Aplicações Web MVC

IFRN – Instituto Federal de Educação, Ciências e Tecnologias do Rio Grande do Norte.

Aplicações Web

ASP.NET MVC

ASP.NET MVC fornece, por meio de design patterns, uma maneira poderosa e alternativa para criar websites dinâmicos.

O ASP.NET MVC implementa o padrão MVC e separa a aplicação em três componentes: model, controller e view.

- O model contém o código da camada de dados.
- O controller recebe as requisições do usuário.
- O view implementa o design da aplicação.

Nota: ASP.NET MVC é framework baseado no .NET que foi liberado em 2009. A versão inicial usava os mesmos web forms (.aspx), que foi usado no ASP.NET framework. Em 2010, um novo engine, chamado **Razor** foi liberada, a qual é mais natural, com a sintaxe HTML. A versão MVC 4.0 é incluída com o Visual Studio 2012.

Aplicação Web

Observe a figura 1.1:

Na figura 1.1 vemos a relação entre o controller, o model e o view. O controller aceita a requisição do usuário, acessa o view e/ou model. O model acessa o banco de dados. Ao final, o conteúdo do view é retornado ao usuário.

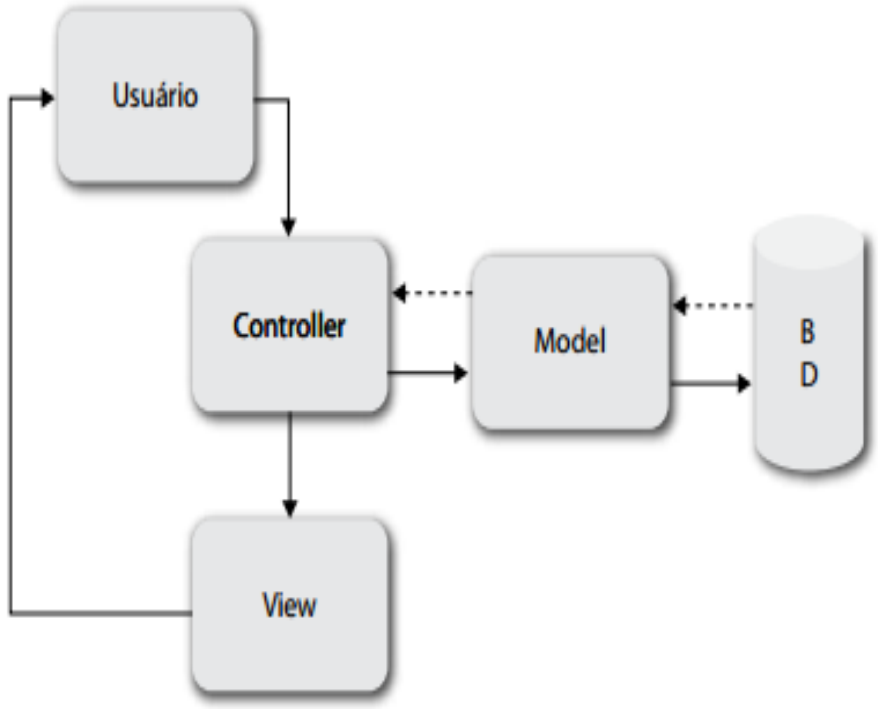


Figura 1.1 – Relação entre o usuário, o controller, o model e o view.

Aplicação Web

Recursos do ASP.NET MVC

O ASP.NET MVC permite, entre outras coisas:

- Controle total sobre o HTML.
- Criação de URLs amigáveis.
- Clara separação entre o design, o código e a camada de dados.
- Validação no cliente e no servidor.
- Definição de filtros de ação.
- Facilidade em implementar aplicações AJAX.
- Uso de sintaxe Razor.
- Uso intensivo de atributos.
- Implementação e criação de HTML Helpers.
- A implementação de unidades de teste.
- Implementa o padrão MVC.

Aplicações Web

MVC

Model-view-controller (**MVC**) é um padrão arquitetural que tem sido usado desde o inicio dos anos 1970. O principal benefício deste padrão é, favorecer o desenvolvimento do sistema em camadas independentes uma das outras.

- **View**: fornece a experiência com o usuário (apresentando dados e interagindo com o usuário).
- **Model**: fornece dados e lógica de negócio.
- **Controller**: manipula as requisições do usuário, passando esses dados para o modelo, invocando a view apropriada.

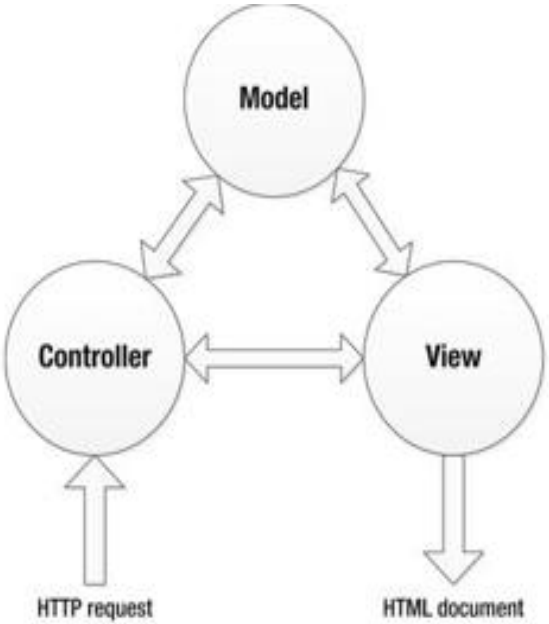


Figura 1.2 Padrão MVC

Aplicação Web

Desenvolvendo uma nova aplicação

- Inicie o Visual Studio 2010+ . Acesse o menu File e clique em New Project ....
- Na caixa de diálogo New Project selecione Visual C# e ASP.NET MVC 4 Web Application. Nomeie o projeto como AppExemplo1. Clique em OK. Conforme figura 1.3.

