
Programação Web

Aulas Teóricas – Capítulo 3 – 3.2

1º Semestre - 2022/2023

Departamento de Engenharia Informática e de Sistemas
Instituto Superior de Engenharia de Coimbra/Instituto Politécnico de Coimbra



**Estes Slides foram reformulados para o presente Ano Lectivo.
No entanto para a elaboração dos mesmos foram incluídos partes de
conteúdos utilizados nesta Unidade Curricular em Anos Lectivos
anteriores os quais tiveram contribuições de diversos docentes,
nomeadamente dos docentes Jorge Barbosa, Cristiana Areias,
Francisco Leite!
A matéria e alguns exemplos seguem as orientações existentes nos
livros indicados na Bibliografia!**

Programação Web

Passagem de Dados, Views Razor Layouts

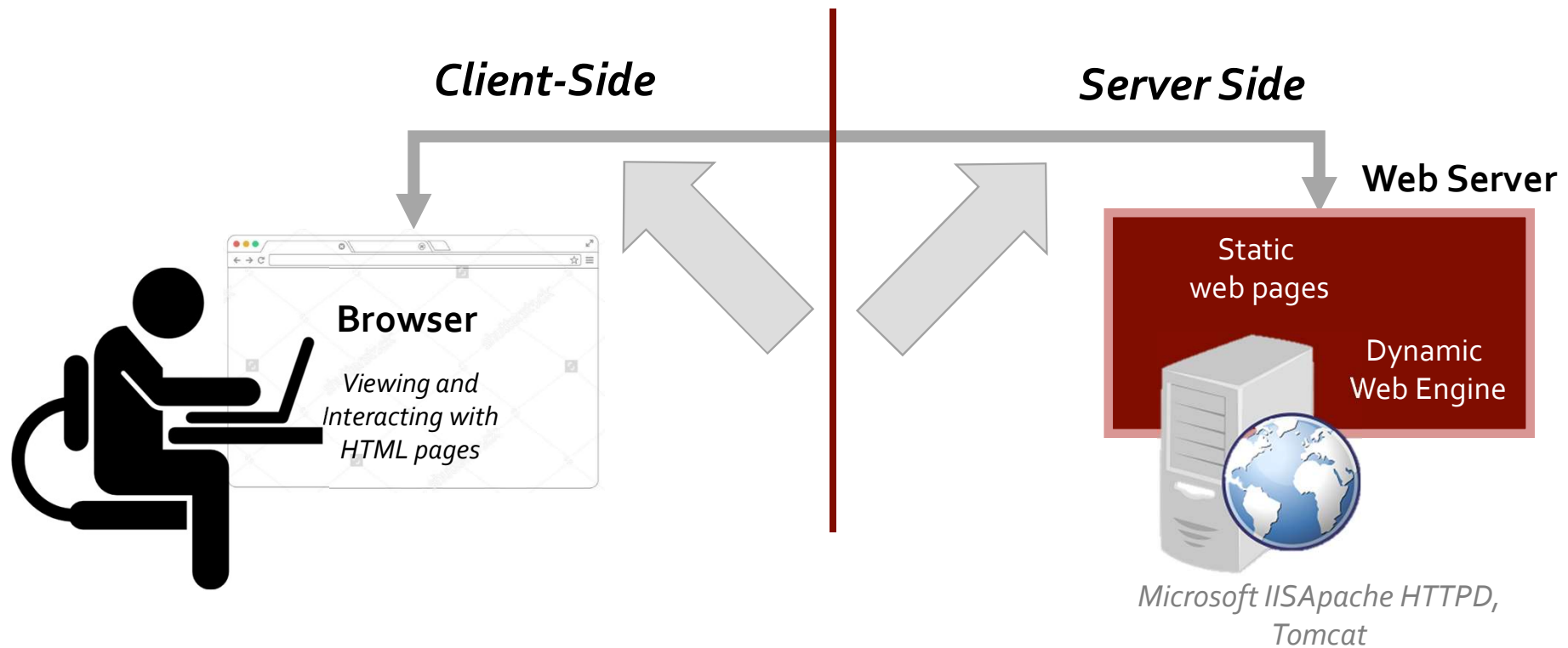
*Departamento de Engenharia Informática e de Sistemas
Instituto Superior de Engenharia de Coimbra/Instituto Politécnico de Coimbra*



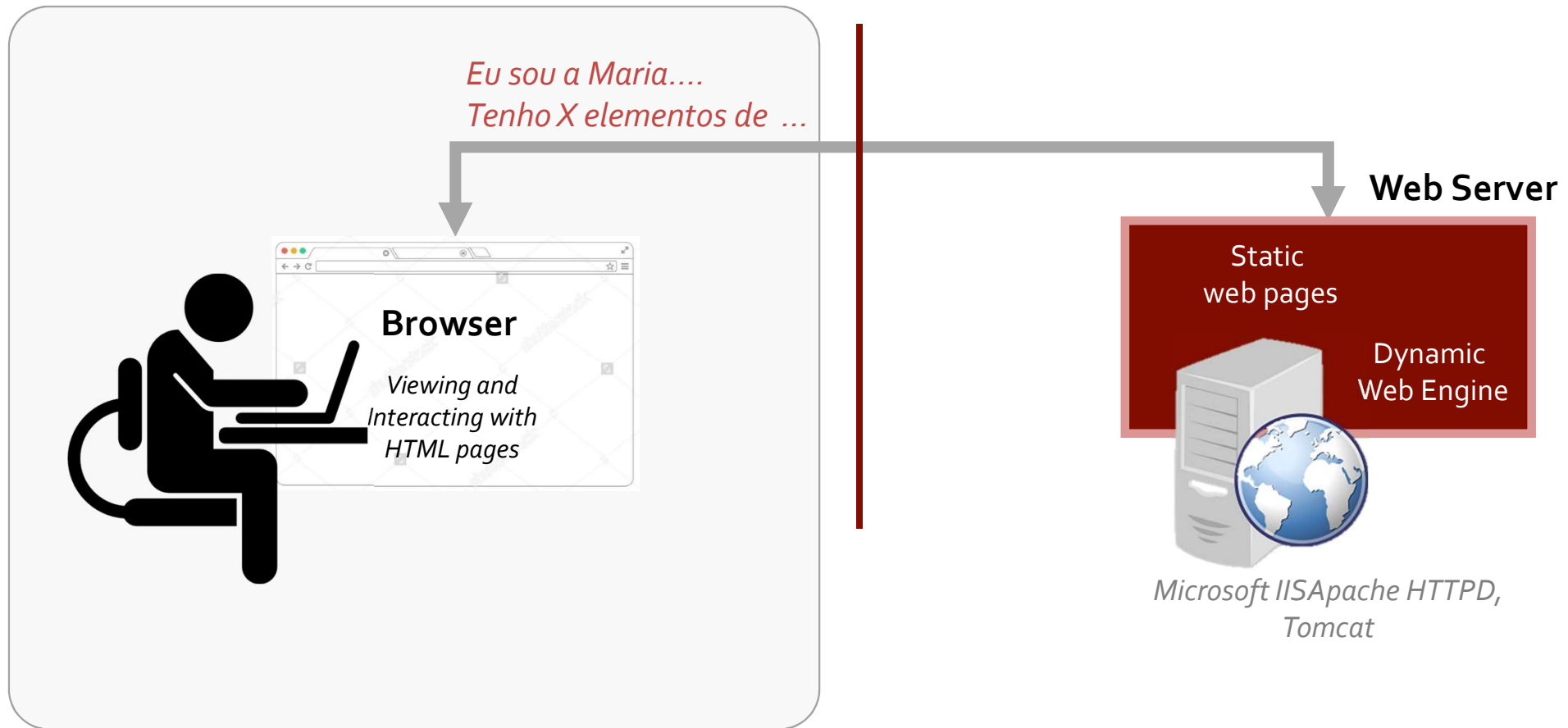
Gestão de Estados

Introdução

Gestão de Estados



Gestão de Estados



Gestão de Estados



Passagem de Dados

Controlador – Vista - Controlador

Passagem de Dados

- A passagem de dados do **Controlador** à **Vista**, e ao próximo *request*, pode ser efetuada de outras formas para além da passagem de dados através de um argumento da *View*.

```
public ActionResult Index()
{
    var alunoTeste = new Aluno() { Nome = "Aluno de Teste" };
    return View(alunoTeste);
}
```

Argumento na View

Passagem de Dados

Passagem de dados para views

- Dados fortemente tipados: viewmodel
- Dados fracamente tipados
 - ***ViewData*** (ViewDataAttribute)
 - ***ViewBag***

Passagem de Dados

Para Dados fortemente tipados (***viewmodel***)

- A abordagem mais robusta é especificar um tipo de modelo na view. Este modelo é comumente referido como um ***viewmodel***. É passada uma instância do tipo ***viewmodel*** para a ***view*** da ***action***
- Usar um ***viewmodel*** para passar dados para uma ***view*** permite que a ***view*** tire proveito da forte verificação de tipo. A tipagem forte (ou fortemente tipada) significa que cada variável e constante tem um tipo explicitamente definido (exemplo, *string*, *int*, *DateTime*, etc)
 - A validade dos tipos usados numa *view* é verificada em tempo de compilação.

Passagem de Dados

Para Dados fracamente tipados: ViewData e ViewBag

- Além dos viewmodels fortemente tipadas, as views têm acesso a uma coleção de dados fracamente tipadas
- Ao contrário dos tipos fortes, os tipos fracos (ou tipos soltos) significam que não se declara explicitamente o tipo de dados que irão ser passados
- Pode-se usar esta abordagem para passar pequenas quantidades de dados para dentro e para fora de controladores e de vistas.

Passagem de dados

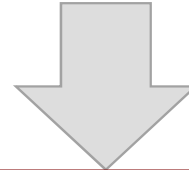
- A ter em conta que:

- **ViewData**

- **ViewBag**

Utilizados para comunicar entre o controlador e a vista correspondente, mas apenas para chamadas ao servidor.

*Torna-se **null** se ocorrer redireccionamento.*



Mecanismo para **manter estado**
entre o controlador e a vista
correspondente



Passagem de dados – Outras formas

- Estas coleção podem ser referenciada por meio das propriedades ***ViewData*** ou ***ViewBag*** em controladores e visualizações
- A propriedade ***ViewData*** é um dicionário de objetos fracamente digitados
- A propriedade ***ViewBag*** é um invólucro em torno de ***ViewData*** que fornece propriedades dinâmicas para a coleção ***ViewData*** subjacente
- Nota: As pesquisas de chave não diferenciam maiúsculas de minúsculas para ViewData e ViewBag.

