Programação Web

Aulas Teóricas – Capítulo 2 – 2.3

1º Semestre - 2023/2024

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

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

Passagem de Dados

Controlador – Vista - Controlador

Passagem de Dados

- A passagem de dados do ***Controlador*** para a ***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 mais complexos: ViewModel
- Dados simples
  - ***ViewData*** (ViewDataAttribute)
  - ***ViewBag***
  - ***TempData***

Passagem de Dados

Para Dados mais complexos (***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 mais simples: ViewData e ViewBag

- 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

- A referenciação dos dados a passar pode ser feita por meio das propriedades ***ViewData*** ou ***ViewBag*** em controladores e views
- 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
- ***ViewData*** e ***ViewBag*** são resolvidos dinamicamente em tempo de execução.
  - Nota: As pesquisas de chave não diferenciam maiúsculas de minúsculas para 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.

  - 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

```
// GET: Cursos/Create
[Authorize(Roles = "Admin")]
0 referências
public IActionResult Create()
{
    ViewData["ListaDeCategorias"] = new SelectList(_context.Categorias.OrderBy(c => c.Disponivel).ToList(), "Id", "Nome");
    return View();
}
```

View

```
<div class="form-group">
    <label asp-for="CategoriaId" class="control-label fw-bold"></label>
    <select asp-for="CategoriaId"
            asp-items="ViewBag.ListaDeCategorias"
            class="form-control">
    </select>
    <span asp-validation-for="CategoriaId" class="text-danger"></span>
</div>
```

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.

