

Visual Studio 2015

- ASP.NET com C#

Fundamentos



Visual Studio 2015

- ASP.NET com C#

Fundamentos



IMPACTA
EDITORADA

Créditos

Copyright © Monte Everest Participações e Empreendimentos Ltda.

Todos os direitos autorais reservados. Este manual não pode ser copiado, fotocopiado, reproduzido, traduzido ou convertido em qualquer forma eletrônica, ou legível por qualquer meio, em parte ou no todo, sem a aprovação prévia, por escrito, da Monte Everest Participações e Empreendimentos Ltda., estando o contrafator sujeito a responder por crime de Violação de Direito Autoral, conforme o art.184 do Código Penal Brasileiro, além de responder por Perdas e Danos. Todos os logotipos e marcas utilizados neste material pertencem às suas respectivas empresas.

"As marcas registradas e os nomes comerciais citados nesta obra, mesmo que não sejam assim identificados, pertencem aos seus respectivos proprietários nos termos das leis, convenções e diretrizes nacionais e internacionais."

Visual Studio 2015 - ASP.NET com C# Fundamentos

Coordenação Geral
Marcia M. Rosa

Revisão Ortográfica e Gramatical
Fernanda Monteiro Laneri

Coordenação Editorial
Henrique Thomaz Bruscagin

Diagramação
Bruno de Oliveira Santos

Atualização
José Eduardo Machado Grasso

Edição nº 1 | 1795_0
Junho/ 2016



Este material constitui uma nova obra e é uma derivação da seguinte obra original, produzida por TechnoEdition Editora Ltda., em Ago/2014: ASP.NET 2013 com C# - Fundamentos

Autoria: José Eduardo Machado Grasso

Informações sobre o treinamento	09
Capítulo 1 - Conceitos básicos	11
1.1. Introdução	12
1.2. Tipos de softwares	12
1.3. Arquitetura cliente-servidor	13
1.3.1. O servidor Web	13
1.3.2. O protocolo HTTP	14
1.3.2.1. Request	17
1.3.2.2. Response	18
1.4. Tecnologias utilizadas nas páginas Web	19
1.4.1. Exemplo de uma página retornada do servidor contendo HTML, CSS e JavaScript	20
1.5. Tecnologias utilizadas no servidor	21
1.6. Versões do ASP.NET	22
1.7. IIS – Internet Information Services	23
1.7.1. IIS – Criando uma aplicação ASP.NET	27
1.8. O Visual Studio	33
1.8.1. Usando o Visual Studio Community	34
1.8.1.1. Criando um projeto Web	34
1.8.1.2. Criando um Web site	39
1.8.1.3. Web project e Web site	42
Pontos principais	43
Teste seus conhecimentos	45
Mãos à obra!	49
Capítulo 2 - Web Pages	63
2.1. Tecnologias utilizadas nas páginas Web	64
2.1.1. A linguagem de marcação HTML	64
2.1.2. Linguagem CSS	68
2.1.3. Linguagem JavaScript	72
2.1.3.1. Noções básicas	72
2.1.3.2. Estrutura	73
2.2. Tecnologias utilizadas no servidor	75
2.3. Web Pages e Razor	75
2.3.1. Criando uma Web Page	76
2.3.2. Loops	78
2.3.3. Estruturas condicionais	80
2.3.4. Render e Layout	82
2.3.5. O caminho da renderização da página	87
2.3.6. Request e Form	87
2.3.7. Helpers	90
2.3.7.1. HTML	90
2.3.7.2. Chart	91
2.3.7.3. Crypto	93
Pontos principais	96
Teste seus conhecimentos	97
Mãos à obra!	101

Visual Studio 2015 - ASP.NET com C# Fundamentos

Capítulo 3 - Web Forms	121
3.1. Introdução	122
3.2. A construção de UIs	122
3.2.1. O processo de renderização	122
3.3. Controles	126
3.3.1. Controles de servidor HTML	126
3.3.2. Controles de servidor Web	127
3.3.2.1. Controles para exibir texto	129
3.3.2.2. Controles para exibir listas	129
3.3.2.3. Controles para exibir imagens	131
3.3.2.4. Controles para agrupar outros controles	131
3.3.3. Controles de validação	132
3.3.3.1. Validação pelo cliente e validação pelo servidor	136
3.3.3.2. Propriedades dos controles de validação	138
3.3.4. Controles de usuário	139
Pontos principais	141
 Teste seus conhecimentos.....	143
 Mãos à obra!	147
 Capítulo 4 - MVC.....	 169
4.1. Introdução	170
4.2. Modelos de desenvolvimento ASP.NET	170
4.3. ASP.NET MVC	171
4.4. Estrutura dos arquivos MVC	173
4.5. Routes	174
4.6. Controller	179
4.7. Views	181
4.7.1. ViewBag	184
4.8. HtmlHelper	185
4.8.1. ActionLink	185
4.8.2. Partial	186
4.8.3. BeginForm	186
4.9. Model e Views tipadas	189
4.10. GET e POST	192
4.11. Ciclo de requisição e resposta no MVC	196
Pontos principais	197
 Teste seus conhecimentos.....	199
 Mãos à obra!	203

Sumário

Capítulo 5 - Gerenciamento do estado da sessão	219
5.1. Introdução	220
5.1.1. Cookies	220
5.1.2. Session	220
5.2. Manipulando cookies	222
5.3. ViewState (Web Forms)	224
Pontos principais	227
Teste seus conhecimentos.....	229
Mãos à obra!	233
Capítulo 6 - Componentes.....	247
6.1. Introdução	248
6.2. NuGet	248
6.3. Bundle/Minification.....	249
6.3.1. Implementando o Bundle	251
6.3.2. Utilizando o Bundle	252
6.4. Bootstrap	258
6.5. FriendlyUrls	275
Pontos principais	280
Teste seus conhecimentos.....	281
Mãos à obra!	285
Capítulo 7 - Acesso a dados ADO.NET.....	301
7.1. Introdução	302
7.2. Providers	302
7.3. Principais classes de acesso a dados	303
7.4. Classes desconectadas.....	304
7.5. Conectando	304
7.6. Enviando comandos ao banco de dados	305
7.7. Lendo informações do banco	308
7.7.1. DbDataReader	309
7.7.2. DataSet	310
7.8. Exibindo informações	312
7.8.1. Web Forms: Usando Web Controls para exibir dados	313
7.8.1.1. GridView	313
7.8.1.2. DropDownList	314
7.8.2. MVC: Usando Views para exibir dados	315
7.8.3. Web Pages: Usando Database para exibir dados	317
Pontos principais	318
Teste seus conhecimentos.....	319
Mãos à obra!	323

Visual Studio 2015 - ASP.NET com C# Fundamentos

Capítulo 8 - Boas práticas em ASP.NET.....	359
8.1. Introdução	360
8.2. Web Controls (Web Forms) e HTML (MVC).....	360
8.2.1. Use Literal no lugar de Label	360
8.2.2. Use HtmlEncoding.....	361
8.2.3. Associe a legenda com o campo.....	362
8.2.4. Não use tabela para formulário	363
8.2.5. Use listas para menus	365
8.3. Performance	368
8.3.1. Desabilite o ViewState (Web Forms).....	368
8.3.2. Use o ViewState (Web Forms)	374
8.4. Código.....	378
8.4.1. Separar funcionalidades	378
8.4.1.1. Separar por métodos.....	378
8.4.1.2. Separar por classes	387
8.4.1.3. Separar por componente	390
8.4.2. Acesso a dados	392
8.4.2.1. String de conexão	393
8.4.2.2. Sempre fechar a conexão	394
8.4.3. DataReader, DataSet e Generics	396
Pontos principais	400
Teste seus conhecimentos.....	401
Mãos à obra!	405
 Capítulo 9 - AJAX.....	 423
9.1. Introdução	424
9.2. Como funciona o AJAX	426
9.3. AJAX com JavaScript	429
9.4. AJAX com jQuery	436
9.4.1. NuGet	436
9.4.2. Usando jQuery	438
9.5. Uso de AJAX com Web Forms	440
9.5.1. Suporte no lado servidor	440
9.6. Exemplo de uso	441
9.7. AJAX com MVC	447
9.8. AJAX Control Toolkit	449
9.8.1. Usando o AJAX Control Toolkit	452
9.8.1.1. Exemplo de Extender: ConfirmButton	456
9.8.1.2. Exemplo de WebControl: TabContainer	457
9.8.2. Exemplos	458
Pontos principais	461
Teste seus conhecimentos.....	463
Mãos à obra!	467

Informações sobre o treinamento

Para o melhor aproveitamento do **curso Visual Studio 2015 – ASP.NET com C# Fundamentos**, é imprescindível ter participado do curso Visual Studio 2015 – C# Fundamentos, ou possuir conhecimentos equivalentes.

1

Conceitos básicos

- ✓ Tipos de softwares;
- ✓ Arquitetura cliente-servidor;
- ✓ Tecnologias utilizadas nas páginas Web;
- ✓ Tecnologias utilizadas no servidor;
- ✓ Versões do ASP.NET;
- ✓ IIS – Internet Information Services;
- ✓ O Visual Studio.



IMPACTA
EDITORA

1.1. Introdução

Neste capítulo, conhiceremos alguns conceitos, tecnologias e ferramentas envolvidos no desenvolvimento de software.

1.2. Tipos de softwares

Existem vários tipos de softwares que atendem às necessidades de gerenciamento de informação. Esses tipos são classificados de acordo com sua arquitetura e finalidade. São eles:

- **Desktop application:** São aplicativos (softwares) que rodam isoladamente dentro de um computador desktop ou notebook e que disponibilizam uma interface com o usuário;
- **Web-based application:** São aplicativos que rodam em um computador servidor e necessitam de um navegador Web para funcionar. O navegador é responsável por exibir as telas e interagir com o usuário, e o servidor é responsável pelo processamento das informações;
- **Services:** São programas que não têm uma interface com o usuário e são iniciados pelo sistema operacional de forma automática;
- **Web services:** Os aplicativos do tipo services podem ser disponibilizados em um servidor Web e executados pela Internet. Um serviço disponível dessa forma é chamado de Web service;
- **Mobile/Tablet application:** É uma variação do aplicativo desktop. O aplicativo roda isoladamente dentro de um smartphone ou tablet.

Essas definições não são absolutas, e os aplicativos podem usar parte de um tipo e parte de outro. Um aplicativo desktop, por exemplo, pode se comunicar com um serviço interno do Windows e, também, com um Web service. O conjunto de tecnologias que definem um aplicativo e como ele é construído é chamado **arquitetura de software**.

1.3. Arquitetura cliente-servidor

A arquitetura de software em que um aplicativo solicita processamento de outro aplicativo ou serviço é chamada **cliente-servidor**. O **cliente** é qualquer aplicativo que solicita processamento de outro programa. O **servidor** é qualquer programa que responde a essa solicitação.

A Internet é um exemplo desta arquitetura. O **cliente** é o navegador Web e o **servidor** é o Web site ou Web service que responde a essa solicitação.



Cliente:

Um computador usando um browser, como Internet Explorer.

Servidor:

Um computador executando um serviço que responde a requisições de um cliente.

1.3.1. O servidor Web

Para desempenhar o papel de servidor, um computador necessita de um programa ou serviço que seja capaz de receber solicitações de aplicativos cliente e de responder a essas solicitações. No Windows, existe um pacote de serviços conhecido como **IIS (Internet Information Services)**. Esses serviços incluem, entre outras coisas, disponibilização de páginas no formato HTML usando o protocolo HTTP, disponibilização de arquivos usando o protocolo FTP, gerenciamento de e-mails, módulos de aplicativos e certificados digitais.

1.3.2. O protocolo HTTP

O **HTTP (HyperText Transfer Protocol)** é um protocolo de comunicação que estabelece transferência de dados entre os navegadores Web e os Web sites. Inicialmente, o propósito do protocolo HTTP era transferir documentos hipertextuais, de forma que os documentos fossem simplesmente relacionados, sem considerar aspectos como as interfaces de usuário baseadas na Web, que são o principal elemento nos sites atuais.

O processo de o navegador chamar o servidor é denominado **request**, e a resposta do servidor a essa solicitação é chamada **response**.

Funcionamento básico do protocolo HTTP



Na versão 1.0 (publicada na década de 1990), o HTTP deixou de ser um simples protocolo de transferência de texto para estabelecer uma comunicação mais complexa. Tal desenvolvimento tinha como objetivo oferecer suporte a diferentes mídias definidas pelo MIME (Multipurpose Internet Mail Extensions). A versão 1.1 é a mais recente.

O protocolo HTTP se baseia em comandos e dados enviados ao servidor. Os comandos são chamados **verbos** e são os seguintes:

- **GET**: Este comando obtém informações do servidor. Usado para pesquisa de dados;
- **POST**: Este comando é utilizado para enviar ao servidor solicitações com dados encapsulados dentro da requisição. É usado para processar informações;

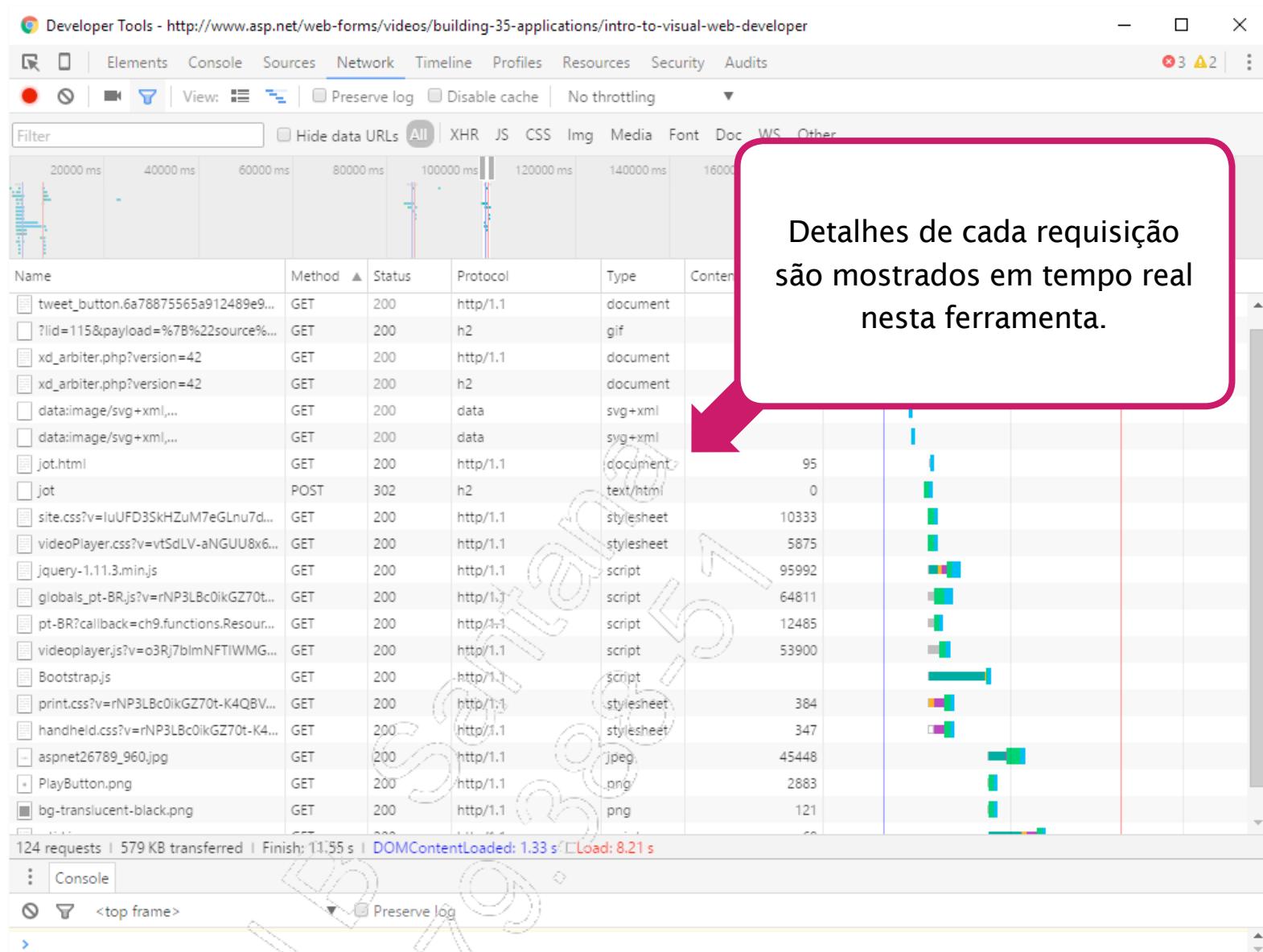
- **HEAD**: Igual ao GET, mas sem o corpo da página. Apenas os cabeçalhos de identificação são retornados. Os cabeçalhos são informações enviadas junto com a página que definem, entre outras coisas, o tipo de conteúdo que será transmitido, o conjunto de caracteres utilizado (encoding), os dados armazenados no servidor (cookies) e o tamanho, em bytes, desses dados;
- **PUT**: Comando semelhante ao POST, usado para incluir ou criar novos dados;
- **DELETE**: Comando para excluir dados;
- **TRACE**: Comando para exibir a requisição que foi enviada ao servidor. Usado para investigar processos;
- **OPTIONS**: Comando que retorna os comandos suportados pelo servidor;
- **CONNECT**: Usado para criar uma conexão criptografada usando o protocolo TCP/IP. Esse processo é chamado **tunnel TCP/IP**;
- **PATCH**: Usado para realizar modificações parciais no servidor.

É importante lembrar que os comandos em si não efetuam nenhum processamento no servidor. O aplicativo instalado no servidor é responsável por interpretar e executar os comandos necessários.

É possível observar os métodos usados nas requisições por meio de recursos para o desenvolvedor encontrados na maioria dos navegadores. No Google Chrome, estando dentro de uma página Web, ao pressionar F12, uma janela de ferramentas aparece.

Visual Studio 2015 - ASP.NET com C# Fundamentos

No menu principal desse aplicativo, na opção **Network**, é possível visualizar todos os detalhes das requisições e respostas:



Conceitos básicos

Entrando no detalhe de uma requisição, é possível visualizar todos os dados enviados (Request) e recebidos (Response) pelo navegador:

The screenshot shows two instances of the Microsoft Edge Developer Tools Network tab. The left instance displays the 'Request' details for a 'GET /Home/Contact HTTP/1.1' request, listing headers like Host, User-Agent, and Accept. The right instance displays the 'Response' details, showing the 'Headers' tab with response headers such as Cache-Control, Content-Encoding, and Content-Type. A callout bubble points from the text below to the right instance.

Cada solicitação e resposta podem ser inspecionadas em todos os detalhes.

1.3.2.1.Request

O formato básico de uma requisição HTTP (Request) é o seguinte:

```
GET /Home/Contact HTTP/1.1
Host: localhost:51021
Accept: text/html

Cookie: usuario=3WrUX
```

A linha 1 informa o método (**GET**), a URL relativa ao servidor (**/Home/Contact**) e o protocolo (**HTTP/1.1**). As informações são separadas por espaço.

A linha 2 informa o endereço do servidor (**Host**).

A linha 3 adiciona uma informação à requisição. Nesse exemplo, apenas um item está definido (**Accept**), mas múltiplas informações podem ser inseridas nessa parte. Nesse caso, o item **Accept** diz ao servidor o formato de dados esperado pelo cliente nesta requisição.

A linha 5 informa os cookies armazenados pelo navegador. Um cookie é um texto armazenado pelo navegador e transmitido para o servidor a cada requisição.

No exemplo anterior, o método GET solicita dados do servidor. Geralmente esse método é utilizado para solicitar uma página. Para enviar dados digitados em um formulário, o método POST pode se utilizar. Nesse caso, a informação digitada pelo usuário é incluída dentro da requisição:

```
POST / Usuario/Cadastro HTTP/1.1
Host: localhost:51021
Content-Length: 37

nome=Maria&email=maria@teste.com.br
```

No exemplo anterior, a linha 3 informa o tamanho da informação, a linha 4 está em branco (deve ser assim) e a linha 5 contém os dados enviados ao servidor.

1.3.2.2. Response

O formato da resposta (Response) no protocolo HTTP é o seguinte:

```
HTTP/1.1 200 OK
Content-Length: 12260
Content-Type: text/html;

<html><body><h1>Teste</h1></body></html>
```

A linha 1 informa o protocolo (**HTTP/1.1**), o status da resposta (**200**) e o texto do status da resposta (**OK**). Os status mais comuns são:

- 200 – OK;
- 204 – Nenhum conteúdo;
- 404 – Não encontrado;
- 401 – Não autorizado;
- 500 – Erro interno do servidor.

A linha 2 informa o tamanho das informações (**Content-Length**) que serão transmitidas. O navegador pode criar uma barra de progresso para informar ao usuário quando a página foi completamente carregada.

A linha 3 define o tipo de conteúdo (**Content-Type**). Nesse caso, é uma página HTML. O servidor pode definir muitos outros dados e inseri-los a partir dessa linha.

A linha 4 está em branco.

Da linha 5 em diante é o conteúdo da página.

1.4. Tecnologias utilizadas nas páginas Web

O servidor pode enviar ao cliente qualquer tipo de conteúdo: textos, imagens ou arquivos binários. Cabe ao navegador entender e gerar uma página a partir dos arquivos recebidos. Todo navegador, por padrão, consegue manipular três tipos de arquivos: HTML, JavaScript e CSS.

- **HTML (HyperText Markup Language)**, ou linguagem de marcação de hipertexto, é uma linguagem criada para construir documentos. O objetivo desta linguagem é definir, em um documento, onde ficam o título, o subtítulo, um parágrafo, uma seção, um vínculo com outro documento, uma área para navegação, entre outras partes. Essa definição é feita usando tags (marcas) no texto;
- **CSS (Cascading Style Sheets)**, ou folhas de estilo em cascata, é uma linguagem criada para definir a aparência de um documento HTML. As folhas de estilo utilizam seletores – para selecionar qual parte do documento HTML vai ser formatada – e palavras reservadas – para definir itens como tamanho, cor, margem, espaçamento e outras propriedades relacionadas à maneira como as informações serão exibidas;
- **JavaScript** é uma linguagem de programação executada pelo navegador, ou seja, do lado do cliente. A linguagem JavaScript pode interagir com os elementos de uma página HTML usando um modelo de objetos chamado **DOM (Document Object Model)** e pode interceptar eventos como movimentos do mouse, gestos (no caso de dispositivos touch) ou carregamento de imagens.

