

# **Programando com ASP.NET MVC**

**Aprenda a desenvolver aplicações web  
utilizando a arquitetura MVC**

**Alfredo Lotar**

Copyright© 2011 da Novatec Editora Ltda.

Todos os direitos reservados e protegidos pela Lei 9610 de 19/02/1998.

É proibida a reprodução desta obra, mesmo parcial, por qualquer processo, sem prévia autorização, por escrito, do autor e da Editora.

Editor: Rubens Prates

Editoração eletrônica: Camila Kuwabata e Carolina Kuwabata

Revisão gramatical: Débora Facin

Capa: Victor Bittow

ISBN: 978-85-7522-283-6

Histórico de impressões:

Outubro/2011      Primeira edição

Novatec Editora Ltda.

Rua Luís Antônio dos Santos 110

02460-000 – São Paulo, SP – Brasil

Tel.: +55 11 2959-6529

Fax: +55 11 2950-8869

Email: [novatec@novatec.com.br](mailto:novatec@novatec.com.br)

Site: [www.novatec.com.br](http://www.novatec.com.br)

Twitter: [twitter.com/novateceditora](https://twitter.com/novateceditora)

Facebook: [facebook.com/novatec](https://facebook.com/novatec)

LinkedIn: [linkedin.com/in/novatec](https://linkedin.com/in/novatec)

# Introdução ASP.NET MVC

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

O ASP.NET MVC implementa o pattern 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.

Observe a figura 1.1:

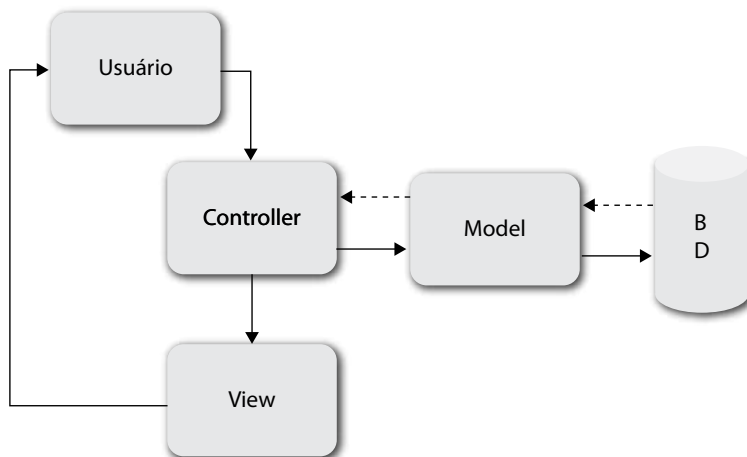


Figura 1.1 – Relação entre o usuário, o controller, o model e o view.

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 o model. O model acessa o banco de dados. Ao final, o conteúdo do view é retornado ao usuário.

Obs.: no decorrer deste livro usamos a palavra controlador em vez de controller. As palavras model e view não foram traduzidas.

## 1.1 Para quem é este livro?

Bem, este não é um livro para iniciantes em programação, ou seja, é necessário conhecimento prévio de C# e/ou ASP.NET Web Forms.

Neste livro não abordamos a sintaxe C#, o ASP.NET Web Forms e as classes do .NET Framework, pois o foco do livro é especificamente o ASP.NET MVC.

Para testar os exemplos abordados no livro, baixe os arquivos de exemplo no website da Novatec: <http://www.novatec.com.br/downloads.php>

Obs.: ASP.NET Web Forms é a denominação usada para o ASP.NET tradicional, com base em formulários e controles. Este nome visa apenas a diferenciá-lo do ASP.NET MVC.

## 1.2 Por que outro ASP.NET?

Porque o ASP.NET Web Forms apresenta uma série de limitações e problemas, por exemplo:

- Gera páginas muito grandes, afetando o carregamento das páginas e o tráfego da rede.
- Temos pouco controle sobre o HTML gerado.
- Os Web Server Controls são processados no servidor e podem afetar o desempenho da aplicação.
- Dificuldade em realizar testes na aplicação.
- Não tem real separação entre o código e o design.

## 1.3 ASP.NET MVC é difícil?

Alguns programadores com quem mantenho contato pela Internet afirmam: o ASP.NET MVC é muito difícil comparado ao ASP.NET Web Forms. Na minha opinião o ASP.NET MVC não é difícil, mas diferente. Por exemplo, o ASP.NET Web Forms é semelhante ao modelo de programação usado pelo Visual Basic ou Delphi, ao qual a maioria dos programadores atuais está acostumado.

No ASP.NET MVC você programa métodos e não eventos. Há também separação entre o código que interage com o usuário, o design e a camada de dados. Tudo isso parece estranho para programadores acostumados a programar eventos em formulários.

Outros programadores afirmam que têm dificuldades em aprender ASP.NET MVC. Acredito que a dificuldade no aprendizado se deve ao fato de a maioria do conteúdo estar em inglês ou em pequenos tutoriais em português. No momento em que escrevo este texto – início de junho de 2011 –, há poucos livros em português sobre ASP.NET MVC.

## 1.4 Quando usar o ASP.NET Web Forms

A seguir listamos alguns motivos para você continuar usando o ASP.NET Web Forms:

- Quando você precisa rapidamente de um recurso visualmente sofisticado, como os obtidos com o controle `GridView`, `DataList`, `ListView` OU `Repeater`.
- Quando há necessidade de manter o estado da página entre requisições.
- Quando for necessário ligar um controle `SqlDataSource` a uma origem de dados.
- Quando sentir saudade do recurso de arrastar e soltar do Visual Studio.

## 1.5 Quando usar o ASP.NET MVC

Alguns motivos para você adotar o ASP.NET MVC:

- Quando você precisa de controle total sobre o HTML.
- Quando há necessidade de unidades de teste no projeto.
- Quando a aplicação necessita de separação entre o design, o código e a camada de dados.
- Quando há obrigação de reduzir o tamanho das páginas geradas.
- Quando é preciso eliminar ou reduzir os Postbacks.
- Quando uma equipe grande desenvolve uma aplicação. Cada um pode-se dedicar a uma parte (controlador, model e view) da aplicação.
- Quando é necessário estender a aplicação constantemente.
- Quando a aplicação requer múltiplas interfaces. Por exemplo, você pode criar um view que exibe uma página HTML e outro que exibe uma página no estilo Silverlight ou em um formato para dispositivos móveis.
- Quando você não se sentir confortável desenvolvendo com formulários. Geralmente, programadores não gostam de fazer o design da aplicação.
- Quando os designers da empresa estiverem com pouco trabalho e você quer que eles criem todas as páginas manualmente, sem os característicos recursos de arrastar e soltar. Maldade, né?

## 1.6 Os dois ASP.NET funcionam juntos?

Sim, aproveite o melhor de cada um. Por exemplo, uma aplicação ASP.NET MVC pode exibir informações em um controle `DataList` do ASP.NET Web Forms.

## 1.7 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 servidor.
- Definição de filtros de ação.
- Facilidade em implementar aplicações Ajax.
- Uso da sintaxe Razor.
- Uso intensivo de atributos.
- Implementação e criação de HTML helpers.
- A implementação de unidades de teste.
- Implementa o pattern MVC.

