Curso de C#
Aula 8

Professores
Célio Alencar de Assis
Fábio Duarte Machado

UNIVALI

O que será abordado Hoje

- ASP.Net MVC
  - View
  - Helpers
  - Controller

ASP.NET MVC

Atualmente, o ASP.NET MVC é o framework para desenvolvimento de aplicações web na plataforma
.NET em maior ascensão. A documentação desse framework pode ser obtida em http://www.asp.net/mvc. O ASP.NET MVC é fortemente baseado no padrão *MVC.*

MVC

O MVC (*model-view-controller) é um padrão de arquitetura que tem por objetivo isolar a lógica* de negócio da lógica de apresentação de uma aplicação.

Esse padrão (ou alguma variação) é amplamente adotado nas principais plataformas de desenvolvimento atuais. Em particular, ele é bastante utilizado no desenvolvimento de aplicações web.

MVC

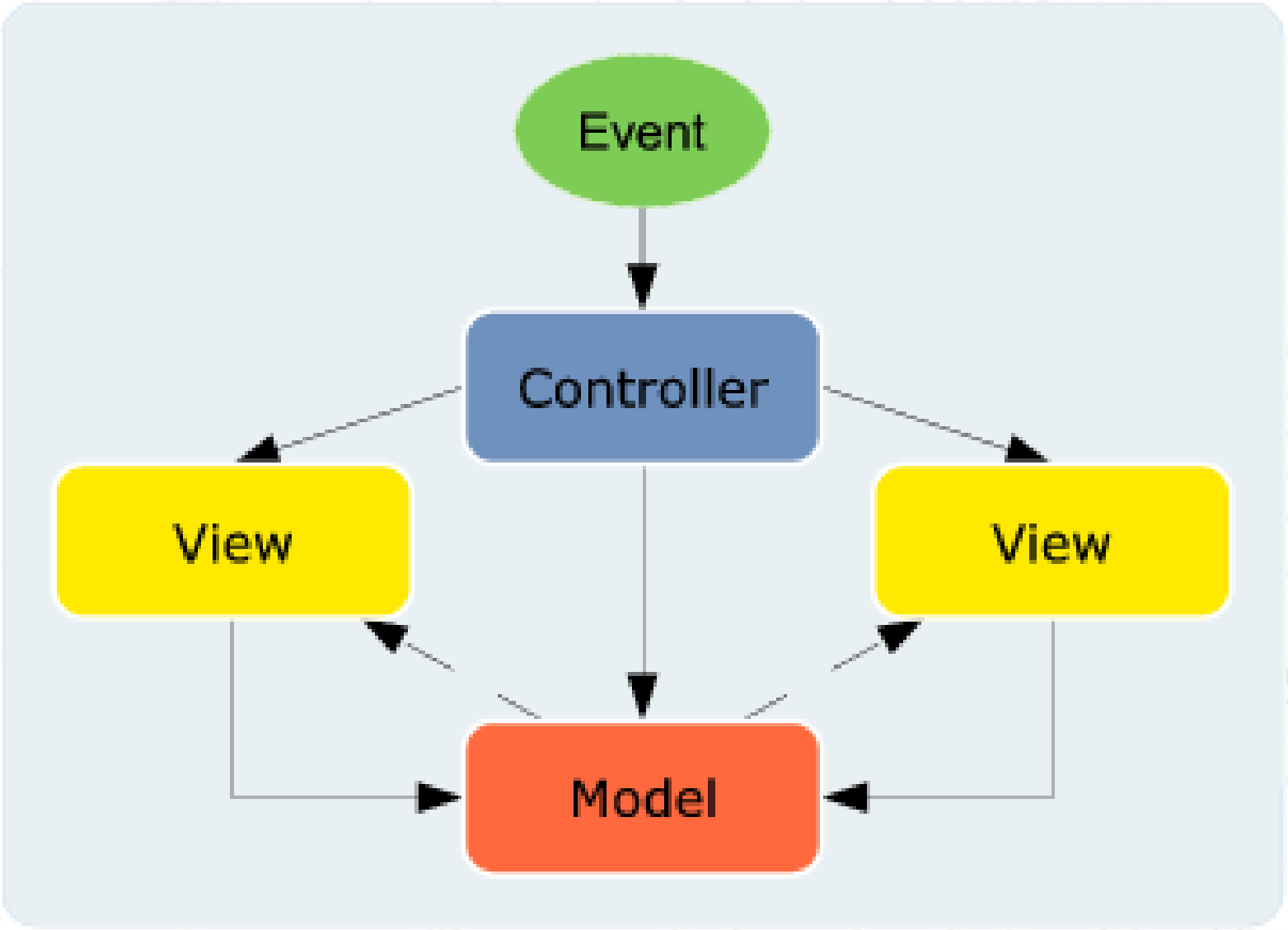
O padrão MVC divide uma aplicação em três tipos de componentes: modelo, visão e controlador.