Além dessas tecnologias citadas, o navegador pode utilizar várias outras por meio de plugins ou recursos nativos. Existe uma tendência lenta e constante que os navegadores utilizem cada vez mais linguagens e componentes padrão de mercado. Um dos últimos grandes avanços nessa área é a capacidade de criar gráficos, exibir vídeos e animações incluídas no HTML5. Antes dessa versão do HTML, essas funcionalidades só estavam disponíveis por meio de componentes de empresas privadas.

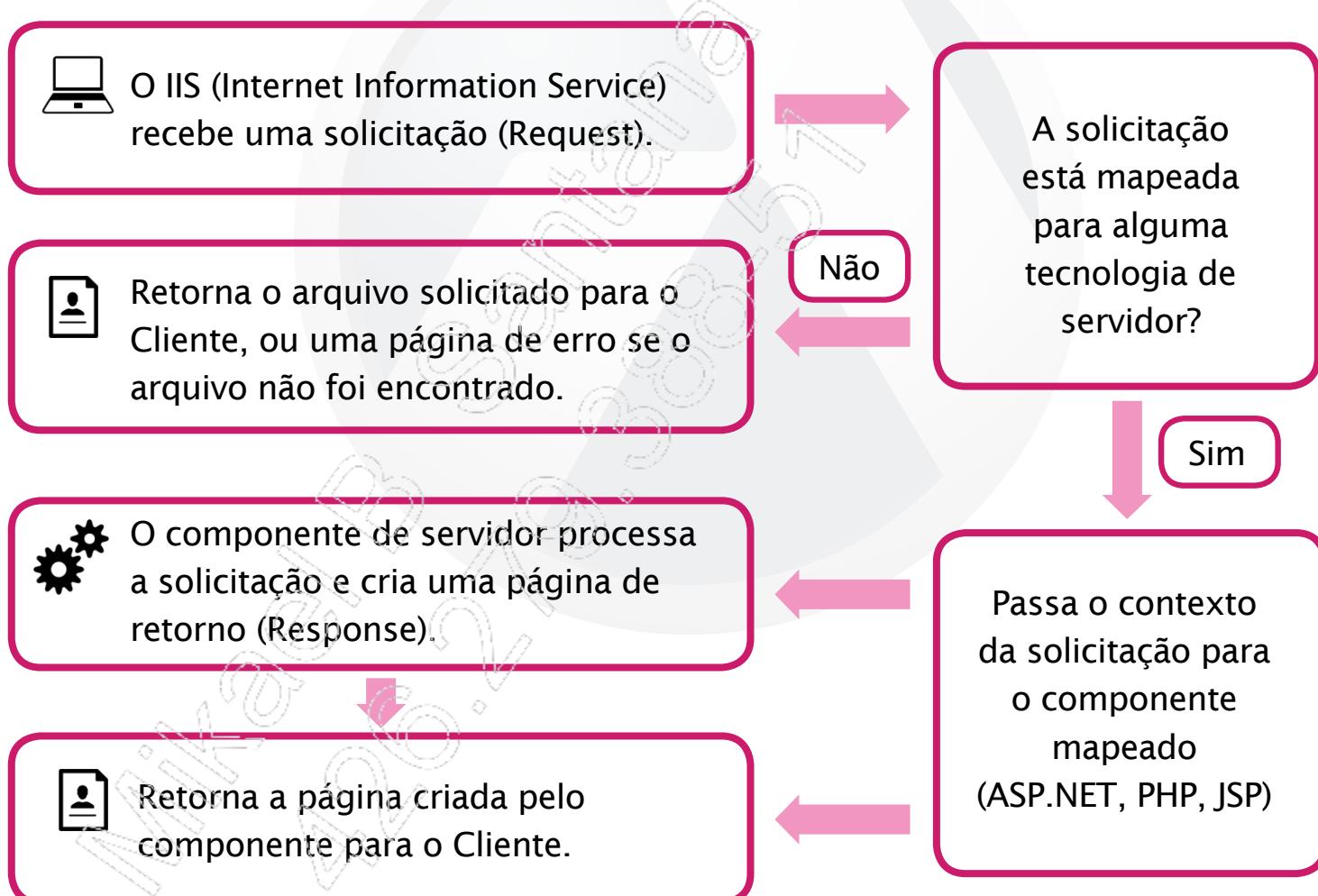
1.4.1. Exemplo de uma página retornada do servidor contendo HTML, CSS e JavaScript



Uma página real pode parecer bem mais complexa do que a do exemplo. Isso ocorre porque, além de o conteúdo ser maior, as folhas de estilo e o JavaScript são normalmente colocados em arquivos separados, existem referências a imagens, bibliotecas de terceiros, metadados e objetos incorporados.

1.5. Tecnologias utilizadas no servidor

O servidor Web é o responsável por atender a solicitações do cliente, localizar o conteúdo solicitado, preparar o documento HTML e retorná-lo. O ASP.NET entra nesse processo depois que o IIS recebeu a solicitação e antes de retornar o conteúdo para o cliente. Esse caminho é chamado de **pipeline** (encanamento) e diversos componentes podem ser colocados nesse processo.



1.6. Versões do ASP.NET

O ASP.NET foi originalmente concebido para ser um componente de servidor que roda dentro do IIS e reúne, em um único componente, todas as funcionalidades para criar aplicativos Web. Essas funcionalidades incluem autenticação, criptografia, gerenciamento de estado, mecanismos de renderização de páginas HTML e integração com o Windows. Até a versão 4.6 do .NET Framework essa foi a abordagem utilizada.

Com a versão 5.0, o ASP.NET e o .NET Framework passam a seguir outro caminho. Em vez de um único componente proprietário com todas as funcionalidades, foi adotada uma política de fragmentar as funcionalidades em diversos componentes isolados, todos open source e multiplataforma. O próprio nome da nova versão passou por diversas mudanças. O sucessor do ASP.NET 4.6 foi chamado de ASP.NET vNext, ASP.NET 5.0 e, finalmente, ASP.NET Core 1.0. O quadro a seguir compara as versões:

ASP.NET 2.0 - 4.6	ASP.NET Core 1.0
Componente único (Module).	Diversos componentes (Middleware).
Código proprietário.	Open source.
Plataforma Windows.	Multiplataforma (Windows, Linux, iOS).
Instalado em um servidor com IIS.	Servidor pode ser distribuído com a aplicação.
Framework maduro, testado e com amplo material disponível para aprendizado. Em aperfeiçoamento há 15 anos.	Novo. No lançamento (2015), diversas funcionalidades não foram incluídas, e o cronograma é de estarem todas disponíveis no final de 2016.

Neste curso, será abordado o framework 4.6, porque a base conceitual é a mesma (IIS, HTML, cliente/servidor, CSS, JavaScript e objetos ASP.NET). As novidades da versão Core 1.0 serão incluídas aos poucos nos próximos cursos, sempre fazendo o paralelo do que mudou e por qual motivo.

1.7. IIS - Internet Information Services

O **IIS (Internet Information Services)** é o serviço do Sistema Operacional Windows responsável por receber e responder a solicitações via protocolo HTTP. Para administrar este serviço, é utilizado o **Gerenciador de Serviços de Informações da Internet (IIS)**, dentro do **Painel de Controle**, em **Ferramentas Administrativas**:



Em linhas gerais, o que esse serviço faz é mapear uma pasta física para uma pasta virtual. A pasta virtual pode ser isolada em uma área de memória protegida e preparada para executar scripts. Quando uma pasta é preparada dessa forma, é chamada de **pasta de aplicação**.

A pasta física é o local real onde está um aplicação Web. Por exemplo:

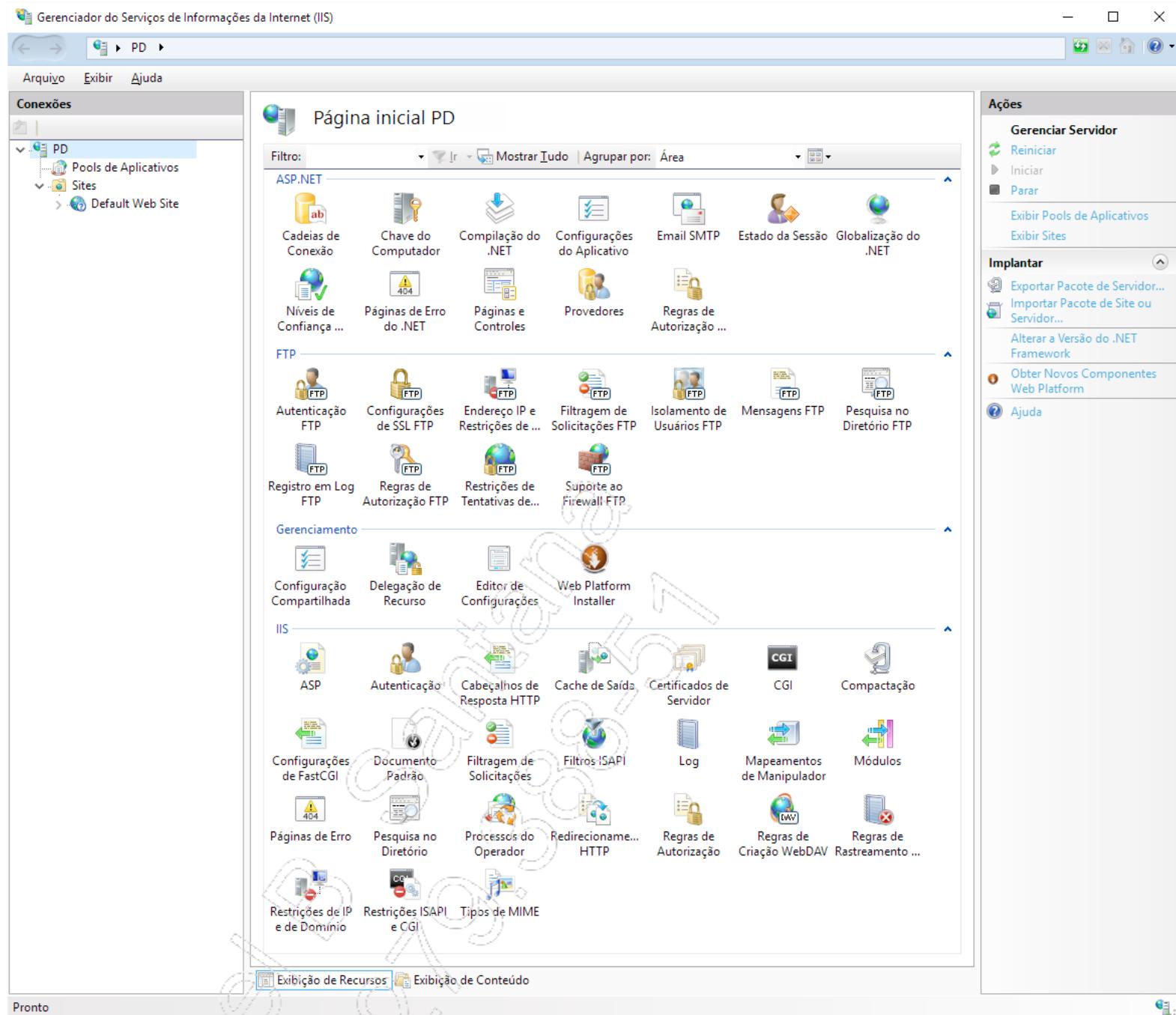
C:\inetpub\wwwroot\MinhaAppWeb

A pasta virtual é a URL (Uniform Resource Location) que dá acesso ao servidor e à aplicação Web. Por exemplo:

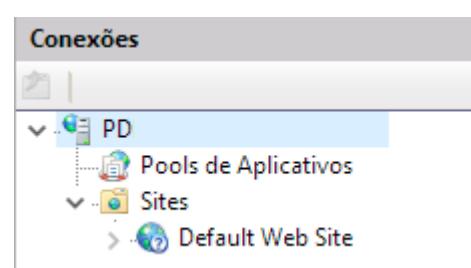
<http://MeuServidor/AppExemplo>

Visual Studio 2015 - ASP.NET com C# Fundamentos

Executando o gerenciador do IIS, obtém-se a seguinte tela:



Na janela **Conexões**, existe o nome do servidor e o Web site padrão, chamado **Default Web Site**. Cada aplicação Web pode compartilhar dados de segurança com outras aplicações. O item **Pool de Aplicativos** define esses grupos.



Na janela principal, existem quatro grupos: **ASP.NET**, **FTP**, **Gerenciamento** e **IIS**. Cada grupo controla determinadas características do servidor Web. Um dos itens controla o mapeamento do manipulador HTTP, que é responsável por decidir se determinada solicitação vai ser atendida enviando diretamente o arquivo ou chamando um componente instalado. Esse item é o **Mapeamentos de Manipulador**:



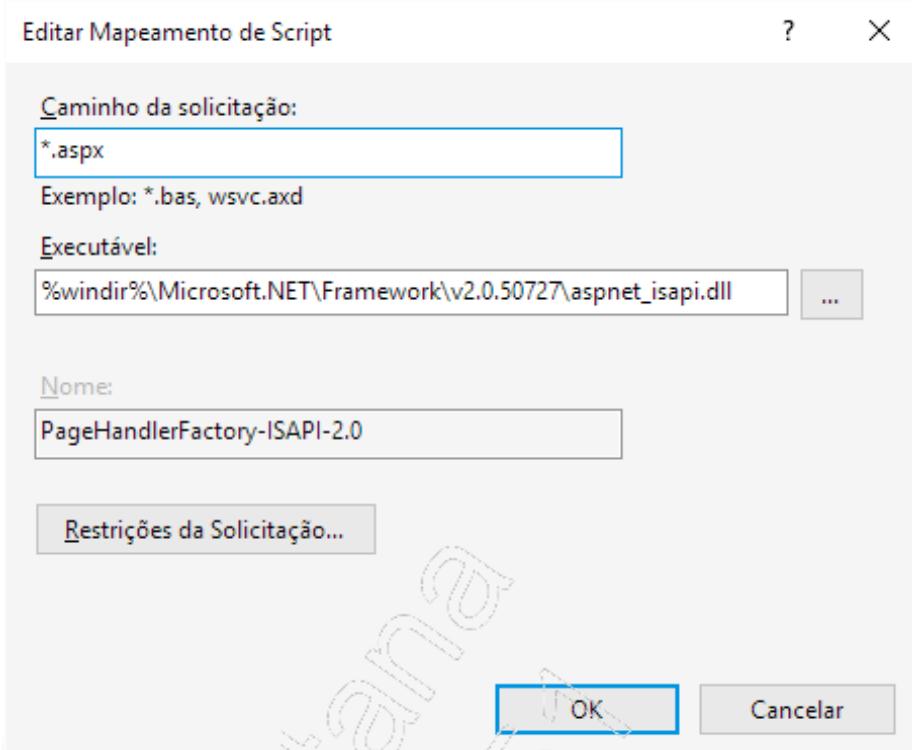
Nessa tela, podemos ver cada extensão de arquivo e o manipulador associado a ele. Por exemplo, se uma página chamada **Index.aspx** for solicitada, o IIS para o controle da resposta (Response) para o ASP.NET. Se a página for **index.php**, o IIS para o controle para o componente PHP (se estiver instalado).

A captura de tela mostra a interface de usuário "Mapeamentos de Manipulador". No topo, há uma barra com o nome da seção e uma descrição: "Use este recurso para especificar os recursos, tais como DLLs e código gerenciado, que manipulam respostas para tipos de solicitações específicos." Abaixo, há uma tabela com os seguintes dados:

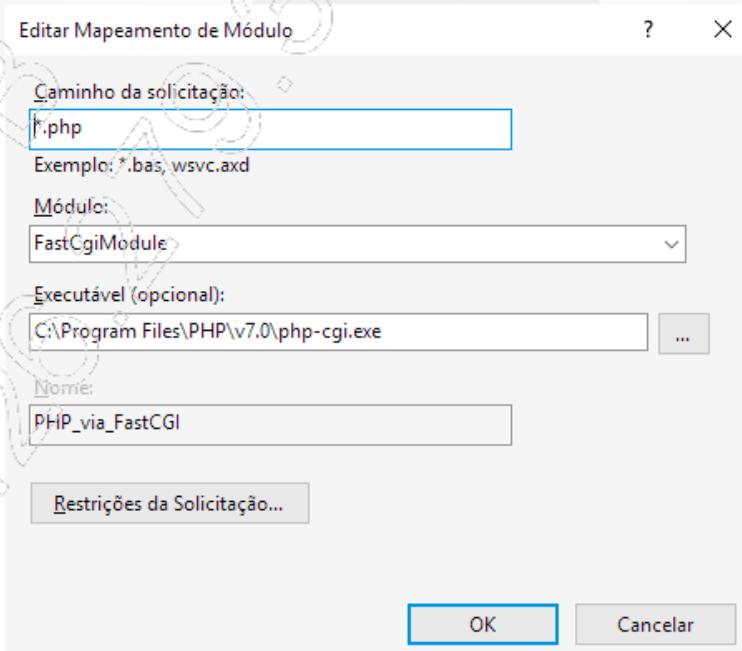
Nome	Caminho	Estado	Tipo de Caminho	Manipulador	Tipo de En...
Habilitado					
PHP_via_FastCGI	*.php	Habilitado	Arquivo ou Pa...	FastCgiModule	Local
PHP53_via_FastCGI	*.php	Habilitado	Arquivo ou Pa...	FastCgiModule	Local
rules-64-ISAPI-2.0	*.rules	Habilitado	Não especifica...	IsapiModule	Local
rules-ISAPI-2.0	*.rules	Habilitado	Não especifica...	IsapiModule	Local
xoml-64-ISAPI-2.0	*.xoml	Habilitado	Não especifica...	IsapiModule	Local
xoml-ISAPI-2.0	*.xoml	Habilitado	Não especifica...	IsapiModule	Local
svc-ISAPI-2.0-64	*.svc	Habilitado	Não especifica...	IsapiModule	Local
svc-ISAPI-2.0	*.svc	Habilitado	Não especifica...	IsapiModule	Local
xamlx-ISAPI-4.0_64bit	*.xamlx	Habilitado	Não especifica...	IsapiModule	Local
xamlx-ISAPI-4.0_32bit	*.xamlx	Habilitado	Não especifica...	IsapiModule	Local

Visual Studio 2015 - ASP.NET com C# Fundamentos

Abrindo o item cujo caminho é ***.aspx**, por exemplo, é possível visualizar os detalhes. Na imagem adiante, é possível verificar que esse tipo de arquivo está associado ao assembly **aspnet_isapi.dll**, do .NET Framework.



O manipulador de uma página PHP é ativado quando as páginas têm a extensão **.php**, conforme mostra a tela:



Cada componente instalado no servidor é chamado pelo IIS para processar o arquivo solicitado. O arquivo geralmente contém scripts de servidor que são analisados, compilados (ou interpretados) e executados para gerar a página HTML final.

É importante destacar que o navegador não sabe qual tecnologia foi utilizada para processar sua solicitação. Do ponto de vista do cliente (navegador), foi solicitado um documento HTML, e foi exatamente isso que retornou do servidor.

Esse é um dos principais motivos do sucesso da Internet e das aplicações que são executadas nesse ambiente. Não importa a tecnologia utilizada, a aplicação Web pode ser usada por qualquer navegador, independente do sistema operacional em que está instalado. Isso é completamente diferente de um software desktop que precisa ser instalado no computador onde será utilizado, com a versão compatível para o seu Sistema Operacional.

1.7.1. IIS – Criando uma aplicação ASP.NET

Estão fora do escopo deste curso todos os detalhes técnicos de funcionamento, instalação e manutenção do IIS em um servidor. Esse processo geralmente é transparente para o programador ASP.NET, e configuração do servidor é parte das tarefas dos técnicos, analistas e gerentes de infraestrutura. Mas, para iniciar no desenvolvimento de aplicativos Web, é importante entender como funciona o IIS nesse cenário.

Em muitas situações, essa tarefa é delegada a uma empresa (provedor) que disponibiliza algum painel de controle simplificado para as configurações relacionadas à aplicação. Outro caminho é a computação nas nuvens, em que a aplicação é publicada em um serviço que cria máquinas virtuais automaticamente e que podem ser redimensionadas conforme a necessidade do aplicativo. O Windows Azure é o serviço oficial da Microsoft para esse tipo de tarefa.