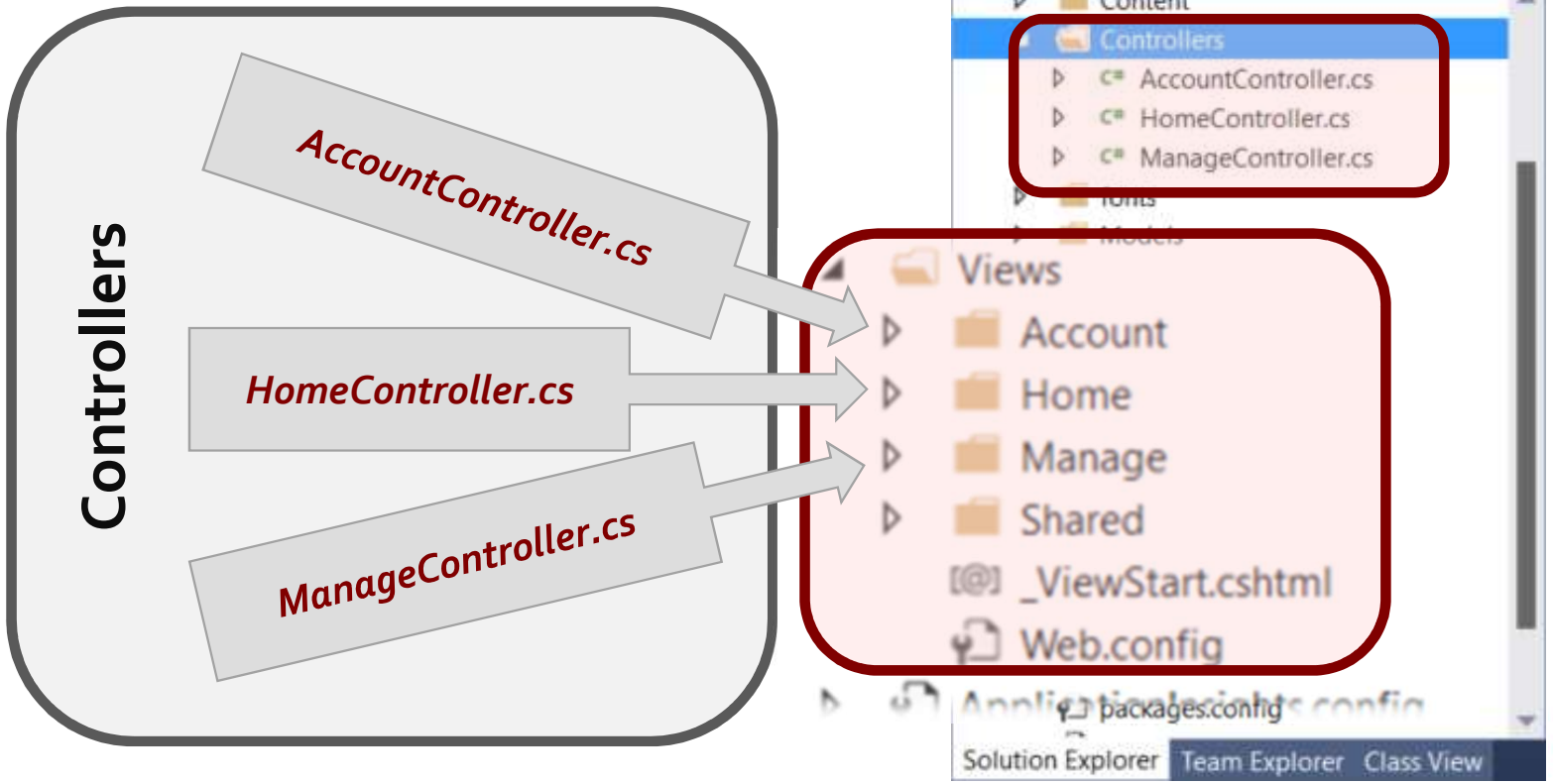
Views

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

Views - Associação com as *Actions*

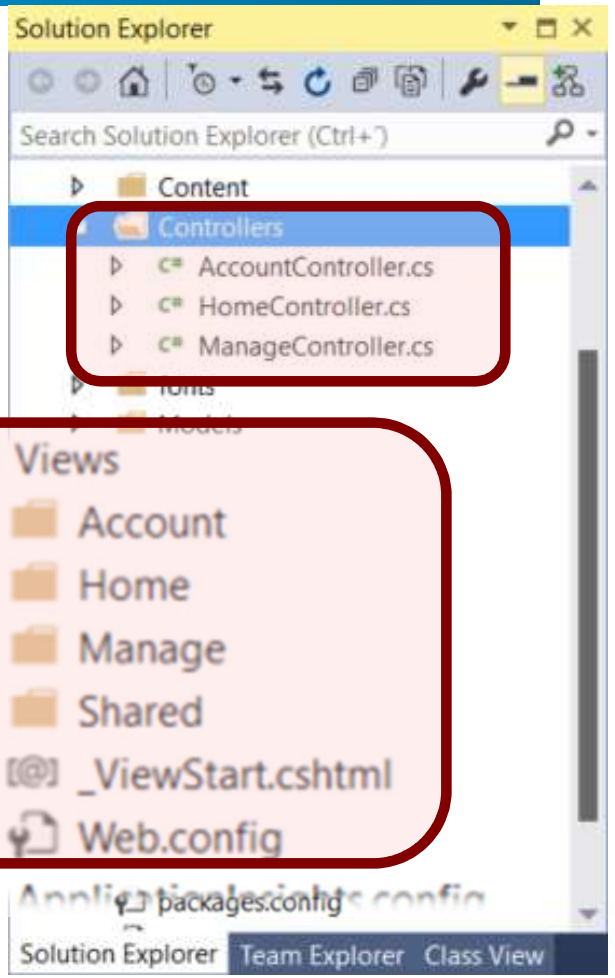
- Pastas com o nome dos controladores

Convenções!



Views - Associação com as *Actions*

- Pastas com o nome dos controladores

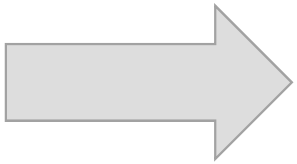


Shared

Vistas que pode ser utilizada por todos os controladores

Qual a *View* apresentada?