Passagem de dados – Outras formas

- ***ViewData*** e ***ViewBag*** são resolvidos dinamicamente em tempo de execução.
- Uma vez que eles não oferecem verificação de tipo em tempo de compilação, ambos são geralmente mais sujeitos a erros do que usar um modelo de visualização.
 - Por esse motivo não é consensual a utilização de ***ViewData*** e ***ViewBag***

Passagem de dados – Outras formas

- ***Em Resumo:***
- ***ViewData*** - *Objeto do tipo dicionário*
 - Acessível usando *strings* como chaves
 - Necessita do *type casting* para tipos complexos
- ***ViewBag*** – *Objecto do tipo dinâmico*
 - Não necessita de *type casting* nem verificações de *null*
 - *Propriedades adicionadas em tempo de execução*

Passagem de dados – Outras formas

- ***TempData***

- O ASP.NET Core expõe o ***Razor Pages TempData*** ou ***Controller TempData***. Esta propriedade armazena dados até que sejam lidos noutra solicitação.
 - Os métodos ***Keep (String)*** e ***Peek (string)*** podem ser usados para examinar os dados sem exclusão no final da solicitação. Manter marcas em todos os itens do dicionário para retenção.
- Assim, ***TempData*** é:
 - Útil para redirecionamento quando os dados são necessários para mais de uma única solicitação.
 - Implementado por provedores ***TempData*** usando *cookies* ou controlo do estado de sessão.

Passagem de dados – Outras formas

- ***Em Resumo:***
- ***TempData*** - Objeto do tipo dicionário
 - Permanece durante o tempo de um pedido HTTP
 - *Pode ser usado para manter dados entre redireccionamentos*

Passagem de dados: ViewData

Controller

```
public ActionResult Vencedor()
{
    var pescador = new Pescador() { Nome = "Nuno Manuel Pereira" };
    ViewData["PescadorData"] = pescador;
    return View();
}
```

View

```
@using MVC1.Models
@model MVC1.Models.Pescador
@{
    ViewBag.Title = "Vencedor";
    Layout = "~/Views/Shared/_Layout.cshtml";
}

<h2>@(((Pescador)ViewData["PescadorData"]).Nome)</h2>
```

Passagem de dados: ViewBag

Controller

```
public ActionResult Vencedor()
{
    var pescador = new Pescador() { Nome = "Nuno Manuel Pereira" };
    ViewBag.PescadorProp = pescador;
    return View();
}
```

View

```
@using MVC1.Models
@model MVC1.Models.Pescador
@{
    ViewBag.Title = "Vencedor";
    Layout = "~/Views/Shared/_Layout.cshtml";
}

<h2>@ViewBag.PescadorProp.Nome</h2>
```

Exemplo do uso do TempData

```
public class CreateModel : PageModel
{
    private readonly RazorPagesContactsContext _context;

    public CreateModel(RazorPagesContactsContext context)
    {
        _context = context;
    }

    public IActionResult OnGet()
    {
        return Page();
    }
}
```

[TempData]

```
public string Message { get; set; }
```

[BindProperty]

```
public Customer Customer { get; set; }
```

...

Exemplo do uso do TempData

...

```
public async Task<IActionResult> OnPostAsync()
{
    if (!ModelState.IsValid)
    {
        return Page();
    }

    _context.Customer.Add(Customer);
    await _context.SaveChangesAsync();
    Message = $"Customer {Customer.Name} added";

    return RedirectToPage("./IndexPeek");
}
```

Exemplo do uso do TempData

@page

@model IndexModel

<h1>Peek Contacts</h1>

```
@{  
    if (TempData.Peek("Message") != null)  
    {  
        <h3>Message: @TempData.Peek("Message")</h3>  
    }  
}
```

@*Content removed for brevity.*@

Neste exemplo, no final da solicitação, **TempData ["Message"]** não é excluído porque **Peek** é usado. Se a página for atualizada é exibido o conteúdo de TempData ["Mensagem"].

Exemplo do uso do TempData

```
@page
@model IndexModel

<h1>Contacts Keep</h1>

@{
    if (TempData["Message"] != null)
    {
        <h3>Message: @TempData["Message"]</h3>
    }
    TempData.Keep("Message");
}

@*Content removed for brevity.*@
```

Este exemplo é similar ao anterior mas usa **Keep** para preservar os dados no final da solicitação:



Views

Views

- A view não deve executar qualquer lógica de negócio, ou interagir com a base de dados directamente.
- Deve trabalhar somente com os dados que lhe são fornecidos pelo controlador.
- A "*separation of concerns*" ajuda a manter um código limpo, testável e de mais facil manutenção.

Views

- Simples
- Usam *ViewData*, *ViewModel*, *ViewBag*
- Bibliotecas JQuery incluídas
- Possibilidade de *Partial Views*
- Html helper class
- Tag Helpers class

Exemplo de View em ASP.NET MVC

View

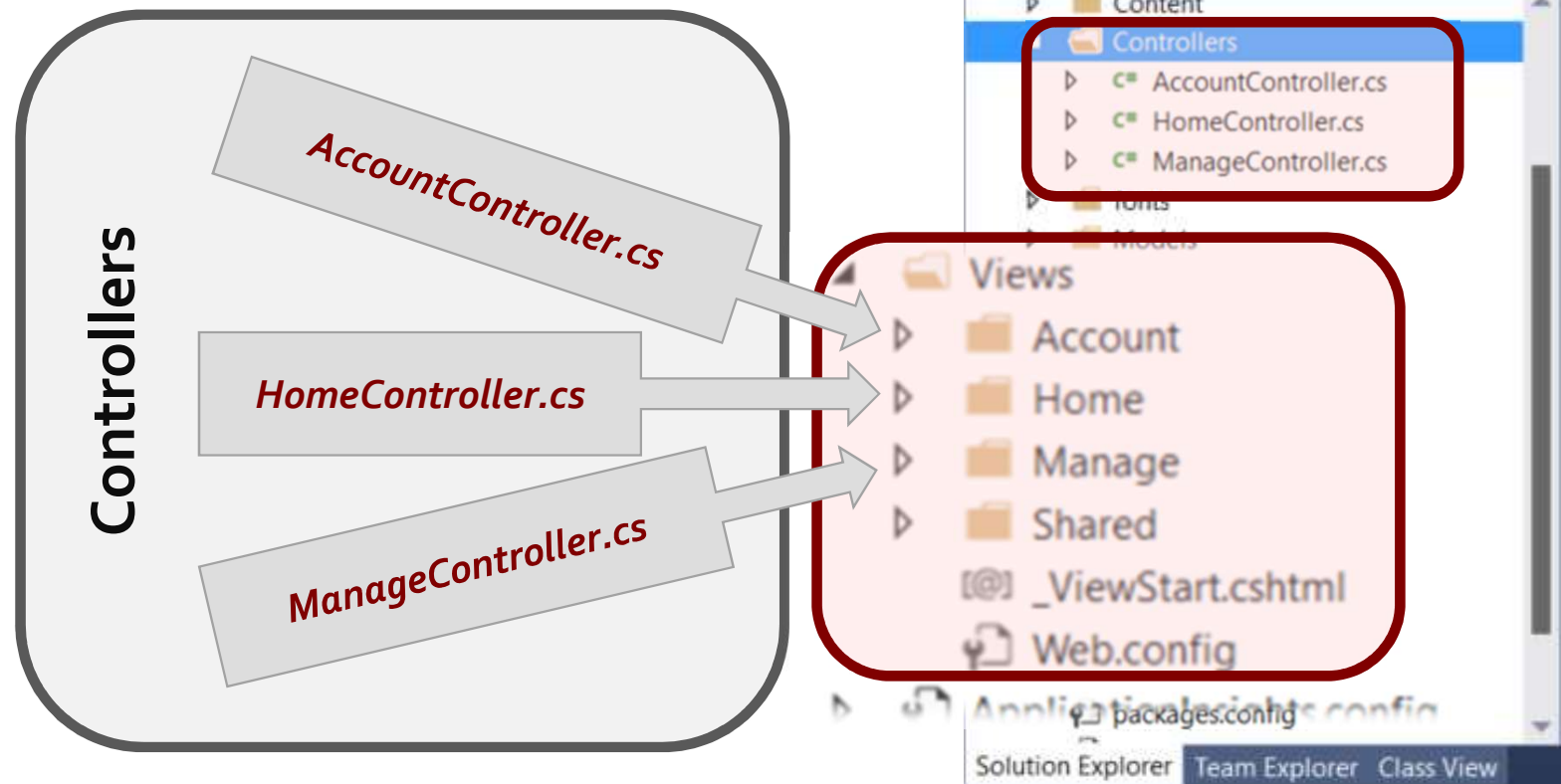
```
@{
    ViewBag.Title = "Home Page";
}
<div class="jumbotron">
    <h1>ASP.NET</h1>
    <p class="lead">ASP.NET is a free web framework for building great Web
sites and Web applications using HTML, CSS and JavaScript.</p>
    <p><a href="http://asp.net" class="btn btn-primary btn-lg">Learn more
&raquo;</a></p>
</div>

<div class="row">
    <div class="col-md-4">
        <h2>Getting started</h2>
        <p>ASP.NET MVC gives you a powerful, patterns-based way to build
dynamic websites that enables a clean separation of concerns and gives you full
control over markupfor enjoyable, agile development.
        </p> ...
    </div>
</div>
```