Para criar uma aplicação ASP.NET, siga os passos descritos a seguir:

1. Crie uma pasta em qualquer drive. Se quiser usar a "pasta oficial" do ASP.NET, o endereço é **c:\inetpub\wwwroot\NomeDoSeuApp**;

Visual Studio 2015 - ASP.NET com C# Fundamentos

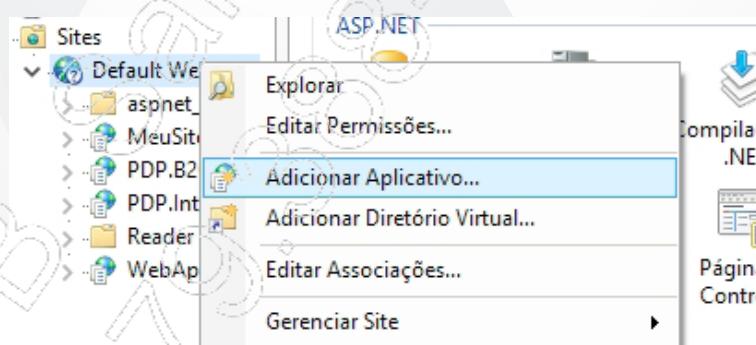
2. Crie um arquivo HTML (usando o Notepad, o Notepad++ ou o editor de sua preferência) chamado **Index.htm** dentro dessa pasta. Use o template mínimo para um arquivo HTML:

```
<!DOCTYPE html>

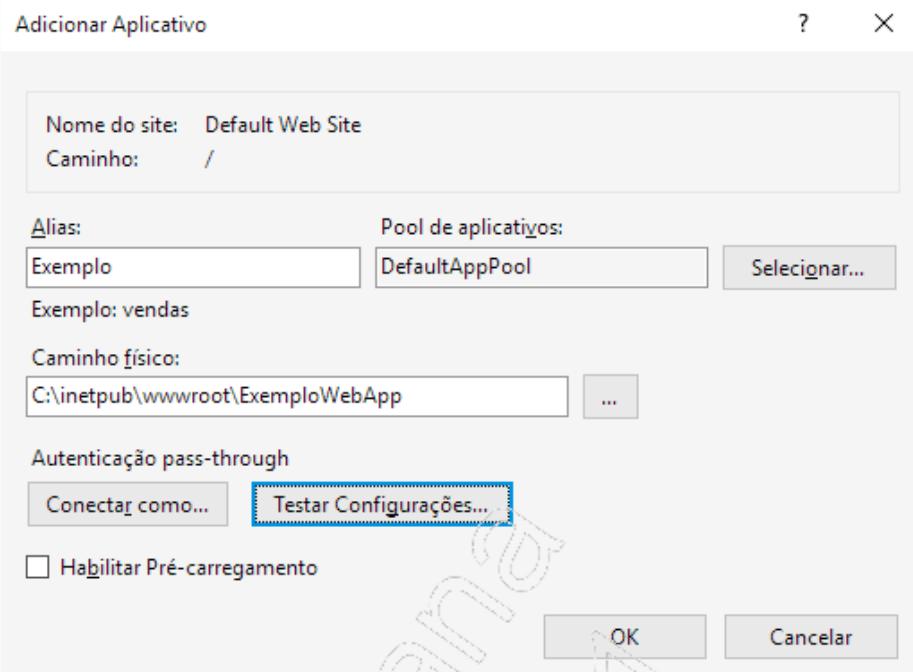
<html>
<head>
    <title></title>
</head>
<body>
    <h1>Exemplo de Arquivo HTML</h1>

    <p>Isto é um parágrafo</p>
</body>
</html>
```

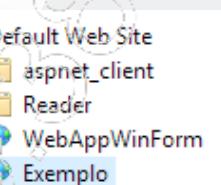
3. No gerenciador do IIS, no menu de contexto de **Default Web Site**, escolha **Adicionar Aplicativo**:



4. Informe o apelido do site (**Alias**) e o caminho físico. O alias pode ser qualquer nome sem espaços ou acentos. É por meio desse nome que o aplicativo será executado. O caminho físico deve ser exatamente aquele onde a pasta foi criada;



5. Verifique se foi criado corretamente:



6. Usando qualquer navegador, entre no endereço usando o seguinte modelo:
http://NomeDoServidor/Alias/Index.htm.

Por exemplo, se o seu servidor se chama **Servidor01** e o apelido do seu aplicativo é **MeuApp**, a URL será **http://Servidor01/MeuApp/Index.htm**.

Quando o servidor é o mesmo computador onde está o navegador, é possível trocar o nome do servidor por localhost ou pelo número de IP.

Visual Studio 2015 - ASP.NET com C# Fundamentos

O arquivo **Index.html** está na lista dos arquivos que abrem automaticamente, por isso pode ser omitido.



O exemplo anterior executa e o browser consegue exibir o arquivo, mas não há nenhum processamento no servidor além do IIS, porque a extensão **htm** não está mapeada para nenhum componente. Nesse caso, o IIS apenas entrega o arquivo solicitado.

Para observar um processamento no servidor, é necessário criar um arquivo que esteja associado a um componente. Por exemplo, as páginas com extensão **.aspx** estão associadas ao ASP.NET, mais precisamente ao **Web Form**, um dos tipos de páginas disponíveis.

Para visualizar um exemplo de processamento no servidor, insira na pasta da aplicação o arquivo a seguir, chamado **ExemploWebform.aspx**. Assim como o arquivo HTML, ele pode ser criado em qualquer editor de texto.

```
<%@ Page Language="C#" %>
<!DOCTYPE html>
<html>
<head>
    <title></title>
</head>
<body>
    <form runat="server">

        <h1>Exemplo Web Form</h1>

        <asp:Calendar runat="server" />

    </form>
</body>
</html>
```

O arquivo do exemplo anterior é um arquivo HTML tradicional com as seguintes alterações:

- A primeira linha está incluída dentro de uma marcação especial usada para ser analisada e executada no servidor: `<% %>`. Essa é uma das duas maneiras de executar informações no servidor. A outra é usando o atributo `runat="server"` em uma tag HTML;
- A primeira linha contém uma diretiva de compilação. Essa instrução fornece ao compilador para página algumas informações. Nesse caso, a informação é que é uma página **@Page** e que a linguagem de programação, se houver scripts, é **C#**. A informação que é uma página é importante porque existem outras estruturas, como **@Control**, **@Master**, **@Register**, entre outras. Essas estruturas serão vistas no decorrer do curso;
- No início do corpo do documento, existe um elemento **form** com o atributo `runat="server"`. Quando uma página ASP.NET do tipo Web Form (extensão **.aspx**) é executada, o mecanismo de execução da página procura por elementos HTML contendo esse atributo. Se o elemento for válido, ele será transformado em um objeto instanciado na memória do servidor, que gera um conteúdo HTML na hora de a página ser criada. Nesse caso, existe um elemento **form** e outro elemento **asp:Calendar** definidos com esse atributo. Esse processo de verificar a página procurando e executando os elementos que devem ser processados no servidor é chamado de **Parse**. O processo de exibir a página no navegador é chamado de **Render**. A renderização é executada pelo navegador (client). O **Parse**, a compilação e a criação do HTML, executados no servidor (server). É importante para o programador Web compreender quem é responsável por cada parte do processo.

Visual Studio 2015 - ASP.NET com C# Fundamentos

Ao executar esse arquivo, o código HTML gerado cria a seguinte tela quando executada no navegador:



O conteúdo HTML gerado pelo servidor é completamente diferente da página ASPX gravada. É possível visualizar o que o navegador recebeu usando o menu de contexto e a opção **Visualizar Código-Fonte da Página**:

A screenshot of a web browser window titled "view-source:localhost/ExemploWebApp.ExemploWebApp.aspx". The main content area displays the raw HTML code generated by the server. It includes the title "Exemplo Web App", the calendar structure with its CSS classes like "table", "tr", "td", and "th", and various JavaScript links and comments. The code is color-coded by the browser's developer tools to highlight different elements.

1.8. O Visual Studio

O Visual Studio é a ferramenta oficial da Microsoft para o desenvolvimento de qualquer tipo de aplicativo. As seguintes versões estão disponíveis:

- **Visual Studio Community 2015**

Versão gratuita com os principais recursos para criar aplicativos Windows, Web, Mobile, Services e aplicações multiplataforma. A licença limita o uso para criações de aplicações open source.

- **Visual Studio Professional 2015**

Versão para criação de aplicações comerciais com os principais recursos para criar aplicativos Windows, Web, Mobile, Services e aplicações multiplataforma.

- **Visual Studio Enterprise 2015**

Versão para criação de aplicações comerciais de qualquer porte com recursos de desenvolvimento, arquitetura, análise de código, teste e design para qualquer tipo de aplicação.

- **Visual Studio Team Services**

Serviços on-line para desenvolvimento em equipe. Gratuito para cinco usuários. Todos as versões do Visual Studio, inclusive a Community, podem utilizar os recursos do Team Services.

- **Visual Studio Test Professional**

Versão para a criação e execução de unidades de testes.

- **Assinatura MSDN**

As assinaturas MSDN permitem utilizar uma versão do Visual Studio com garantia de atualização durante o tempo do contrato. Um vasto material de estudo é disponibilizado neste modo de licenciamento.

- **Visual Studio Code**

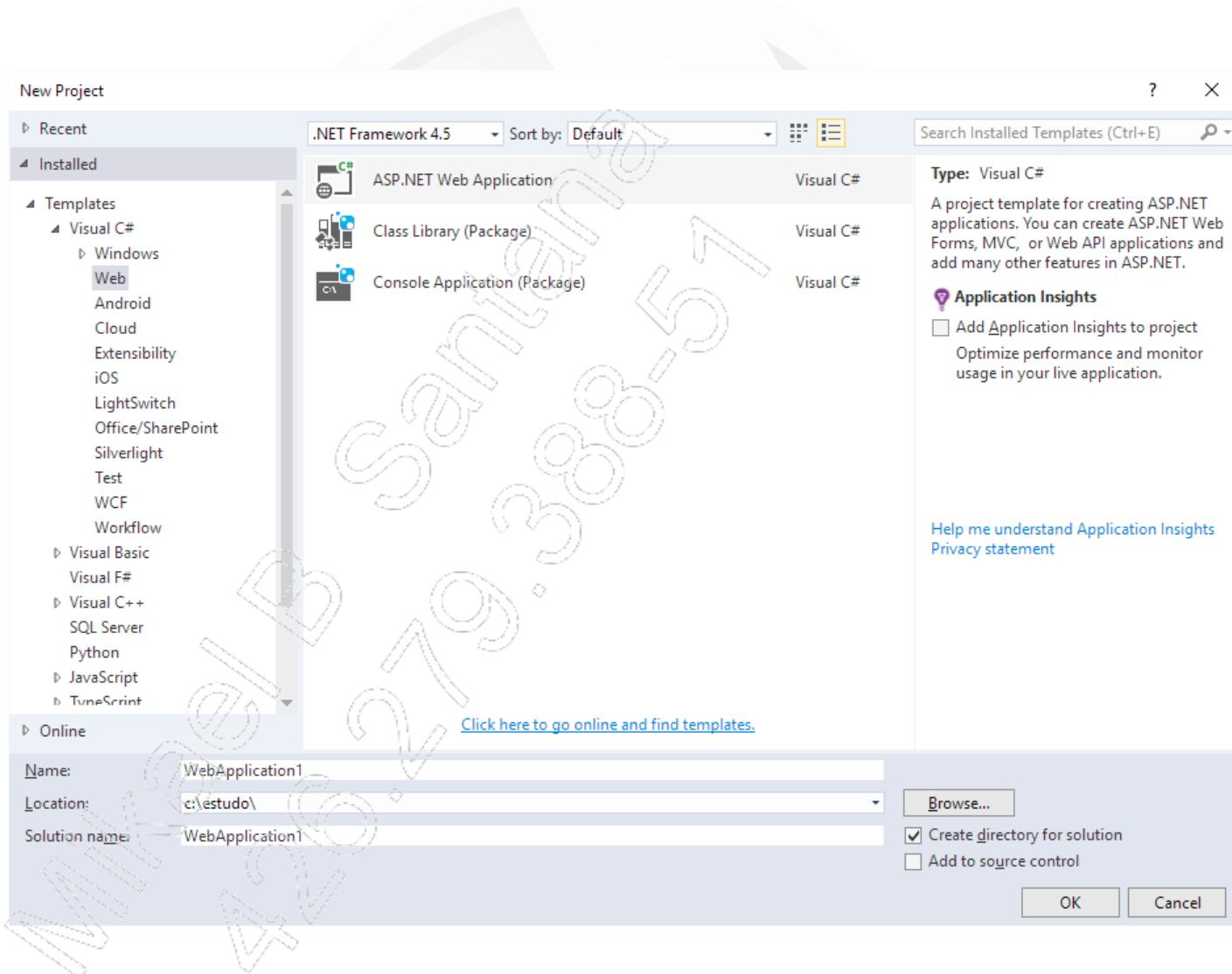
Ferramenta multiplataforma para desenvolvimento de aplicativos usando o framework Core 1.0 e integração facilitada com componentes open source.

1.8.1. Usando o Visual Studio Community

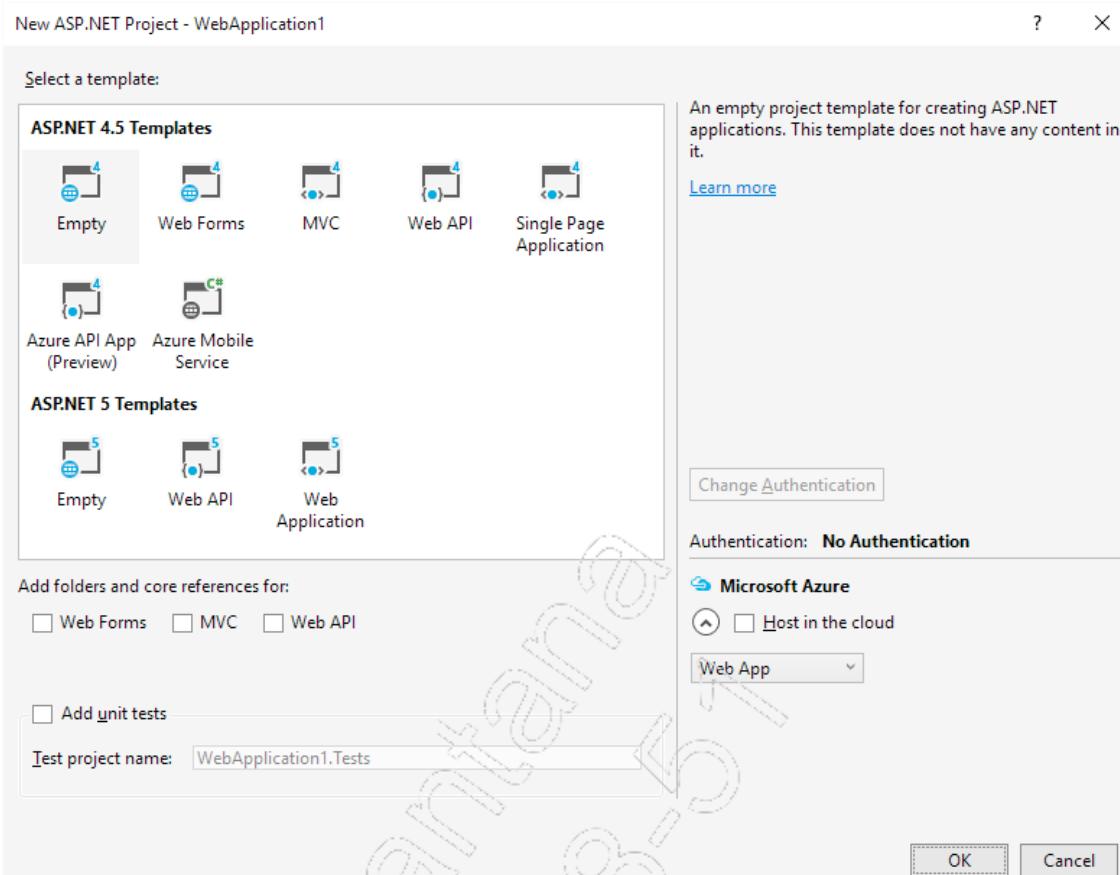
Neste curso, serão utilizados os recursos da versão Community 2015 do Visual Studio. Essa versão é equivalente à versão Professional 2015 e permite criar qualquer tipo de aplicativo e utilizar qualquer tipo de componente.

1.8.1.1. Criando um projeto Web

Um projeto Web é uma série de arquivos organizados que geram uma aplicação Web quando publicados em um servidor Web. Usando o comando de menu **File / New Project**, é possível escolher a linguagem (C#) e o menu Web.



Uma aplicação Web pode utilizar diversas arquiteturas de desenvolvimento e incluir diferentes componentes. Isso é definido na tela seguinte:



Os principais modelos são os seguintes:

- **Empty**

Um projeto vazio. Cabe ao programador definir a arquitetura e inserir os arquivos necessários.

- **Web Form**

Um tipo de projeto muito parecido com Windows Forms, utilizando eventos.

- **MVC**

Um projeto do tipo Model-View-Controller, em que as funcionalidades de cada parte do programa são claramente divididas.

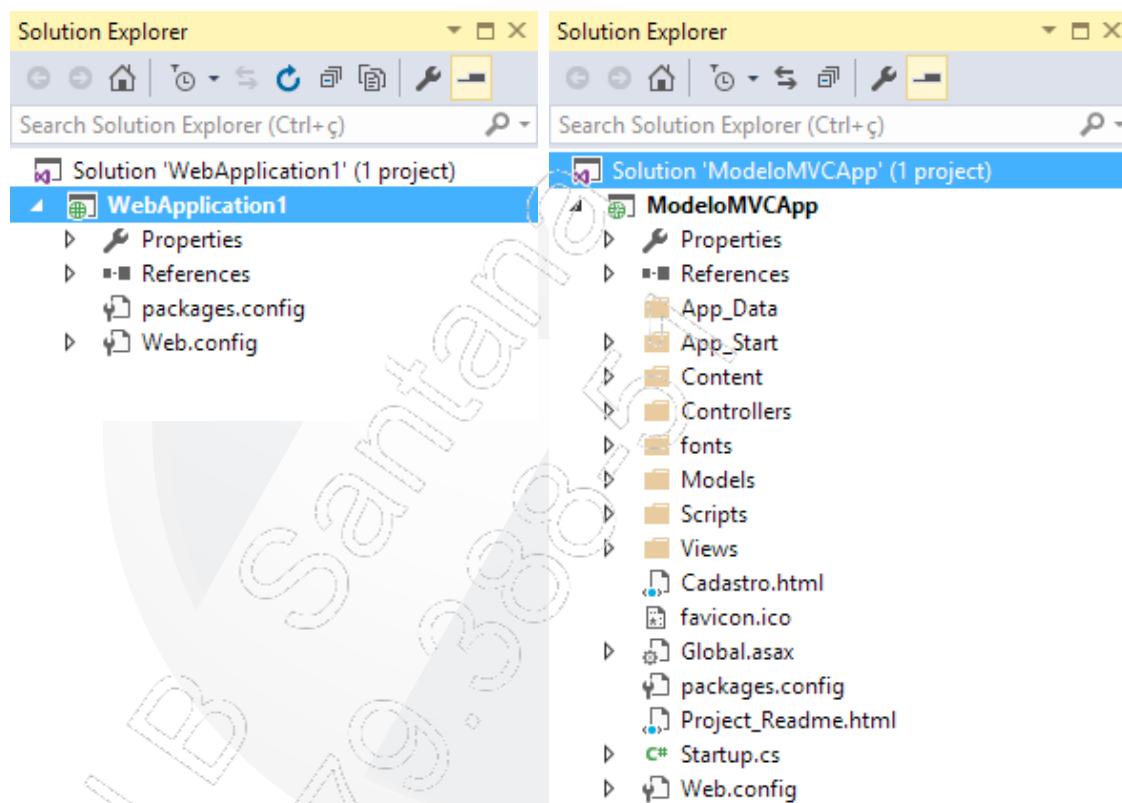
- **Web API**

Um projeto para criar serviços disponibilizados na rede.

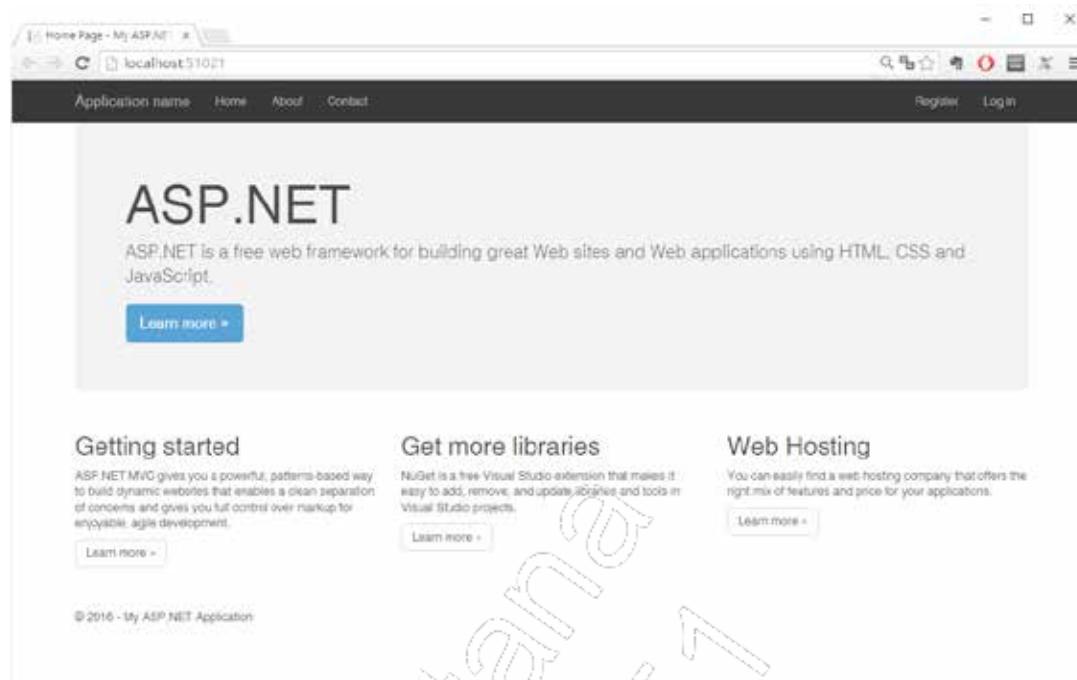
- **Single Page App**

Projeto em que um arquivo HTML é carregado uma única vez e toda interação do usuário é feito usando JavaScript e arquivos no formato JSON e XML.