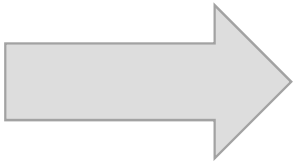
```
public ActionResult Index()
{
    return View();
}
```



Renderiza a vista

Index.cshtml

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



Renderiza a vista

OutraView.cshtml

View Index de Curso – Especificar modelo

```
@model IEnumerable<AutoWEB.Models.Curso>

@{
    ViewData["Title"] = "Cursos";
    Layout = "~/Views/Shared/_Layout2.cshtml";
}

<h1>
    @ViewData["Title"]
    @if (@User.IsInRole("Admin"))
    {
        <small><a class="btn btn-primary btn-sm" asp-action="Create">Adicionar</a></small>
    }
</h1>
```

View Index de UserRolesManager

```
@using AutoWEB.Models
@model IEnumerable<AutoWEB.ViewModels.UserRolesViewModel>
@{
    ViewData["Title"] = "Index";
    Layout = "~/Views/Shared/_Layout2.cshtml";
}
<h1>User Roles</h1>
```

Criação de View Model

View Models

- Nem todos os modelos de domínio mapeiam diretamente o que se pretende apresentar numa *View*
- Um View Model é um modelo que permite fornecer informação à *view sem ser de uma só classe*

Class View Model Agendamento

```
using AutoWEB.Models;
using System.ComponentModel.DataAnnotations;

namespace AutoWEB.ViewModels
{
    public class AgendamentoViewModel
    {
        [Display(Name = "Data de Início", Prompt = "yyyy-mm-dd")]
        [Required(ErrorMessage = "Indique a Data de Início!")]
        8 referências
        public DateTime? DataInicio { get; set; }

        [Display(Name = "Data de Fim", Prompt = "yyyy-mm-dd")]
        [Required(ErrorMessage = "Indique a Data de Fim!")]
        8 referências
        public DateTime? DataFim { get; set; }

        [Display(Name = "Tipo de Aula", Prompt = "Escolha o tipo de aula que pretende")]
        4 referências
        public int TipoDeAulaId { get; set; }
    }
}
```