Views - Associação com as *Actions*

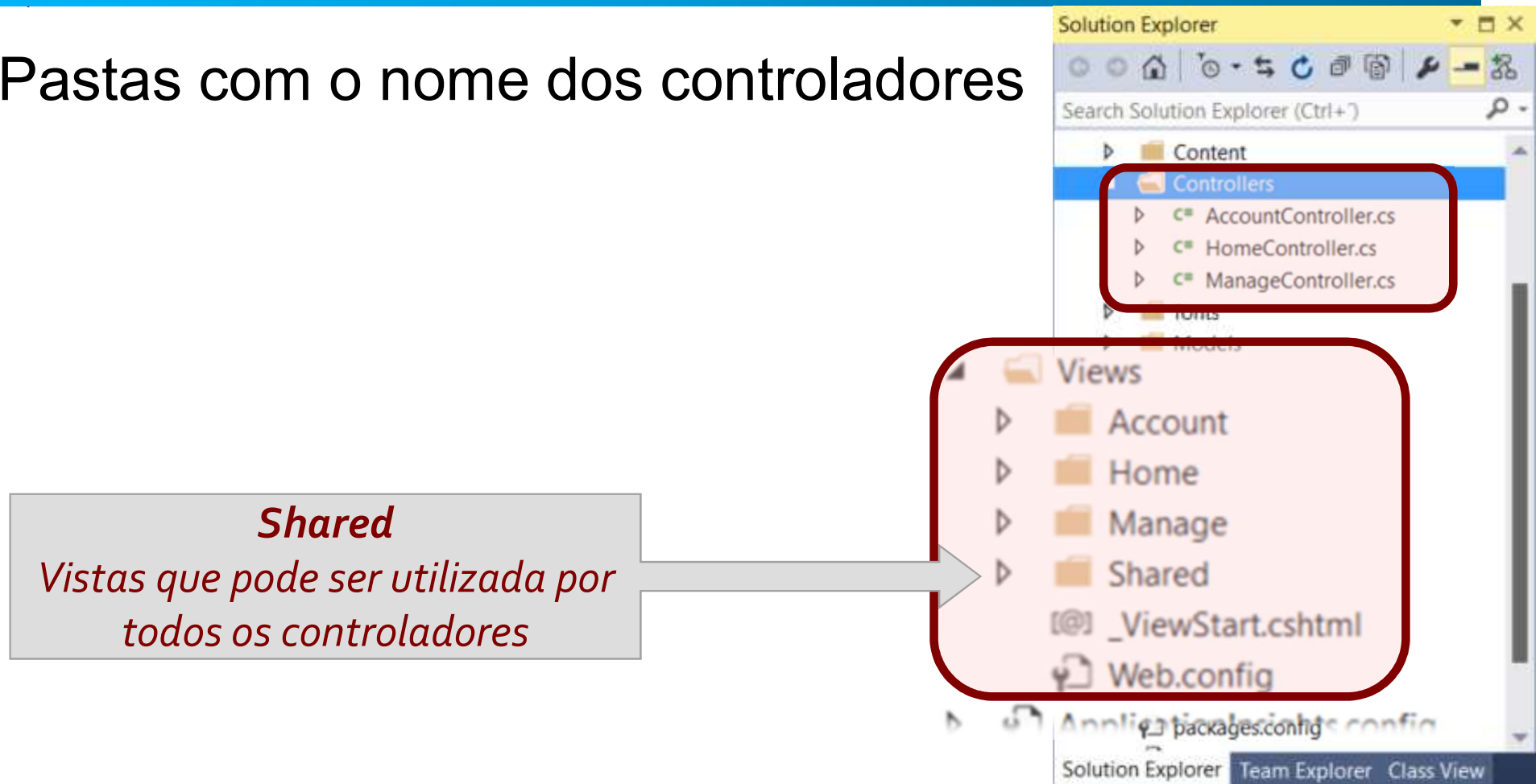
- Pastas com o nome dos controladores

Convenções!



Views - Associação com as *Actions*

- Pastas com o nome dos controladores



Qual a *View* apresentada?

```
public ActionResult Index()  
{  
    return View();  
}
```

/pescadores/Index

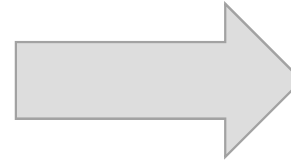


/pescadores/OutraView

```
public ActionResult Index()  
{  
    return View("OutraView");  
}
```

Qual a *View* apresentada?

```
public ActionResult Index()  
{  
    return View();  
}
```



Renderiza a vista
Index.cshtml

```
public ActionResult Index()  
{  
    return View("OutraView");  
}
```



Renderiza a vista
OutraView.cshtml

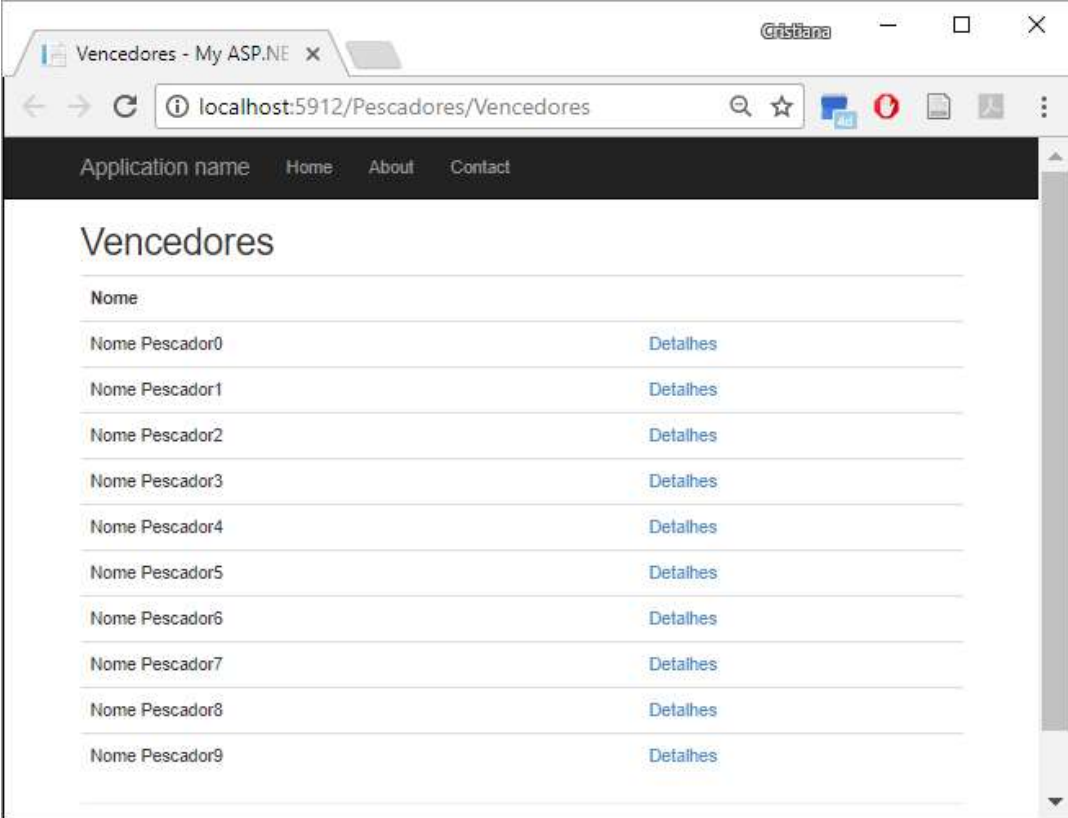
Views

```
public ActionResult Index()  
{  
    return View("~/Views/Exemplos/Index.cshtml ");  
}
```

<http://domínio/pescadores/Index>

View Vencedores

- Lista de Vencedores



Nome	
Nome Pescador0	Detalhes
Nome Pescador1	Detalhes
Nome Pescador2	Detalhes
Nome Pescador3	Detalhes
Nome Pescador4	Detalhes
Nome Pescador5	Detalhes
Nome Pescador6	Detalhes
Nome Pescador7	Detalhes
Nome Pescador8	Detalhes
Nome Pescador9	Detalhes

Views – *Strongly Typed*

```
public class Pescador
{
    public int Id { get; set; }
    public string Nome { get; set; }
}
```

Model
Pescador

```
public ActionResult Vencedores()
{
    var pescadores = new List<Pescador>();
    for (int i=0; i<10;i++)
        pescadores.Add(new Pescador() { Nome = "Nome Pescador"+i });
    return View(pescadores);
}
```

View Vencedores – Especificar modelo

```
@model IEnumerable<Aula2TMVC.Models.Pescador>
@{
    ViewBag.Title = "Vencedores";
}
<h2>Vencedores</h2>

<table class="table">
    <tr>
        <th >@Html.DisplayNameFor(model => model.Nome)</th>
        <th></th>
    </tr>
    @foreach (var item in Model) {
        <tr>
            ...
```

View Vencedores – Especificar modelo

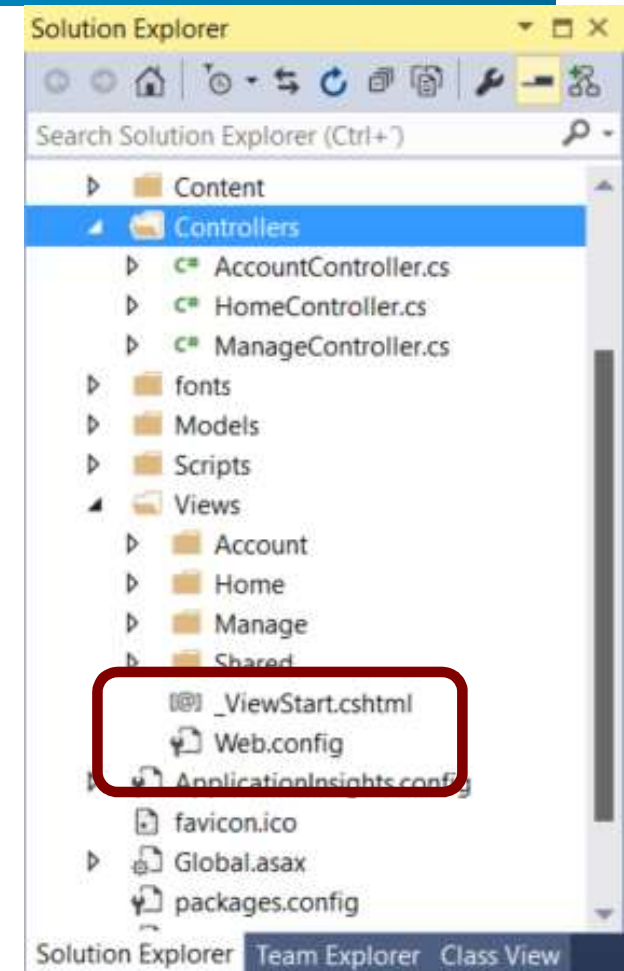
```
@model IEnumerable<Aula2TMVC.Models.Pescador>
```

```
@using Aula2TMVC.Models;  
@model IEnumerable<Pescador>
```

Web.config

```
<system.web.webPages.razor>
  <host factoryType="System.Web.Mvc.MvcWebRazorHo
  <pages pageBaseType="System.Web.Mvc.WebViewPage
    <namespaces>
      <add namespace="System.Web.Mvc" />
      <add namespace="System.Web.Mvc.Ajax" />
      <add namespace="System.Web.Mvc.Html" />
      <add namespace="System.Web.Optimization"/>
      <add namespace="System.Web.Routing" />
      <add namespace="Aula2TMVC" />
      <add namespace="Aula2TMVC.Models" />
    </namespaces>
  </pages>
</system.web.webPages.razor>
```

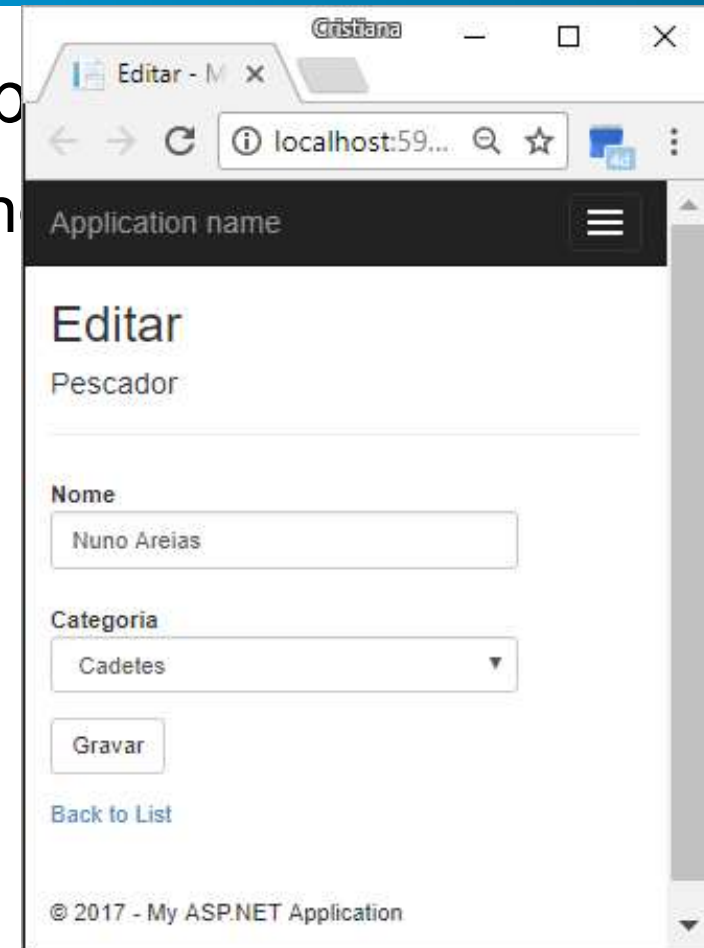
```
@model IEnumerable<Pescador>
```



E se se pretender que uma vista apresente dados que não são mapeados diretamente do modelo Pescador?