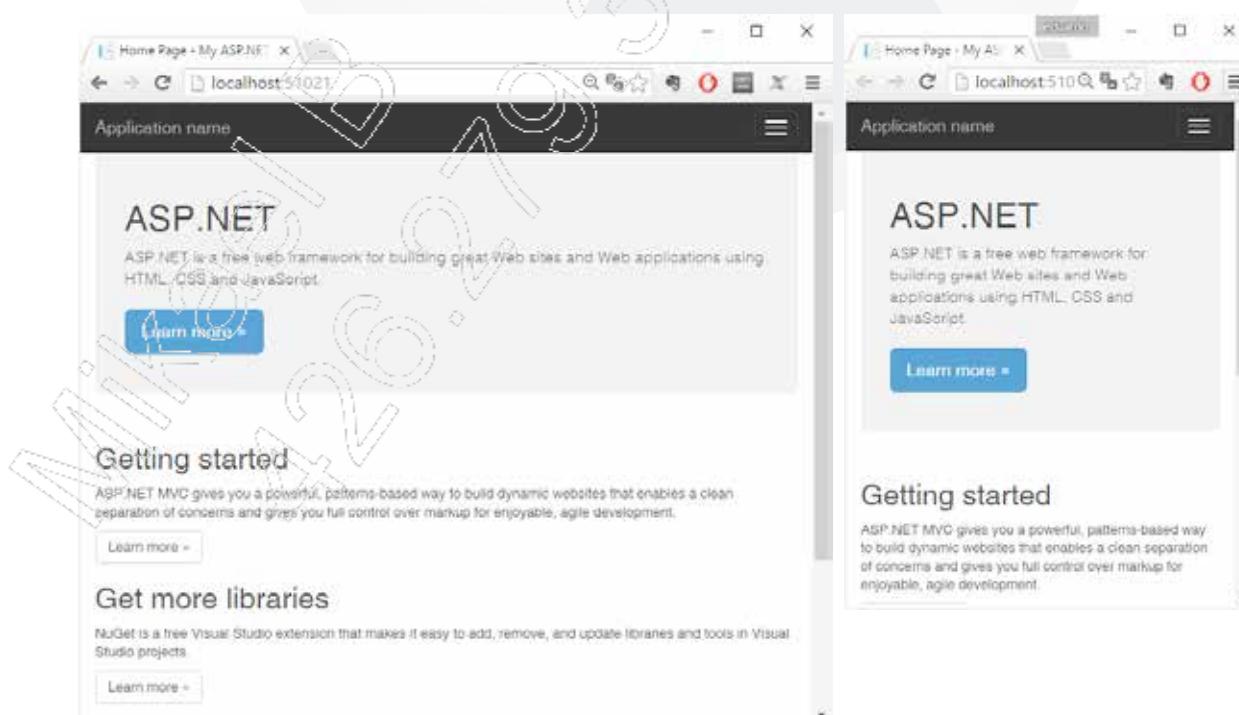
Ainda nesta página, é possível criar aplicativos que fiquem hospedados no Azure, aplicativos Core 1.0, projetos de teste e definir componentes de autenticação. Neste curso, serão abordados a arquitetura Web Forms, MVC e Web Pages. Adiante, um exemplo de um projeto vazio e de um projeto do tipo MVC.



No caso do Empty Project, nem um arquivo HTML é incluído. No caso de Web Form e MVC, a parte visual do aplicativo fica a cargo dos arquivos CSS e toda uma rotina de autenticação é criada:



O design do aplicativo é responsivo, se adaptando a qualquer tamanho de monitor. Isso se deve a uma biblioteca CSS chamada **Bootstrap**.



Visual Studio 2015 - ASP.NET com C# Fundamentos

Um sistema de cadastramento usando o banco de dados SQL Server e um mecanismo chamado ASP.NET Identity é utilizado para gerenciar usuários:

The image displays two side-by-side browser windows showing the 'Register' and 'Log in' pages of an ASP.NET application.

Register Page (Top Window):

- Page title: Register - My ASP.NET App
- URL: localhost:51021/Account/Register
- Header: Application name, Home, About, Contact, Register, Log in
- Content:
 - Register heading: Register.
 - Text: Create a new account.
 - Form fields:
 - Email: [text input]
 - Password: [password input]
 - Confirm password: [password input]
 - Buttons: Register
- Footer: © 2016 - My ASP.NET Application

Log in Page (Bottom Window):

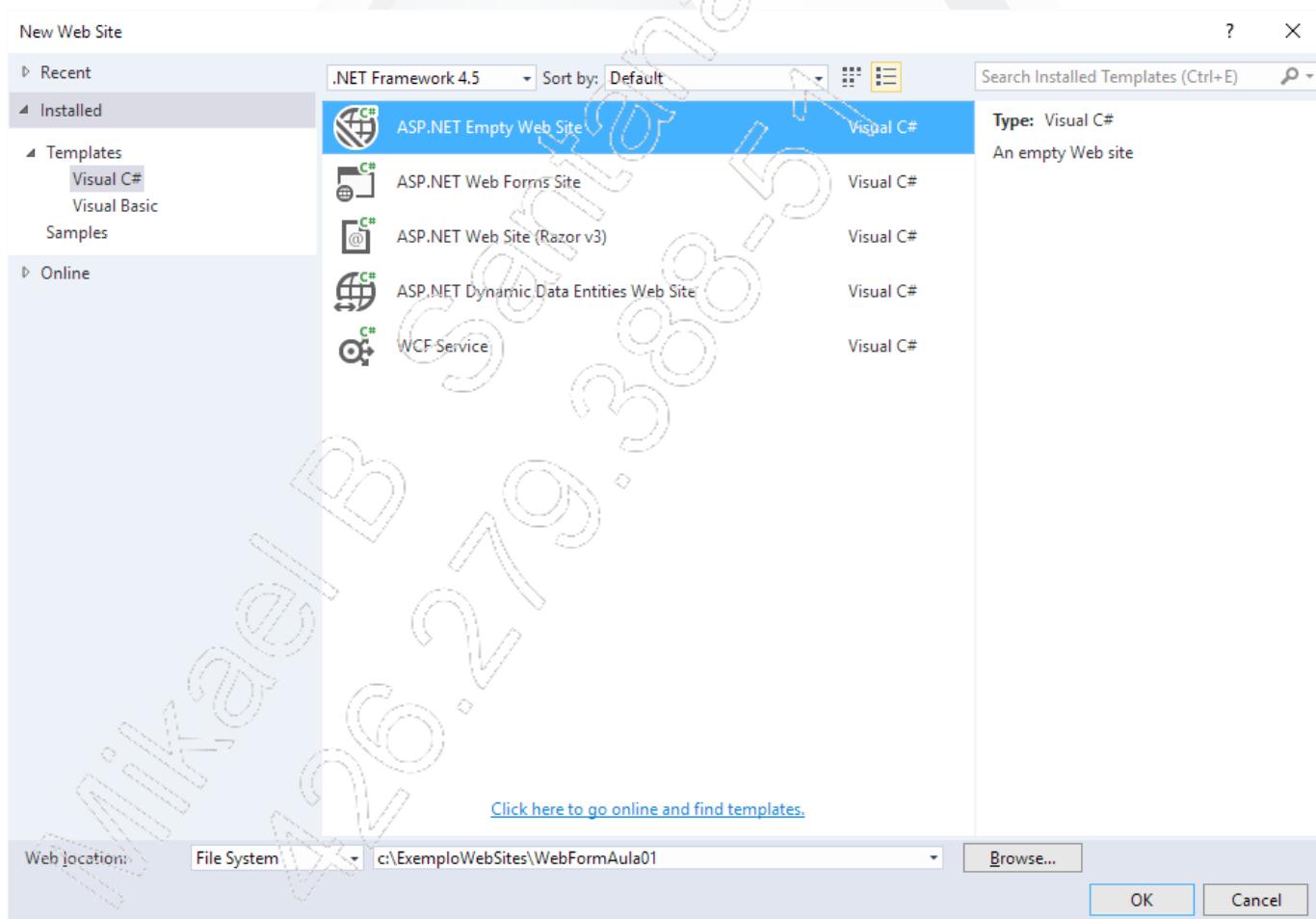
- Page title: Log in - My ASP.NET App
- URL: localhost:51021/Account/Login
- Header: Application name, Home, About, Contact, Register, Log in
- Content:
 - Log in heading: Log in.
 - Text: Use a local account to log in.
 - Form fields:
 - Email: [text input]
 - Password: [password input]
 - Remember me? [checkbox]
 - Buttons: Log in
- Text links:
 - Register as a new user
 - Use another service to log in.
- Text at bottom: There are no external authentication services configured. See [this article](#) for details on setting up this ASP.NET application to support logging in via external services.
- Footer: © 2016 - My ASP.NET Application

1.8.1.2.Criando um Web site

Um Web site é uma pasta contendo todos os arquivos que fazem parte de uma aplicação Web. Não existe arquivo de projeto ou arquivos de configuração do Visual Studio. A pasta pode ser copiada diretamente em um servidor Web. Não é obrigatório passar pelo processo de publicação, como é em um aplicativo Web. Para publicar uma nova página basta copiar os arquivos novos para o servidor. Não são todos os tipos de templates que aceitam um Web site como ponto de partida para um aplicativo Web.

Veja os passos para criar um Web site:

No menu **File**, escolha **New Web Site**:



Os principais modelos são os seguintes:

- **ASP.NET Empty Web Site**

Uma pasta vazia. Cabe ao programador definir a arquitetura e inserir os arquivos necessários.

Visual Studio 2015 - ASP.NET com C# Fundamentos

- **ASP.NET Web Forms Site**

Web site usando Web Forms, que são muito parecidos com Windows Forms, utilizando eventos.

- **ASP.NET Web Site (Razor v3)**

Web site usando Web pages, que são páginas mesclando HTML e código no servidor.

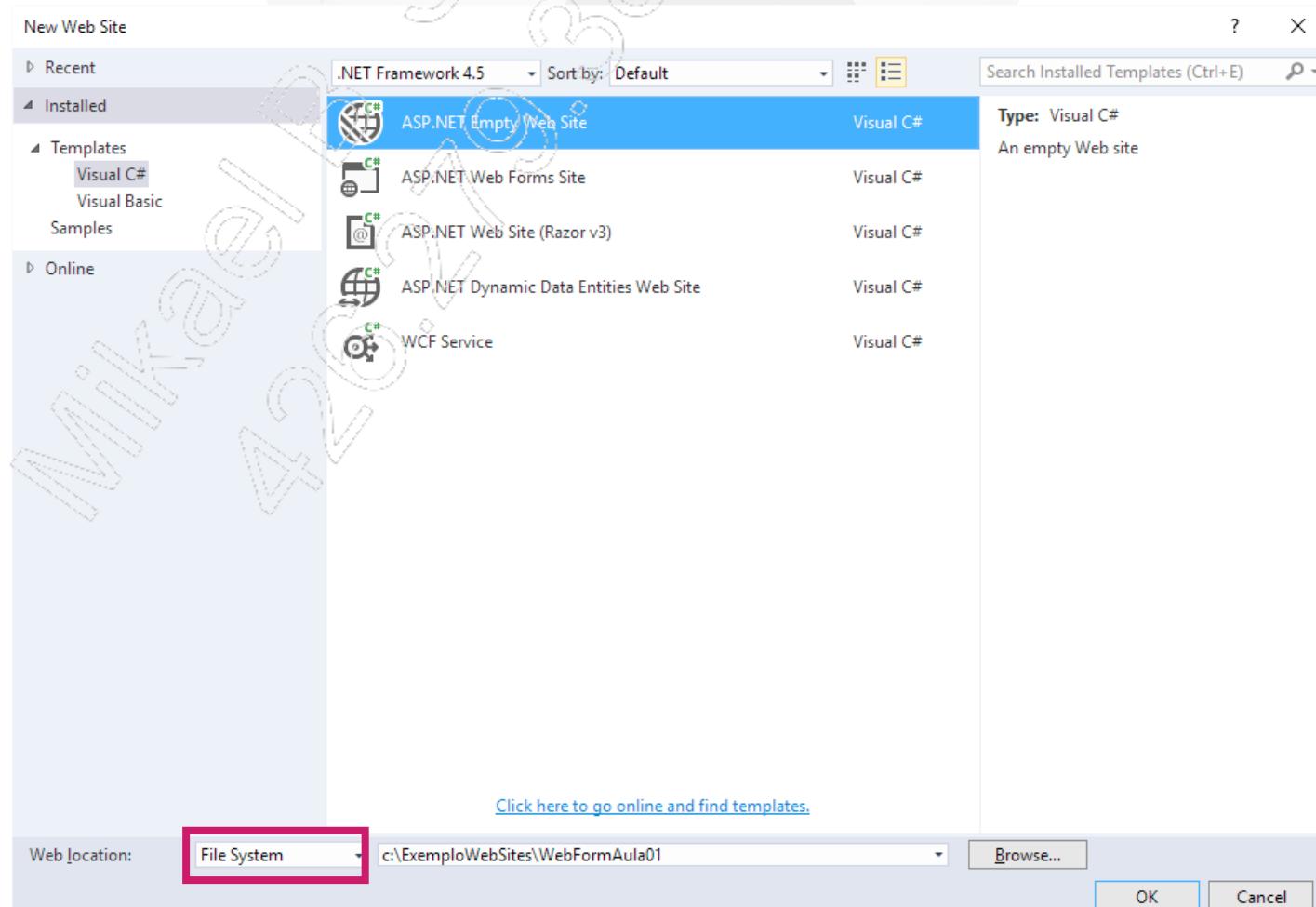
- **ASP.NET Dynamic Data Entities Web Site**

Web site que cria automaticamente páginas para administrar informações em banco de dados.

- **WCF Service**

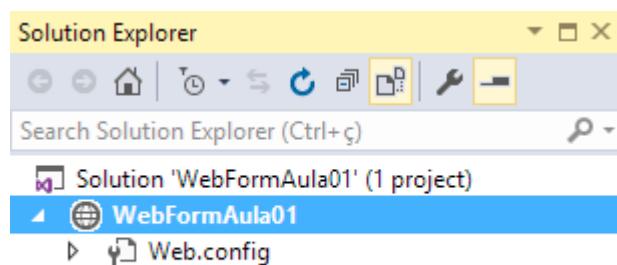
Web site que retorna informações (dados) em formatos diversos.

A parte importante é a maneira que esse Web site será executado. Podem ser usados o IIS, o protocolo FTP ou o sistema de arquivos do Windows. A parte inferior da janela é usada para definir esse item (Web location).

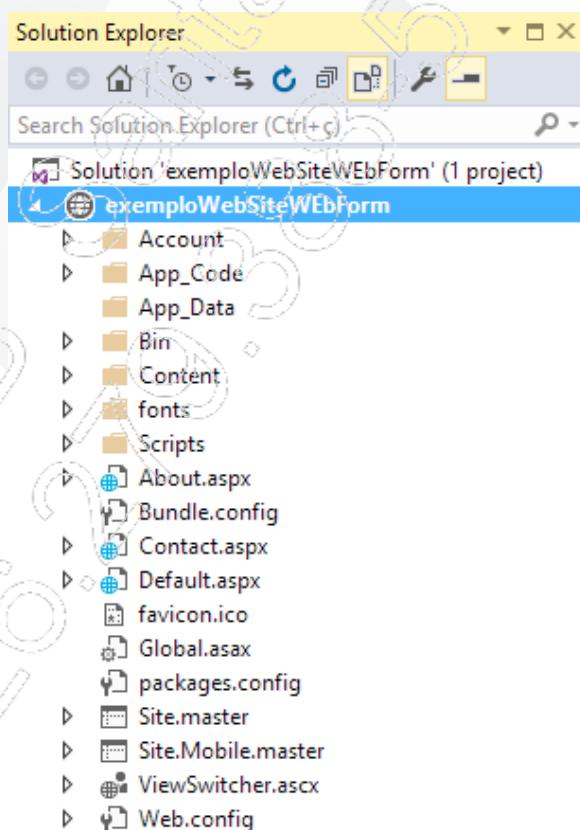


Uma vez criada a aplicação, os componentes internos são os mesmos de um projeto Web. No Solution Explorer, é possível ver todos os arquivos. O Visual Studio cria automaticamente um arquivo de solution.

Veja um exemplo de um Web site vazio:



Veja, agora, um exemplo de um Web site usando um modelo Web Forms:



1.8.1.3. Web project e Web site

O quadro a seguir compara os dois modos de criar aplicações Web no Visual Studio:

Web project	Web site
Um arquivo (xxx.csproj) define os arquivos que fazem parte do projeto.	Uma pasta é definida e todos os arquivos dentro dessa pasta fazem parte do site.
Deve ser compilado e publicado em um servidor Web.	A pasta já é um site; pode ser compilado, mas não é necessário. Funciona se for simplesmente copiado para o servidor.
As classes podem ficar em qualquer lugar do projeto. Como tudo será compilado, a localização física dos arquivos não é importante.	Os arquivos de classes devem ser colocados em uma pasta App_Code para funcionar. Se a pasta for compilada, uma dll será copiada para a pasta bin para publicação.
É perfeito para fazer parte de uma grande solução, pois a adição e a remoção de referências obedecem ao mesmo modelo de programas Windows.	Foge do padrão para múltiplos projetos e suas referências. Pode dar problemas com programas de controle de versão.

De uma maneira geral, o projeto Web é melhor para grandes aplicações e para interação com outros projetos. O Web site é mais apropriado quando o Web site de produção é constantemente atualizado e quanto o projeto é de pequeno porte.

Pontos principais

Atente para os tópicos a seguir. Eles devem ser estudados com muita atenção, pois representam os pontos mais importantes do capítulo.

- Existem diversos tipos de software que atendem às necessidades de gerenciamento de informação, dentre os quais podemos destacar aplicativos desktop, aplicativos Web, serviços e aplicativos mobile;
- A arquitetura cliente-servidor se baseia no conceito de **requisição** de um aplicativo **cliente** e **resposta** de um aplicativo **servidor**. O ambiente Web é um exemplo de arquitetura cliente-servidor;
- O HTTP (HyperText Transfer Protocol) consiste em um protocolo que define a comunicação entre os navegadores Web e os Web sites;
- HTML, CSS e JavaScript são as tecnologias que o navegador entende e que são usadas para criar páginas Web;
- HTML (HyperText Markup Language) é a linguagem utilizada para definir a estrutura de uma página Web;
- CSS (Cascading Style Sheets) é a linguagem utilizada para definir a aparência de uma página HTML;
- JavaScript é uma linguagem de programação cujos programas são executados por um navegador Web e é utilizada para criar interações com o usuário;
- ASP.NET é a parte da plataforma .NET responsável pela criação de Web sites e serviços Web;
- Nos sistemas operacionais da Microsoft, o IIS é o servidor mais utilizado para interceptar solicitações HTTP na porta 80, que é a porta padrão para o recebimento de solicitações HTTP;
- Existem três modos de desenvolver aplicativos Web usando ASP.NET: Web Pages, Web Forms e MVC;

- Web Pages usa o conceito de uma única página em que existe HTML, que é executado no cliente, e scripts, que são executados no servidor. Web Forms usa um modelo de programação orientada a eventos, em que Web Controls são usados para gerar HTML, CSS e JavaScript, encapsulando grande parte da complexidade de criar uma página Web. E MVC (Model-View-Controller) usa um conhecido padrão de desenvolvimento que fornece controle total sobre o HTML gerado no servidor e separa completamente o processamento da página do código-fonte que é interpretado e exibido pelo navegador;
- Existem dois tipos de desenvolvimento Web no Visual Studio: projeto Web e Web site. Um projeto Web é um conjunto de arquivos que precisa ser compilado e publicado em um servidor. Um Web site é uma pasta definida como aplicativo e pode ser simplesmente copiada para um servidor;
- A versão 2015 do Visual Studio utiliza o ASP.NET Core 1.0, que permite criar aplicações multiplataformas.

1

Conceitos básicos

Teste seus conhecimentos

Mikael B
426.279.67



IMPACTA
EDITORA

1. Qual é o tipo de aplicação que é executada usando recursos específicos do sistema operacional (API) em que está sendo executada para gerar a interface com usuário?

- a) Web application
- b) Desktop application
- c) Resources application
- d) System application
- e) Web service

2. Qual tecnologia é utilizada para definir a aparência das telas de uma aplicação Web?

- a) Class Library
- b) HTTPS
- c) CSS
- d) Request
- e) FTP

3. Como se chamam a solicitação de uma aplicação cliente para o servidor e a resposta do servidor?

- a) HTTP e FTP.
- b) Response e Request.
- c) Request e Response.
- d) HTML e CSS.
- e) HTTP e IIS.

4. Qual é o comando (verb) frequentemente utilizado quando dados de um formulário são enviados ao servidor com o objetivo de alterar informações?

- a) GET
- b) DELETE
- c) ALT
- d) HEADER
- e) POST

5. Quais os três modelos de programação disponíveis pelo ASP.NET?

- a) HTML, CSS e JavaScript.
- b) IIS, Request e Response.
- c) Web site, Web app e Web project.
- d) Web Forms, Web Pages e MVC.
- e) WebControls, HtmlControls e UserControls.

1

Conceitos básicos

Mãos à obra!