## 1.8 Prepare seu computador

Os exemplos deste livro podem ser desenvolvidos com o Visual Studio 2010 ou Visual Web Developer 2010 Express.

*<http://www.microsoft.com/netframework>*

*<http://www.microsoft.com/visualstudio>*

*<http://www.microsoft.com/express/downloads>*

Baixe e instale também o ASP.NET MVC. Acesse o website *<http://www.asp.net/mvc>*

Obs.: nos arquivos de exemplo, você encontra uma lista com todos os links indicados no livro. Assim, não há necessidade de digitá-los.

### 1.8.1 Preparando o IIS

Após a instalação dos softwares anteriores, instale o Internet Information Services (IIS).

Se você planeja usar seu computador somente para testar os exemplos deste livro, não é necessário instalá-lo. Nesse caso, use o ASP.NET Development Server, o qual é instalado com o Visual Studio 2010 ou Visual Web Developer 2010 Express.

Uma aplicação ASP.NET MVC roda no contexto da máquina local sob a conta AUTORIDADE NT\SERVIÇO DE REDE.

## 1.8.2 Acessando uma aplicação ASP.NET MVC

Os arquivos da aplicação web acessados pelo navegador são disponibilizados pelo servidor web – o servidor IIS. Todos os arquivos da aplicação ASP.NET MVC acessados pelo navegador devem fazer parte de um diretório virtual.

Para testar a aplicação web no seu computador, recomenda-se empregar a seguinte sintaxe:

```
http://[nomedoservidor]/[DiretórioVirtual]/[método]
```

Exemplo:

```
http://alfredo/livro/
```

ou

```
http://localhost/livro/
```

```
http://localhost/livro/capitulo1
```

de modo que `livro` é o diretório virtual, e `capitulo1`, o subdiretório. Somente o diretório-raiz, no caso `livro`, precisa ser virtual.

Obs.: navegador é o termo usado neste livro para nomear softwares de exibição de páginas de websites como o Internet Explorer, o Firefox, o Opera, o Google Chrome etc.

## 1.8.3 Criando um diretório virtual

- Abra o **Windows Explorer**, uma vez que um diretório virtual pode ser criado facilmente nele.
- Navegue até o diretório-raiz da aplicação web.
- Clique com o botão direito no diretório escolhido e, em seguida, clique em **propriedades**.
- Selecione a aba **Compartilhamento da web** e, em seguida, selecione **compartilhar essa pasta**.
- Defina um alias para a pasta. O alias é o nome do diretório virtual que será usado na URL do navegador. Observe a figura 1.2.

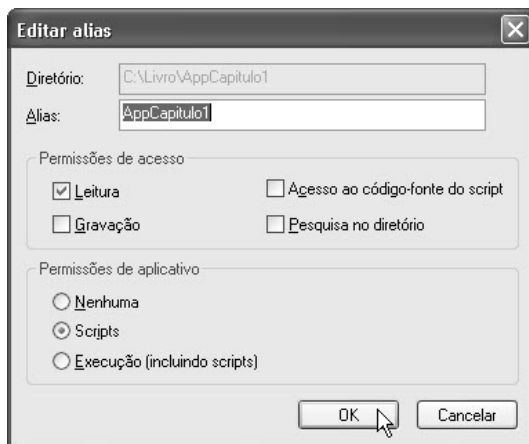


Figura 1.2 – Criando um diretório virtual.

## 1.9 Desenvolvendo uma nova aplicação

Inicie o Visual Studio 2010 ou o Visual Web Developer 2010 Express. Acesse o menu **File** e clique em **New Project ...**.

Na caixa de diálogo **New Project** selecione **Visual C# e ASP.NET MVC 3 Web Application**. Nomeie o projeto como `AppCapitulo1`. Clique em **OK**. Conforme figura 1.3.

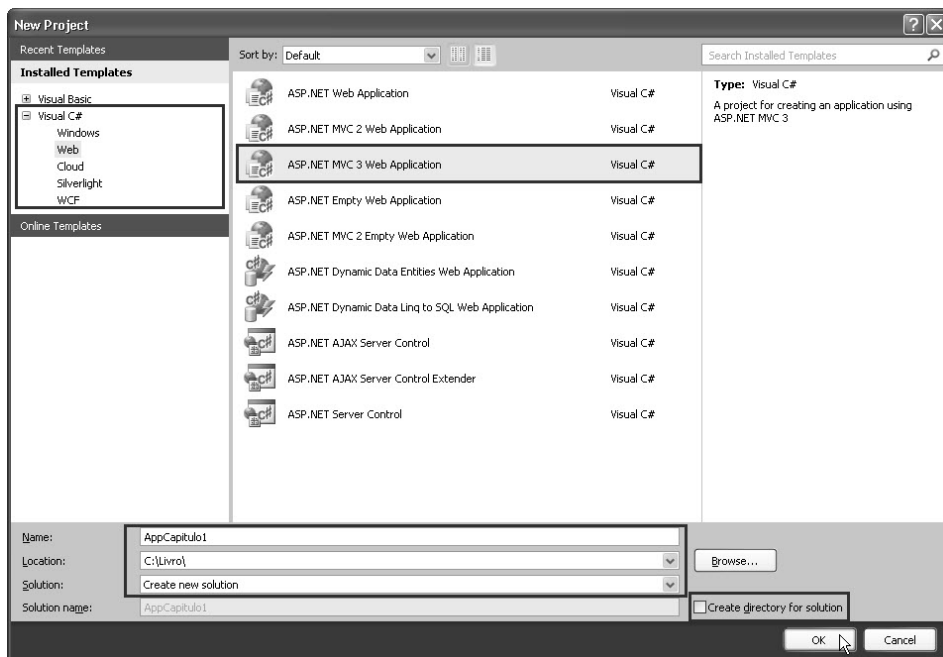


Figura 1.3 – Criando uma nova aplicação ASP.NET MVC.

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

- Empty contém a estrutura básica de arquivos e diretórios usados por uma aplicação ASP.NET MVC.
- Internet Application contém além dos recursos do template 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.

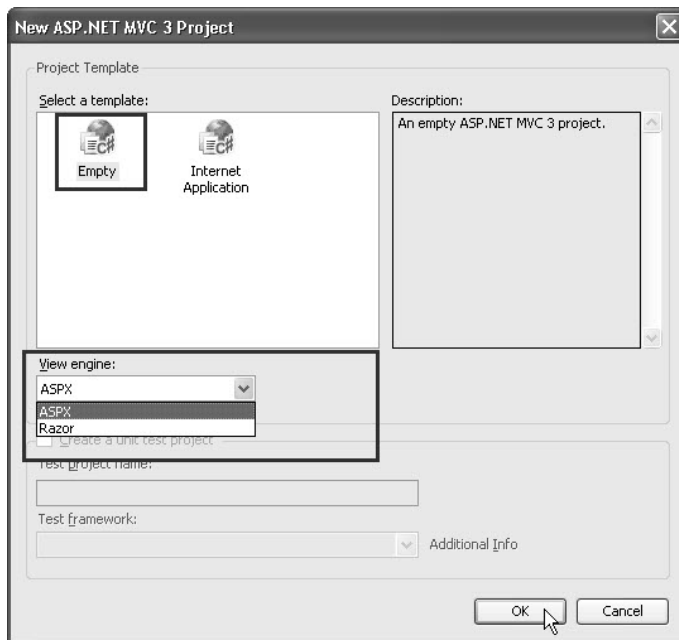


Figura 1.4 – Definindo o template e o mecanismo de exibição da aplicação.