Modelo: Encapsula os dados e as funcionalidades da aplicação.

Visão: É responsável pela exibição de informações, cujos dados são obtidos do modelo.

Controlador: recebe as requisições do usuário e aciona o modelo e/ou a visão.

MVC



Rotas

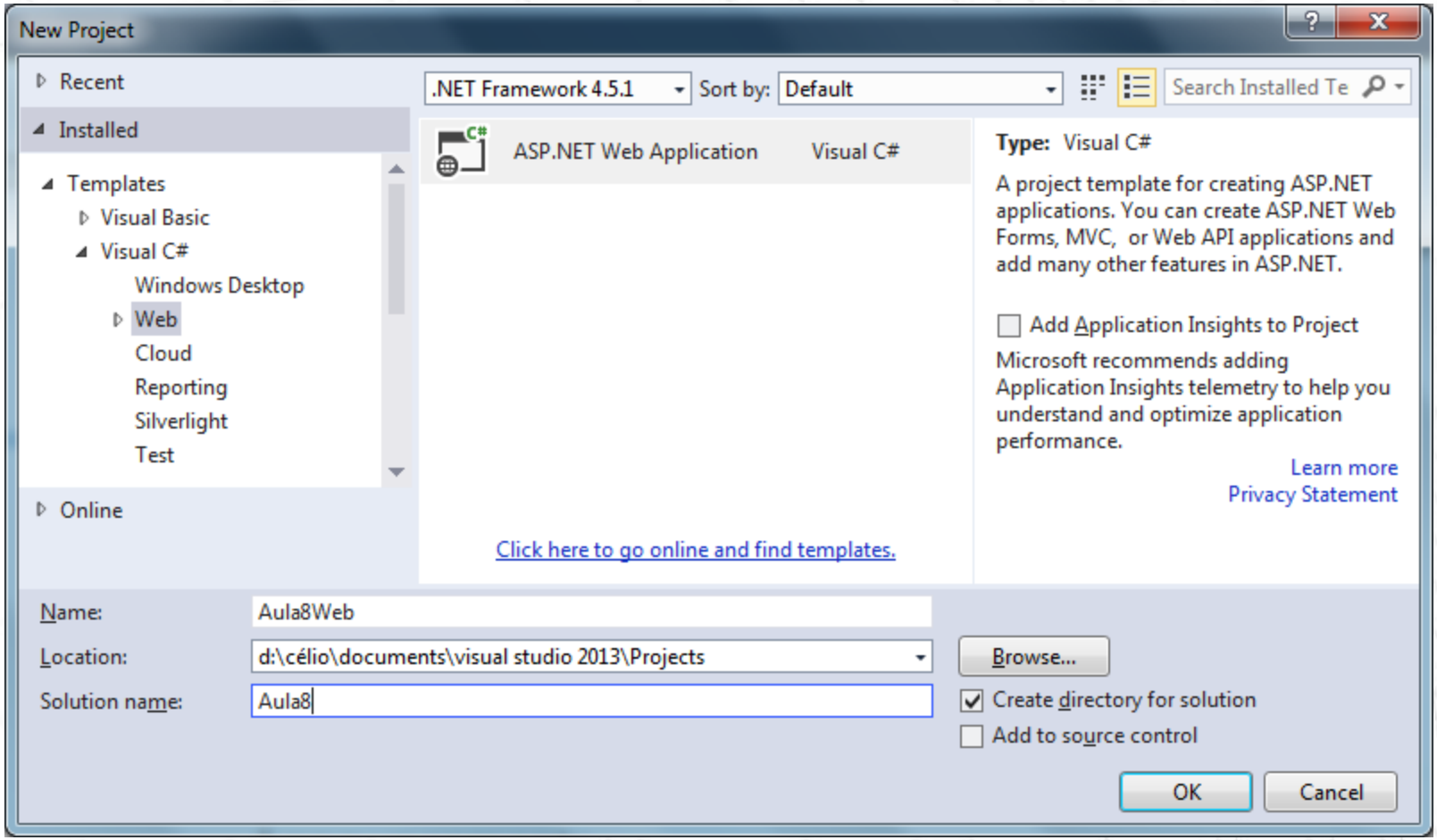
Na arquitetura de WebForms, temos uma *url* (geralmente subdividida em pastas no servidor) que, no final das contas, aponta para um arquivo físico no servidor (.*aspx*) sendo que neste, estão implementadas todas as ações solicitadas pelo *browser*. No ASP.NET MVC o modelo é um pouco diferente, haja vista que a requisição é tratada por um elemento controlador e este decide como gerar a visualização, portanto, ao invés de apontarmos para uma página (física) apontamos para uma ação dentro de um elemento conceitual, o controlador. Assim, precisamos realmente de uma rota para definir o caminho final dos dados.