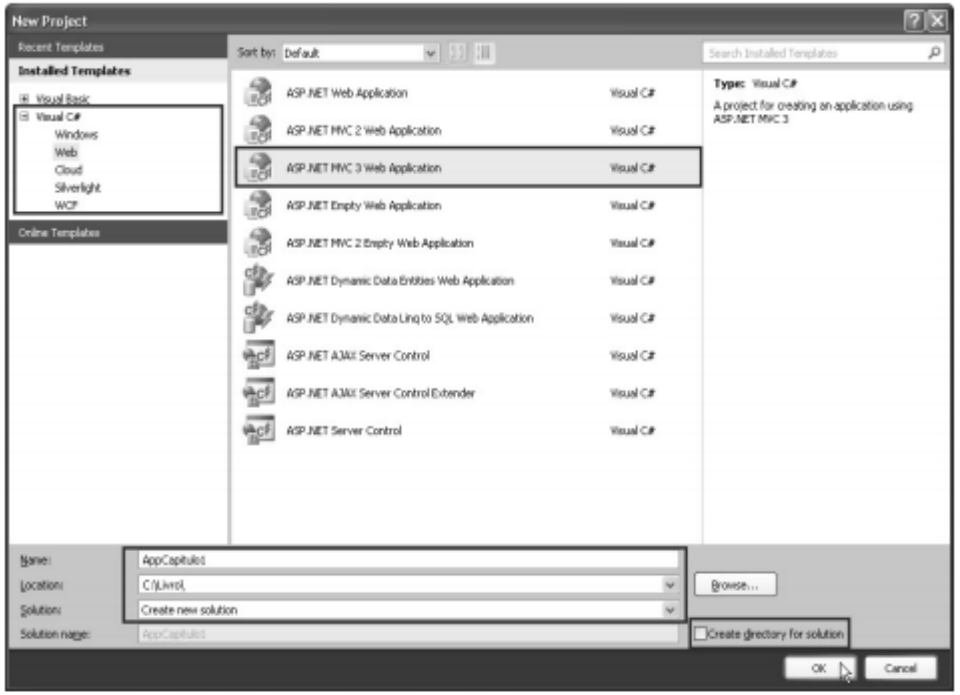


Figura 1.3 – Criando uma nova aplicação ASP.NET MVC.

Aplicação Web

Em seguida, escolha um dos dois templates para iniciar seu projeto:

- Empty conté a estrutura básica de arquivos e diretórios usados por uma aplicação ASP.NET MVC.
- Internet aplication contém além dos recursos de Empty arquivos e diretórios usados na autenticação de usuários e formatação.

Na caixa de combinação View Engine:. Selecione o mecanismo de exibição **ASPX**. Observe a figura 1.4.

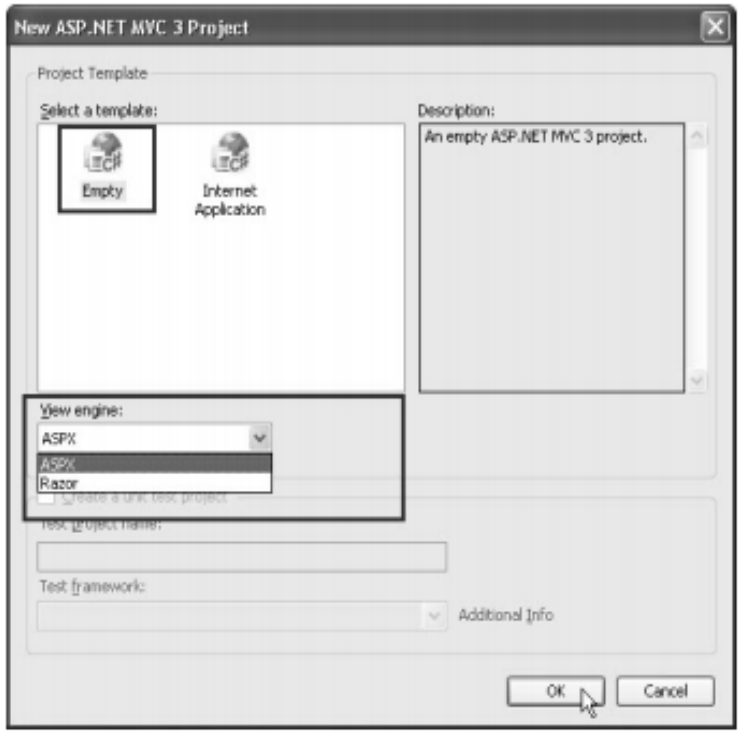


Figura 1.4 – Definindo o template e o mecanismo de exibição da aplicação.