Exemplo

- Consideremos o seguinte exemplo
 - Editar a categoria do pescador, onde há uma lista de categorias possíveis



The screenshot shows a web browser window with the address bar displaying 'localhost:59...'. The page title is 'Editar - M x'. The main content area is titled 'Editar Pescador' and contains a form with the following elements:

- A text input field labeled 'Nome' containing the value 'Nuno Areias'.
- A dropdown menu labeled 'Categoria' with 'Cadetes' selected.
- A 'Gravar' (Save) button.
- A 'Back to List' link.
- A footer indicating '© 2017 - My ASP.NET Application'.

Exemplo

- Será que é possível recorrer à **ViewBag** ?
 - Vantagens?



gens?

ViewBag - ListBox

```
public ActionResult Editar(int? id)
{
    Pescador pescador = ObtemPescadorId(id); // Metodo que obtem um determinado pescador
    List<SelectListItem> tipos = new List<SelectListItem>() {
        new SelectListItem { Text = "Cadetes", Value = "1" },
        new SelectListItem { Text = "Infantis", Value = "2" },
        new SelectListItem { Text = "Juvenis", Value = "3" },
        new SelectListItem { Text = "Juniors", Value = "4" },
        new SelectListItem { Text = "Seniores", Value = "5" } };
    if (pescador != null)
        ViewBag.Tipo = tipos;
    else
        return HttpNotFound();

    return View(pescador);
}
```

DropDownListFor

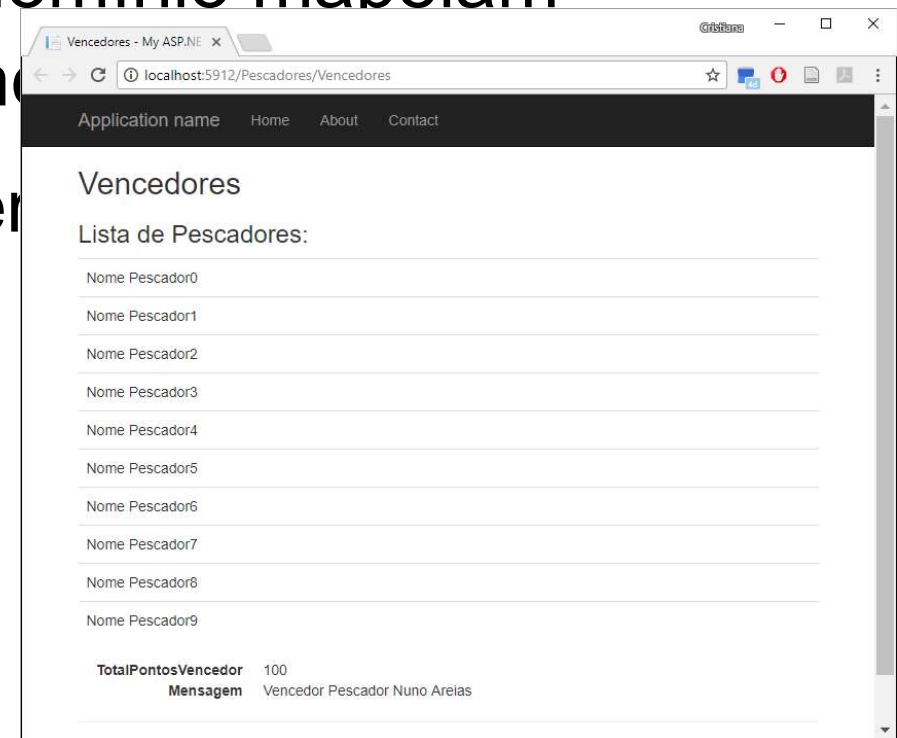
```
...  
</div>  
<div class="form-group">  
    @Html.LabelFor(x => x.Categoria, htmlAttributes: new { @class = "control-label col-md-2" })  
  
    <div class="col-md-10">  
        @Html.DropDownListFor(x => x.Categoria,  
                                new  
SelectList(ViewBag.Tipo, "Value", "Text"),  
                                new { @class = "form-control" })  
    </div>  
</div>  
  
<div class="form-group">  
...
```



Criação de View Model

View Models

- Nem todos os modelos de domínio mapeiam diretamente o que se pretende
- Modelo que permite fornecer



Action Vencedores

```
public class PescadoresVencedoresViewModel
{
    public IEnumerable<Pescador> Pescadores { get; set; }
    public int TotalPontosVencedor { get; set; }
    public string Mensagem { get; set; }
}

public ActionResult Vencedores()
{
    var pescadoresVencedoresViewModel = new PescadoresVencedoresViewModel();

    var pescadores = new List<Pescador>();
    for (int i = 0; i < 10; i++)
        pescadores.Add(new Pescador() { Nome = "Nome Pescador" + i });

    pescadoresVencedoresViewModel.Pescadores = pescadores as IEnumerable<Pescador>;
    pescadoresVencedoresViewModel.TotalPontosVencedor = 100;
    pescadoresVencedoresViewModel.Mensagem = "Vencedor Pescador Nuno Areias";
    return View(pescadoresVencedoresViewModel);
}
```

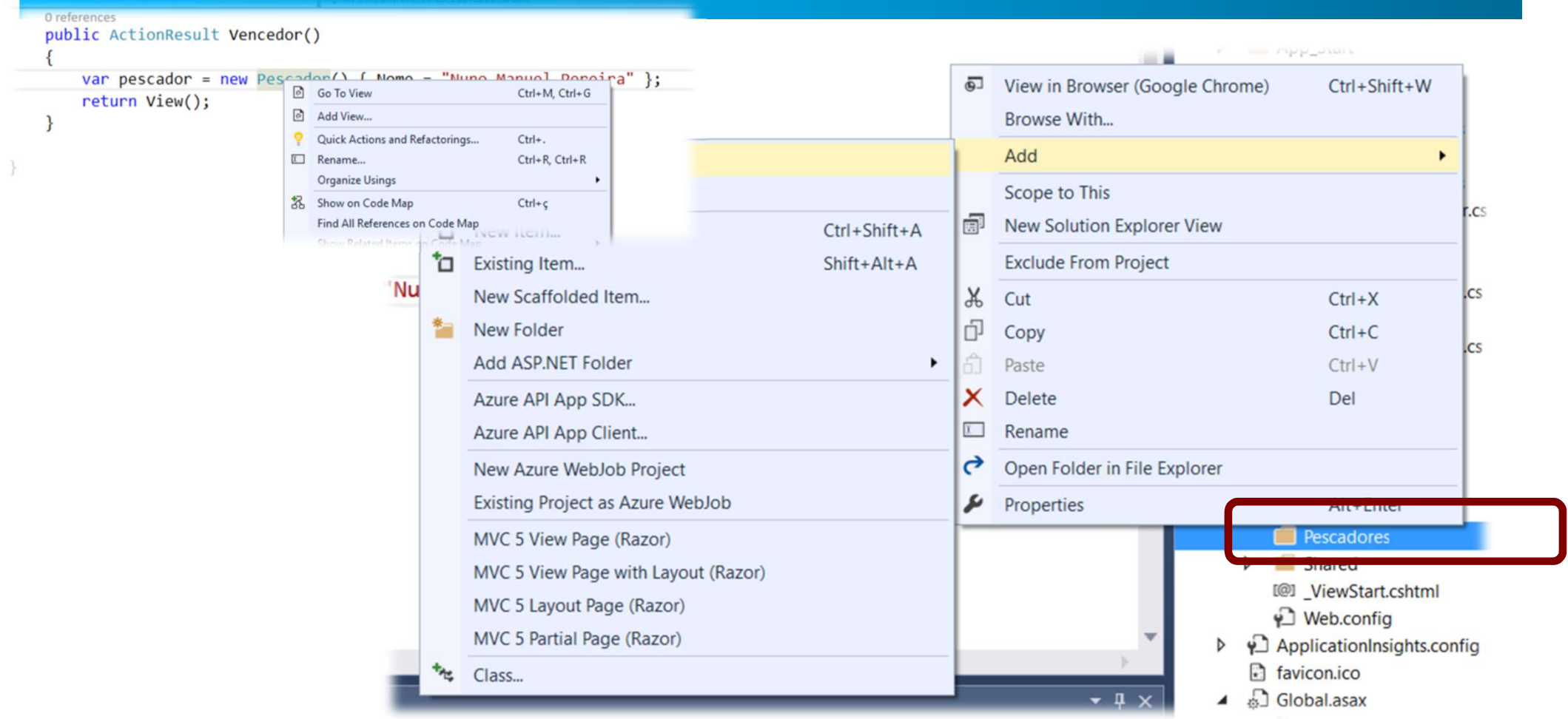
View Vencedores

```
@model Aula2TMVC.Models.PescadoresVencedoresViewModel
@{
    ViewBag.Title = "Vencedores";
}
<h2>Vencedores</h2>
<h3>Lista de Pescadores:</h3>
<table class="table">
@foreach (var pescador in Model.Pescadores) {
    <tr> <td>@pescador.Nome</td> </tr>
}
</table>
<dl class="dl-horizontal">
    <dt> @Html.DisplayNameFor(model => model.TotalPontosVencedor) </dt>
    <dd>@Html.DisplayFor(model => model.TotalPontosVencedor)</dd>
    <dt>@Html.DisplayNameFor(model => model.Mensagem)</dt>
    <dd>@Html.DisplayFor(model => model.Mensagem)</dd>
</dl>
```

Criação de Views

Templates

Criação de View



Criação de uma *View Template*

The image displays two screenshots of the 'Add View' dialog box in Visual Studio, illustrating the steps to create a view template.