Mikael B. Soutana
426.279.67



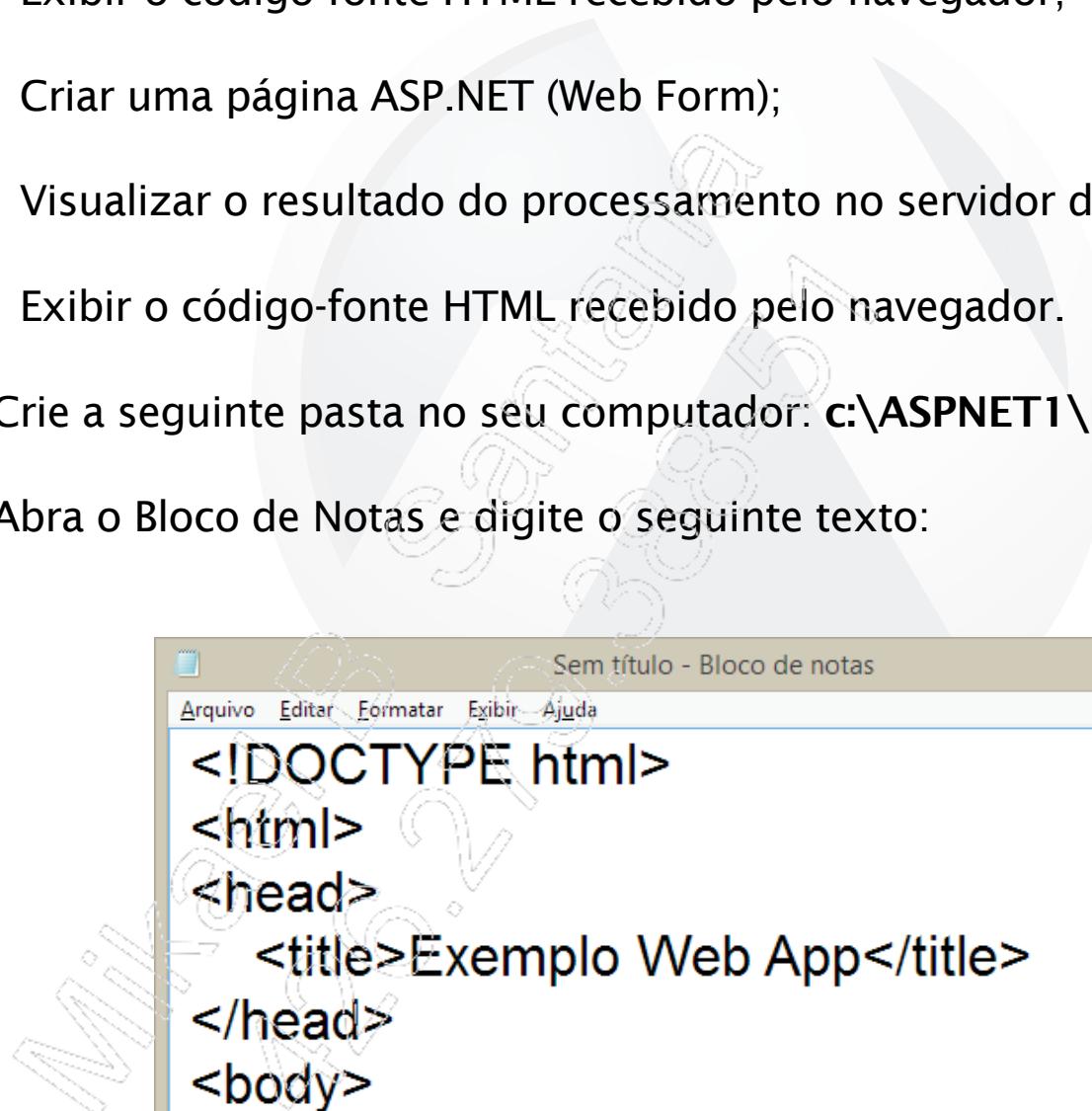
IMPACTA
EDITORA

Laboratório 1

Objetivos:

- Criar um aplicativo ASP.NET usando o IIS;
- Criar uma página HTML;
- Visualizar uma página HTML criada nesse aplicativo;
- Exibir o código-fonte HTML recebido pelo navegador;
- Criar uma página ASP.NET (Web Form);
- Visualizar o resultado do processamento no servidor dessa página;
- Exibir o código-fonte HTML recebido pelo navegador.

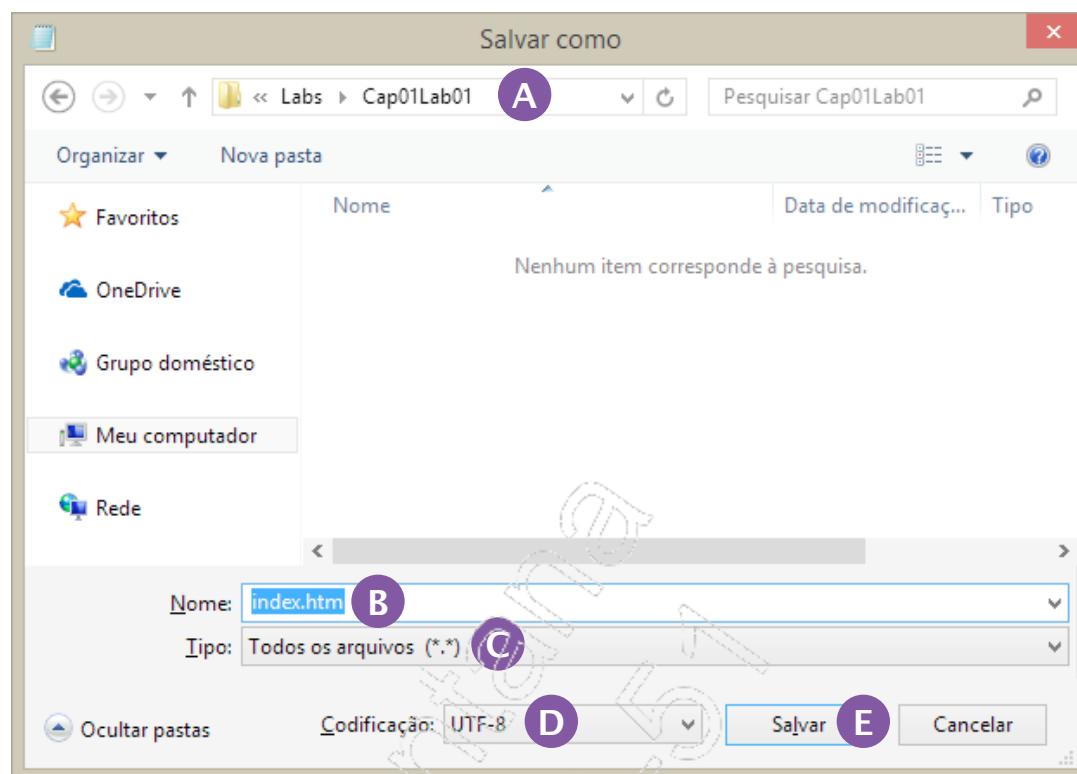
1. Crie a seguinte pasta no seu computador: **c:\ASPNET1\Labs\Cap01Lab01**;
2. Abra o Bloco de Notas e digite o seguinte texto:



```
<!DOCTYPE html>
<html>
<head>
    <title>Exemplo Web App</title>
</head>
<body>
    <h1>Exemplo Web App</h1>

    </body>
</html>
```

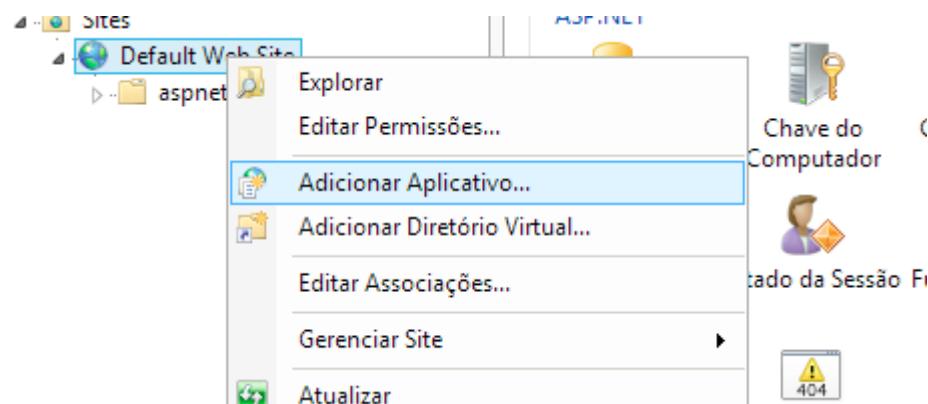
3. Clique em **Arquivo** e em **Salvar** para abrir a janela de gravação. Escolha **index.htm** para o **Nome** do arquivo, **Todos os arquivos (*.*)** para o **Tipo** e **UTF-8** para a **Codificação**. Selecione a pasta criada e clique em **Salvar**;



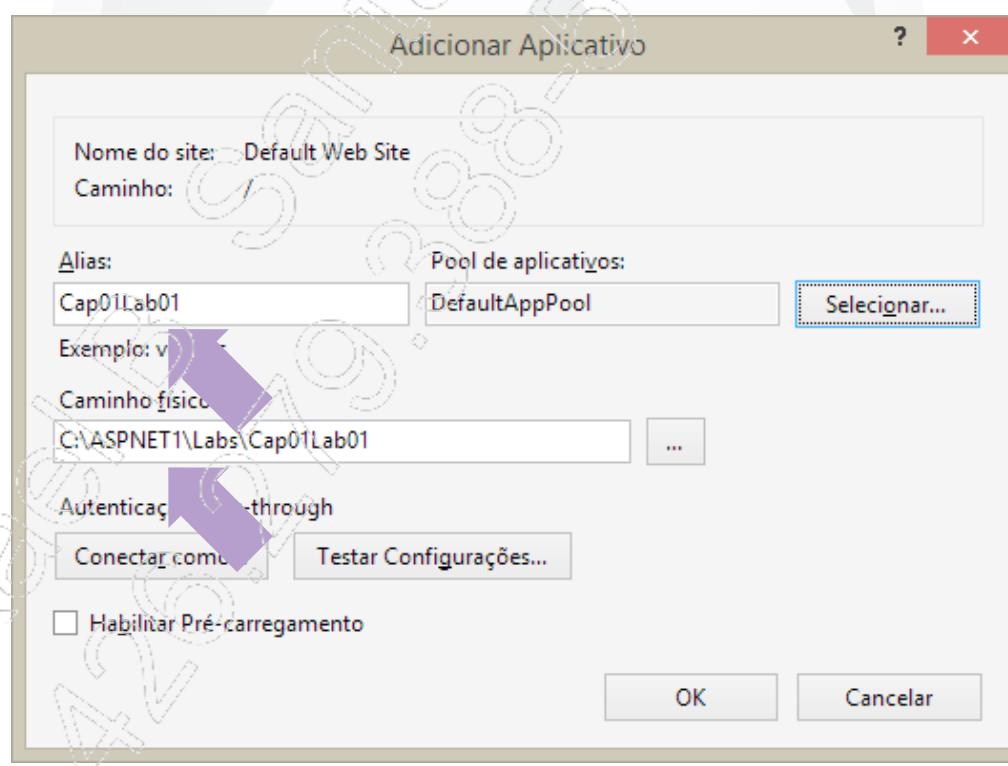
- A – Localização da pasta;
- B – Nome do arquivo. **index.htm** é um nome especial para o Internet Information Services;
- C – Tipo do arquivo. Se não mudar essa opção, o Bloco de Notas vai adicionar a extensão **.txt** ao nome do arquivo;
- D – O padrão de codificação de caracteres padrão no Bloco de Notas é **ANSI**. O padrão **UTF-8** permite exibir corretamente caracteres acentuados e símbolos em qualquer idioma;
- E – Clique em **Salvar** para finalizar.

Visual Studio 2015 - ASP.NET com C# Fundamentos

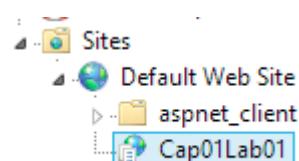
4. Abra o Painel de Controle, o grupo **Ferramentas Administrativas** e o **Gerenciador de Serviços de Informações da Internet (IIS)**. No painel **Conexões**, localize **Sites**, **Default Web Site**, abra o menu de contexto e escolha a opção **Adicionar Aplicativo**;



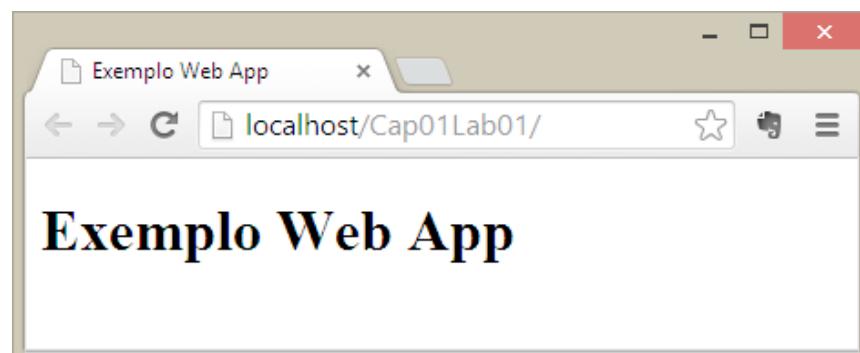
5. Defina o nome virtual do aplicativo (**Alias**) como **Cap01Lab01**, o **Caminho físico** da pasta como aquele que foi criado no início deste laboratório e confirme. Por enquanto, não se preocupe com as outras opções;



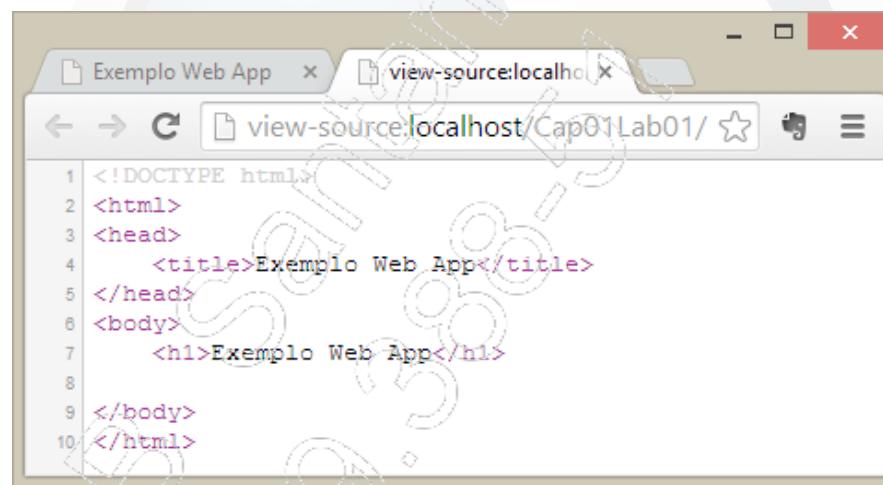
6. O gerenciador do IIS deve ter criado uma pasta com o nome da aplicação Web;



7. Abra o navegador e digite o endereço <http://localhost/Cap01Lab01/> para visualizar a página;



8. Abra o menu de contexto do navegador e escolha **Visualizar Código-Fonte** ou **Exibir código-fonte da página**. Cada navegador pode ter uma mensagem ligeiramente diferente para o comando de exibir o código-fonte enviado pelo servidor;



9. Abra o Bloco de Notas, digite o texto a seguir e salve o arquivo como **teste.aspx** na pasta física da sua aplicação:

A screenshot of a note editor window titled "teste.aspx - Bloco de notas". The menu bar includes "Arquivo", "Editar", "Formatar", "Exibir", and "Ajuda". The main content area contains the following ASPX source code:

```
<!DOCTYPE html>
<html>
<head>
    <title>Exemplo Web App</title>
</head>
<body>
    <form runat="server">
        <h1>Exemplo Web App</h1>
        |   <asp:Calendar runat="server" />
    </form>
</body>
</html>
```

Visual Studio 2015 - ASP.NET com C# Fundamentos

Lembre-se de tomar cuidado com o tipo e a codificação ao gravar com o Bloco de Notas.

10. Visualize o arquivo por meio da aplicação Web criada, usando o endereço <http://localhost/Cap01Lab01/teste.aspx>;



11. Visualize o código-fonte enviado ao browser. Repare que é completamente diferente do código-fonte armazenado no servidor.

The screenshot shows two windows. The main window is a browser displaying the source code of the page. The code includes HTML, CSS, and JavaScript. It features a calendar control with a specific style applied. The second window, titled "teste.aspx - Bloco de notas", contains the original server-side code for the page, which is much more complex and includes event handlers and validation logic.

```
<!--//-->
</script>

<div class="aspNetHidden">
    <input type="hidden" name="__EVENTVALIDATION" id="__EVENTVALIDATION"
    value="/WEdRCJyI3dcLmcUp007e4mQjLqtsaGhrcsf/opWeVPGcs+r3p/nhdP0vEkrK0NtQm/9KOImHPJWSEn1DfpxFg3qzTJ2z
    L82c1HNV36J1N+3UEnrMaIn+DetcrhB7zhNlQgminyxX33fqa{4kx3g9M8sm452f6ICAYUSC2WqCetFafEOGhoTRT1za
    3Trs977CvrlNxR8m/2HsPTU/SdmqfJ3c0c94xrl5px8mng4u78512147bFdx1d8NNHTGm05J638UVNzQw4gVf0imLqslT91
    w4QlnR13GThcr8w@yVliwx655ccrf127o35fllia25v12Bqdspl09071c10cpbV2V5SA/bhch00vjpMcKhpipsf1bdnxCTH
    69t81-dmcRA4UNh4vPb7cshJ7afV7Rbcq14f567XQamewhA8eqghfChvweocoD1z'pqW
    2h0Dcefrw888vhlmn-fcaJucseirhWm8-ubp-vrmnw2t2s12g9yfokcas48m0t8erwv948ALWrdexkpro1m6s5Pw1tct3hN
    ntrm148Kve3s3v12n5K8mV/XkE/v1c14w31ct0f7v1Nw3bp77Us1t8f7v9Pd3x0E0fLsLHm0rN1Alldc2yHtaFxt
    ryKclzUxfXs#3Kg8H2iwcetf12n5K8mV/XkE/v1c14w31ct0f7v1Nw3bp77Us1t8f7v9Pd3x0E0fLsLHm0rN1Alldc2yHtaFxt
    gpc270icUx2dU17ecRjyJsd8#p9gCqhsKcmk4ctpmuKyS0ph#pi/v1kWhntsdu1w1z87E8K0cqg#G1N6CHeKle#CP8Chfc0
    jecogciQg3CTULit/qg#qjxkxkallj)jeyuKetmehoibgokanjalyWd31418XkourBnhQwQV3/49RHC6u#Wrghtms1ngRm#vXW
    kqYFUqg/_jnHah3in/7CMinA=="/>
</div>
<h1>Exemplo Web App</h1>

<table cellspacing="0" cellpadding="2" border="1" style="width:1px; border-collapse:collapse;">
    <tr><td colspan="7" style="background-color:silver;"><table cellspacing="0"
    style="width:100%;border-collapse:collapse;">
        <tr><td style="width:11%;><a href="javascript:_doPostBack('ctl01','V5261')>
        style="color:Black" title="Ir para o mês anterior"><input type="button" value="</a></td><td align="center"
        style="width:70%;>julho de 2014</td><td align="right" style="width:18%;><a href="javascript:_doPostBack('ctl01','V5262')>
        style="color:Black" title="Ir para o mês seguinte"><input type="button" value="</a></td></tr>
        <tr><th align="center" abbr="domingo" scope="col">dom</th><th align="center" abbr="segunda-feira" scope="col">ter</th><th align="center" abbr="terça-feira" scope="col">qua</th><th align="center" abbr="quinta-feira" scope="col">qui</th><th align="center" abbr="sexta-feira" scope="col">sab</th><th align="center" abbr="sábado" scope="col">sab</th><th align="center" abbr="domingo" scope="col">dom</th></tr>
        <tr><td href="javascript:_doPostBack('ctl01','5294')" style="color:Black" title="30 de junho">30</td><td align="center" style="width:11%;><a href="javascript:_doPostBack('ctl01','5295')>
        style="color:Black" title="1 de julho"><input type="button" value="</a></td><td align="center" style="width:70%;>julho</td><td align="right" style="width:18%;><a href="javascript:_doPostBack('ctl01','5296')>
        style="color:Black" title="2 de julho"><input type="button" value="</a></td></tr>
        <tr><td href="javascript:_doPostBack('ctl01','5297')" style="color:Black" title="3 de julho"><input type="button" value="</a></td><td align="center" style="width:11%;><a href="javascript:_doPostBack('ctl01','5298')>
        style="color:Black" title="4 de julho"><input type="button" value="</a></td><td align="center" style="width:70%;>julho</td><td align="right" style="width:18%;><a href="javascript:_doPostBack('ctl01','5299')>
        style="color:Black" title="5 de julho"><input type="button" value="</a></td></tr>
        <tr><td href="javascript:_doPostBack('ctl01','5300')" style="color:Black" title="6 de julho"><input type="button" value="</a></td><td align="center" style="width:11%;><a href="javascript:_doPostBack('ctl01','5301')>
        style="color:Black" title="7 de julho"><input type="button" value="</a></td><td align="center" style="width:70%;>julho</td><td align="right" style="width:18%;><a href="javascript:_doPostBack('ctl01','5302')>
        style="color:Black" title="8 de julho"><input type="button" value="</a></td></tr>
    </table></td></tr>
</table>
```

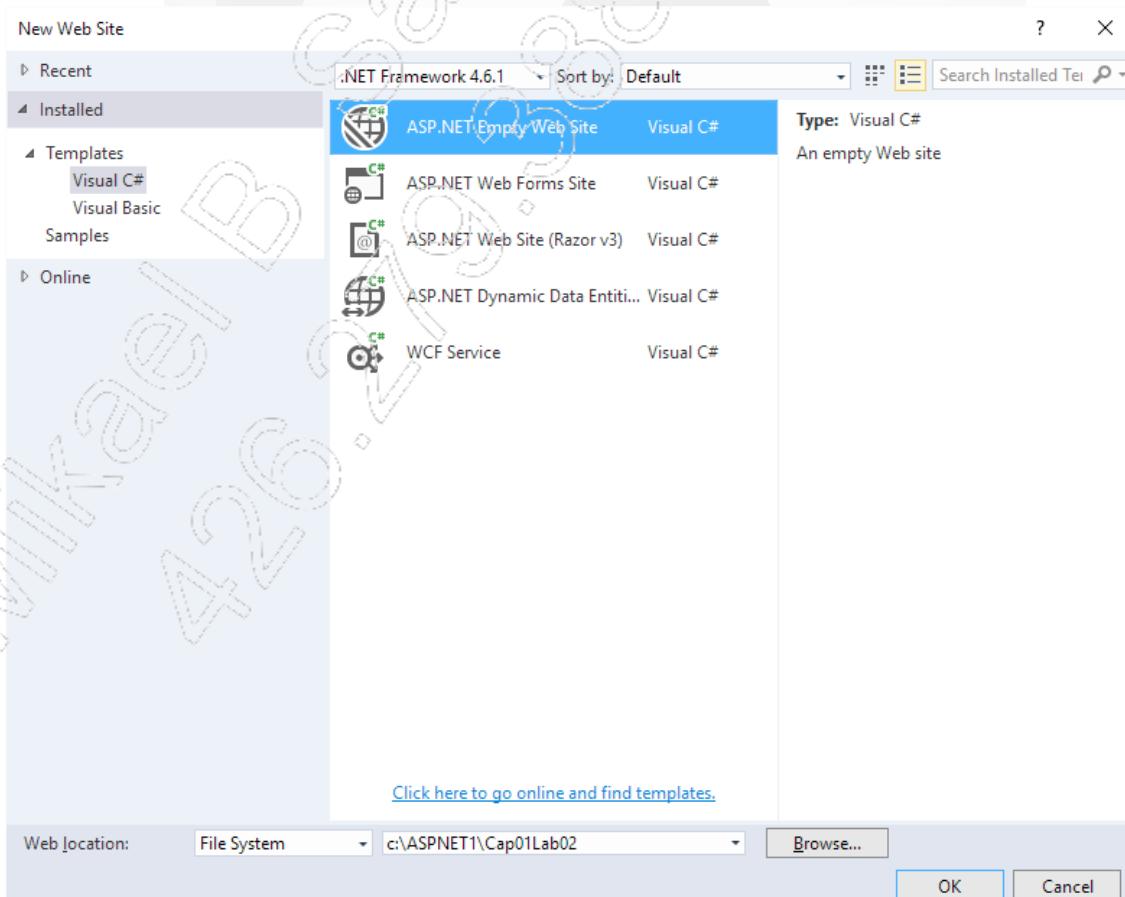
Laboratório 2

Objetivos:

- Criar um Web site usando o Visual Studio;
- Criar uma página **.cshtml** (Web page) no Visual Studio;
- Compilar e visualizar no browser;
- Exibir a data e a hora do carregamento da página usando o mecanismo Razor.