Rotas

O *framework* ASP.MVC traz como sugestão uma rota pré-definida, conforme apresenta na figura abaixo. Entretanto, o desenvolvedor possui o poder de "customizar" inclusive a rota do aplicativo. Evidentemente que, para maioria dos casos, a rota proposta pela *framework* atende as necessidades, entretanto é importante frisar: no modelo MVC o desenvolvedor é que manda e não a *framework*.

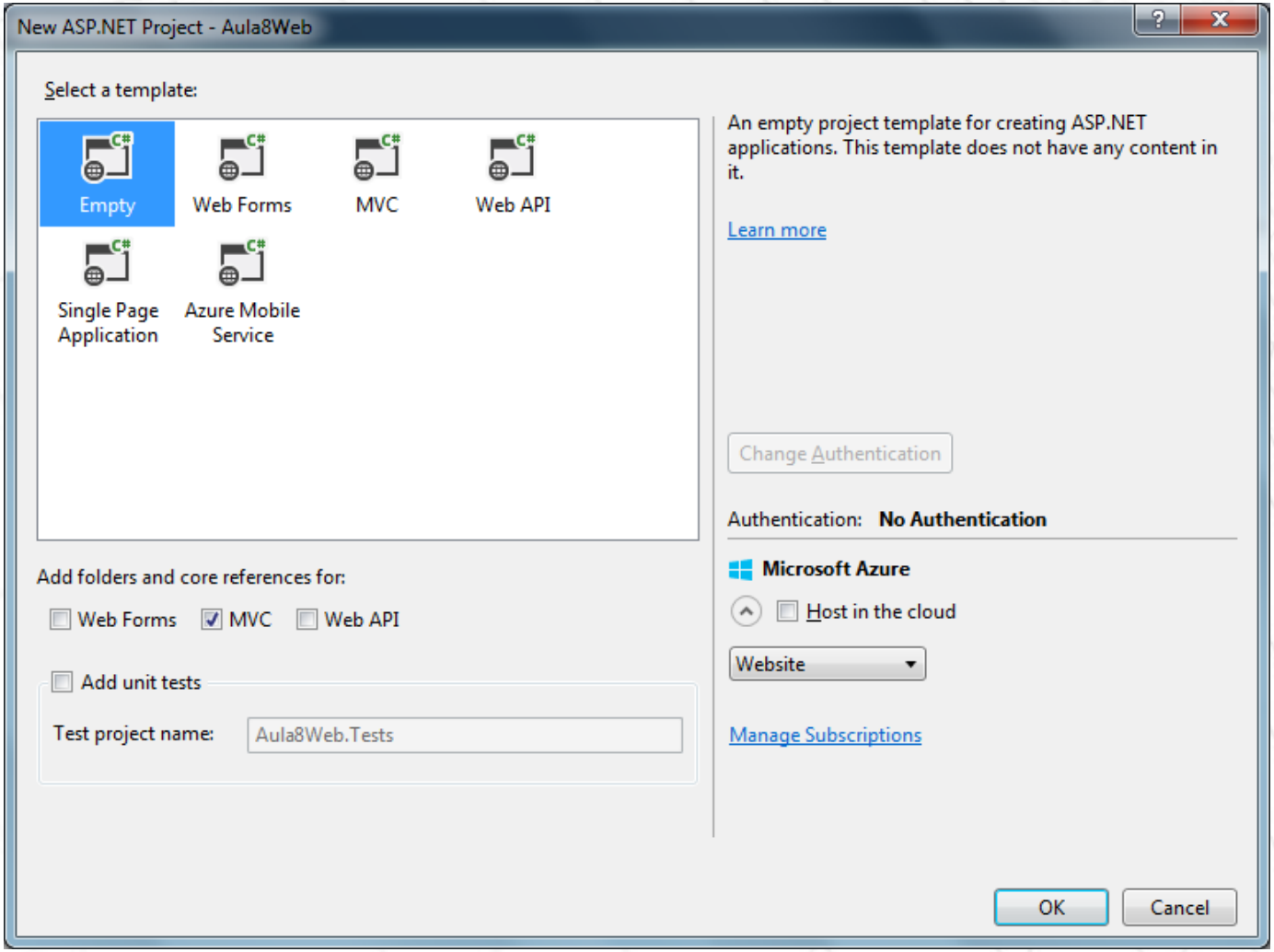
```
10  public class RouteConfig
11  {
12      public static void RegisterRoutes(RouteCollection routes)
13      {
14          routes.IgnoreRoute("{resource}.axd/{*pathInfo}");
15          routes.MapRoute(
16              name: "Default",
17              url: "{controller}/{action}/{id}",
18              defaults: new { controller = "Principal", action = "Index", id = UrlParameter.Optional }
19          );
20      }
21  }
```