Obs.: o Visual Studio 2010 Professional ou superior permite a criação de uma unidade de teste.

Quando criamos uma nova aplicação com o template Empty, é automaticamente adicionada a seguinte estrutura de arquivos e diretórios. Observe a figura 1.5.



Figura 1.5 – Estrutura de arquivos e diretórios do website ASP.NET MVC.

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.   |

A figura 1.6 lista os arquivos de cada diretório da aplicação:

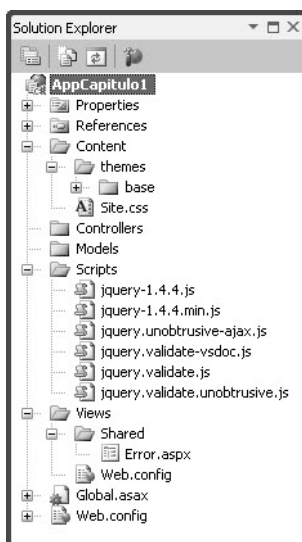


Figura 1.6 – Arquivos e diretórios da aplicação ASP.NET MVC.

A figura 1.6 exibe somente alguns arquivos do diretório Scripts.

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.  |

## 1.10 Executando a aplicação

Para executar a aplicação, pressione **F5** ou acesse **Start Debugging (F5)** na barra de ferramentas do Visual Studio 2010 ou Visual Web Developer 2010 Express. Observe a figura 1.7:

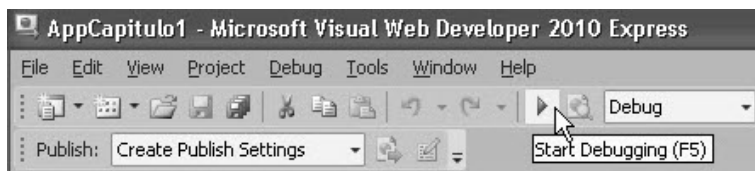


Figura 1.7 – Executando a aplicação.

Em seguida, é gerada a saída exibida na figura 1.8.



Figura 1.8 – Saída exibe uma página inválida.

Bem, esse erro acontece porque não temos nenhuma informação ou página para exibir no navegador. Para resolver esse problema adicionamos um controlador ao projeto.

## 1.11 Controladores

Os controladores MVC manipulam e respondem às entradas e interações do usuário e também interagem com as outras partes da aplicação, como views e modelos.

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 1.9.

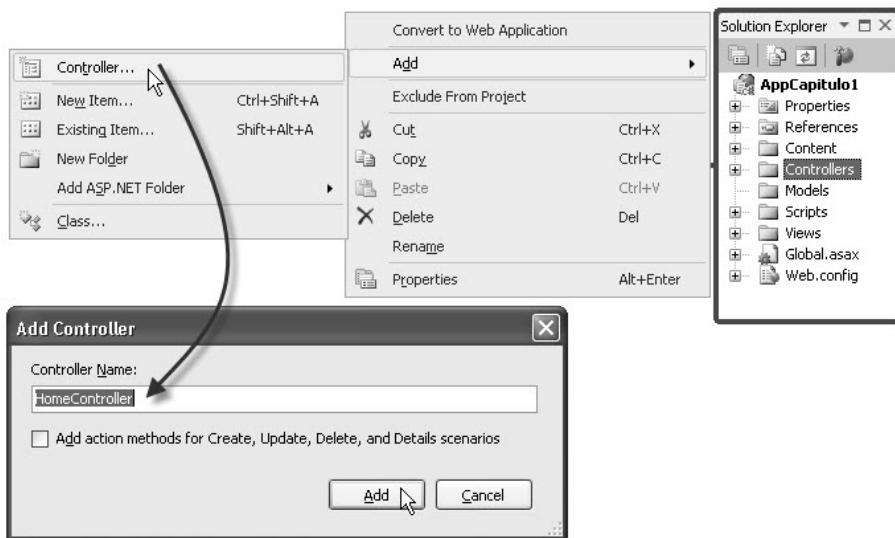


Figura 1.9 – Adicionando um novo controlador.

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/Home/Details/5>

<http://localhost:1573/Home/Edit/5>

Obs.: os action method são denominados também métodos de controlador.

Quando clicamos no botão **Add** da caixa de diálogo **Add Controller**, o Visual Studio 2010 ou Visual Web Developer 2010 Express acrescentam 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() {
            return View();
        }
    }
}
```

Pressione **F5**. A aplicação retorna novamente uma página de erro, pois ainda não associamos nenhum view – responsável pela exibição do conteúdo ao usuário.

Para resolver o problema, substitua o trecho de código:

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

Por:

```
public ActionResult Index() {
    return Content("<h1>01á mundo!</h1>");
}
```

Se preferir use:

```
public string Index() {
    return "<h1>01á mundo!</h1>";
}
```

Ou:

```
public void Index() {
    Response.Write("<h1>01á mundo!</h1>");
}
```

Pressione **F5** novamente. O resultado vemos na figura 1.10.

O mesmo resultado é obtido quando digitamos:

`http://localhost:1029/`

`http://localhost:1029/Home`

`http://localhost:1029/Home/index`



Figura 1.10 – Nosso primeiro exemplo em ação.

O ASP.NET MVC usa roteamento de URLs e as regras são registradas no método `RegisterRoutes` dentro do arquivo `Global.asax`:

```
public static void RegisterRoutes(RouteCollection routes) {
    routes.IgnoreRoute("{resource}.axd/{*pathInfo}");
    routes.MapRoute(
        "Default", // Route name
        "{controller}/{action}/{id}", // URL with parameters
        new { controller = "Home", action = "Index", id = UrlParameter.Optional }
    );
}
```

`controller` é o nome da classe do controlador. `action` é o nome do método invocado. E `id` é o parâmetro opcional embutido na URL e usado para passar argumentos para o método.

O formato de URL `{controller}/{action}/{id}` registrado no `Global.asax` permite as seguintes variações de URLs:

`http://localhost:1029/`

`http://localhost:1029/Home`

`http://localhost:1029/Home/index`

`http://localhost:1029/Home/Create`

`http://localhost:1029/Home/Details/5`

`http://localhost:1029/Home/Edit/25`

Obs.: baixe os arquivos de exemplo no site da Novatec Editora, assim você não precisa digitar grandes quantidades de código.

### 1.11.1 Método com parâmetros

Acrescente o método Details à classe HomeController:

```
public ActionResult Details(int id) {  
    return Content("<h1>Olá mundo com id = " + id + "</h1>");  
}
```

Exemplo:

```
using System;  
using System.Collections.Generic;  
using System.Linq;  
using System.Web;  
using System.Web.Mvc;  
  
namespace AppCapitulo1.Controllers {  
    public class HomeController : Controller {  
        public ActionResult Index() {  
            return Content("<h1>Olá mundo!</h1>");  
        }  
  
        public ActionResult Details(int id) {  
            return Content("<h1>Olá mundo com id = " + id + "</h1>");  
        }  
    }  
}
```

Quando pressionamos **F5** ou digitamos no navegador a URL:

`http://localhost:NúmeroDaPorta/Home/Details/5`

É retornada a página exibida na figura 1.11:

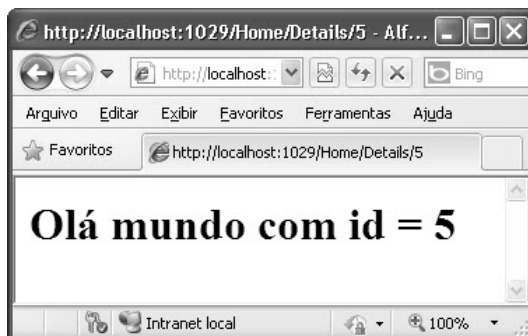


Figura 1.11 – URL com parâmetros.

Obs.: 1029 é o número da porta usada pelo ASP.NET Development Server. Esse valor não é fixo. Ao pressionar **F5** no seu computador o número da porta será outro. Altere o número da porta ou coloque a aplicação num diretório virtual e digite: `http://localhost/nomediretóriovirtual/Home/Details/5`

## 1.12 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 1.12:

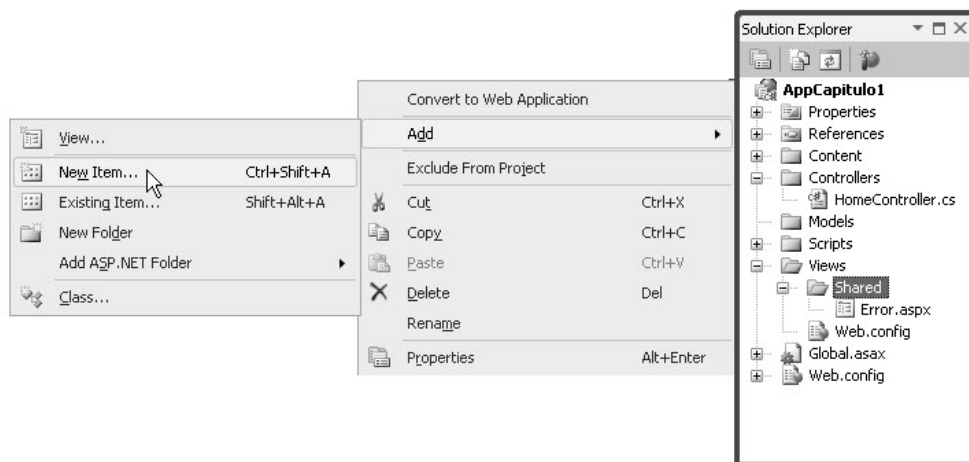


Figura 1.12 – Acrescentando uma nova master page.

Na caixa de diálogo **Add New Item** selecione **MVC 3 View Master Page (ASPX)**. Nomeie como Site.Master. Clique em **Add**. Observe a figura 1.13.

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>
```

Um controle `ContentPlaceholder` define regiões da página que não têm conteúdo comum a todas as páginas. Por exemplo, os produtos mais vendidos.

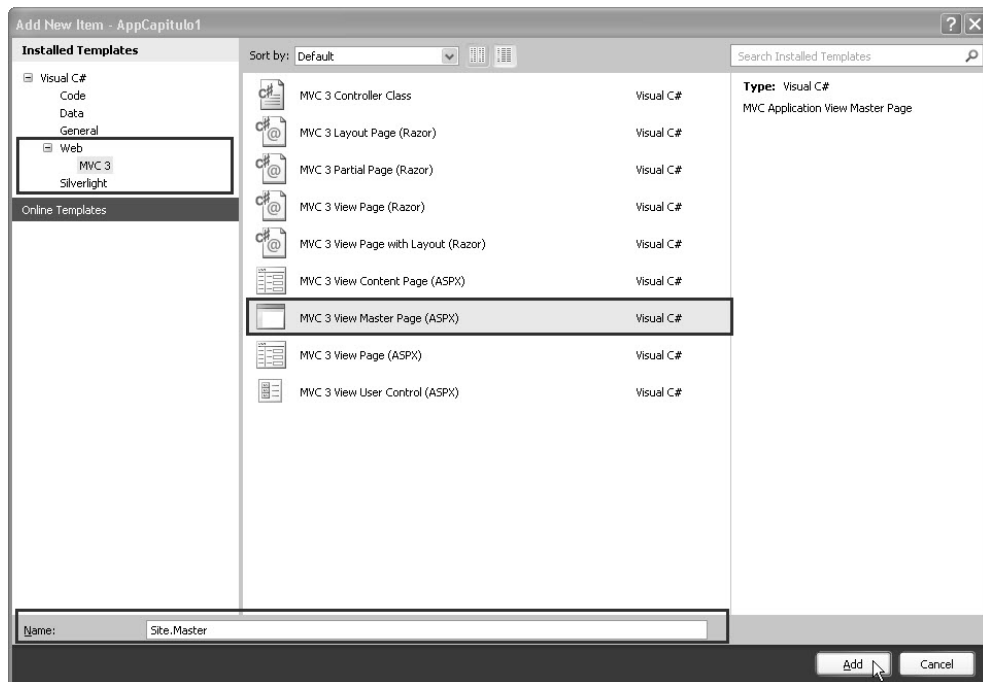


Figura 1.13 – Caixa de diálogo Add New Item.

Para formatar o website usamos folhas de estilo CSS. Acrescente à tag `head` da master page uma referência para a tag `link`:

```
<head runat="server">
    <link href="/Content/Site.css" rel="stylesheet" type="text/css"/>
    <title><asp:ContentPlaceholder ID="TitleContent" runat="server"/></title>
</head>
```

O arquivo `Site.css` é adicionado ao diretório `Content` quando criamos um novo projeto ASP.NET MVC e possui os seguintes seletores CSS:

```
body {
    font-size: 75%;
    font-family: Verdana, Tahoma, Arial, "Helvetica Neue", Helvetica, Sans-Serif;
    color: #232323;
    background-color: #fff;
}

/* Estilos para formulários
-----*/
fieldset {
    border:1px solid #ddd;
```

```
padding:0 1.4em 1.4em 1.4em;
margin:0 0 1.5em 0;
}
legend {
font-size:1.2em;
font-weight: bold;
}
textarea {
min-height: 75px;
}
.editor-label {
margin: 1em 0 0 0;
}
.editor-field {
margin:0.5em 0 0 0;
}
/* Estilos para validação
-----*/
.field-validation-error {
color: #ff0000;
}
.field-validation-valid {
display: none;
}
.input-validation-error {
border: 1px solid #ff0000;
background-color: #ffebee;
}
.validation-summary-errors {
font-weight: bold;
color: #ff0000;
}
.validation-summary-valid {
display: none;
}
```

### 1.12.1 Configurando a master page

Inicialmente, usamos a master page para formatar o topo das páginas da nossa aplicação ASP.NET MVC.