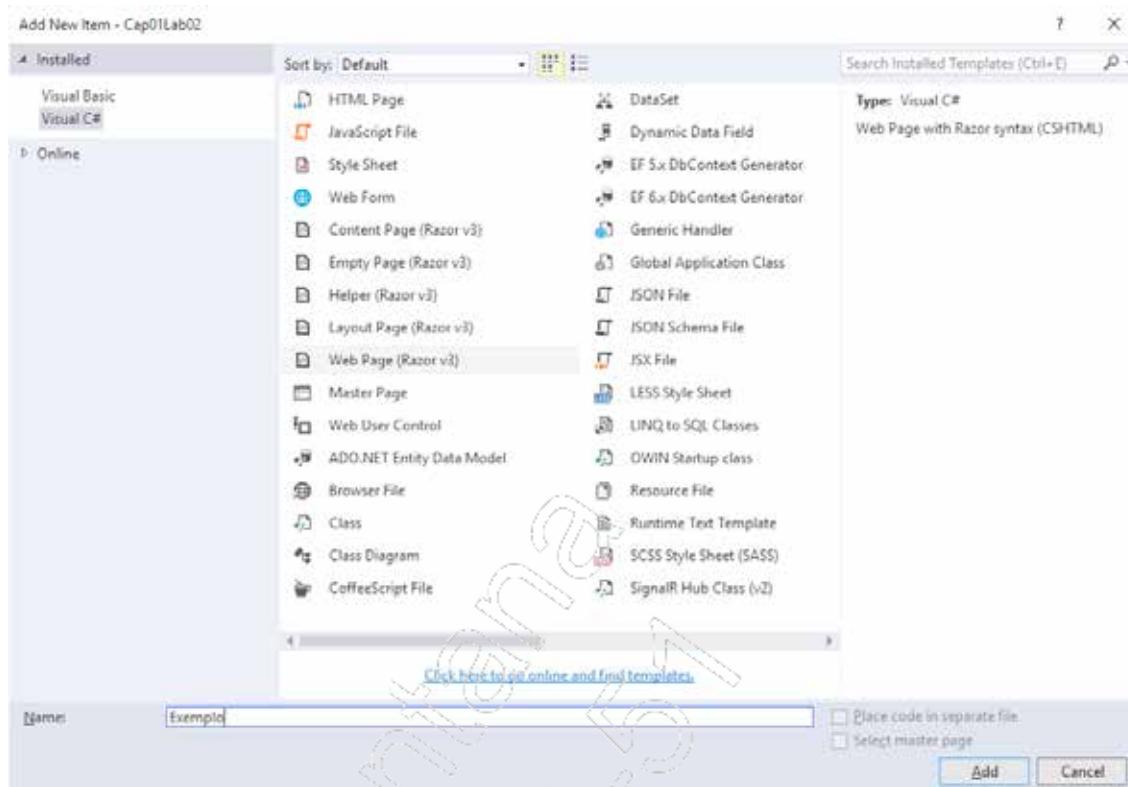
1. Abra o Visual Studio;

2. Crie um Web site de nome **Cap01Lab02**, usando-o na pasta de laboratórios. Para isso, use os comandos **File**, **New Web Site** e digite o nome da pasta, escolha o tipo **File System** e o modelo **Empty Web Site**;



Visual Studio 2015 - ASP.NET com C# Fundamentos

3. No menu **Web Site**, escolha **Add New Item** e escolha **Visual C# e Web Page**. Deixe as outras opções como padrão. Defina o nome como **Exemplo** e clique em **Add** para confirmar:



4. No código HTML gerado, insira o título e a informação de data e hora:

```
<!DOCTYPE html>
<html>
  <head>
    <title></title>
  </head>
  <body>

    <h1>Exemplo Web Page</h1>

    Data e Hora do Servidor: @DateTime.Now

  </body>
</html>
```

5. Teste o código pressionando CTRL + F5.

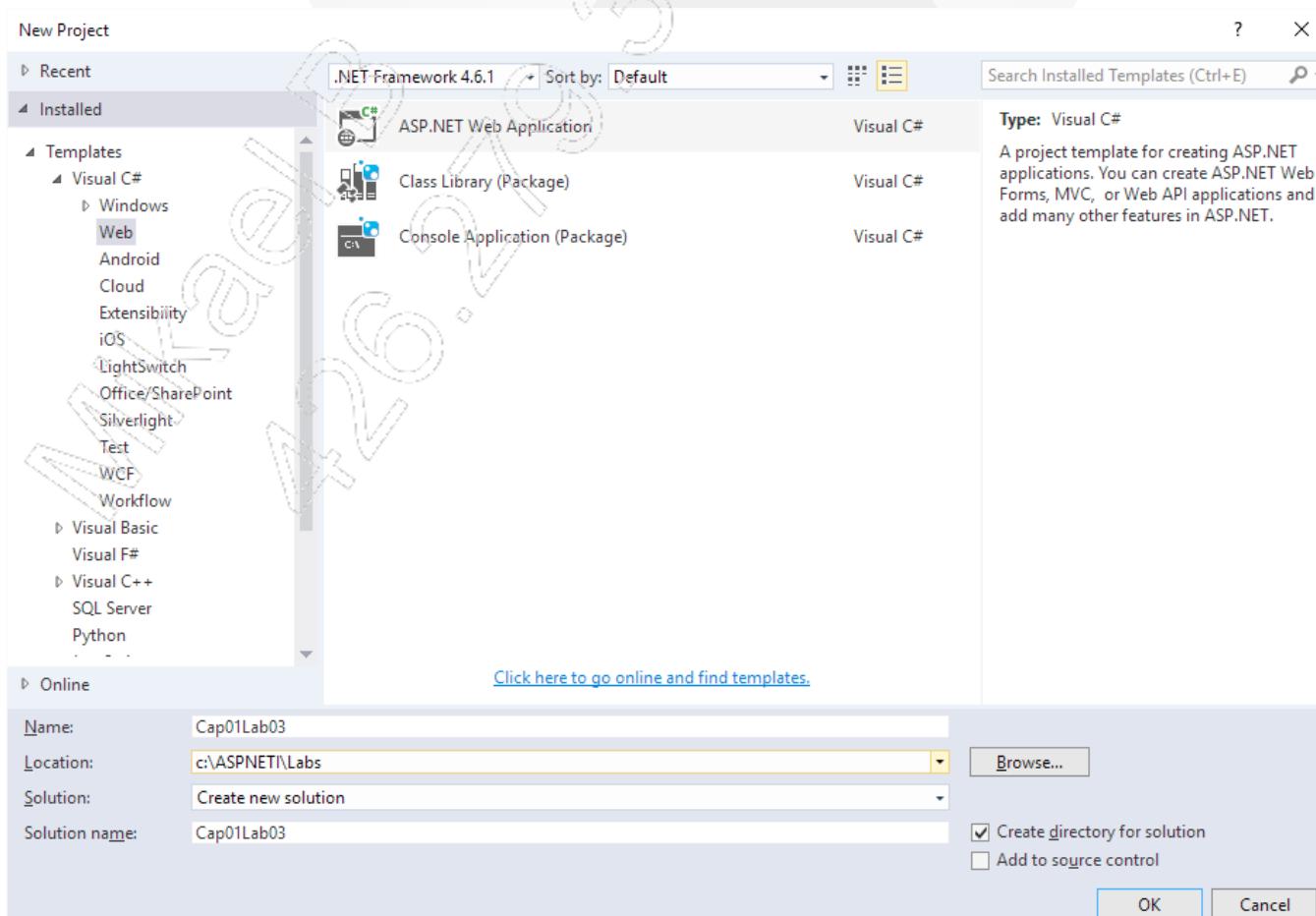
Laboratório 3

Objetivos:

- Criar um projeto Web;
- Criar uma página **.aspx** (Web Form) no Visual Studio;
- Identificar o code-behind;
- Exibir a data e a hora do carregamento da página usando o Web control **Label** e o evento **Page_Load**;
- Compilar e visualizar no browser.

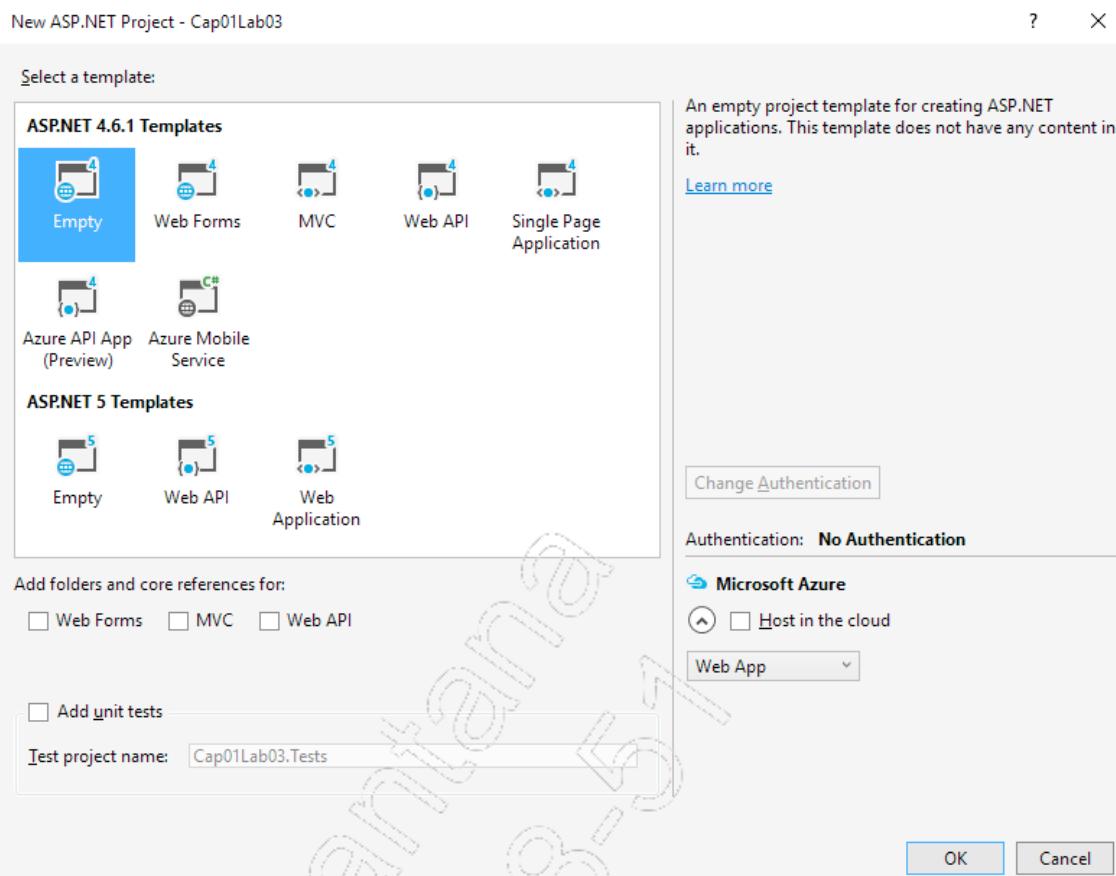
1. Abra o Visual Studio;

2. Crie um projeto Web (não um Web site) de nome **Cap01Lab03**, usando-o na pasta de laboratórios. Para isso, use os comandos **File, New Project**. Na caixa de diálogo, escolha **Templates C#, ASP.NET Web Application**. Escolha **Cap01Lab03** para o nome do projeto e **c:\ASPNET\Labs** para pasta;

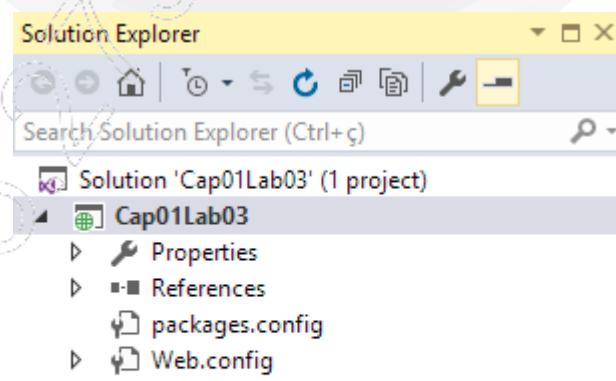


Visual Studio 2015 - ASP.NET com C# Fundamentos

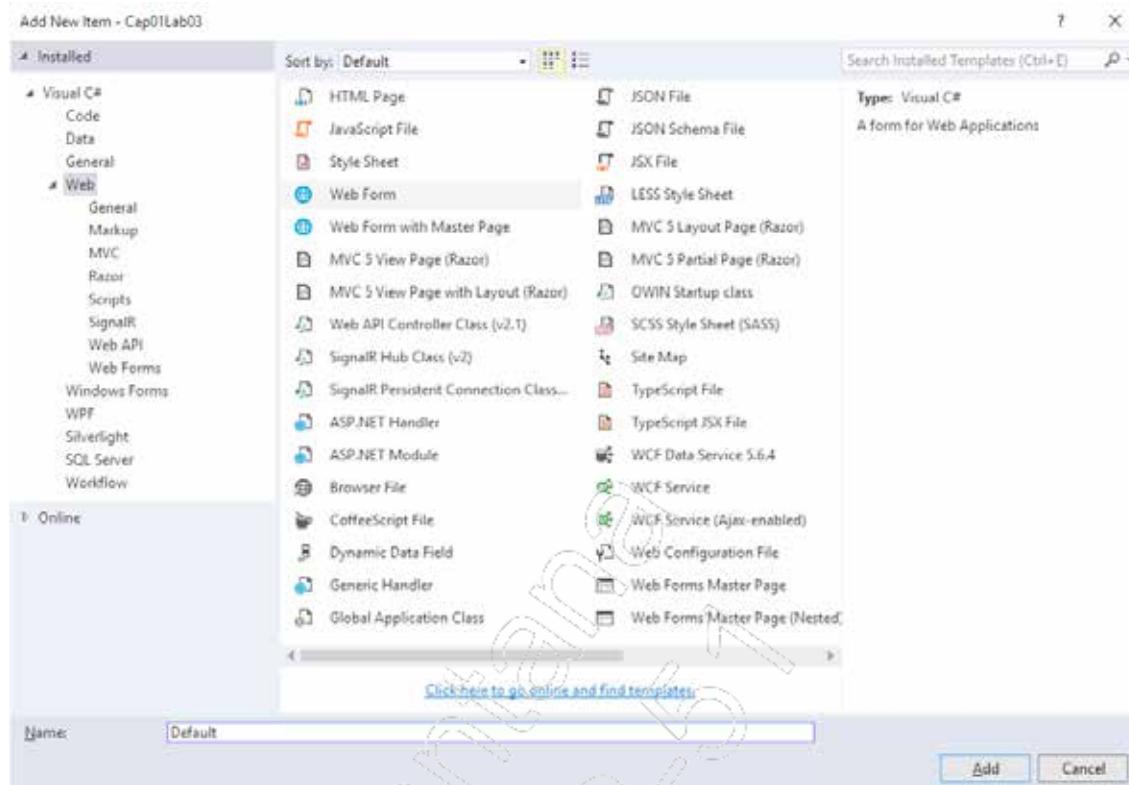
3. No tipo de projeto, escolha **Empty**. Deixe as outras instruções nos valores padrão. Confirme clicando em **OK**;



4. Repare, na janela **Solution Explorer**, que o número de arquivos é maior do que um projeto do tipo Web site:



5. Escolha, no menu **Project**, a opção **Add Item**. Na caixa de diálogo, escolha **Web Form** e nomeie-o como **Default**. Clique em **Add** para confirmar. Deixe as outras opções no padrão;



6. Insira o título e um Web control do tipo Label, após a primeira tag Div:

```
<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="Default.aspx.cs" Inherits="Cap01Lab03.Default" %>

<!DOCTYPE html>

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title></title>
</head>
<body>
    <form id="form1" runat="server">
        <div>
```

<h1>Exemplo de Web Form</h1>

<asp:Label runat="server" ID="mensagemLabel"></asp:Label>

```

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

Visual Studio 2015 - ASP.NET com C# Fundamentos

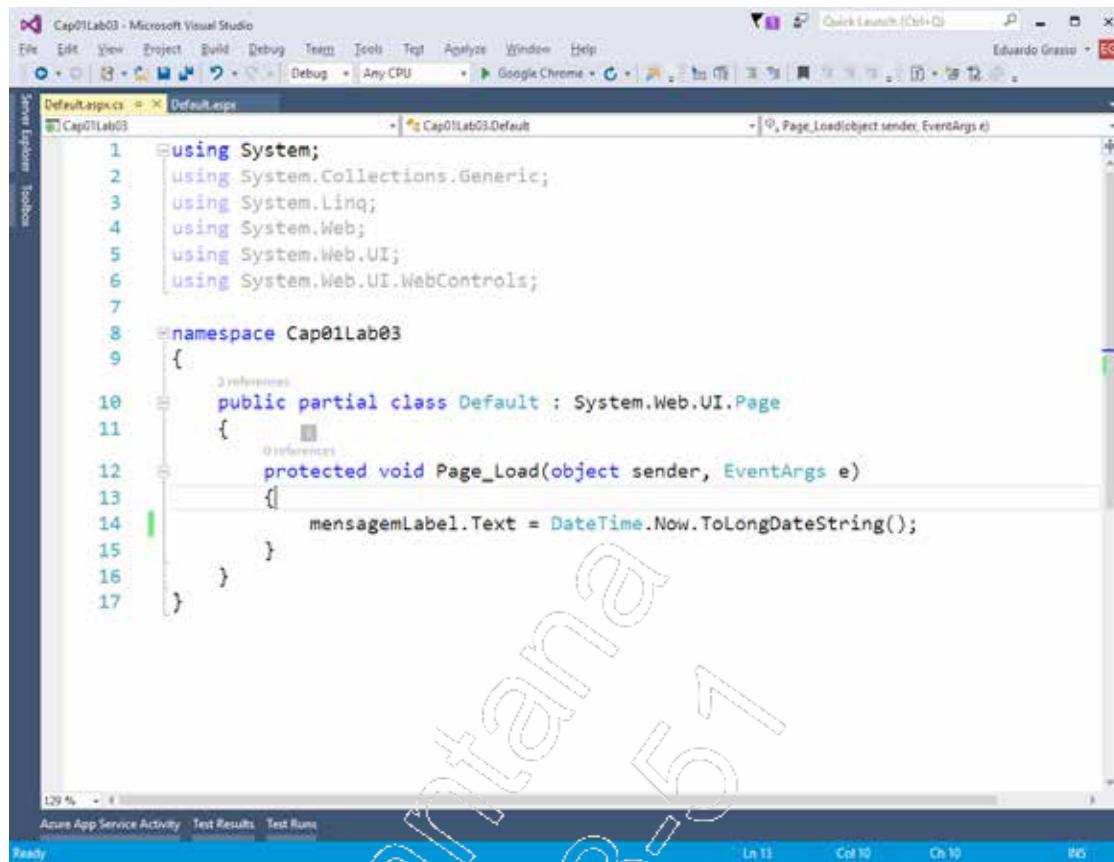
7. Pressione F7 para abrir o código associado a esse formulário. Altere a propriedade Text do label chamado **mensagemLabel** para exibir a data e a hora:

```
namespace Cap01Lab03
{
    public partial class Default : System.Web.UI.Page
    {
        protected void Page_Load(object sender, EventArgs e)
        {
            mensagemLabel.Text = DateTime.Now.ToString("dd/MM/yyyy HH:mm:ss");
        }
    }
}
```

8. Teste o programa pressionando CTRL + F5;

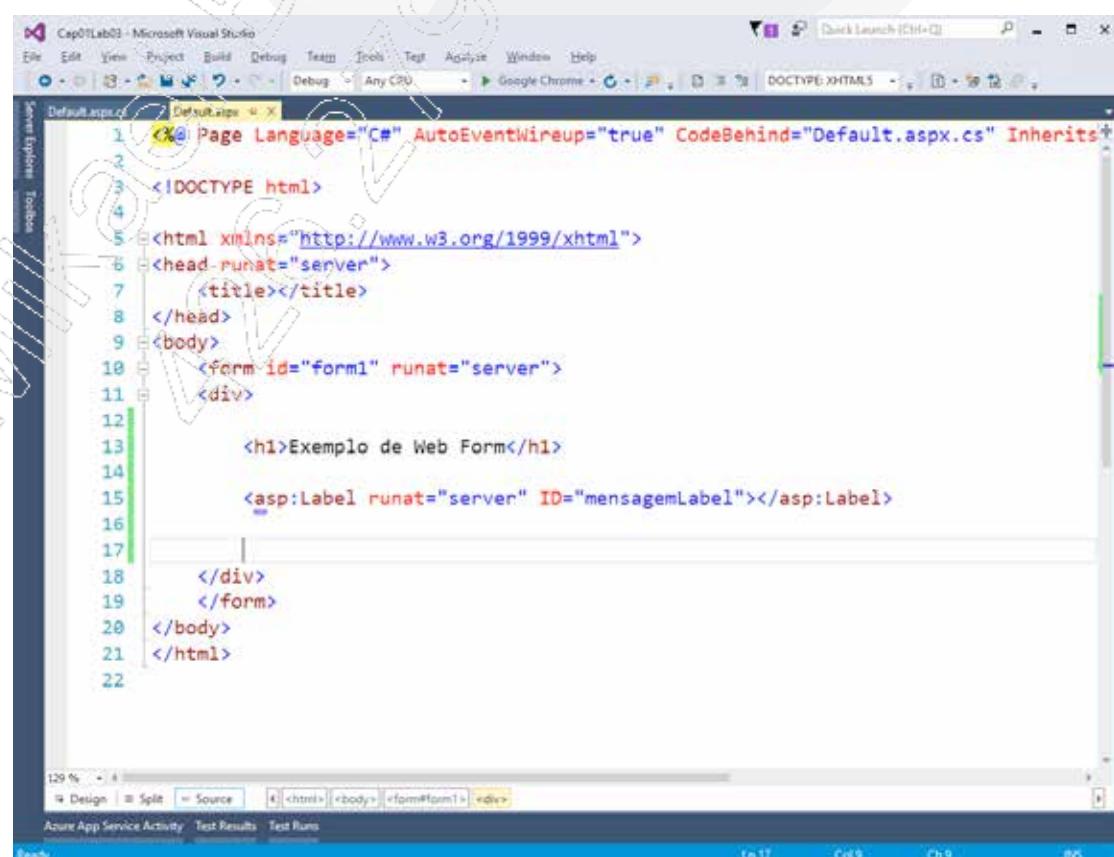


9. Feche o navegador e volte ao Visual Studio. Repare nas abas que permitem navegar por dois arquivos: o código HTML e o code-behind, que é uma classe associada ao HMTL;



```
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Web;
5  using System.Web.UI;
6  using System.Web.UI.WebControls;
7
8  namespace Cap01Lab03
9  {
10     public partial class Default : System.Web.UI.Page
11     {
12         protected void Page_Load(object sender, EventArgs e)
13         {
14             mensagemLabel.Text = DateTime.Now.ToString();
15         }
16     }
17 }
```

10. Usando as guias, abra o arquivo **default.aspx**. Repare nos botões no canto inferior da tela. Permite alternar entre a visualização do código HTML e do code-behind.