Criando um Projeto Asp.Net MVC

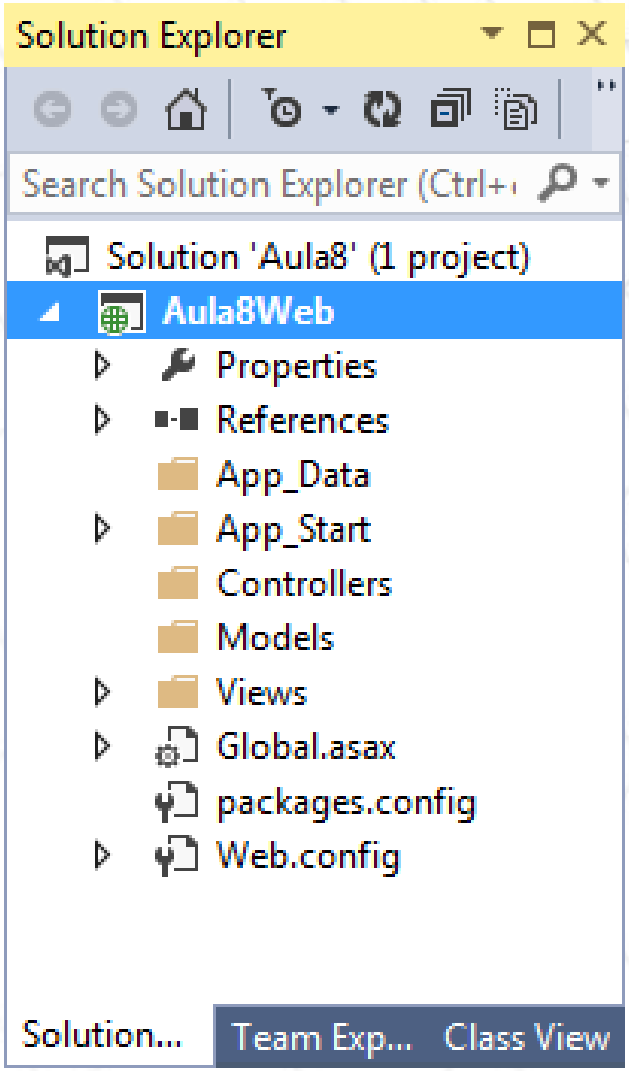


UNIVALI

Criando um Projeto Asp.Net MVC



Estrutura do Projeto



Razor e ASPX

Até a segunda versão do ASP.NET MVC, as telas (ou páginas) de uma aplicação web eram desenvolvidas em ASPX. A partir da terceira versão desse framework, podemos usar a linguagem Razor para construir essas telas. A principal característica da Razor é ser concisa e simples, diminuindo o número de caracteres e as tradicionais tags de scripts do ASP.NET MVC (<% %>).

