total que nos referimos no começo da aula.

```
<div class="col-md-2">
                Total: R$ <span class="pull-right" total>
                    @(Model.Sum(i => i.Quantidade * i.PrecoUnitario))
```

Nós removeremos este cálculo e o substituiremos por `Model.Total`.

```
<div class="col-md-2">
                Total: R$ <span class="pull-right" total>
                    @(Model.Total)
```

Precisamos, ainda, fornecer uma instância a nova classe `CarrinhoViewModel` a partir do cotroller, afinal é ele que irá retornar uma view e injetar o objeto que servirá de modelo. Em gerenciador de soluções abriremos o arquivo `PedidoController.cs` e localizaremos a action que retorna `Carrinho`. Percebam que ela está injetando na view o parâmetro `pedidoRepository.GetPedido().Itens`, que é uma lista de pedidos. Podemos substituir este trecho por `CarrinhoViewModel`.

```
 public IActionResult Carrinho(string codigo)
        {
```

```
            if (!string.IsNullOrEmpty(codigo))
            {
                pedidoRepository.AddItem(codigo);
            }

                        return View(pedidoRepository.GetPedido().Itens);
        }
```

Selecionaremos `pedidoRepository.GetPedido().Itens` e extrairemos uma variável local, para que nosso código se torne mais legível. Pressionaremos "Ctrl + ." e escolheremos a opção "Introduzir local para "pedidoRepository.GetPedido().Itens".

```
    public IActionResult Carrinho(string codigo)
        {
            if (!string.IsNullOrEmpty(codigo))
            {
                pedidoRepository.AddItem(codigo);
            }

                        List<ItemPedido> itens = pedidoRepository.GetPedido().Itens;
                        return base.View(itens);
        }
```

Com a variável criada que capta os itens, montaremos uma instância de `CarrinhoViewModel` , que receberá como parâmetro `itens` .

```
    public IActionResult Carrinho(string codigo)
        {
            if (!string.IsNullOrEmpty(codigo))
            {
                pedidoRepository.AddItem(codigo);
            }

                        List<ItemPedido> itens = pedidoRepository.GetPedido().Itens;

                        CarrinhoViewModel carrinhoViewModel = new CarrinhoViewModel(itens);
                        return base.View(carrinhoViewModel);
        }
```

Executaremos nossa aplicação pressionando o atalho "F5". Seremos direcionados para a página principal e selecionaremos um produto para adicionarmos o carrinho. Já na página de carrinho aumentaremos o número de itens e veremos que o valor total é modificado cada vez que um intem é incluído.


Conseguimos aplicar o padrão novo que fornece informações para a view com a vantagem de não precisar ser gravado no banco de dados como tabela.