Left Screenshot: The 'View name' is 'Vencedores'. The 'Template' dropdown menu is open, showing options: 'Create', 'Delete', 'Details', 'Edit', 'Empty', 'Empty (without model)', and 'List'. 'Empty (without model)' is selected. The 'Model class' is empty. Under 'Options', 'Create as a partial view' is unchecked, 'Reference script libraries' is checked, and 'Use a layout page' is checked.

Right Screenshot: The 'View name' is 'Vencedores'. The 'Template' is 'Details'. The 'Model class' is 'PescadoresVencedoresViewModel (Aula2TMVC.Models)'. Under 'Options', 'Create as a partial view' is unchecked, 'Reference script libraries' is checked, and 'Use a layout page' is checked. A text box for the layout page is empty.

Criação da *View* - Vencedores

Add View

View name: Vencedores

Template: Details

Model class: PescadoresVencedoresViewModel (Aula2TMVC.Models)

Options:

- ☐ Create as a partial view
- ☒ Reference script libraries
- ☒ Use a layout page:

(Leave empty if it is set in a Razor _viewstart file)

Add Cancel

Criação da *View* - Vencedores

Add View

View name: Vencedores

Template: Details

Model class: PescadoresVencedoresViewModel (Aula2TMVC.Models)

Options:

- ☐ Create as a partial view
- ☒ Reference script libraries
- ☒ Use a layout page:

(Leave empty if it is set in a Razor _viewstart file)

Add Cancel

Criação da *View* - Vencedores

Add View

View name: Vencedores

Template: Details

Model class: PescadoresVencedoresViewModel (Aula2TMVC.Models)

Options:

- ☐ Create as a partial view
- ☒ Reference script libraries
- ☒ Use a layout page:

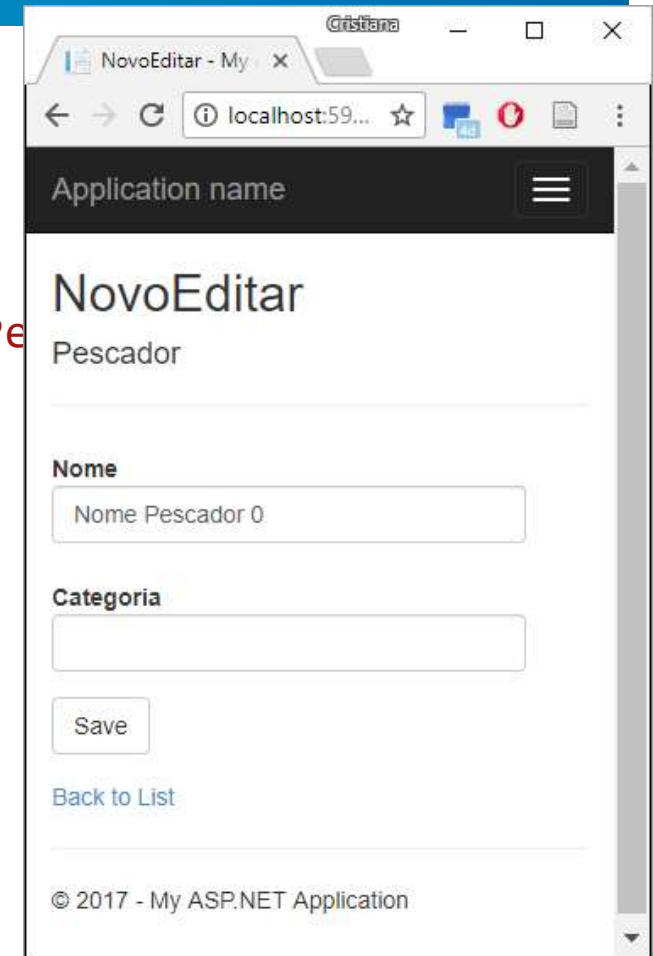
(Leave empty if it is set in a Razor _views)

Buttons: Add, Cancel

Não necessário caso se opte por recorrer ao default leayout pois este esta já especificado no ficheiro _ViewStart.cshtml

Action NovoEditor

```
public ActionResult NovoEditor(int id)
{
    var pescadores = new List<Pescador>();
    for (int i = 0; i < 10; i++)
        pescadores.Add(new Pescador() { Nome = "Nome Pe
    return View(pescadores[0]);
}
```



View NovoEditor - Scaffolding

```
@model Aula2TMVC.Models.Pescador

@{
    ViewBag.Title = "NovoEditor";
}
<h2>NovoEditor</h2>

@using (Html.BeginForm())
{
    @Html.AntiForgeryToken()

    <div class="form-horizontal">
        <h4>Pescador</h4>
        <hr />
        @Html.ValidationSummary(true, "", new { @class = "text-danger" })
        @Html.HiddenFor(model => model.Id)

        <div class="form-group">
            @Html.LabelFor(model => model.Nome, htmlAttributes: new { @class = "control-label col-md-2" })
            <div class="col-md-10">
                @Html.EditorFor(model => model.Nome, new { htmlAttributes = new { @class = "form-control" } })
                @Html.ValidationMessageFor(model => model.Nome, "", new { @class = "text-danger" })
            </div>
        </div>
    </div>
}
```

View NovoEditor - Scaffolding

...

```
<div class="form-group">
    @Html.LabelFor(model => model.Categoria, htmlAttributes: new { @class = "control-label col-md-2" })
    <div class="col-md-10">
        @Html.EditorFor(model => model.Categoria, new { htmlAttributes = new { @class = "form-control" } })
    >
    @Html.ValidationMessageFor(model => model.Categoria, "", new { @class = "text-danger" })
</div>
</div>

<div class="form-group">
    <div class="col-md-offset-2 col-md-10">
        <input type="submit" value="Save" class="btn btn-default" />
    </div>
</div>
</div>
}
<div>
    @Html.ActionLink("Back to List", "Index")
</div>
@section Scripts {
    @Scripts.Render("~/bundles/jqueryval")
}
```




Razor View Engine

Razor

- Introduzido com o ASP.NET MVC 3
- Sintaxe simplificada
- Minimiza a quantidade de sintaxe e caracteres extra
- Codifica automaticamente qualquer *output* enviado através do @ de forma a prevenir ataques *cross site scripting*
- Razor suporta expressões e blocos de código
 - Razor View Engine

Razor – Carater @

Informa o RazorView Engine que os caracteres seguintes são código, não HTML

@{

```
Layout = "~/Views/Shared/_Layout.cshtml";
```

}

Caracter @

```
@{ var mensagem = "Programação Web"; }
```

```
<p>O valor da Mensagem é: @mensagem</p>
```

```
@{
```

```
    var boasVindas= "Bem vindo ao Razor!";
```

```
    var diaSemana= DateTime.Now.DayOfWeek;
```

```
    var mensagem = boasVindas + " Hoje é " + diaSemana;
```

```
}
```

Razor Syntax

Algum erro?

```
@{  
    var nome = "Cristiana";  
    var Apelido = "Areias";  
    <div>  
        Nome: @nome  
        Apelido: @apelido  
    </div>  
}
```



Razor Syntax

```
@{  
    var nome = "Cristiana";  
    var apelido = "Areias";  
    <div>  
        Nome: @nome <br />  
        Apelido: @apelido  
    </div>  
}
```



Razor Syntaxe

@{**Algum erro?**

```
var txt = "";
```

```
if (DateTime.Now.Hour > 12)
```

```
    txt = "Boa tarde!";
```

```
else
```

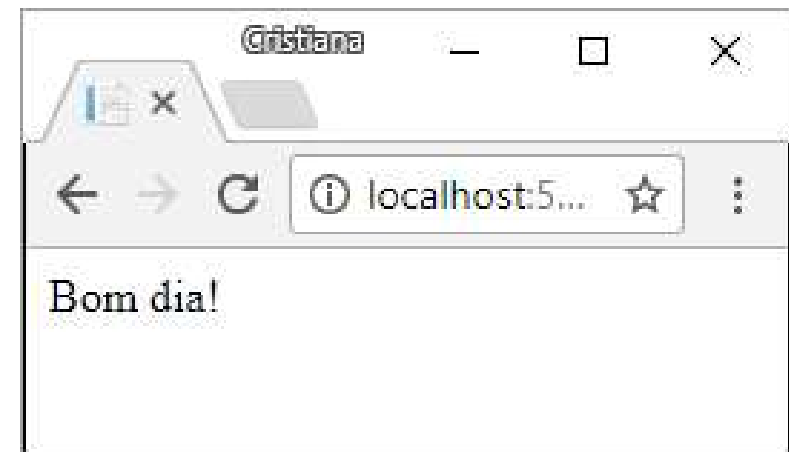
```
    txt = "Bom dia!";
```

```
}
```



Razor Syntaxe

```
@{  
    var txt = "";  
    if (DateTime.Now.Hour > 12)  
    {  
        txt = "Boa tarde!";  
    }  
    else  
    {  
        txt = "Bom dia!";  
    }  
    @txt  
}
```



Razor – Algumas Regras

- Blocos de código estão delimitados com `@{ ... }`
- Expressões *Inline* (variáveis e funções) iniciam com `@`
- Declarações de código terminam com `;`
- Variáveis são declaradas com a palavra chave *var*
- Strings devem estar entre aspas
- Código C# é *case sensitive*
- Ficheiros Razor tem extensão `.cshtml`

Razor: Exemplo

```
@{
    Layout = null;

    var totalMensagem = "";
    var num1="";
    var num2="";
    if (IsPost)
    {
        {   num1 = Request["text1"];
            num2 = Request["text2"];
            var total = num1.AsInt() + num2.AsInt();
            totalMensagem = "Total = " + total;
        }
    }
}

<!DOCTYPE html>

<html> ...
```

Razor: Exemplo

```
... <!DOCTYPE html>
```

```
<html>
```

```
<body style="background-color: lightgray; font
```

```
<form action="" method="post">
```

```
<p><label for="text1">1ºNumero:</label><br>
```

```
<input type="text" name="text1" value=
```

```
</p>
```

```
<p><label for="text2">2º Number:</label><b
```