Adicione ao arquivo `Site.css` o trecho de código a seguir:

```

body {
    font-size: 75%;
    font-family: Verdana, Tahoma, Arial, "Helvetica Neue", Helvetica, Sans-Serif;
    color: #232323;
    background-color: #fff;
    margin-left: 0;
    margin-right: 0;
    margin-top: 0;
    margin-bottom: 0;
}

table {
    border-collapse: collapse;
    border: 0;
}

td {
    padding: 0;
}

```

Acrescente as tabelas a seguir ao arquivo Site.Master:

```

<table style="text-align: center; border: 0; width: 100%">
    <tr>
        <td style="background-color: #006486; height: 5px;" colspan="4">
        </td>
    </tr>
    <tr>
        <td style="height: 65px; width: 15px;">
            <strong>LOGO Website</strong>
        </td>
        <td style="text-align: right;">
            <table style="width: 100%; border: 0px;">
                <tr>
                    <td style="width: 100%; text-align: right; font-size: 40px; font-family: Arial;">
                    </td>
                </tr>
            </table>
        </td>
    </tr>
</table>
<table style="width: 100%; text-align: center; border: 0;">
    <tr>
        <td style="background-color: #006486; height: 5px;">
        </td>
    </tr>
</table>

```

## Exemplo:

```

<%@ Master Language="C#" Inherits="System.Web.Mvc.ViewMasterPage" %>
<!DOCTYPE html>
<html>
  <head runat="server">
    <link href="/Content/Site.css" rel="Stylesheet" type="text/css" />
    <title>
      <asp:ContentPlaceHolder ID="TitleContent" runat="server" />
    </title>
  </head>
  <body>
    <div>
      <table style="text-align: center; border: 0; width: 100%">
        <tr>
          <td style="background-color: #006486; height: 5px;" colspan="4">
          </td>
        </tr>
        <tr>
          <td style="height: 65px; width: 15px;">
            <strong>LOGO Website</strong>
          </td>
          <td style="text-align: right;">
            <table style="width: 100%; border: 0px;">
              <tr>
                <td style="width: 100%; text-align: right; font-size: 40px; font-family: Arial;">
                </td>
              </tr>
            </table>
          </td>
        </tr>
      </table>
      <table style="width: 100%; text-align: center; border: 0;">
        <tr>
          <td style="background-color: #006486; height: 5px;">
          </td>
        </tr>
      </table>
      <asp:ContentPlaceHolder ID="MainContent" runat="server">
      </asp:ContentPlaceHolder>
    </div>
  </body>
</html>

```

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>" %>
```

A figura 1.14 mostra a master page `Site.Master` em ação:

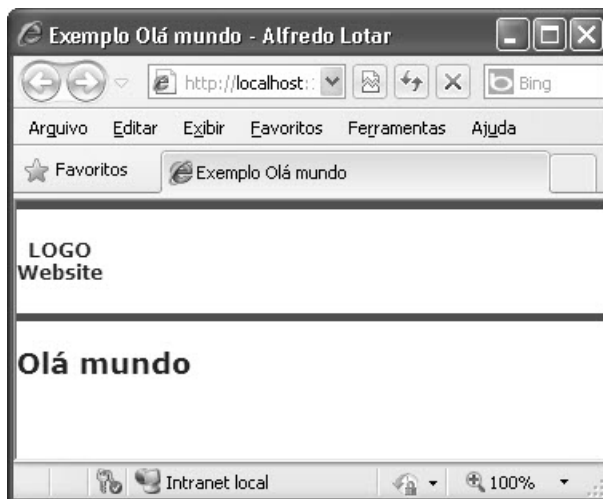


Figura 1.14 – Cabeçalho da página formatada pela master page.

Obs.: baixe os arquivos de exemplo no site da Editora Novatec e, em seguida, copie o código anteriormente mencionado.

## 1.13 Views

View ou exibição – é responsável pela apresentação do conteúdo HTML da página. Geralmente, é definida por intermédio de arquivos `.aspx`, `.ascx`, `.master`, `.cshtml`.

Criar uma exibição a partir de um método de controlador MVC. É o modo mais fácil, rápido e simples.

Clique com o botão direito do mouse no nome do método e, em seguida, clique em **Add View...** Observe a figura 1.15.

Na caixa de diálogo **Add View** defina o nome do view e também o mecanismo de exibição – `ASPX` no nosso caso. Marque a caixa de seleção com a legenda "Use a layout or master page:" e determine o nome da master page. Observe a figura 1.16.

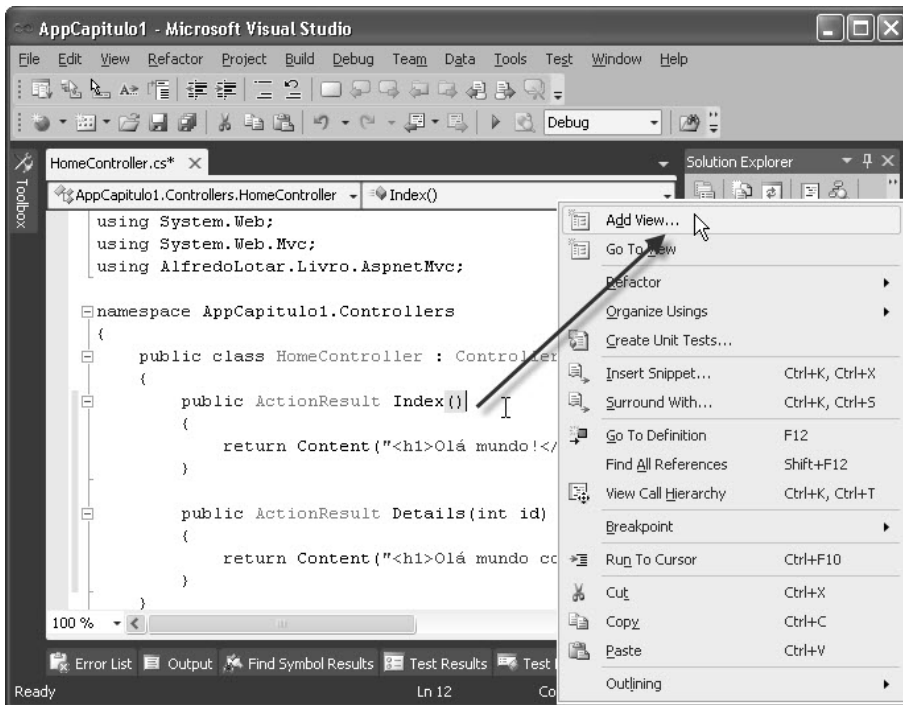


Figura 1.15 – Adicionando um novo View.

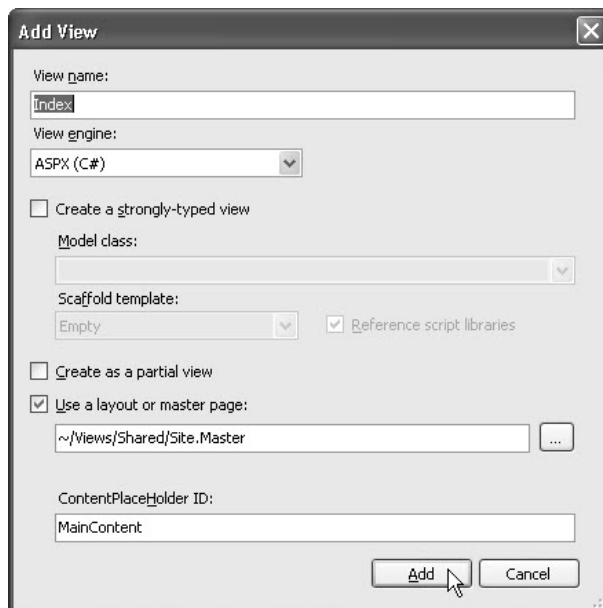


Figura 1.16 – Caixa de diálogo Add View.

O novo view é acrescentado ao subdiretório Home do diretório Views. Observe a figura 1.17:

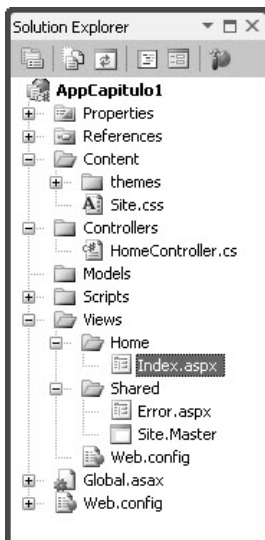


Figura 1.17 – View Index.aspx.

Quando abrimos o arquivo Index.aspx vemos o seguinte conteúdo:

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

<asp:Content ID="Content1" ContentPlaceHolderID="TitleContent" runat="server">
    Index
</asp:Content>

<asp:Content ID="Content2" ContentPlaceHolderID="MainContent" runat="server">
    <h2>Index</h2>
</asp:Content>
```