Aplicação Web

Os diretórios-padrão da aplicação ASP.NET MVC são:

Diretório	Descrição
App_Data	Contém os arquivos do banco de dados (.MDF, .MDB), documentos xml.
Content	Contém os arquivos estáticos como imagens e arquivos CSS.
Controllers	Contém as classes do controlador. Recebe as requisições do usuário. Ex.: o usuário digita o endereço da Internet de uma página ou clique num link
Models	Contém as classes da camada de dados. Manipula informações do banco de Dados
Scripts	Contém os arquivos JavaScript, jQuery, arquivos .js em geral.
Views	Contém em geral arquivos .aspx, .ascx, .master, .cshtml responsáveis pela exibição do conteúdo ao usuário

Aplicação Web

Além dos diretórios-padrão, listados anteriormente, há diretórios com significados especiais:

Diretório	Descrição
Areas	O ASP.NET MVC cria uma nova estrutura de diretórios e arquivos. Ideal para dividir uma aplicação grande ou criar uma área restrita no website.
App_GlobalResources	Contém os arquivos de recursos globais (.resx e .resources). Esses arquivos contêm imagens, textos, arquivos etc.
App_LocalResources	Contém os arquivos de recursos locais (.resx e .resources). Esses arquivos contêm imagens, textos, outros arquivos etc.
App_Browsers	Contém os arquivos com a extensão .browser. Os arquivos .browser gerenciam as informações sobre os recursos implementados pelo navegador.
App_Themes	Contém os arquivos .skin e .css responsáveis pela aparência da aplicação.

Aplicação Web

Controller

Para adicionar um novo controlador à aplicação web, acesse a janela Solution Explorer. Clique com o botão direito do mouse no diretório Controllers. Em seguida, aponte para Add e clique em Controller.... Na caixa de diálogo Add Controller digite o nome do controlador – HomeController. Observe a figura 1.5

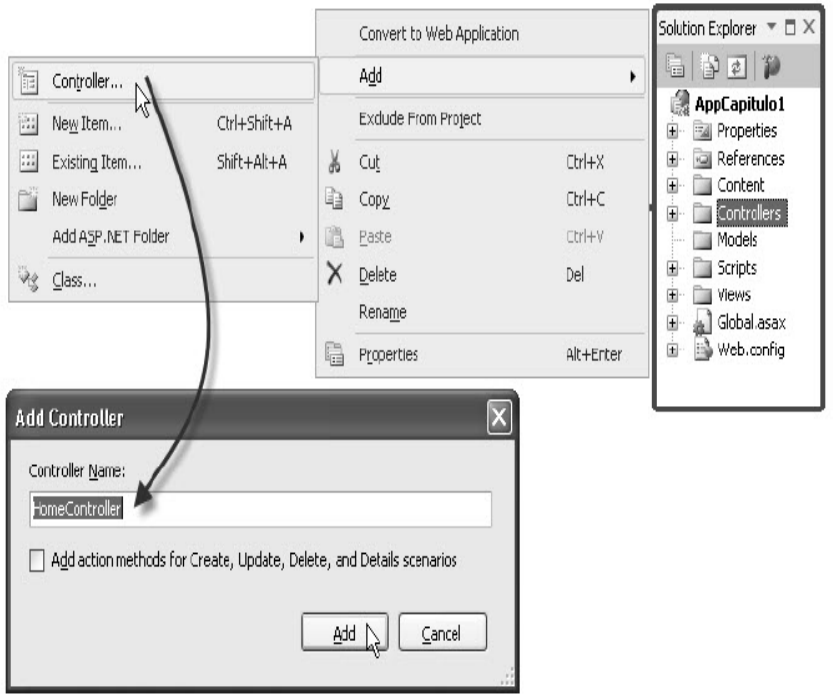


Figura 1.5 Adicionando um novo controller.

Aplicação Web

O ASP.NET MVC requer controladores com o sufixo Controller. Exemplo: HomeController, DetailsController, EditController etc. O prefixo é usado nas entradas e interações do usuário.

Exemplo:

http://localhost:1573/Home
http://localhost:1573/Details
http://localhost:1573/Edit