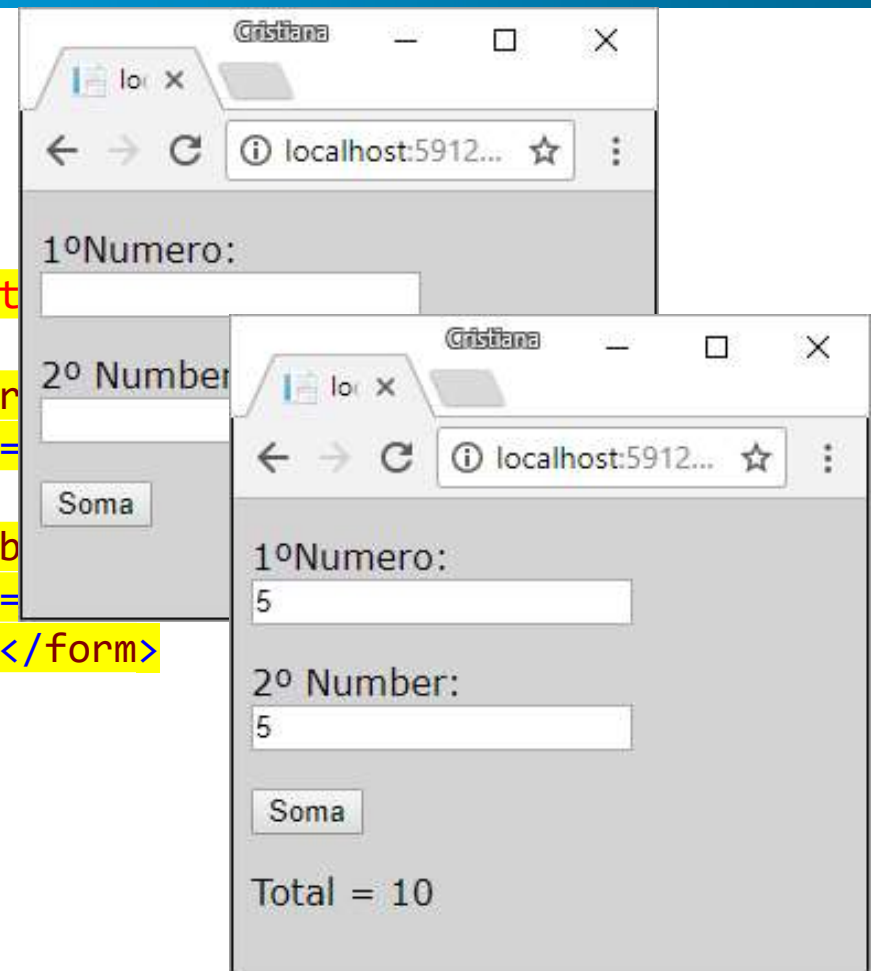
```
<input type="text" name="text2" value=
```

```
<p><input type="submit" value="Soma"></p></form>
```

```
<p>@totalMensagem</p>
```

```
</body>
```

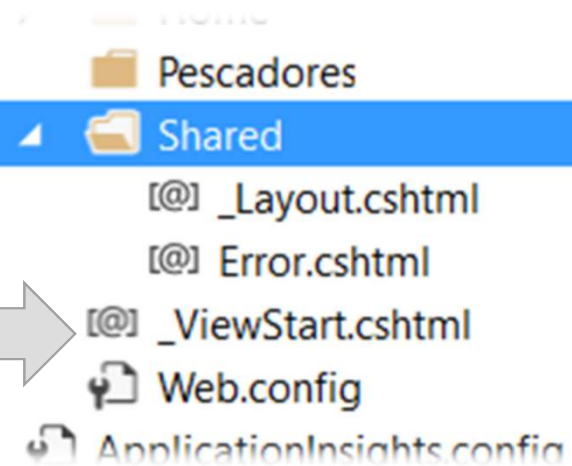
```
</html>
```



Razor

Informa o RazorView Engine que os caracteres seguintes são código, não HTML

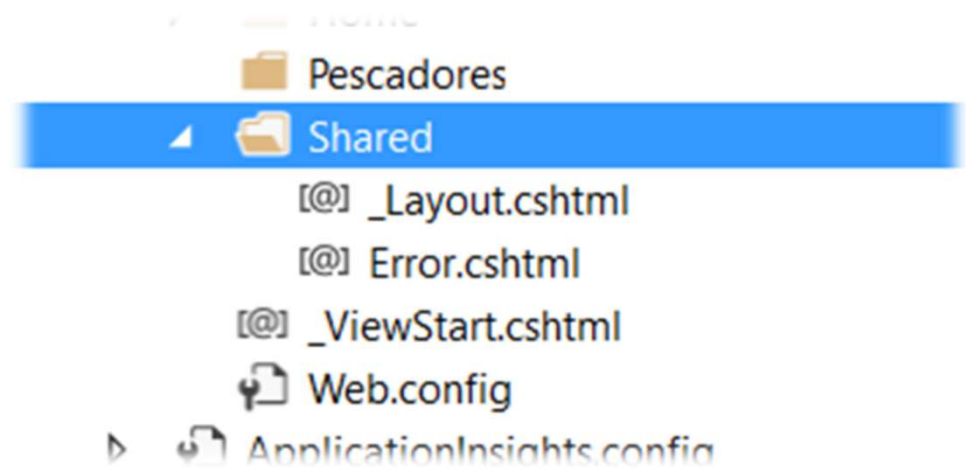
```
@{  
    Layout = "~/Views/Shared/_Layout.cshtml";  
}
```



Layouts

Layout com Razor

- `_Layout.cshtml`
 - Define o que se quer apresentar em todas as páginas da aplicação
 - Recorre a métodos herdados para especificar áreas de conteúdo
 - **RenderBody**
 - **RenderSection**



_Layout.cshtml

- A *view _Layout* deve ser especificada na pasta *Shared* existente na secção Views, permitindo assim definir o que se pretende que apareça em qualquer página da aplicação
- Tem o corpo do documento HTML, link de navegação, menus...



RenderBody

- RenderBody permite especificar onde será apresentado o conteúdo individual das *views*
- Obrigatório a sua chamada numa vista layout

_Layout.cshtml

```
<html>
<head>
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>@ViewBag.Title - My ASP.NET Application</title>
  @Styles.Render("~/Content/css")
  @Scripts.Render("~/bundles/modernizr")
</head>
<body>
  <div class="navbar navbar-inverse navbar-fixed-top">
    <div class="container">
      <div class="navbar-header">
        <button type="button" class="navbar-toggle" data-toggle="
          <span class="icon-bar"></span>
          <span class="icon-bar"></span>
          <span class="icon-bar"></span>
```

_Layout.cshtml

```
<ul class="nav navbar-nav">
  <li>@Html.ActionLink("Home", "Index", "Home")</li>
  <li>@Html.ActionLink("About", "About", "Home")</li>
  <li>@Html.ActionLink("Contact", "Contact", "Home")</li>
</ul>
</div>
</div>
</div>
<div class="container body-content">
  @RenderBody()
  <hr />
  <footer>
    <p>&copy; @DateTime.Now.Year - My ASP.NET Application</p>
  </footer>
</div>
```

Aplicação do *layout* nas Views

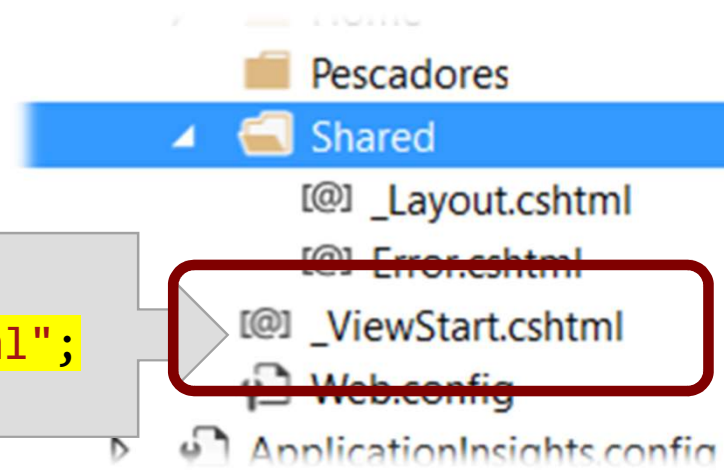
```
@{  
    Layout = "~/Views/Shared/_Layout.cshtml";  
}
```

Exemplo de uma *View*: About

```
@{  
    ViewBag.Title = "About";  
}  
<h2>@ViewBag.Title.</h2>  
<h3>@ViewBag.Message</h3>  
  
<p>Use this area to provide additional information.</p>
```

_ViewStart

```
@{  
    Layout = "~/Views/Shared/_Layout.cshtml";  
}
```

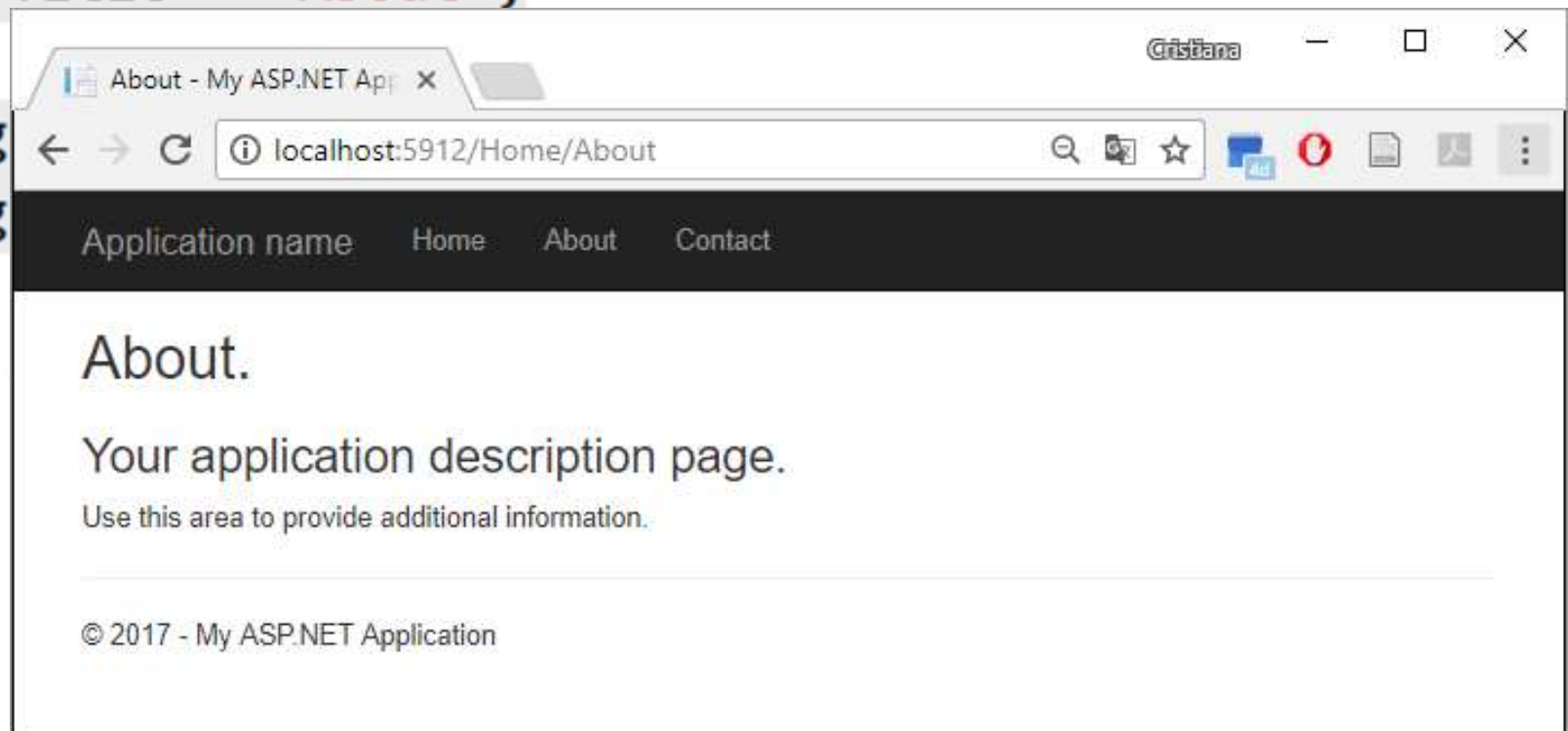


View About

```
@{  
    ViewBag.Title = "About";
```

```
}  
<h2>@ViewBag  
<h3>@ViewBag
```

```
<p>Use this
```