### 1.13.1 Exibindo informações num view

Normalmente, passamos informações de um método de controlador para um view por intermédio da propriedade ViewBag. Cujas sintaxe é:

```
ViewBag.QualquerNome = Conteúdo;
```

Exemplo:

```
ViewBag.Nome = "Alfredo Lotar";
ViewBag.Editora = "Novatec";
ViewBag.Idade = 37;
ViewBag.Data = DateTime.Now;
```

Para exibir a mensagem "ASP.NET MVC!" na tela, acrescentamos ao método `Index` da classe `HomeController` a seguinte linha:

```
ViewBag.Mensagem = "ASP.NET MVC!";
```

Exemplo:

```
using System.Web.Mvc;
namespace AppCapitulo1.Controllers {
    public class HomeController : Controller {
        public ActionResult Index() {
            ViewBag.Mensagem = "ASP.NET MVC!";
            return View();
        }

        public ActionResult Details(int id) {
            return View();
        }
    }
}
```

Em seguida, adicionamos a linha a seguir:

```
<%=ViewBag.Mensagem%>
```

Ao arquivo `Index.aspx`. Exemplo:

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

<asp:Content ID="Content1" ContentPlaceHolderID="TitleContent" runat="server">
    Index
</asp:Content>

<asp:Content ID="Content2" ContentPlaceHolderID="MainContent" runat="server">
    <h2><%=ViewBag.Mensagem%></h2>
</asp:Content>
```

A saída no ASP.NET MVC é gerada pela expressão inline: `<%= expressão %>`

Obs.: as versões 1 e 2 do ASP.NET MVC usam a expressão `<%= expressão %>` em vez de `<%= expressão %>`.

### 1.13.2 Acrescentando um link ao view

O modo mais usado para linkar páginas no ASP.NET MVC é por intermédio do método `ActionLink`. Exemplo:

```
<%=Html.ActionLink("Volta para página inicial", "Index") %>
```

O primeiro parâmetro do método `ActionLink` define o texto do link; o segundo, o nome do método que deve ser executado; o terceiro parâmetro opcional embutido na URL é usado para passar argumentos para o método.

Para exemplificar, altere o método `Details` da classe `HomeController` para:

```
public ActionResult Details(int id) {
    ViewBag.Mensagem = "Detalhes id = " + id;
    return View();
}
```

Em seguida, associe um novo view para o método `Details`. Observe a figura 1.18.

Na página `Details.aspx` acrescente as linhas a seguir:

```
<h2>
    <%:ViewBag.Mensagem%>
</h2>
<%:Html.ActionLink("Volta para página inicial", "Index") %>
```

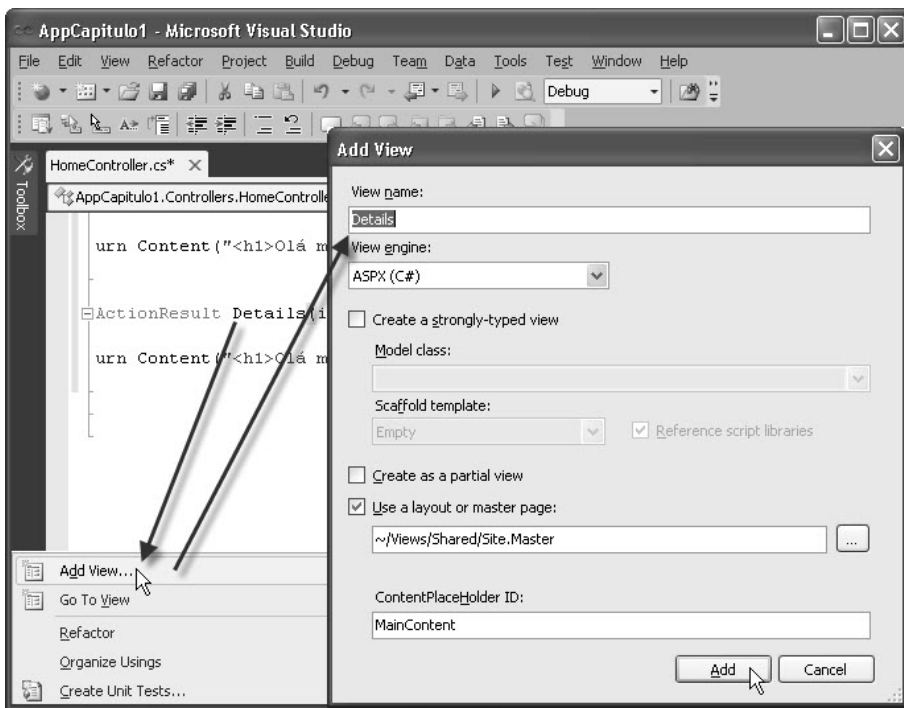


Figura 1.18 – Criando um novo View.

Exemplo:

```
<%@ Page Title="" Language="C#" MasterPageFile="~/Views/Shared/Site.Master"
    Inherits="System.Web.Mvc.ViewPage<dynamic>" %>
<asp:Content ID="Content1" ContentPlaceHolderID="TitleContent" runat="server">
    Detalhes
</asp:Content>
<asp:Content ID="Content2" ContentPlaceHolderID="MainContent" runat="server">
    <h2>
        <%=ViewBag.Mensagem%>
    </h2>
    <%=Html.ActionLink("Volta para página inicial", "Index") %>
</asp:Content>
```

E no arquivo Index.aspx acrescente a linha:

```
<%=Html.ActionLink("Página detalhes", "Details", new {id = 5}) %>
```

Navegue entre as páginas e veja os links funcionando. Observe a figura 1.19:

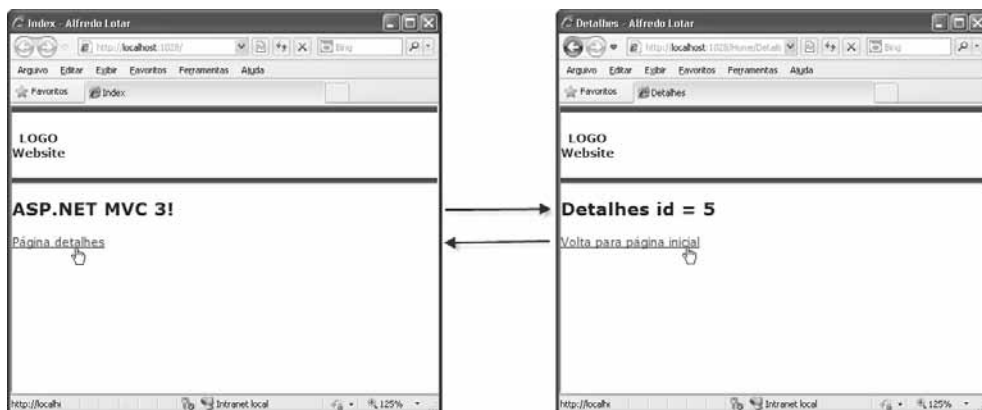


Figura 1.19 – Testando links desenvolvidos com o método ActionLink.

## 1.14 Models

O model contém as classes da camada de dados, ou seja, contém as rotinas de acesso e manipulação de dados provenientes de diferentes origens de dados, por exemplo, banco de dados SQL Server, Oracle, MySQL, documentos XML etc.

Os arquivos da camada de dados são armazenados no diretório Models.

Para facilitar a manutenção, os testes e evitar a duplicação de códigos, divida, por exemplo, a camada de dados em: um ou mais objetos Entity Framework e os manipule por intermédio de uma classe e uma interface. No capítulo 4 abordaremos o assunto novamente.

### 1.14.1 Acrescentando uma nova classe

Para ilustrar o que foi abordado até o momento, criamos uma nova classe no diretório Models, conforme a figura 1.20:

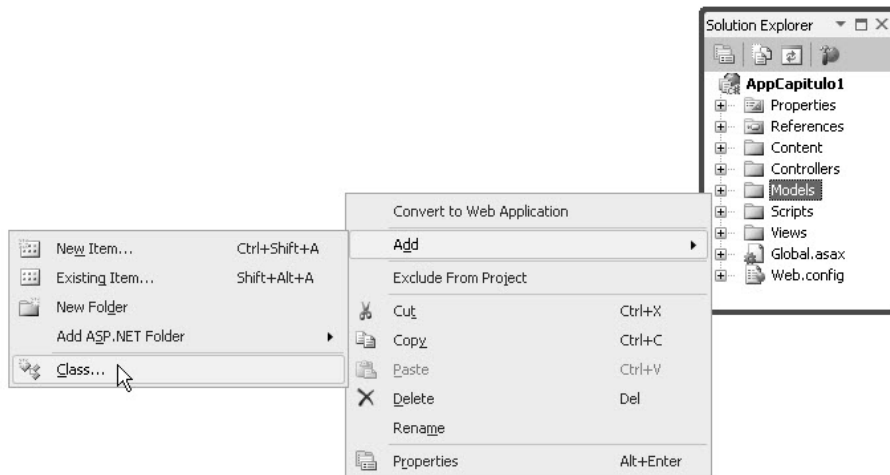


Figura 1.20 – Acrescentando uma nova classe.

Em seguida, na caixa de diálogo **Add New Item** definimos o nome da classe. Observe a figura 1.21:

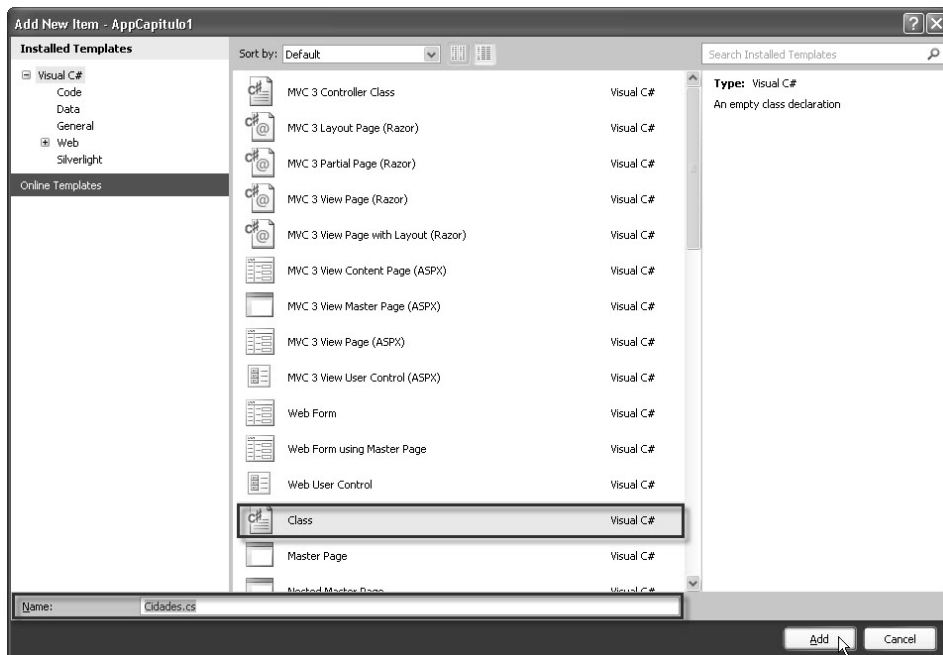


Figura 1.21 – Caixa de diálogo Add New Item.

Ao clicar no botão **Add** da caixa de diálogo **Add New Item** é acrescentado ao diretório `Models` o arquivo `Cidades.cs` com o seguinte código:

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

namespace AppCapitulo1.Models {
    public class Cidades {
    }
}
```

O passo seguinte é alterar o código gerado. Alteramos o nome da namespace de:

```
namespace AppCapitulo1.Models {
```

Para:

```
namespace AlfredoLotar.Livro.AspnetMvc {
```

E acrescentamos três propriedades:

```
public int CidadeID { get; set; }
public string Nome { get; set; }
public string Estado { get; set; }
```