```
1 <%@ Page Language="C#" AutoEventWireup="true" CodeBehind="Default.aspx.cs" Inherits="Cap01Lab03.Default" %>
2
3 <!DOCTYPE html>
4
5 <html xmlns="http://www.w3.org/1999/xhtml">
6 <head runat="server">
7     <title></title>
8 </head>
9 <body>
10    <form id="form1" runat="server">
11        <div>
12            <h1>Exemplo de Web Form</h1>
13
14            <asp:Label runat="server" ID="mensagemLabel"></asp:Label>
15
16        </div>
17    </form>
18 </body>
19 </html>
```

Visual Studio 2015 - ASP.NET com C# Fundamentos

```
2
3  <!DOCTYPE html>
4
5  <html xmlns="http://www.w3.org/1999/xhtml">
6  <head runat="server">
7      <title></title>
8  </head>
9  <body>
10     <form id="form1" runat="server">
11         <div>
12             <h1>Exemplo de Web Form</h1>
13         </div>
14     </form>
15     <!-- mensagemLabel -->
16 </body>
17 </html>
```

```
2
3  <!DOCTYPE html>
4
5  <html xmlns="http://www.w3.org/1999/xhtml">
6  <head runat="server">
7      <title></title>
8  </head>
9  <body>
10     <form id="form1" runat="server">
11         <div>
12             <h1>Exemplo de Web Form</h1>
13             <asp:Label runat="server" ID="mensagemLabel"></asp:Label>
14         </div>
15     </form>
16     </body>
17 </html>
```

2

Web Pages

- ✓ Tecnologias utilizadas nas páginas Web;
- ✓ Tecnologias utilizadas no servidor;
- ✓ Web Pages e Razor.



IMPACTA
EDITORA

2.1. Tecnologias utilizadas nas páginas Web

O navegador pode receber, em princípio, três tipos de informação do servidor quando solicita uma página Web: o código HTML, que é responsável pela estrutura do documento; o código CSS, que determina a aparência da página; e o código JavaScript, que permite manipular eventos e elementos da página.

É comum, em desenvolvimento, o uso de bibliotecas (também chamadas de frameworks) para facilitar e tornar mais produtivo o processo de criação de páginas. A programação em JavaScript, por exemplo, demanda uma grande atenção por parte do programador devido a detalhes de sua arquitetura. A biblioteca **jQuery** facilita o trabalho do programador, fornecendo mecanismos inteligentes de seleção, criação e alteração de elementos HTML. Outro exemplo é a biblioteca **Bootstrap**, que resolve diversos problemas de compatibilidade e padronização quando as folhas de estilo (CSS) são utilizadas para criar aplicações que rodam em diferentes dispositivos. Os modelos de projetos do Visual Studio utilizam diversas bibliotecas open source. Existem várias bibliotecas disponíveis para todo tipo de tarefa.

As tecnologias citadas até aqui se referem a o que o navegador recebe do servidor e o que é executado no navegador, ou seja, no lado cliente de uma aplicação client/server. Neste capítulo, veremos um breve resumo dessas tecnologias e o tipo de processamento simples no servidor que são as Web Pages.

2.1.1. A linguagem de marcação HTML

HTML significa **HyperText Markup Language**, ou linguagem de marcação de hipertexto. Um hipertexto é um documento com vínculos, chamados de links, para outros documentos.

A linguagem HTML determina a estrutura do documento. Isso significa informar que uma parte do texto é um título, uma parte é um formulário e outra parte é um rodapé. Isso é feito por meio de marcas (tags). Uma tag indica onde começa e onde acaba um elemento do texto. Por exemplo, a tag **H1** indica o título principal da página. Existem seis níveis de títulos, de H1 até H6. A sintaxe é a seguinte:

```
<h1> Isto é um título </h1>
<h2> Isto é um subtítulo </h2>
```

Algumas tags não podem conter textos internamente. A tag **BR** (linha em branco) é um exemplo. Nesse caso, em vez de abrir e fechar a tag, usa-se uma sintaxe mais curta, desta forma:

```
<br />
```

Em vez desta forma:

```
<br> </br>
```

As tags podem conter **atributos**, que são utilizados para adicionar informação ao elemento HTML. O exemplo para exibir imagens, **img**, necessita do nome e da localização do arquivo. Veja o exemplo:

```

```

Um documento HTML precisa ter, no mínimo, a definição da versão do HTML, uma tag que envolva todo o documento (**html**), uma tag que defina o cabeçalho (**head**), uma tag para o título da página (**title**) dentro dessa tag e uma tag que defina o corpo do texto (**body**). A seguir, um exemplo da página HTML com o mínimo de elementos:

```
<!DOCTYPE html>
<html>
  <head>
    <title></title>
  </head>

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

Visual Studio 2015 - ASP.NET com C# Fundamentos

A seção **head** é utilizada para definir vínculos com outros arquivos, metadados e o título da janela. A tag **body** define a parte visível do documento:

```
<!DOCTYPE html>
<html>
    <head>
        <title>Minha Página</title>
    </head>

    <body>
        <h1>Bem-vindo ao Web site</h1>

    </body>
</html>
```

As tags HTML a seguir são frequentemente utilizadas em um documento:

Tag	Uso	Exemplo
h1...h6	Títulos do documento	<h1>loja online</h1>
p	Parágrafo	<p>Preencha o formulário abaixo:</p>
table, th, tr, td	Tabela	<table> <tr><th>Nome</th></tr> <tr><td>Maria</td></tr> <tr><td>José</td></tr> </table>
a	Vincular documentos	Página 2
div, span, section	Áreas do documento	<div id="texto"> Isto é um exemplo de áreas de um documento</div> <section> Área do login...</section>
form	Formulário	<form method="post" action="processar.aspx">....
input	Entrada de dados	<input type="text" id="nome"> <input type="submit" value="Enviar" />
select	Entrada de dados por meio de uma lista	<select id="cidade"> <option value="1">São Paulo</option> <option value="2">Londres</option> </select>

Tag	Uso	Exemplo
label	Legenda de um campo	<label for="nome">Digite o nome</label>
ul, li	Listas	 Item 1 Item 2
br, hr	Linha em branco e linha preenchida	 <hr />

A seguir, um exemplo de uma página HTML usando os elementos **H1** (título), **P** (parágrafo), **BR** (linha em branco) e **A** (âncora):

```

<!DOCTYPE html>
<html>
<head>
  <title>minha página</title>
</head>
<body>
  <!-- Isto é um comentário -->

  <!-- Um título (h1) -->
  <h1>Bem-vindo ao Curso de ASP.NET</h1>

  <!-- um parágrafo (p) -->
  <p>Isto é um parágrafo. </p>

  <!-- Um HyperLink (a). É necessário o atributo "href" -->
  <a href="pagina.htm">
    Isto é um HyperLink para uma página neste site.</a>

  <!-- linha em branco (br) -->
  <br/>

  <!-- para outro site o caminho deve ser completo -->
  <a href="http://www.microsoft.com">
    Isto é um HyperLink para uma página em outro site</a>

```

```
<!-- linha em branco -->
<br />

<!--
    para abrir em outra janela, usa-se o atributo "target"
-->
<a href="pagina.htm" target="_blank">
    Isto é um HyperLink para uma página em uma nova janela.
</a>

<!-- linha em branco -->
<br />

</body>
</html>
```

2.1.2. Linguagem CSS

É uma linguagem de folha de estilos utilizada para definir a aparência de um Web site. Uma ou mais páginas podem ser vinculadas a uma folha de estilos, e esse processo permite manter no Web site uma uniformidade na identidade visual.

A CSS funciona selecionando elementos e aplicando neles estilos de formatação. Essas regras para definir um ou mais elementos são chamadas **seletor**.

Existem três seletores frequentemente utilizados: **seletor de tipo**, **seletor de classe** e **elemento com ID**. Esses seletores podem ser combinados para gerar especificações de seleção bastante específicas.

- **Seletor de tipo**

Neste seletor, o nome do elemento HTML é utilizado. Todo elemento dentro do documento receberá a formatação definida. No exemplo a seguir, todo elemento H1 da página será exibido com a cor azul:

```
h1 { color:blue; }
```

- **Seletor de classe**

Neste tipo, os elementos com o atributo **class** definido com um determinado nome serão selecionados. No exemplo a seguir, todo elemento com o atributo **class** definido para **texto** será exibido com a cor azul:

```
.texto { color:blue; }
```

O elemento HTML deve estar marcado desta forma:

```
<p class="texto"> ... </p>
```

- **Elemento com ID**

Neste tipo, os elementos com o atributo **id** definido com um determinado nome serão selecionados. No exemplo a seguir, o elemento com o atributo **id** definido para **mensagem** será exibido com a cor azul:

```
#mensagem
{
    color:blue;
}
```

O elemento HTML deve estar marcado desta forma:

```
<p id="mensagem"> ... </p>
```

O que torna os seletores CSS interessantes é a possibilidade de combiná-los de diversas maneiras, para selecionar apenas os elementos desejados. Por exemplo, para selecionar todos os parágrafos dentro de um elemento chamado **conteúdo**, o seletor CSS é este:

```
#conteudo p
{
    color:red;
}
```

No exemplo anterior, os parágrafos contidos em outros elementos não são afetados.

Visual Studio 2015 - ASP.NET com C# Fundamentos

O seguinte exemplo mostra uma página HTML vinculada a uma folha de estilos:

- **exemplo.html**

```
<!DOCTYPE html>

<html>

<head>

    <title>Exemplo</title>

    <link href="estilos.css" rel="stylesheet" />

</head>

<body>

    <h1>Minha página</h1>

    <p class="texto">Teste da folha de estilo aplicada nesta página Web. Este
       parágrafo está definido com o estilo chamado "texto" </p>

    <div id="rodape">
        Esta é uma div chamada "Rodapé"
    </div>

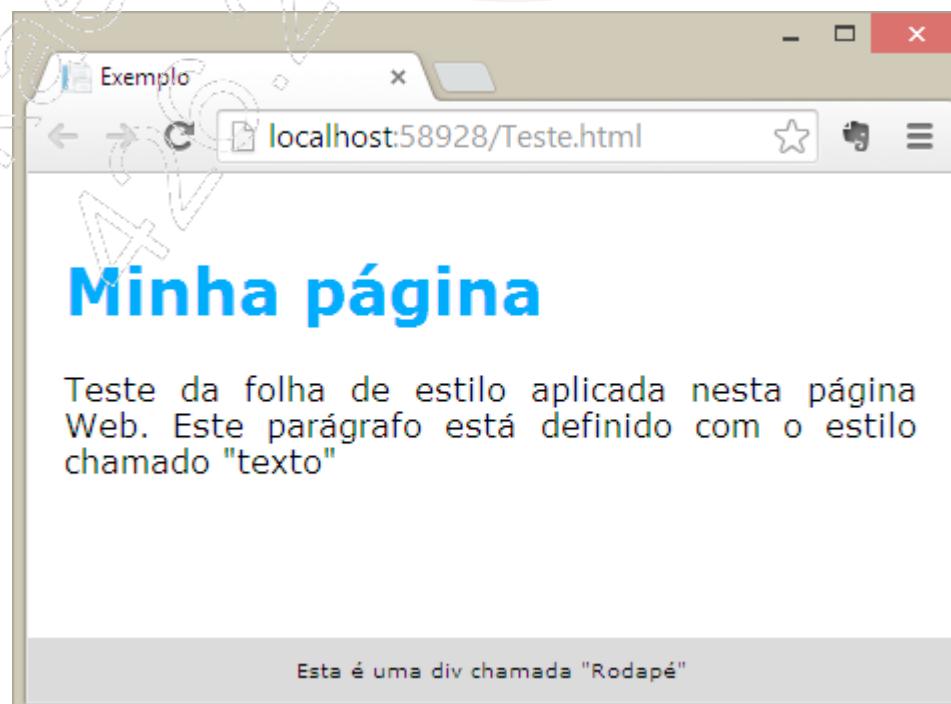
</body>

</html>
```

- **estilos.css**

```
body {  
    font-family:Verdana;  
    padding:10px;  
}  
  
h1 {  
    color:#0094ff;  
}  
  
.texto {  
    text-align:justify;  
}  
  
#rodape {  
    position:absolute;  
    bottom:0px;  
    left:0px;  
    right:0px;  
    background-color:#cecece;  
    padding:10px;  
    font-size:10px;  
    text-align:center;  
}
```

O resultado da página exibida no navegador é o seguinte:



Sem a folha de estilos, a página é exibida desta forma:



2.1.3. Linguagem JavaScript

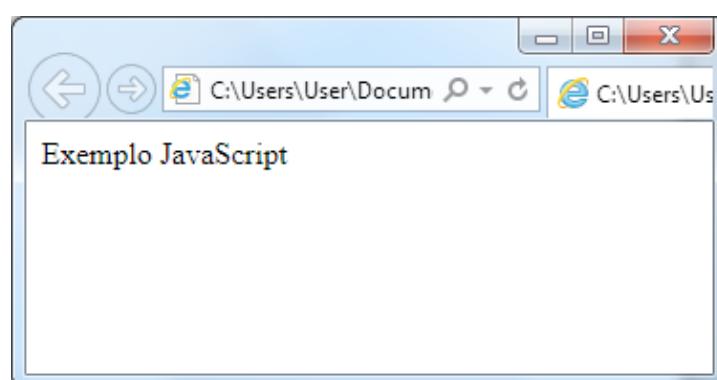
É uma linguagem de script orientada a objeto que pode ser executada em várias plataformas. É leve, concisa e de fácil integração com outras aplicações.

Podemos definir JavaScript como uma linguagem interpretada pelo navegador, assim, é possível criar um script a partir de qualquer editor de texto (Bloco de Notas, por exemplo) e executá-lo em qualquer navegador atual.

2.1.3.1. Noções básicas

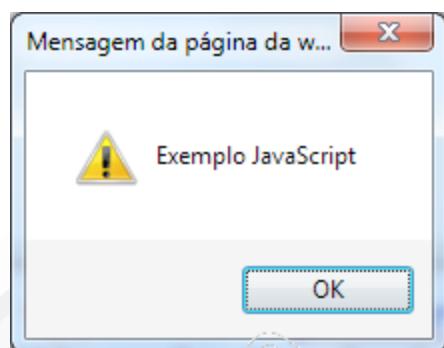
Para criarmos um script, são necessárias as declarações `<script>` no início e `</script>` no final do bloco de instruções.

```
<script type="text/javascript">  
  
    document.write("Exemplo JavaScript");  
  
</script>
```



Para mostrarmos uma mensagem para o usuário, utilizamos a instrução **alert**, de acordo com o exemplo a seguir:

```
<script type="text/javascript">
    alert("Exemplo JavaScript");
</script>
```



2.1.3.2. Estrutura

O JavaScript é uma linguagem fracamente tipada, ou seja, não tem tipos definidos de dados. A tipagem ocorre dinamicamente quando um valor é associado a uma variável. Por exemplo:

```
var x=10
```

A variável **x**, nesse caso, será do tipo numérico inteiro porque o valor informado é um número inteiro.

A estrutura de sintaxe é baseada no uso de funções. Essas funções podem ter outras funções dentro, podem ser passadas por parâmetro e podem ser associadas a variáveis.

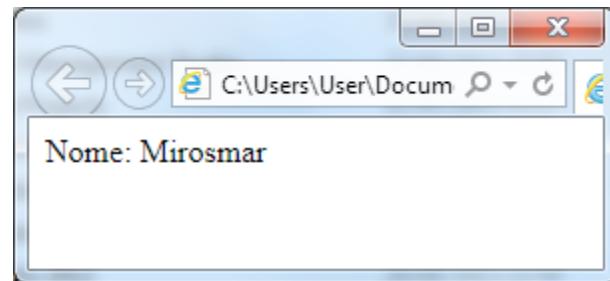
A criação de funções em JavaScript é feita usando a palavra reservada **function**. Veja o exemplo a seguir:

```
<script type="text/javascript">

    function ExibirNome() {
        document.write("Nome: Mirosmar");
    }
}
```

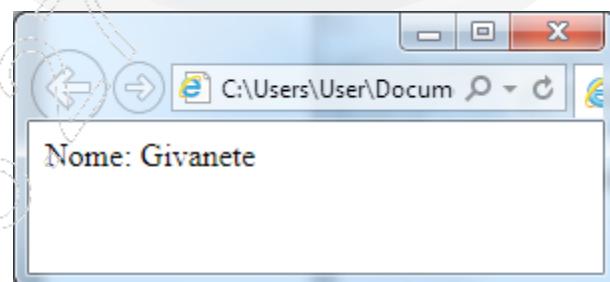
Visual Studio 2015 - ASP.NET com C# Fundamentos

```
//chamando a função  
ExibirNome();  
  
</script>
```



Podemos passar parâmetros nas funções. Veja:

```
<script type="text/javascript">  
  
    function ExibirNome(nome) {  
        document.write("Nome: " + nome);  
    }  
  
    //chamando a função  
    ExibirNome("Givanete");  
  
</script>
```



2.2. Tecnologias utilizadas no servidor

HTML, CSS e JavaScript são executadas pelo navegador Web. O uso de JavaScript para processar as interações do usuário deixa a aplicação mais rápida do que seria se toda a interação tivesse que ser processada pelo servidor. Tarefas tipicamente de servidor são processamento com banco de dados, validação de credenciais e execução de regras de negócio de uma aplicação. As tarefas normalmente executadas pelo navegador são renderização da página, resposta visual à interação do usuário e solicitação de processamento por meio de formulários HTML ou via programação em JavaScript.

O ASP.NET apresenta três tipos de arquitetura para criar páginas em uma aplicação Web: **Web Pages**, **Web Forms** e **MVC**. Neste capítulo, serão abordadas as Web Pages.

2.3. Web Pages e Razor

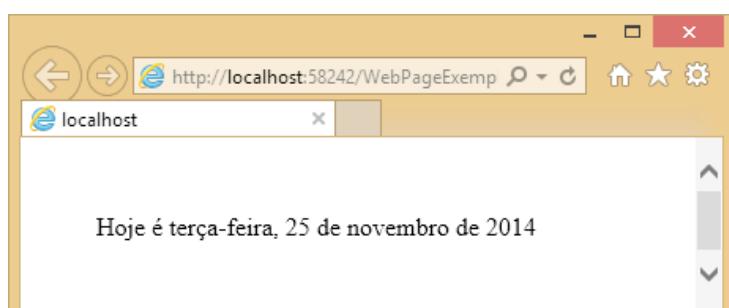
Web Pages são páginas HTML contendo códigos que são executados no servidor antes de serem enviadas para o cliente. A maneira de inserir comandos dentro do documento HTML é conhecida por linguagens como PHP e a primeira versão do ASP. O mecanismo de renderização da página do tipo Web Page é chamado de **Razor**.

Este tipo de componente é chamado de **View Engine**. Frequentemente, a palavra **Razor** é usada, também, para se referir à sintaxe que esse componente utiliza para mesclar HTML e Server-Side Scripts.