Aplicação Web

Quando clicamos no botão Add da caixa de diálogo Add Controller, o Visual Studio acrescenta o arquivo HomeController.cs ao diretório Controllers e o seguinte código:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Mvc;
namespace AppCapitulo1.Controllers {
public class HomeController : Controller {
        // GET: /Home/
        public ActionResult Index() {
                ViewBag.Message = "Olá mundo!";
                return View();
        }
    }
}
```

Aplicação Web

Master pages

Para exibir elementos comum a todas as páginas, use master pages. Uma master page pode facilmente exibir menus, logos, banners, e seu layout pode conter textos estáticos, elementos HTML, imagens e web server controls.

Para acrescentar uma master page a um projeto ASP.NET MVC, siga os passos descritos a seguir:

Clique com o botão direito do mouse no diretório /Views/Shared. Selecione Add e, em seguida, aponte para New Item…. Observe a figura 1.6:

Aplicação Web

Adicionando uma Master Page.

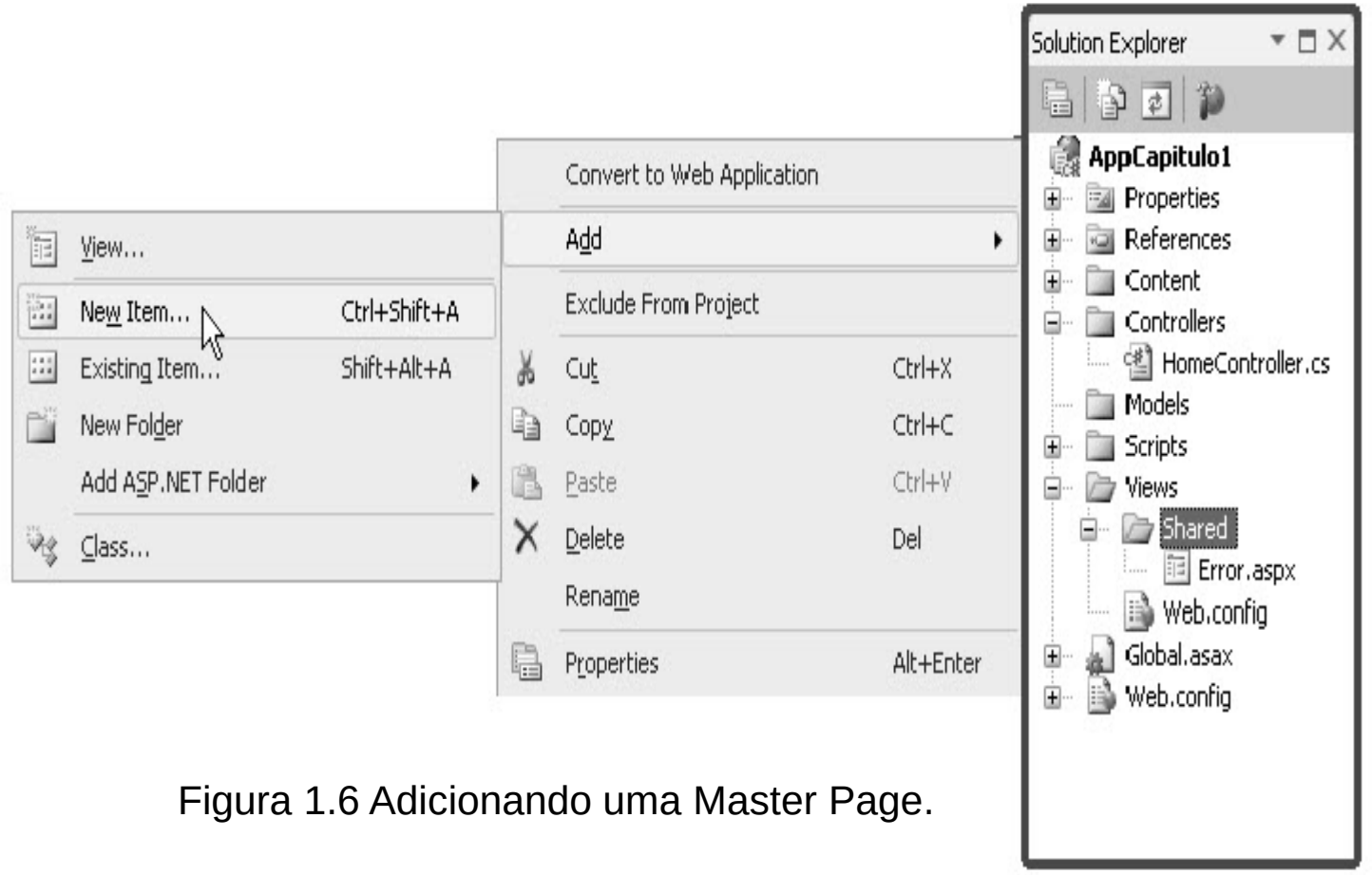


Figura 1.6 Adicionando uma Master Page.

Aplicação Web

Na caixa de diálogo **Add New Item** selecione **MVC 4 View Master Page (ASPX)**. Nomeie como Site. Master. Clique em **Add**.

O arquivo Site.Master contém o seguinte código:

```
<%@ Master Language="C#" Inherits="System.Web.Mvc.ViewMasterPage" %>
<!DOCTYPE html>
<html>
<head runat="server">
        <title><asp:ContentPlaceHolder ID="TitleContent" runat="server" /></title>
</head>
<body>
        <div>
                <asp:ContentPlaceHolder ID="MainContent" runat="server">
                </asp:ContentPlaceHolder>
        </div>
</body>
</html>
```

Aplicação Web

Para aplicar a master page em um view, basta adicionar o atributo MasterPageFile à diretiva @ page:.

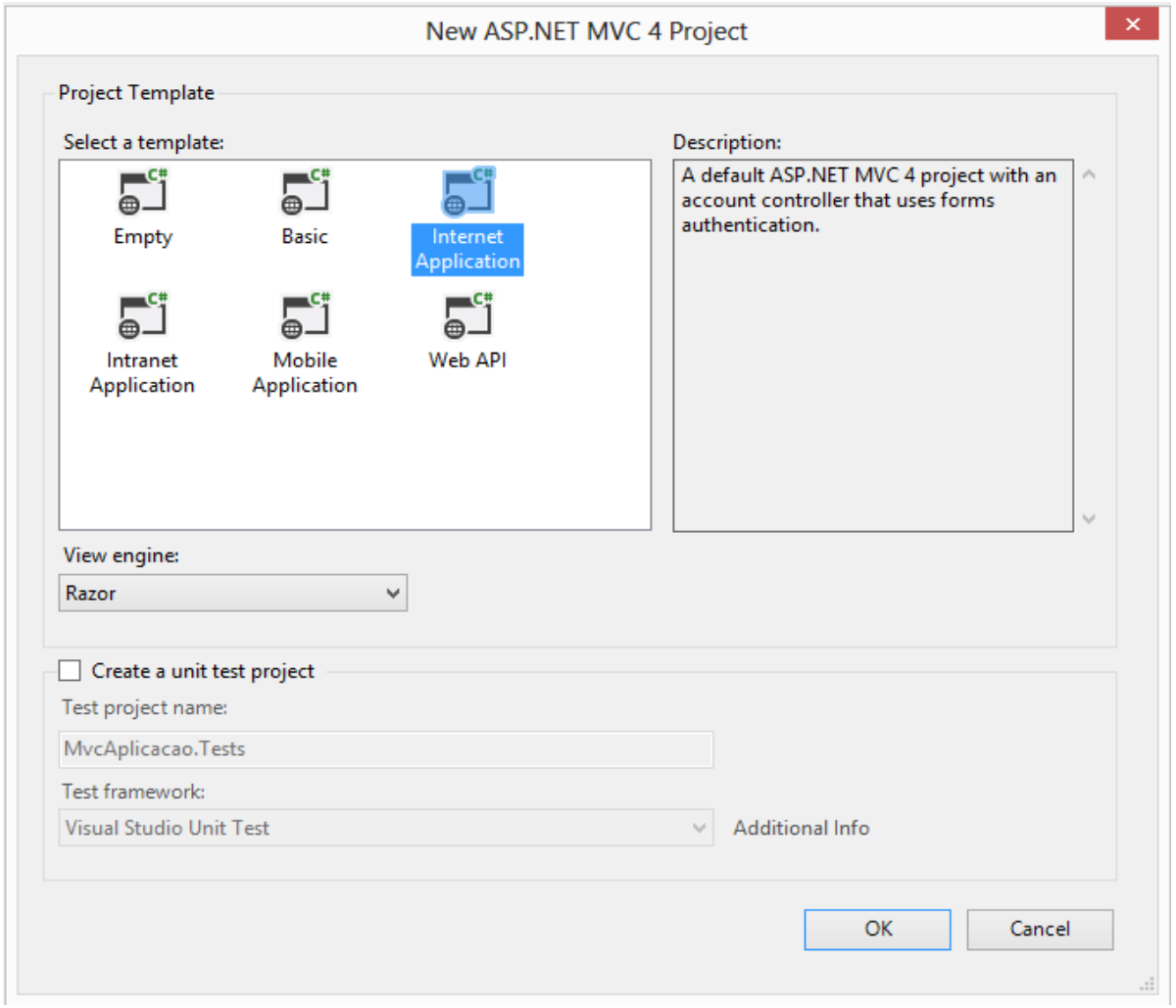
```
<%@ Page Title="" Language="C#" MasterPageFile="~/Views/Shared/Site.Master"
                                Inherits="System.Web.Mvc.ViewPage<dynamic>" %>
```

Aplicação Web

Views

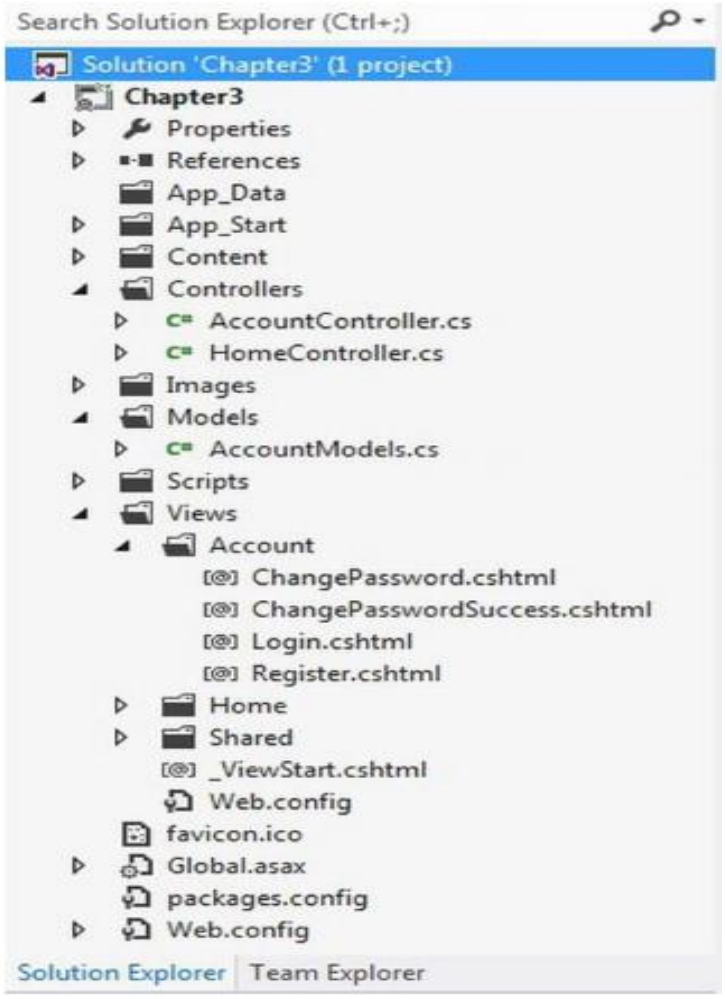
Criando um Projeto ASP MVC

1. Inicie o Visual Studio 2012.
2. Selecione o template ASP.NET MVC 4
3. Selecione o Internet Application (autenticação no form)
4. Assegure-se que o engine Razor está selecionado.



Aplicações Web

Após o projeto ter sido criado, você verá um número de pastas no Solution Explorer. Note que há pastas separadas para controllers, models e views, como mostra a figura ao lado.



Aplicações Web

Explorando a View Razor

Para uma rápida demonstração da sintaxe do Razor, vamos dá uma olhada no arquivo Register.cshtml, o quel você encontra na pasta Views\Account. Este implementa a view para a página de registro de usuário.