RenderSection

- Chamada opcional numa *Layout View*
- Podem existir uma ou mais *rendersection*
- Fornece uma *content view* como a view Index, permitindo aplicar conteúdo em outras secções da página
- A secção necessita de ter um nome e estar definida no ficheiro *content view*
- É possível que a secção exista apenas em algumas *views*

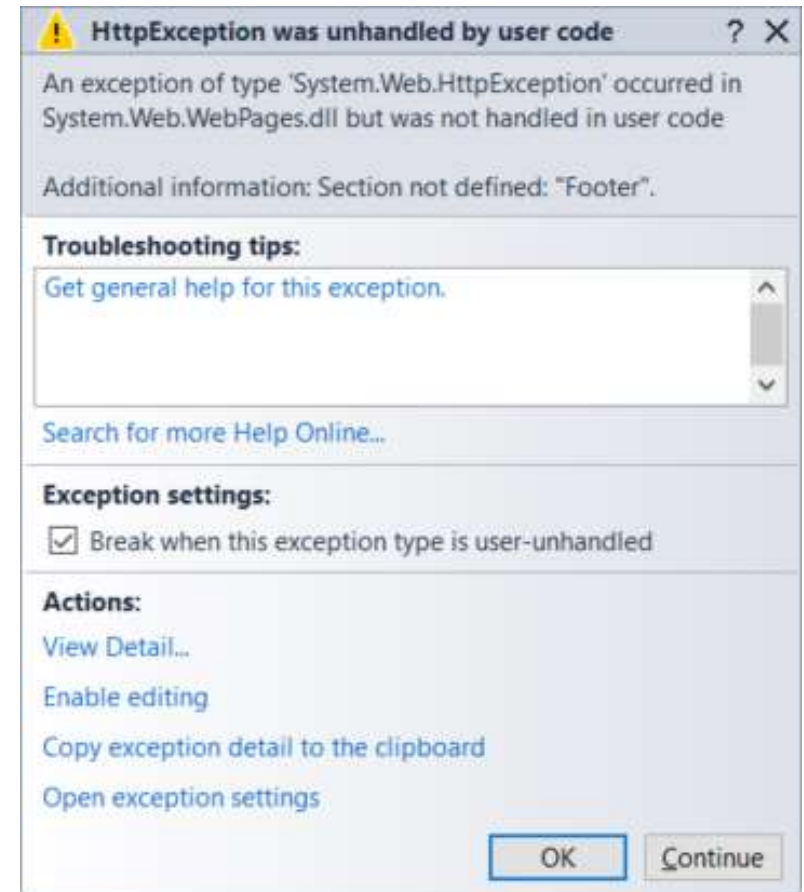
RenderSection

```
        <li>@Html.ActionLink("Home", "Index", "Home")</li>
        <li>@Html.ActionLink("About", "About", "Home")</li>
        <li>@Html.ActionLink("Contact", "Contact", "Home")</li>
    </ul>
</div>
</div>
</div>
<div class="container body-content">
    @RenderBody()
    <hr />
    <footer>
        <p>&copy; @RenderSection("Footer")</p>
    </footer>
</div>

    @Scripts.Render("~/bundles/jquery")
    @Scripts.Render("~/bundles/bootstrap")
    @RenderSection("scripts", required: false)
</body>
```


Não especificação de secção

- Se a *RenderSection* não for especificada como opcional



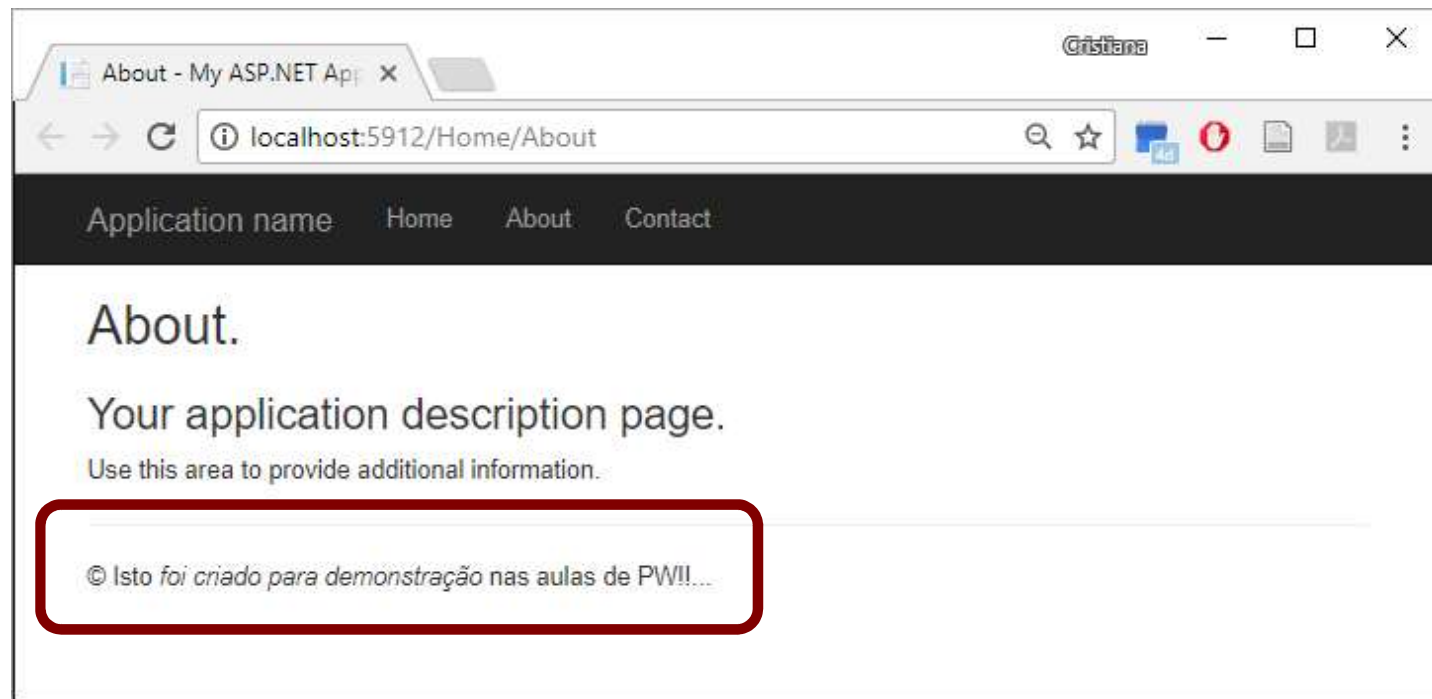
Alteração da View: About

```
@{
    ViewBag.Title = "About";
}
<h2>@ViewBag.Title.</h2>
<h3>@ViewBag.Message</h3>

<p>Use this area to provide additional information.</p>

@section Footer {
    Isto <em>foi criado para demonstração</em> nas aulas de PW!!...
}
```

RenderSection



Partial Views

- Permite especificar código HTML e C# num ficheiro para que depois possa ser reutilizado em várias vistas;
- Pode ser usado simplesmente para simplificar uma vista;
- Por convenção, deve-se especificar no prefixo do nome, o caracter _

Partial Views

- Para *renderizar* uma vista parcial, deve-se recorrer ao *html helper* `Html.Partial`
 - Deve ser passado o nome da vista parcial e o modelo que necessita

```
@foreach (var item in Model)
{
    @Html.Partial("_Rever", item)
}
```

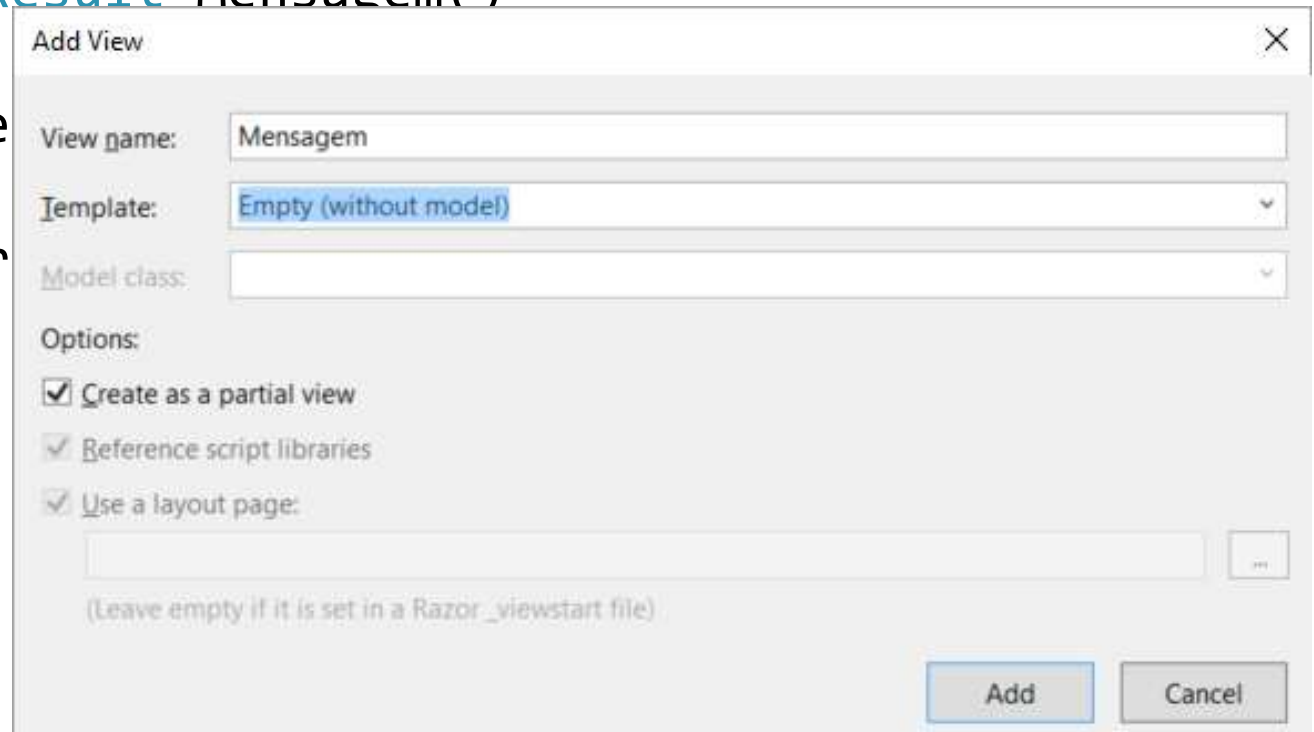
Partial Views

- Normalmente, especifica-se uma vista parcial na pasta onde esta irá ser utilizada.
- Caso seja utilizada por várias vistas, deverá ser colocada na pasta *shared*
- Útil em cenários em que se usa Ajax