View Agendamento - Pedido

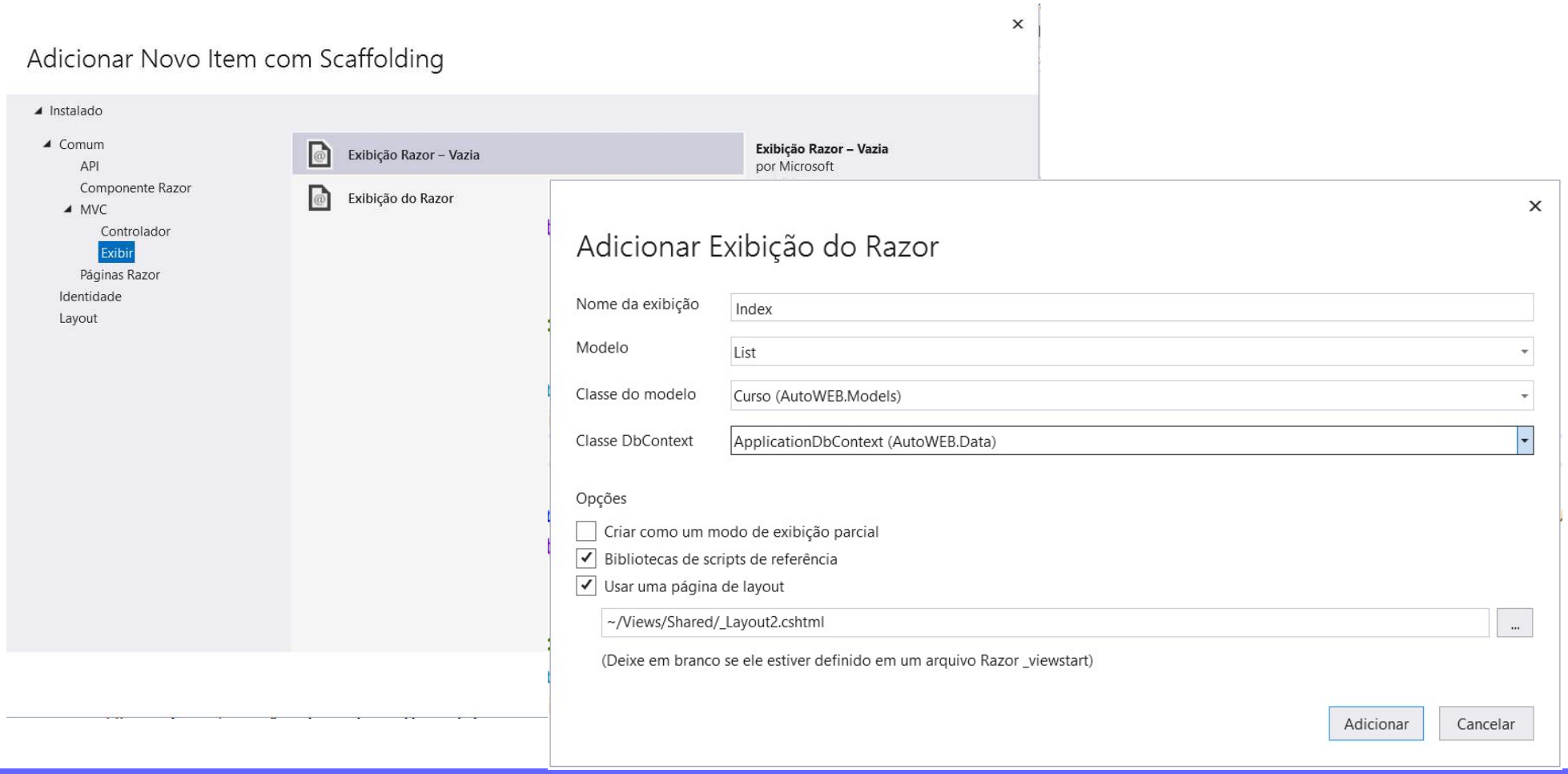
```
@model AutoWEB.ViewModels.AgendamentoViewModel
@{
    Layout = "~/Views/Shared/_Layout2.cshtml";
}
<h1>Agendamento - Pedido</h1>
<hr />
@if(User.Identity.IsAuthenticated){
<div class="row">
    <div class="col-md-4">
        <form asp-action="Calcular">
            <div asp-validation-summary="All" class="text-danger"></div>
            <div class="form-group">
                <label asp-for="TipoDeAulaId" class="control-label fw-bold"></label>
                <select asp-for="TipoDeAulaId" class="form-control" asp-items="ViewBag.TipoDeAulaId"></select>
            </div>

            <div class="form-group col-md-6">
                <label asp-for="DataInicio" class="control-label fw-bold"></label>
                <input asp-for="DataInicio" class="form-control" />
                <span asp-validation-for="DataInicio" class="text-danger"></span>
            </div>
            <div class="form-group col-md-6">
                <label asp-for="DataFim" class="control-label fw-bold"></label>
                <input asp-for="DataFim" class="form-control" />
            </div>
            <div class="col-md-12">
                <span asp-validation-for="DataFim" class="text-danger"></span>
            </div>
           <br/>
            <div class="form-group">
                <input type="submit" value="Calcular" class="btn btn-success" /> |
                    <a class="btn btn-primary" asp-action="OsMeusAgendamentos">Voltar à Listagem</a>
            </div>
            <br />
        </form>
    </div>
</div>
}
else
{
    <p class="alert alert-warning">
        Só utilizadores autenticados é que
        podem efetuar agendamentos. Por favor, autentique-se!
    </p>
}
```