```
<fieldset>
    <legend>Registration Form</legend>
    <ol>
        <li>
            @Html.LabelFor(m => m.UserName)
            @Html.TextBoxFor(m => m.UserName)
        </li>
        <li>
            @Html.LabelFor(m => m.Password)
            @Html.PasswordFor(m => m.Password)
        </li>
        <li>
            @Html.LabelFor(m => m.ConfirmPassword)
            @Html.PasswordFor(m => m.ConfirmPassword)
        </li>
    </ol>
    <input type="submit" value="Register" />
</fieldset>
```

Na sintaxe do Razor um @ indica a marcação que será usada. O códogo irá gerar HTML em runtime. Observe que cada um desses métodos usa uma expressão lambda **(m => m.UserName**), que especifica um elemento de dado que será associado ao modelo.

O modelo que é usado pela view, é definido pela a instrução seguinte, no topo do arquivo:

@model MvcAplicacao.Models.RegisterModel

Aplicação Web

Se você olhar o arquivo AccountModels.cs, você irá encontrar a definição da classe RegisterModel. Esta classe tem quatro propriedades publicas:

- UserName
- Email
- Password
- ConfirmPassword

Cada uma dessas propriedade tem atributos metadata tal como Required e DataType, que são usados para gerar o HTML correto.

Aplicação Web

```
public class RegisterModel
   {
      [Required]
      [Display(Name = "User name")]
      public string UserName { get; set; }

      [Required]
      [StringLength(100, ErrorMessage = "The {0} must be at least {2} characters long.", MinimumLength = 6)]
      [DataType(DataType.Password)]
      [Display(Name = "Password")]
      public string Password { get; set; }

      [DataType(DataType.Password)]
      [Display(Name = "Confirm password")]
      [Compare("Password", ErrorMessage = "The password and confirmation password do not match.")]
      public string ConfirmPassword { get; set; }
   }
```

Aplicação Web

Usando o Editor Templates

1. Abra o arquivo Register.cshtml.
2. Adicione o campo Email, usando o EditorFor, para isso. Veja o código a seguir.
3. No Solution Explorer, clique de direita na pasta Views\Shared e selecione Add → New Folder, entre com EditorTemplate para o nome do folder. Isso implica que ele estará disponível para todas as view do projeto.
4. Clique de direita na pasta Views\Shared\EditorTemplate e selecione Add → View
5. Entre com EmailAddress como view name e assegure que todos os checkbox estão desabilitados. Como mostra a figura na página seguinte.

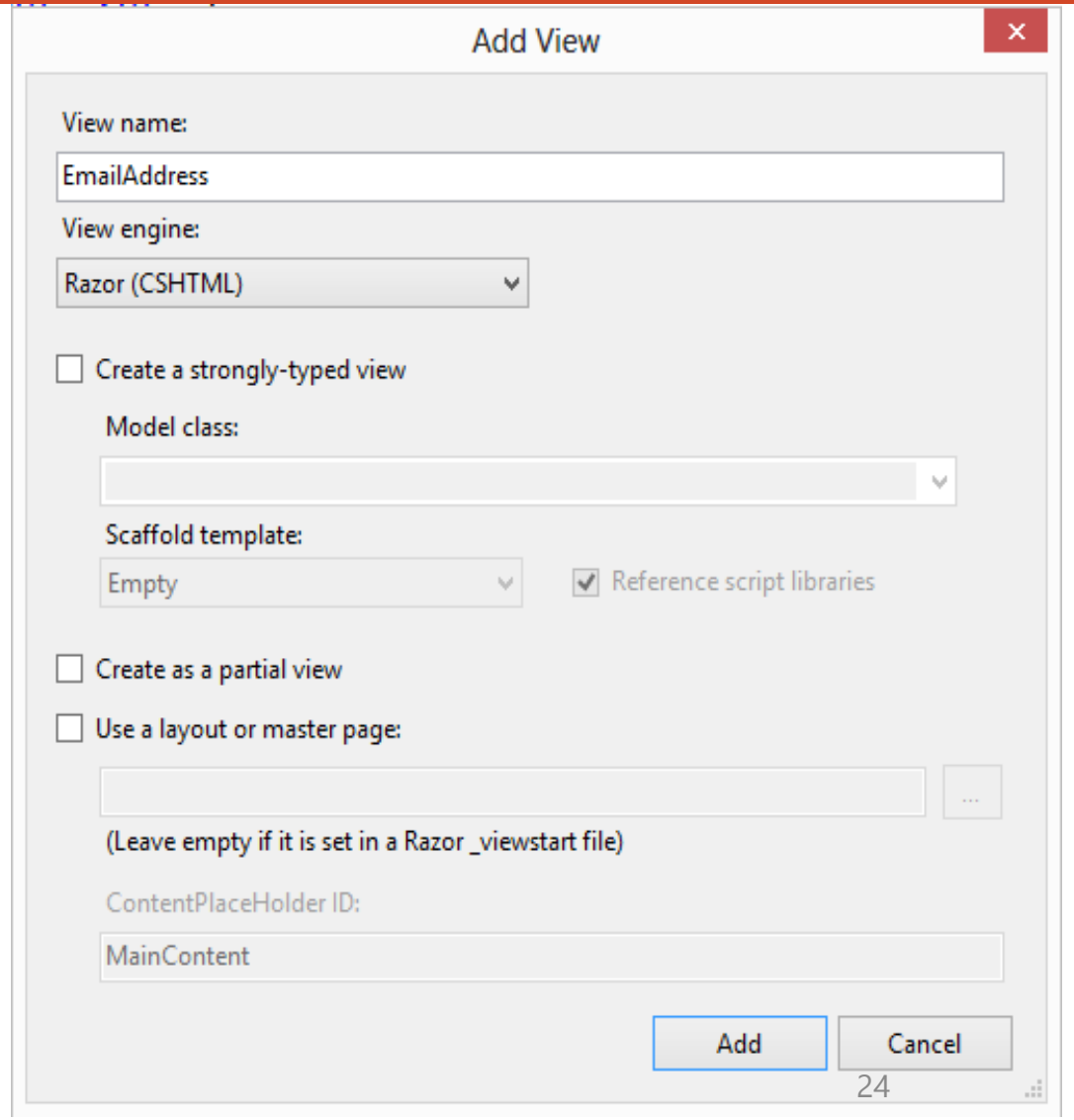
Aplicação Web

Isso irá gerar uma página nomeada EmailAddress.cshtml. Delete o conteúdo da entrada e substitua pelo seguinte código:

```
<div>
        @Html.TextBox("",null, new { @class = "text-box single-line", type="email", placehold="Enter an e-mail address"})
    </div>
```

6. Salve o projeto e execute para ver as mudanças.

Deve acontecer um erro, Porque?



Aplicação Web

O erro aconteceu, por que falta acrescentar o atributo **Email** ao modelo **AccountModels.cs**.

7. Abra o modelo AccountModels.cs e acrescente o atributo Email.

```
public class RegisterModel
    {
        [Required]
        [Display(Name = "User name")]
        public string UserName { get; set; }

        [Display(Name="Email Address")]
        public string Email { get; set; }
.
.
.
```

8. Salve e execute novamente o projeto e, dê uma olhada no código gerado.

Aplicação Web

Adicionando uma página Feedback

Vamos agora criar um form e usar esta página para demonstrar novos recursos HTML5. inicialmente vamos criar um modelo e implementar uma view baseado nesse modelo. Então, iremos adicionar um controller, que agirar como um link para a nova página.

Portanto, usualmente, uma aplicação envolve adicionar um modelo, adicionar uma view, e criar ou modificar um controller. O padrão MVC permite que isso seja desenvolvido separadamente e em grandes projetos, tem diferentes pessoas responsáveis por cada um desse módulos.

Aplicação Web

Criando o Modelo Feedback

Um modelo define os elementos de dados que podem ser incluído em sua página. Construindo o modelo, você pode implementar a visão.

1. No Solution Explorer, clique de direita na pasta Models e selecione Add → Class e, entre com FeedbackModel para o nome.
2. Entre com o seguinte código para implementação da classe.

Aplicação Web

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.ComponentModel.DataAnnotations;

namespace MvcAplicacao.Models
{
    public class FeedbackModel
    {
        [Display(Name = "Name", Prompt = "Enter your full name"), Required]
        public string Name { get; set; }

        [Display(Name = "Average Score", Prompt = "Your average score"), Range(1.0, 100.0), Required]
        public decimal Score { get; set; }

        [Display(Name = "Birthday"), DataType(DataType.Date)]
        public DateTime? Birthday { get; set; }

        [Display(Name = "Home page", Prompt = "Personal home page"), DataType(DataType.Url), Required]
        public string Homepage { get; set; }

        [Display(Name = "Email", Prompt = "Preferred-e-mail address"), DataType(DataType.EmailAddress), Required]
        public string Email { get; set; }

        [Display(Name = "Phone number", Prompt = "Contact phone number"), DataType(DataType.PhoneNumber), Required]
        public string Phone { get; set; }

        [Display(Name = "Overall Statisfaction")]
        public string Satisfaction { get; set; }
    }
```

Aplicação Web

Definindo a visão Feedback

Agora iremos definir uma nova visão baseado no modelo criado anteriormente. Inicialmente, será um form simples, e então, posteriormente serão adicionados mais campos. Então, você irá criar um link para a página e um controle para manipulá-lo.

1. No Solution Explorer, expanda a pasta Views. Clique de direita na pasta Home e selecione Add → View links. Entre com Feedback para o name, e selecione FeedbackModel, selecione “Create a strongly-typed view” como mostra a figura na página seguinte.