Criação de uma *Partial View*

```
public class HomeController : Controller
{
    public ActionResult Mensagem()
    {
        ViewBag.Me

        return Par
    }
}
```

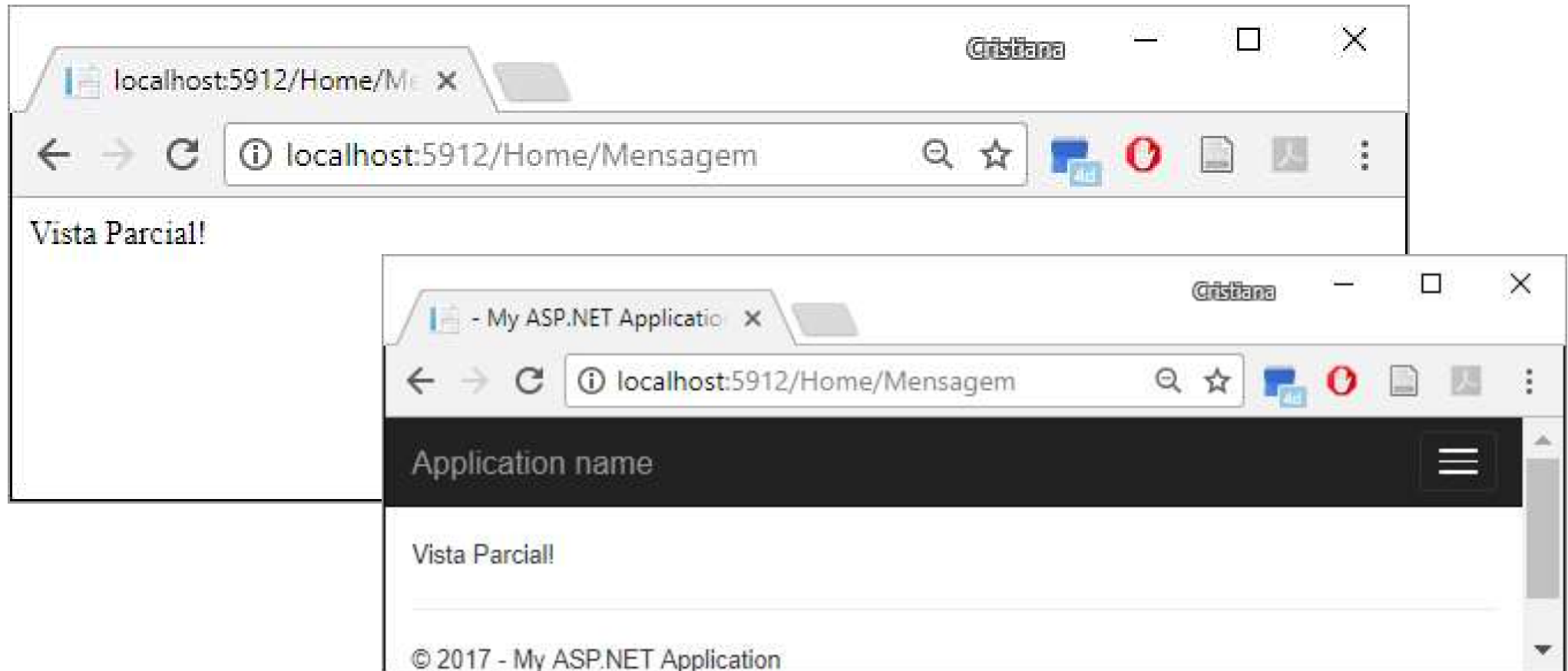


The screenshot shows the 'Add View' dialog box with the following configuration:

- View name: Mensagem
- Template: Empty (without model)
- Model class: (empty)
- Options:
 - ☒ Create as a partial view
 - ☒ Reference script libraries
 - ☒ Use a layout page:

The 'Add' button is highlighted in blue.

Qual o output?



Tag Helpers

- **HTML Helpers e URL Helpers**

- Nas views e a partir do ASP.Net MVC utilizavam-se os **HTML Helpers** e os **URL Helpers** com o objetivo de facilitar a criação de views.
 - Os HTML Helpers podem ser invocados a partir da propriedade **Html** de uma view
 - URL Helpers podem ser invocados via propriedade **Url**. Os *URL Helpers também estão disponíveis nos controladores.*
 - *Também se podem utilizar os Ajax Helpers via propriedade Ajax.*

Tag Helpers

- O processamento por parte do **Razor** da maioria dos **HTML Helpers** origina a renderização de tags **HTML**
 - Por exemplo, a renderização pelo **Razor** do **Html.TextBox** resultaria em:

```
@Html.TextBox("nome")
```

```
<input id="nome" name="nome"  
type="text" value="" />
```

Tag Helpers

- **Tag Helpers** - são um novo recurso disponível no ASP.Net Core
 - Continua a ser possível a utilização de *Html.Helpers*
- Os **Tag Helpers** possibilitam a obtenção dos mesmos resultados escrevendo controles que interagem com Models, Views, etc
 - A sua sintaxe é mais próxima da do *HTML* (com atributos em seus elementos)

Tag Helpers

- A renderização pelo **RAZOR** de uma **TextBox** utilizando o **Tag Helper** `<input ... >` resultaria em:

```
<input type="text" asp-  
for="Nome" />
```

```
<input id="nome" name="nome"  
type="text" value="" />
```

Tag Helpers

■ *Tag Helpers*

- A sintaxe dos *Tag Helpers* é semelhante à sintaxe do *HTML*, elementos e atributos, mas é processado pelo *Razor* no servidor.
- Os *TagHelpers* são uma alternativa aos *Html Helpers*, mas disponibilizam algumas funcionalidades difíceis ou mesmo impossíveis de se obter com esses outros métodos
- Cada *Tag Helper* tem um comportamento diferente e diferentes opções.

Tag Helpers

- ***Tag Helpers***

- A utilização dos ***Tag Helpers*** é disponibilizada quando se cria um projecto no *MS Visual Studio* mas também podem ser adicionados de *assembly/namespace* recorrendo-se à diretiva **@addTagHelper** nos ficheiros **cshtml** das views

@addTagHelper "*", Microsoft.AspNet.Mvc.TagHelpers"

Tag Helpers

- A utilização de **Tag Helpers** em formulários possibilita a criação de código de views mais limpo do que quando se utilizam os *Html Helpers*.
- Esta diferença pode ser percebida no exemplo abaixo onde para a criação de uma textbox se utilizam com o mesmo objectivo e base na propriedade **Nome** de um **Model**, um **Html Helper *Html.EditorFor*** e um **Tag Helper *input***

Tag Helpers

<!--Cria um input com uma classe adicional para a propriedade Nome usando um Html Helper-->

```
@Html.EditorFor(C => C.Nome, new { htmlAttributes = new { @class = "form-control" } })
```

<!--Cria um input com uma classe adicional para a propriedade Nome usando um Tag Helper-->

```
<input asp-for="Nome" class="form-control" />
```

A renderização pelo *RAZOR* de um e de outro obtém-se o mesmo resultado mas é mais fácil utilizar-se **Tag Helper**

Tag Helpers

- Os **Tag Helper** permitem adicionar atributos *HTML* mais facilmente à tag input os quais serão incluídos no *HTML* gerado
 - A inclusão da classe **'form-control'** é natural e de fácil entendimento se comparada com a sintaxe do *Html Helper EditorFor*
 - O facto de o **Intellisense** do VS estar disponível para os *Tag Helpers* também facilita a sua utilização

Tag Helpers

- A geração do HTML para o elemento Form utilizando-se TagHelper é também mais simples e de mais fácil utilização como se pode ver no exemplo
- Basta especificar o *Controller* e a *Action* que tratarão o formulário
- Por default também é gerado automaticamente o *token anti-forgery* mas esta possibilidade pode ser desabilitada

Tag Helpers

```
<form asp-action="Create" asp-controller="Aluno">  
    //Os elementos do formulário  
</form>
```

- A renderização deste **Tag Helper** geraria o seguinte **HTML**:

```
<form action="Create" method="post">  
    //Os elementos do formulário  
    <input name="__RequestVerificationToken" type="hidden"  
value="ABCDEFabcd34456 ....."</n>>  
</form>
```

Tag Helpers

```
<form asp-controller="Create" asp-anti-forgery="false" asp-  
action="Aluno">  
    //Os elementos do formulário  
</form>
```

- O mesmo exemplo mas desabilitando-se a geração do **token anti forgery** definindo no *Tag Helper* a propriedade “**false**”

Tag Helpers

- ***Tag Helpers*** habitualmente utilizados em formulários
 - Input Tag Helper
 - Text Area Tag Helper
 - Validation Tag Helper
 - Label Tag Helper
 - Select Tag Helper
- É possível aos programadores criarem os seus próprios **TagHelpers** e adicioná-los aos já existentes no ASP .NET Core

Tag Helpers

HTML Helpers	Tag Helpers
<pre>@using (Html.BeginForm("Criar")) { @Html.AntiForgeryToken() //conteúdo }</pre>	<pre><form asp-anti-forgery="false" asp-action="Criar"> <!--conteúdo--> </form></pre>
<pre>@Html.LabelFor(model => model.Nome, new { @class = "control-label col-md-2" })</pre>	<pre><label asp-for="Nome" class="control-label col-md- 2" /></pre>
<pre>@Html.EditorFor(model => model.Nome)</pre>	<pre><input type="text" asp-for="Nome" /></pre>
<pre>@Html.ValidationMessageFor(model => model.Nome)</pre>	<pre></pre>
<pre>@Html.DropDownListFor(model=>model.Tags)</pre>	<pre><select asp-for="Tags" asp- items="(IEnumerable<SelectListItem>)Model.Tags" size="20" class="form-control"> <option value="">-- Selecione --</option> </select></pre>

Tag Helpers

HTML Helpers	Tag Helpers
@Html.ValidationSummary(true)	<pre><div asp-validation-summary="All" id="valida_data" class="form-group"> Mensagem </div></pre>
@Html.HiddenFor(model=>model.Id)	<pre><input type="hidden" asp-for="Id" /></pre>