Criando Variáveis

```
@{
  string nome = "K19";
  string telefoneDaK19 = "2387-3791";
  string enderecoDaK19 = "Av. Brigadeiro Faria Lima";
  int numeroDaK19 = 1571;
}
```

Criando variáveis em Razor

```
<%
  string nome = "K19";
  string telefoneDaK19 = "2387-3791";
  string enderecoDaK19 = "Av. Brigadeiro Faria Lima";
  int numeroDaK19 = 1571;
%>
```

Criando variáveis em ASPX

Acessando variáveis

```
<p>Telefone da Univali : @telefone</p>
```

Razor

```
<p>Telefone da Univali : <%= telefone%></p>
```

ASPX

Condicionais (if e else)

```
@if ( numero == numeroDaUnivali )
{
  <p>Chegou na Univali !</p>
}
Else
{
  <p>Continue andando .</p>
}
```

Razor

Condicionais (if e else)

```
<% if( numero == numeroDaUnivali )
{ %>
<p>Chegou na Univali !</p>
<%
}
else
{
%>
<p>Continue andando .</p>
<% } %>
```

ASPX

ViewBag

A ViewBag é utilizada para transmitir dados da camada de controle para a camada de apresentação.

```
ViewBag.HoraDoServidor = DateTime.Now.ToShortTimeString ();
```

Adicionando um item na ViewBag pelo controller

Na camada de apresentação, os itens armazenados na ViewBag podem ser acessados facilmente através da sintaxe da Razor.

```
A hora do servidor é @ViewBag .HoraDoServidor
```

Acessando umitem da ViewBag

Strogly Typed Views

Uma alternativa à utilização da ViewBag é o recurso strongly typed views. Com esse recurso podemos fixar, em tempo de compilação, o tipo do objeto que será transferido da camada de controle para a camada de apresentação. Uma vez que o tipo desse objeto foi fixado, o compilador é capaz de verificar a validade do código, evitando eventuais erros de execução. Além disso, o Visual Studio pode auxiliar o desenvolvimento da aplicação com os recursos de *IntelliSense e de refatoração.*

```
public ActionResult Acao ()
{
   Usuario usuario = new Usuario { Nome = "Fulano", Email = "fulano@univali.br" };
   return View ( usuario);
}
```

Passando um objeto para a camada de apresentação

```
@using Aula8Web.Models
@model Usuario

<h2>Nome do Usuário: @Model.Nome</h2>
```

Utilizando a propriedade Model

Helpers

A função das páginas CSHTML é gerar hipertexto XHTML para enviar aos navegadores dos usuários.

Os arquivos .cshtml misturam tags XHTML com scripts de servidor escritos em C# (ou outra linguagem de programação suportada pelo .NET). Essa mistura pode prejudicar a legibilidade do código em alguns casos. Além disso, a manutenção da aplicação pode se tornar mais complexa.

Considere a criação de um link utilizando diretamente as tags HTML.

```
<a href ="/ Editora / Index /">Lista de editoras </a>
```

Para facilitar a manutenção do código das páginas CSHTML, o ASP.NET MVC oferece os chamados **HTML Helpers.** A função de um HTML Helper é encapsular um código XHTML. Por exemplo, para adicionar um link, podemos usar o método ActionLink do objeto Html, ao invés de usar a tag <a> do HTML diretamente. No código abaixo, criamos um link para a ação responsável por listar as editoras.

```
@Html . ActionLink (" Lista de editoras ", " Index ", " Editora ")
```