Aplicação Web

2. Assegure que Razor esteja selecionado.
3. A nova view é gerada com uma simples div.

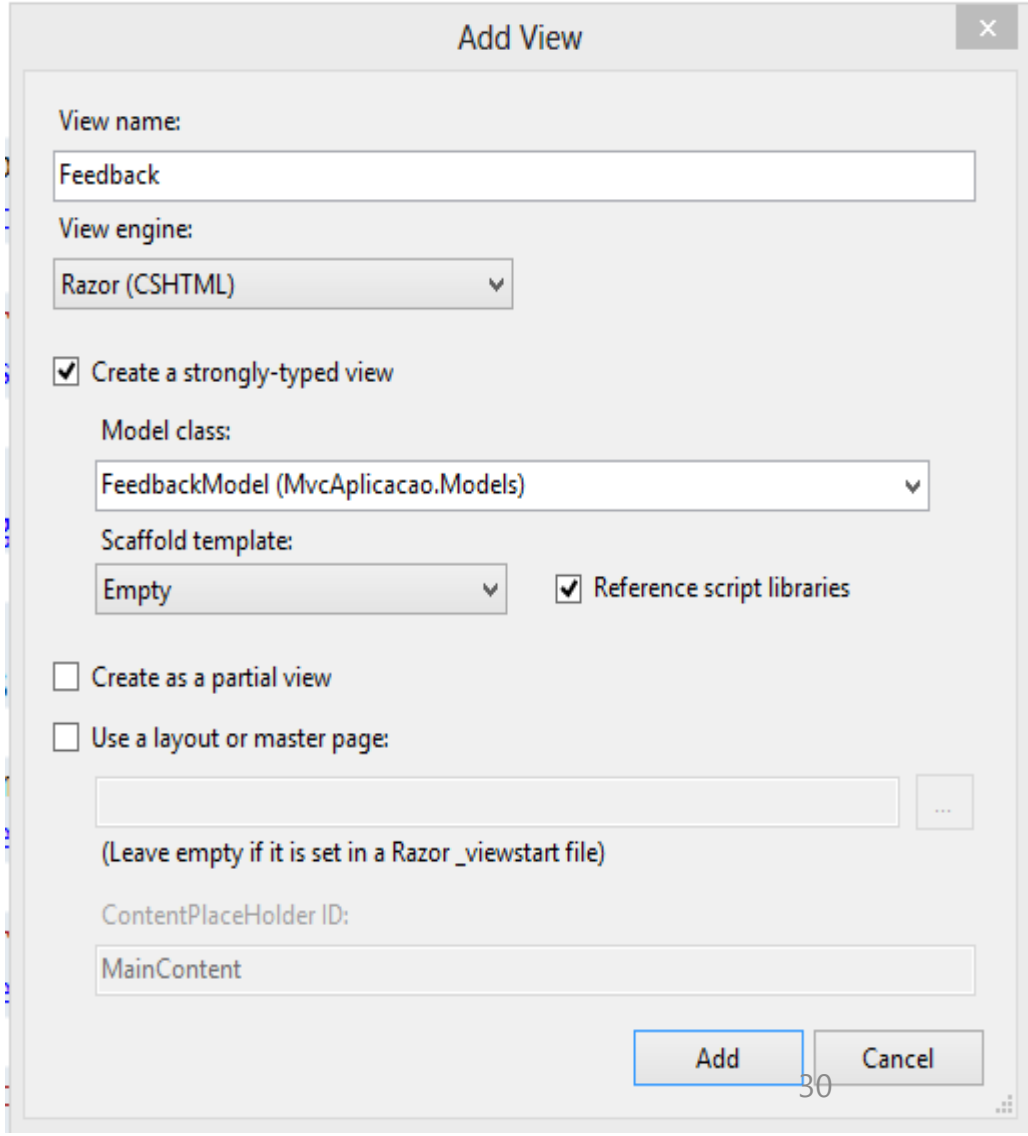
```
@model MvcAplicacao.Models.FeedbackModel

@{
    Layout = null;
}

<!DOCTYPE html>

<html>
<head>
    <meta name="viewport" content="width=device-width" />
    <title>Feedback</title>
</head>
<body>
    <div>

    </div>
</body>
</html>
```



Aplicação Web

Definindo o formulário

```
@model MvcAplicacao.Models.FeedbackModel

@{
    Layout = null;
}

<!DOCTYPE html>

<html>
<head>
    <meta name="viewport" content="width=device-width" />
    <title>Feedback</title>
</head>
<body>
    <div>
        @using (Html.BeginForm((string)ViewBag.FormAction, "Home"))
        {
            <fieldset>
                <legend>Feedback</legend>
                <div>
                    @Html.EditorFor(m => m.Email)
                    <p><input type="submit" value="Submit" /></p>
                </div>
            </fieldset>
        }
    </div>
</body>
</html>
```

Aplicação Web

1. Views são invocadas por um controller, assim você irá necessitar adicionar um controller action que irá carregar a página.
2. Abra a classe HomeController.cs, que você encontra na pasta Controllers.
3. Adicione o seguinte método.

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

4. Finalmente, você necessita de um link que dispara este controller action. Abra o arquivo _Layout.cshtml na pasta View\Shared.

5. Adicione o seguinte código.

```
<li>@Html.ActionLink("Feedback","Feedback","Home")</li>
```

6. Salve e execute o projeto.