Criação de Views

Templates

Criação de uma *View Template*



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 Razor View Engine que os carateres 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;
 }
```

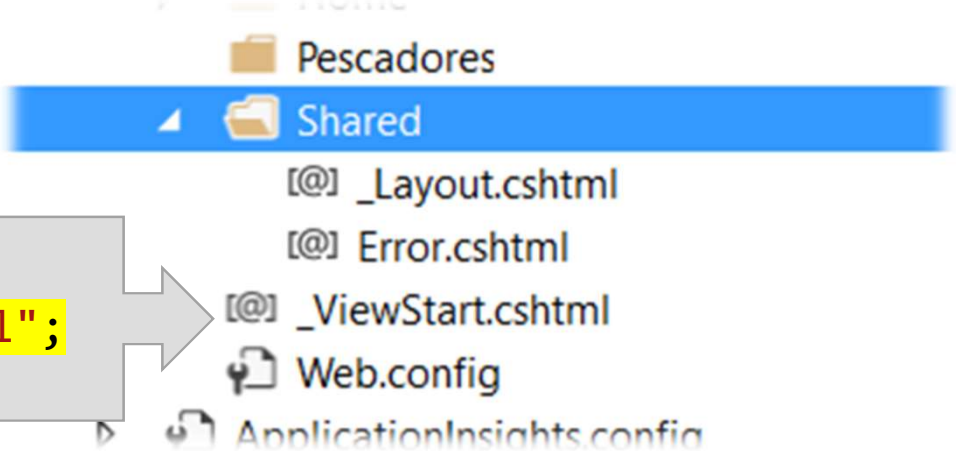
Razor – Algumas Regras

- 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

Informa o Razor View Engine que os carateres seguintes são código, não HTML

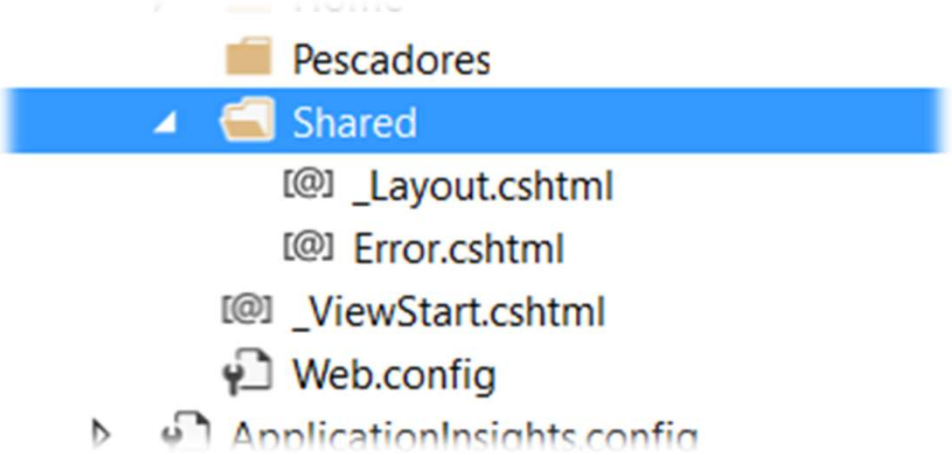
```
@{
    Layout = "~/Views/Shared/_Layout.cshtml";
}
```



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

Aplicação do *layout* nas *Views*

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

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 possivel que a secção exista apenas em algumas *views*

Partial Views

- Permite especificar código Razor e Tag Helpers ou HTML Helpers 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*

Tag Helpers

- **HTML Helpers**
  - Nas views e a partir do ASP.Net MVC utilizavam-se os **HTML Helpers** com o objetivo de facilitar a criação de views.
    - Os HTML Helpers podem ser invocados a partir da propriedade **Html** de uma view

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 percepcionada 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

- 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

- ***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