ActionLink Helper

O helper ActionLink é utilizado para gerar os links das páginas de uma aplicação web. Esse helper pode ser utilizado de diversas maneiras. A forma mais simples de utilizá-lo é passar a ele dois parâmetros: o texto e a ação associados ao link desejado. Nesse caso, o link gerado pelo helper ActionLink estará associado ao controlador correspondente a página na qual o link foi inserido.

```
@Html . ActionLink (" TEXTO PARA O USUÁRIO ", " ACTION " )
```

Criando um link com controlador implícito

```
@Html . ActionLink (" TEXTO PARA O USUÁRIO ", " ACTION ", " CONTROLADOR " )
```

Criando um link com controlador explícito

```
@Html . ActionLink (" TEXTO PARA O USUÁRIO ", " ACTION ", new { Inicio = 0, Final = 10 })
```

Acrescenta parâmetros de url

BeginForm e *EndForm* Helpers

No ASP.NET MVC, há um conjunto de HTML Helpers para facilitar a criação de formulários nas páginas de uma aplicação web.
Para criar um formulário, utilizamos o helper BeginForm. Para terminar um formulário, utilizamos o helper EndForm.

```
@{ Html . BeginForm () ;}
<! -- elementos do formulário -->
@{ Html . EndForm () ;}
```

Também podemos utilizar o comando using para garantir que os formulários sejam fechados.
Nesse caso, não é necessário adicionar o helper EndForm.

```
@using ( Html . BeginForm ()) {
<! -- elementos do formulário -->
}
```

Outros Helpers

```
@Html.TextBox (" Nome ", " Digite o seu nome “)
```

```
@Html.TextArea (" Mensagem ", " Digite uma mensagem ")
```

```
@Html.RadioButton (" CidadeNatal ", " São Paulo ", true )
@Html.RadioButton (" CidadeNatal ", " Natal ", false ) @Html
@Html.RadioButton (" CidadeNatal ", " Piracicaba ", false )
```

```
@Html.Hidden ("Id", 1)
```

```
@Html.Password (" Password ", " senha ")
```

Strongly Typed Helpers

Se os HTML Helpers forem utilizados de maneira análoga à mostrada anteriormente, a probabilidade de ocorrer um erro de digitação é alta. A forma que os HTML Helpers foram aplicados não permite que o compilador verifique a existência das propriedades associadas aos elementos dos formulários.

```
@Html.CheckBox (" Aceito ", false )
```

O código anterior supõe que o objeto recebido como parâmetro pela ação associada ao formulário onde o checkbox foi inserido possua uma propriedade chamada Aceito. Essa propriedade pode não existir. Contudo, nenhum erro de compilação seria gerado nesse caso. Para evitar esse tipo de problema, podemos utilizar a seguinte sintaxe em telas fortemente tipadas:

```
@model Aula8Web. Models.Contrato
...
@Html . CheckBoxFor (c => c. Aceito )
...
```

Outros Strongly Typed Helpers

```
@Html.TextBoxFor( x => x.Nome)
```

```
@Html.TextAreaFor( x => x.Descricao)
```

```
@Html.RadioButtonFor(x => x.CidadeNatal, " São Paulo ", true )
@Html.RadioButtonFor(x => x.CidadeNatal, " Natal ", false )
@Html.RadioButtonFor(x => x.CidadeNatal, " Piracicaba ", false )
```

```
@Html.HiddenFor(x => x.Id)
```

```
@Html.PasswordFor(x => x.Senha)
```

EditorFor

```
public class Editora
{
public int Id { get ; set ; }
public string Nome { get ; set ; }
public string Email { get ; set ; }
public bool IsAtivo { get ; set ; }
public string Descricao { get ; set ; }
}
```

```
@Html.TextBoxFor( model => model . Nome )
@Html.TextBoxFor( model => model . Email )
@Html.CheckBoxFor( model => model . IsAtivo)
```

```
@Html.EditorFor( model => model . Nome )
@Html.EditorFor( model => model . Email)
@Html.EditorFor( model => model . IsAtivo )
```

EditorForModel

```
public class Editora
{
public int Id { get ; set ; }
public string Nome { get ; set ; }
public string Email { get ; set ; }
public bool IsAtivo { get ; set ; }
public string Descricao { get ; set ; }
}
```

```
@model Aula8Web.Models.Editora
@{
ViewBag.Title = " Edição de Editora ";
}
<h2 >Edição de Editora </h2 >
@using (Html . BeginForm ()) {
    @Html.EditorForModel ()
<input type =" submit " value =" Salvar" />
}
```