A seguir temos o código da classe `Cidades` com as alterações:

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

namespace AlfredoLotar.Livro.AspnetMvc {
    public class Cidades {
        public int CidadeID { get; set; }
        public string Nome { get; set; }
        public string Estado { get; set; }
    }
}
```

Antes de acessar os membros da classe `Cidades` é preciso compilá-la. Observe a figura 1.22:

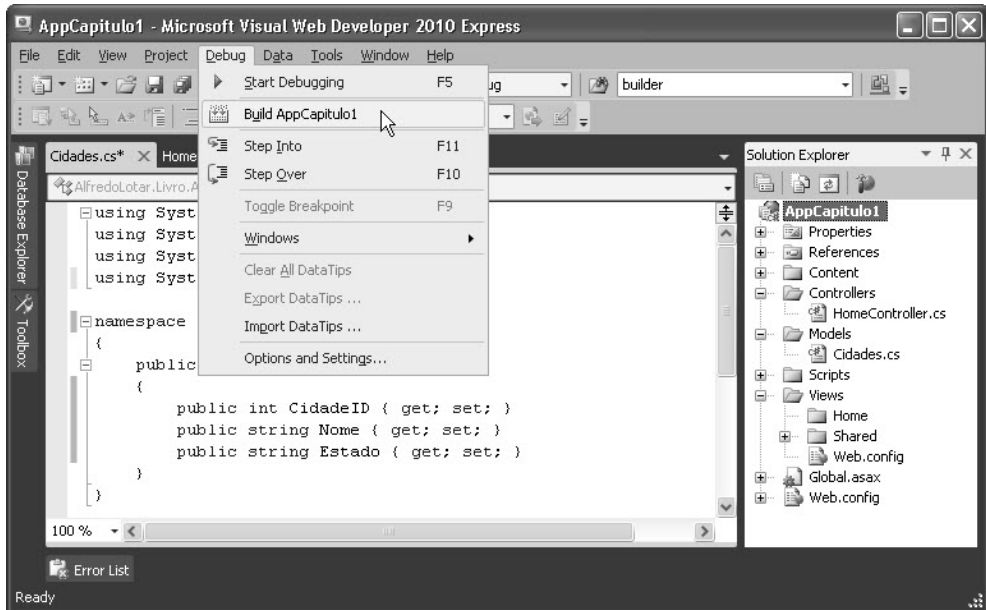


Figura 1.22 – Compilando a aplicação a partir do menu Debug.

### 1.14.2 Criando uma instância da classe Cidades

O próximo passo do nosso exemplo é a criação de uma instância da classe `Cidades` em um método de controlador qualquer. Nomeamos o método como `Index`. Se preferir, use qualquer outro nome.

Exemplo:

```
using System.Web.Mvc;
using AlfredoLotar.Livro.AspNetMvc;

namespace AppCapitulo1.Controllers {
    public class HomeController : Controller {
        public ActionResult Index() {
            var model = new Cidades() {
                CidadeID=1,
                Nome="São Paulo",
                Estado="SP"
            };
            return View(model);
        }
    }
}
```

### 1.14.3 Acrescentando um view

Clique com o botão direito do mouse no método `Index` e, em seguida, clique em **Add View...** Na caixa de diálogo **Add View** marque a caixa de seleção **Create a strongly-typed view** e selecione na caixa de combinação **Model Class:** `Cidades (AlfredoLotar.Livro.AspNetMvc)`, conforme a figura 1.23:

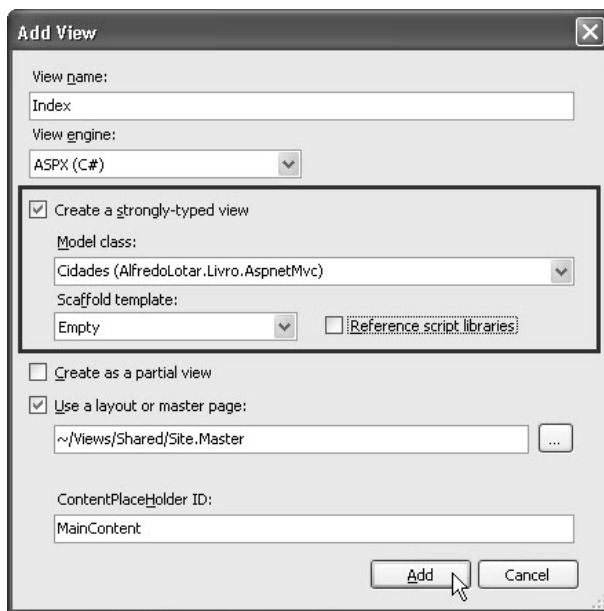


Figura 1.23 – Caixa de diálogo Add View.

Após clicar em **Add** é adicionado o arquivo `Index.aspx` ao diretório `Views\Home` com o seguinte código:

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

<asp:Content ID="Content1" ContentPlaceHolderID="TitleContent" runat="server">
    Index
</asp:Content>

<asp:Content ID="Content2" ContentPlaceHolderID="MainContent" runat="server">
    <h2>Index</h2>
</asp:Content>
```

A classe usada pelo view é declarada na diretiva `@ Page`:

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

No view, as propriedades da classe `Cidades` são acessadas por intermédio do objeto `Model`:

```
<%:Model.CidadeID %>  
<%:Model.Nome %>  
<%:Model.Estado %>
```

Exemplo:

```
<@ Page Title="" Language="C#" MasterPageFile="~/Views/Shared/Site.Master"  
    Inherits="System.Web.Mvc.ViewPage<AlfredoLotar.Livro.AspNetMvc.Cidades>" %>  
  
<asp:Content ID="Content1" ContentPlaceHolderID="TitleContent" runat="server">  
    Cidades  
</asp:Content>  
  
<asp:Content ID="Content2" ContentPlaceHolderID="MainContent" runat="server">  
    <h2>Cidades</h2>  
    <%:Model.CidadeID %>  
    <%:Model.Nome %>  
    <%:Model.Estado %>  
</asp:Content>
```

A saída no navegador você vê na figura 1.24:

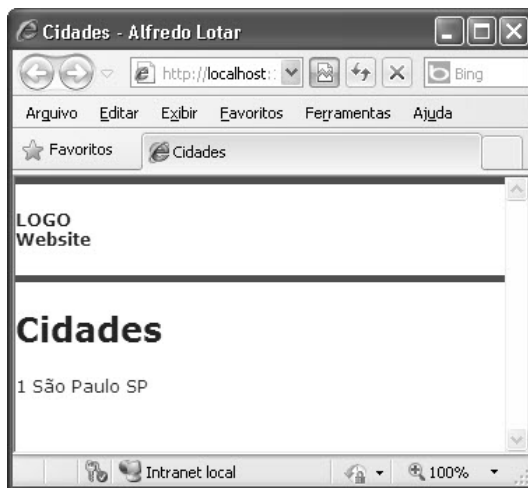


Figura 1.24 – Exemplo em ação.