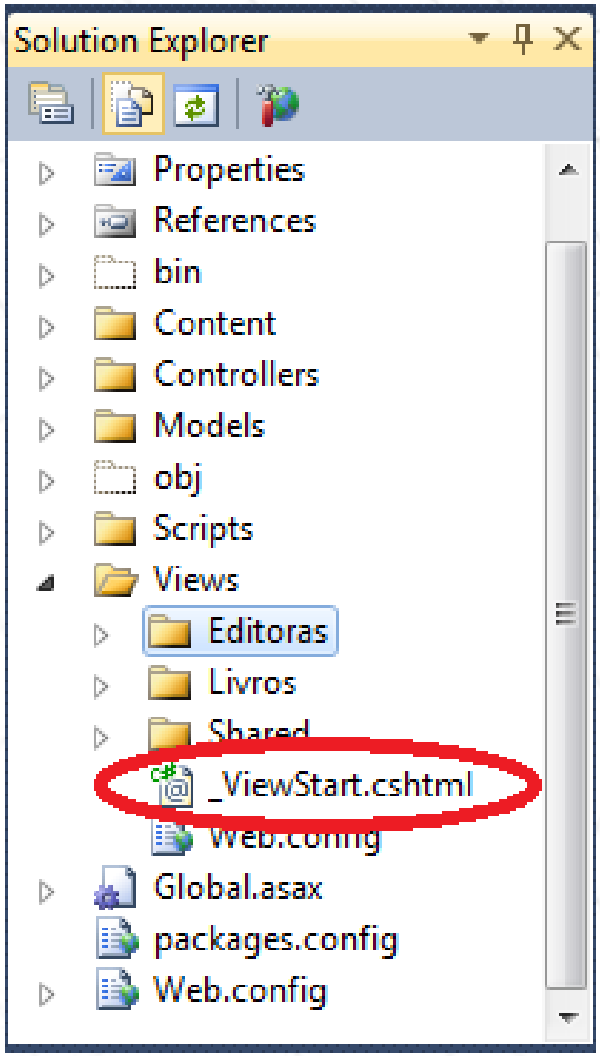
Personalizando o EditorFor e o EditorForModel

```
public class Usuario
{
[ HiddenInput ( DisplayValue = false )]
public int Id { get ; set ; }
[ Display ( Name =" Nome do Usuário ")]
public string Nome { get ; set ; }
[ DataType ( DataType . EmailAddress )]
public string Email { get ; set ; }
[ DataType ( DataType . Password )]
public string Senha { get ; set ; }
[ DataType ( DataType . MultilineText )]
public string Descricao { get ; set ; }
[ DataType ( DataType . Date )]
[ DisplayFormat ( ApplyFormatInEditMode = true , DataFormatString = " {0: yyyy -MM -dd}")]
public DateTime DataDeCadastro { get ; set ; }
}
```

Layouts

É comum que as páginas de uma aplicação web possuam conteúdo em comum (por exemplo, um cabeçalho ou um rodapé). O conteúdo em comum pode ser replicado em todas as páginas através do CTRL+C e CTRL+V. Porém, essa não é uma boa abordagem, pois quando alguma alteração precisa ser realizada, todos os arquivos devem ser modificados. Também é comum que as páginas de uma aplicação web possuam um certo padrão visual. Daí surge o conceito de **Layouts.**

_ViewStart.cshtml



Section (seções em layout)

```
<! DOCTYPE html >
<html >
<head >
<title >@ViewBag . Title </ title >
   @Styles . Render ("~/ Content / Site . css ")
   @Scripts . Render ("~/ Scripts / jquery -1.5.1. min .js")
</ head >
<body >
   <div id=" header ">
      @Html . ActionLink (" Editoras ", " Index ", " Editora ")
      @Html . ActionLink (" Livros ", " Index "," Livro ")
   </ div >
   <div id=" sidebar "> @RenderSection (" Sidebar ",false )</ div >
   <div id=" content "> @RenderBody () </ div >
   <div id=" footer ">K19 Treinamentos </ div >
</ body >
</ html >
```

@section (seções em layout)

```
@model Aula8Web.Models.Editora
@{
Layout = "~/ Views / Shared / _Layout . cshtml ";
// Define o título específico desta página
ViewBag . Title = " Edição de Editora ";
}
<h2 >Edição de Editora </h2 >
@using ( Html . BeginForm ()) {
@Html . EditorForModel ()
<input type =" submit " value =" Salvar " />
}
@section Sidebar {
<p>Sidebar do cadastro de Edição de Editora </p>
}
```

Camada de Controle

No ASP.NETMVC as urls são mapeadas para métodos (ações) em classes que definem os chamadas controladores. As requisições enviadas pelos navegadores são processadas pelos controladores.
O processamento realizado por um controlador para tratar uma requisição consiste basicamente em:

- Recuperar os dados enviados pelo usuário através de formulários.
- Interagir com a camada de modelo.
- Acionar a camada de apresentação para construir a página HTML que deve ser enviada para o usuário como resposta à sua requisição.

Para que uma classe seja considerada um controlador, ela deve seguir algumas regras.

- O nome da classe deve ter o sufixo “Controller”.
- A classe deve implementar a interface System.Web.Mvc.IController ou herdar da classe System.Web.Mvc.Controller.

Actions

Os controladores e as ações são elementos fundamentais de uma aplicação ASP.NET MVC. Um controlador pode conter diversas ações. As ações são utilizadas para processar as requisições realizadas pelos navegadores. Para criar uma ação, é necessário definir um método public dentro da classe de um controlador. Os parâmetros desse método podem ser utilizados para receber os dados enviados pelos usuários através de formulários HTML. Esse método deve devolver um ActionResult que será utilizado pelo ASP.NET MVC para definir o que deve ser executado depois que a ação terminar.

Quando um usuário faz uma requisição HTTP através de um navegador, o ASP.NET MVC verifica na tabela de rotas o controlador e a ação associados à url da requisição realizada. Essa tabela é definida no arquivo RouteConfig.cs e inicializada no arquivo Global.asax.

ActionResult

Quando uma ação termina, o método correspondente deve devolver um ActionResult. O valor devolvido indica para o ASP.NET MVC o que deve ser executado depois da ação.

ViewResult: Devolve uma página da camada de apresentação.

Considerando uma aplicação ASP.NET MVC em C# e Razor, o método View(), ao ser chamado sem parâmetros, executará o seguinte processo para determinar qual arquivo deve ser utilizado para construir a página de resposta.

1. Se o arquivo Views\Teste\Acao.cshtml existir, ele será utilizado.
2. Caso contrário, se o arquivo Views\Shared\Acao.cshtml existir, ele será utilizado.
3. Se nenhum desses arquivos existir, uma página de erro será devolvida.

Outras ActionResult

PartialViewResult: Devolve uma página parcial da camada de apresentação.
Ex.: return PartialView ();

RedirectResult: Redireciona o navegador para uma URL específica.
Ex.: return Redirect("http://univali.br");

RedirectToAction: Redireciona para outra ação da camada de controle.
Ex.: return RedirectToAction (" OutraAction ", " OutroController ");

ContentResult: Devolve texto.
Ex.: return Content (" Texto "," text \ plain ");

JsonResult: Devolve umobjeto no formato JSON.
Ex.: return Json (editora);

JavaScriptResult: Devolve código Javascript.
Ex: return JavaScript ("\$('\# divResultText '). html (' JavaScript Passed ');");

FileResult: Devolve dados binários.
Ex.: return File (@"c:\ relatorio . pdf ", " application \ pdf ");

EmptyResult: Devolve uma resposta vazia.
Ex.: return EmpytResult();

1

Implementar uma aplicação que esta no site no Link Abaixo.
http://www.macoratti.net/13/04/mvc4_app.htm

Referencias Bibliográficas

- k19-k32-desenvolvimento-web-com-aspnet-mvc – PDF Anexo.
- http://fabriciosanchez.com.br/2/desenvolvimento/asp-net-mvc/
- http://www.macoratti.net/13/04/mvc4_app.htm