O símbolo usado para abrir uma parte do script para ser executada no servidor é o caractere arroba (@). Veja o exemplo:

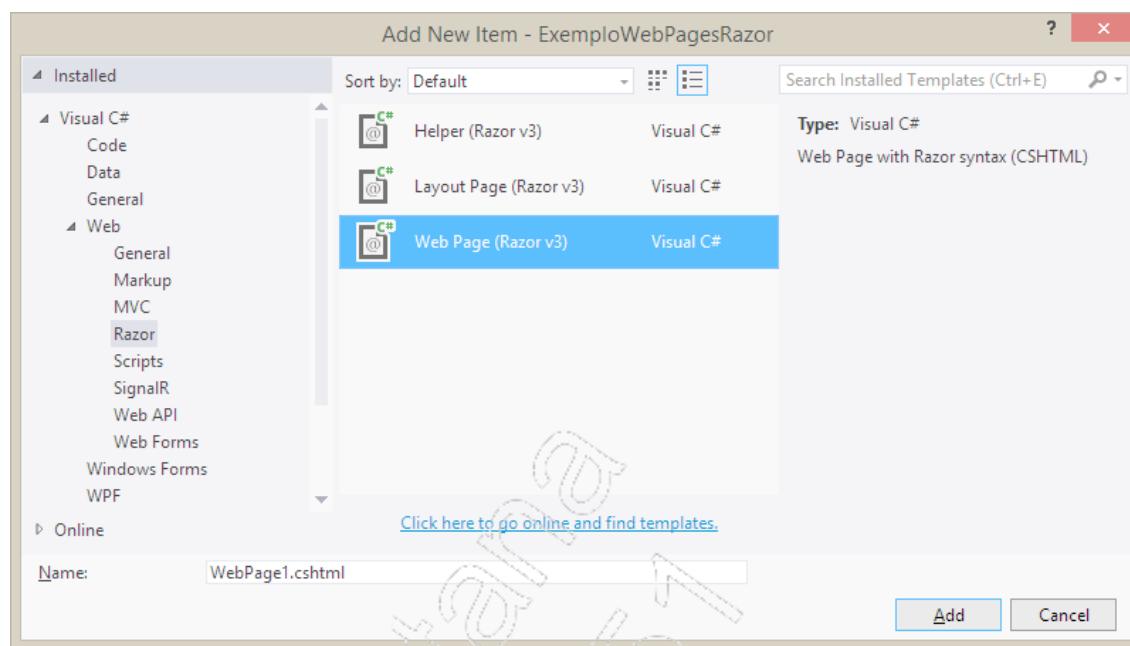
```
<p>Hoje é @DateTime.Now </p>
```

Não existe tag de fechamento. Quando um caractere que não deveria estar no local é encontrado, o mecanismo do Razor entende que terminou a expressão. O resultado enviado para o navegador é o seguinte:



2.3.1. Criando uma Web Page

Em qualquer aplicação Web, para adicionar uma Web Page, escolha, por meio do comando do menu **Project, Add New Item**, a opção **Web Page**.



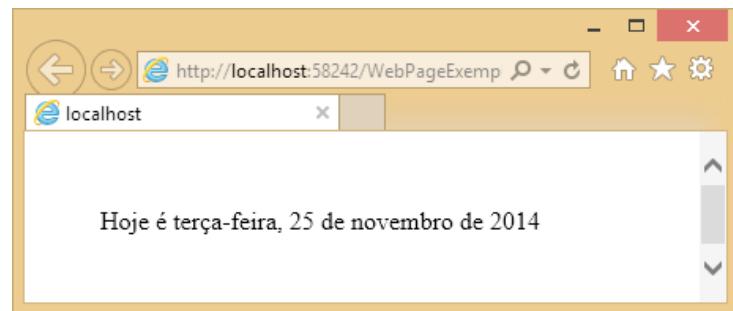
O tipo principal é uma Web Page (.cshtml) que foi criada com o objetivo de ser a mais simples possível. Uma página típica não tem nenhum caractere que uma página HTML não teria.



Os dois tipos de estruturas disponíveis na sintaxe do Razor são o caractere arroba (@), para indicar um código em uma linha, e as chaves ({}), para indicar um bloco de código.

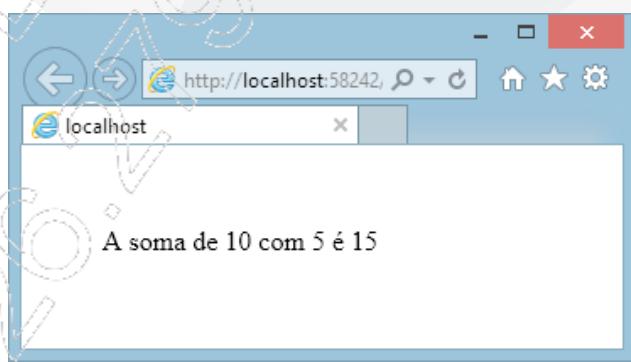
Veja um exemplo de uso do caractere arroba (@) em uma linha:

```
Hoje é @DateTime.Now
```



Veja, agora, um exemplo de bloco de código usando chaves ({ }):

```
@{  
    var x = 10;  
    var y = 5;  
}  
  
A soma de @x com @y é @(x+y)
```

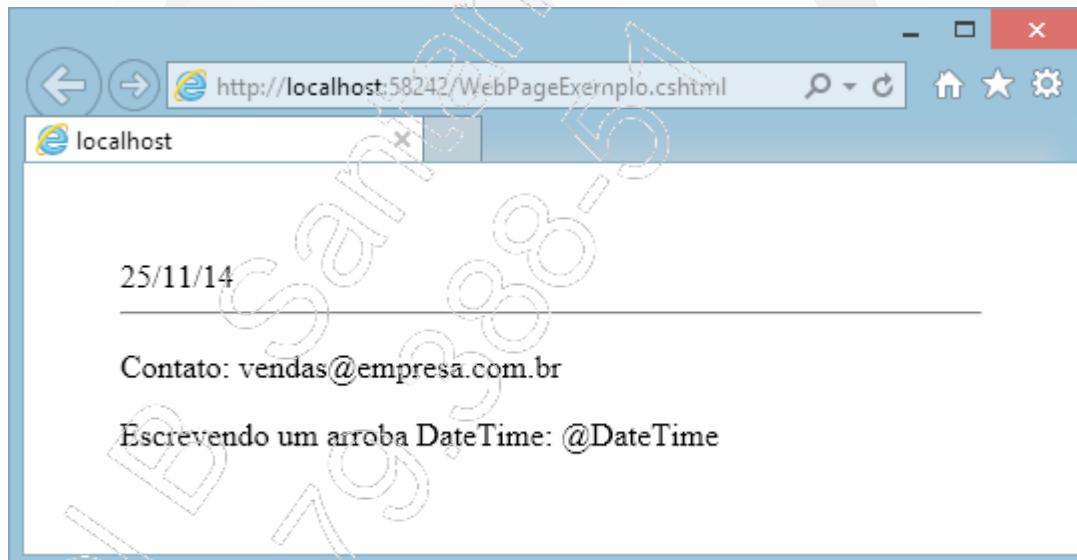


Em alguns casos, o próprio mecanismo de interpretação do Razor percebe quando um sinal de arroba não é um comando, por exemplo, quando fizer parte de um e-mail. Esse recurso não é infalível, mas funciona em quase todos os casos. Se quiser, porém, escrever o sinal de arroba e, em seguida, uma palavra que seja um comando válido e não quiser que o Razor interprete como comando, basta inserir dois sinais de arroba.

O exemplo adiante mostra um sinal de arroba dentro de um e-mail, portanto, automaticamente não será interpretado como comando, e um conjunto de dois sinais de arroba que será escrito literalmente na página, também não sendo interpretado:

```
@DateTime.Now.ToShortDateString()  
<hr/>  
<p>  
    Contato: vendas@empresa.com.br  
</p>
```

Escrevendo um arroba DateTime: @@DateTime



2.3.2. Loops

Razor não é uma linguagem de programação e sim uma view engine que utiliza qualquer linguagem, contanto que exista um plugin disponível. Usando C#, a sintaxe para criar loops é a seguinte:

- Loop **for**

```
@for (int i = 1; i <= 10; i++)  
{  
    <p>  
        O valor de i é  
        <span>@i</span>  
    </p>  
}
```

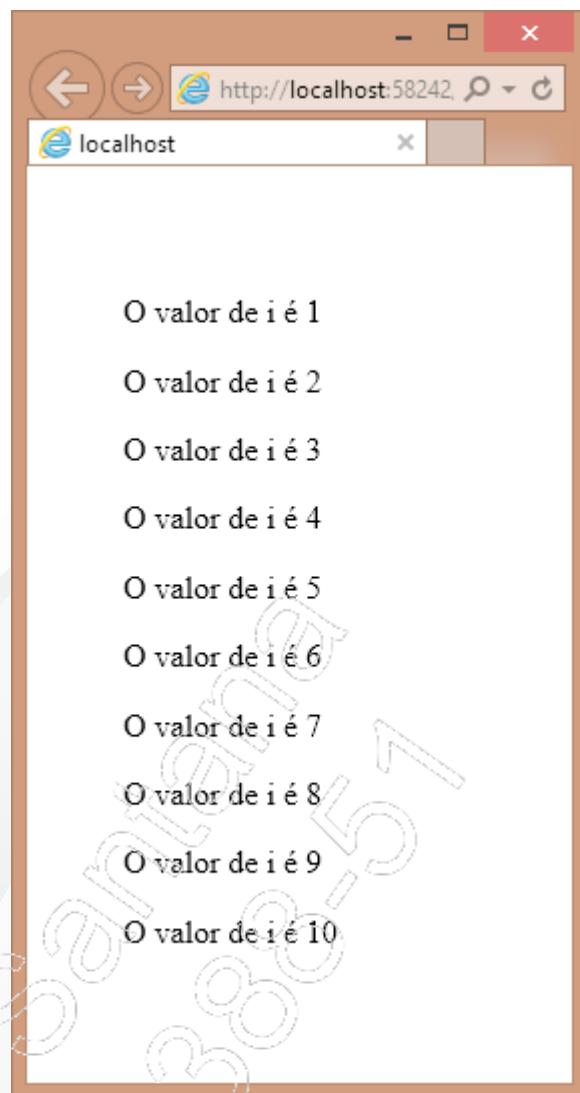
- Loop **foreach**

```
@{  
    var numeros = new int[] { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };  
    foreach (int i in numeros)  
    {  
        <p>  
            O valor de i é  
            <span>@i</span>  
        </p>  
    }  
}
```

- Loop **while**

```
@{ int i = 1;  
    while (i <= 10)  
    {  
        <p>  
            O valor de i é  
            <span>@i</span>  
        </p>  
        i = i + 1;  
  
    }  
}
```

Em todos os casos, o valor do resultado no browser é o mesmo. Veja:



É interessante notar que não existe uma regra para terminar um trecho de código e iniciar um em HTML. O Razor é inteligente o suficiente para deduzir quando é um código de servidor e quando é HTML. No exemplo anterior, o fato de existir um parágrafo (`<P>`) no meio do código indica que é HTML e, ao encontrar chaves (`{}`) ou uma expressão, ele retorna para o código. É claro que, às vezes, é necessário intervir nessa interpretação, como o caractere arroba (@) colocado para exibir a variável `i`, mas é muito menos do que ter que definir isso todas as vezes, como ocorre com as outras tecnologias de servidor.

2.3.3. Estruturas condicionais

Uma das melhores características do Razor é a simplicidade das suas regras de sintaxe, que são apenas duas: um caractere arroba (@) inicia um processamento no servidor em uma linha ou expressão, e um par de chaves (`{ }`) inicia o processamento no servidor em um bloco de texto.

Uma vez indicada uma dessas duas marcas, o processamento do Razor View Engine entra em ação e é inteligente o suficiente para saber quando algo é HTML ou linguagem de programação.

Vejamos, a seguir, suas estruturas condicionais:

- Estrutura **if**

```
@if (DateTime.Now.DayOfWeek == DayOfWeek.Friday)
{
    <span>
        Hoje é sexta-feira.
        Agora são @DateTime.Now.ToShortTimeString()
    </span>
}
```

- Estrutura **switch**

```
@switch (DateTime.Now.Hour)
{
    case 12:
    case 13:
        <p>Hora do Almoço</p>
        break;
    case 18:
    case 19:
        <p>Hora do Jantar</p>
        break;
}
```

2.3.4. Render e Layout

Normalmente, em uma aplicação Web, grande parte do código HTML é compartilhada entre todas as páginas: cabeçalho, menus, rodapés. Esses elementos podem ser isolados em páginas separadas e inseridos em outra página por meio do método **RenderPage()**.

- **RenderPage()**

O método **RenderPage()** insere um arquivo externo na posição do comando.

- Arquivo: **exemplo.cshtml**

```
<!DOCTYPE html>
<html>
  <head>
    <title>Exemplo</title>
  </head>
  <body>

    @RenderPage("titulo.cshtml")

    <p>Bem vindo ao site</p>

    @RenderPage("rodape.cshtml")

  </body>
</html>
```

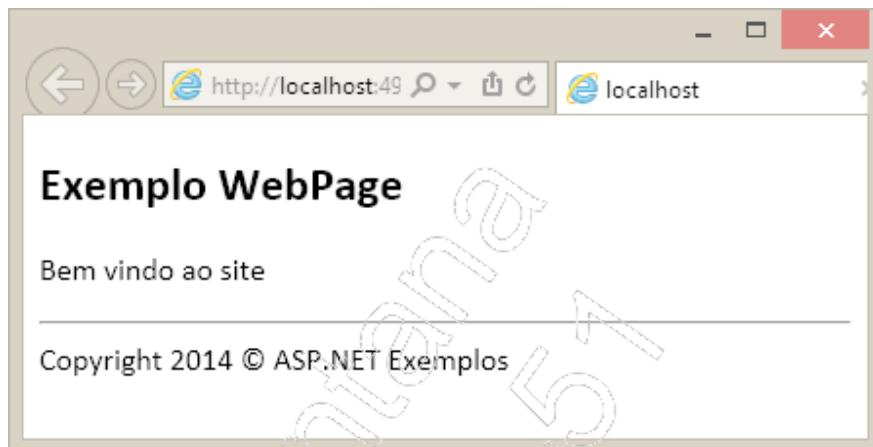
- Arquivo: **titulo.cshtml**

```
<h1>Exemplo WebPage</h1>
```

- Arquivo: **rodape.cshtml**

```
<hr/>
Copyright @DateTime.Now.Year &copy; ASP.NET Exemplos
```

Ao executar a página, o Razor View Engine monta os três arquivos em um só, que é renderizado para ser enviado ao browser. Veja:



Olhando o código-fonte que foi enviado ao navegador, é possível perceber que não foi inserido nada além do conteúdo dos arquivos. Veja:

```
<!DOCTYPE html>
<html>
  <head>
    <title></title>
  </head>
  <body>
    <h2>Exemplo WebPage</h2>

    <p>Bem vindo ao site</p>

    <hr/>Copyright 2014 &copy; ASP.NET Exemplos

  </body>
</html>
```

Se estiver executando o Visual Studio no modo Debug, pode haver a inserção de um módulo JavaScript chamado **BrowserLink**, que é responsável por atualizar o navegador quando há uma alteração no arquivo fonte. Quando em modo de produção, no modo **Release**, esse módulo não será inserido.

O método **RenderPage()** não é a melhor opção para garantir um layout consistente em um aplicativo Web com muitas páginas. Em todas as páginas do aplicativo, seria necessário incluir uma chamada ao método para renderizar o título e outra para renderizar o rodapé. Se a estrutura padrão mudasse, por exemplo, se fosse incluído um navegador, todas as páginas teriam que incluir uma renderização para o arquivo do menu.

O conceito de **Layout** resolve o problema de padronização em Web Pages, do mesmo modo que as Master Pages nos Web Forms.

- **RenderBody()**

O método **RenderBody()** renderiza uma página configurada como página de conteúdo dentro de uma página configurada como página de layout.

Para ser uma página que sirva de modelo para as outras não tem nenhuma configuração especial, basta ter uma chamada ao método **RenderBody()**. A página que será renderizada deve indicar esta página de layout como primeira declaração.

- Página modelo: **_Layout.cshtml**

```
<!DOCTYPE html>
<html>
  <head><title>Minha App</title></head>
  <body>
    <h1>Tituto Minha App</h1>

    @RenderBody()

    <hr/>
    <p>Desenvolvido para o curso de ASP.NET(c) 2014 </p>
  </body>
</html>
```

- Página que utiliza o modelo: **exemplo.cshtml**

```
@{  
    Layout = "~/Layout.cshtml";  
}  
  
<p>  
Aqui é o corpo da página que será renderizado  
pelo método RenderBody() da página de Layout...  
</p>
```

Não é obrigatório utilizar o arquivo iniciando com o caractere sublinhado (_), mas é uma prática comum, porque os arquivos que iniciam dessa forma não podem ser visualizados pelo usuário, e o arquivo que serve de modelo geralmente é usado apenas para esse propósito, portanto não teria sentido isolá-lo.



- **RenderSection()**

O método **RenderSection** renderiza o conteúdo de uma seção nomeada. Este método deve estar dentro de um arquivo de layout. O primeiro parâmetro deste método é o nome da área nomeada, e o segundo é se esta área é opcional. Se no segundo parâmetro for marcado **false**, a página não precisará ter uma área nomeada.

- Página modelo: `_Layout.cshtml`

```
<!DOCTYPE html>
<html>
  <body>
    <p>Texto do Arquivo Layout....</p>

    @RenderBody()

    <p>Mais texto do arquivo de Layout....</p>

    @RenderSection("Mensagem", false)

  </body>
</html>
```

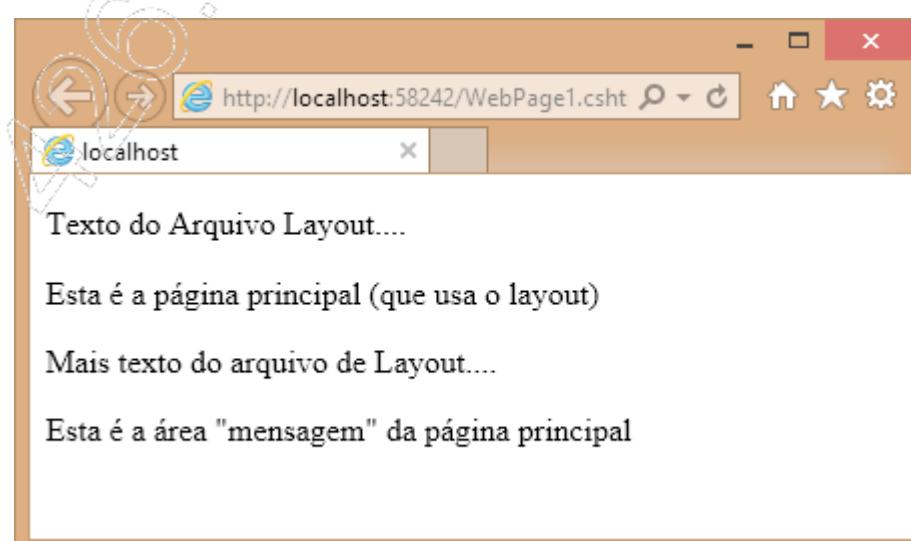
- Página que usa o modelo: `WebPage.cshtml`

```
@{
  Layout = "_Layout.cshtml";
}

<p>Esta é a página principal (que usa o layout)</p>

@section mensagem{
  <p>Esta é a área "mensagem" da página principal</p>
}
```

Veja o resultado ao ser renderizado:



O fato de podermos definir se a renderização é obrigatória ou não é muito importante, porque, se fosse sempre obrigatório, todas as páginas teriam de ter uma seção nomeada. Dessa forma, se a página não tem uma seção com o nome definido, o método não é executado e não provoca nenhum erro.

2.3.5. O caminho da renderização da página

O caminho percorrido desde a solicitação da página pelo aplicativo cliente até a disponibilização desta pelo servidor passa por diversas etapas. As principais são as seguintes:

- **_AppStart.cshtml**

Se for criada uma página com este nome e a extensão **.cshtml** ou **.vbhtml**, o ASP.NET irá executá-la antes de qualquer coisa. Esse arquivo deve ser colocado na raiz do site. É comum usar esse arquivo para definir o arquivo de layout.

- **_PageStart.cshtml**

Dentro de uma pasta, se houver um arquivo com este nome, ele será executado antes dos outros.

2.3.6. Request e Form

Grande parte do sucesso do protocolo HTTP e da linguagem HTML é a versatilidade na maneira de transmitir e receber informações pela Internet. As Web Pages disponibilizam propriedades parecidas com as dos Web Forms para manipular e/ou consultar informações vindas do navegador.

- **IsPost e Request**

IsPost é a propriedade herdada da classe **WebPage** que retorna se a requisição for de um formulário com o método **POST**.

A propriedade **Request** retorna os detalhes da requisição, e os dados do formulário podem ser obtidos por meio da propriedade **Request.Form**.

```
<!DOCTYPE html>
<html>
    <head>
        <title></title>
    </head>
    <body>
        <form action="" method="post">
            <label>Nome:</label>
            <input name="nome" />
            <button type="submit">Enviar</button>
        </form>

        @if(IsPost)
        {
            var nome=Request.Form["nome"];
            <p>Bem vindo, @nome</p>
        }
    </body>
</html>
```

Os dados obtidos pela coleção **Form** podem ser obtidos pela coleção indexadora da classe **Request**:

```
var nome=Request.Form["nome"];
```

pode ser:

```
var nome=Request["nome"];
```

Quando é usada a propriedade indexadora, o item **nome** não é procurado apenas no **Form**, mas também no **QueryString** e **Cookies**. É importante ter cuidado com essa característica, principalmente quando existe a alteração de dados.

Procurar em todas as coleções torna o programa mais vulnerável, pois fica muito fácil passar um parâmetro errado pela **QueryString**, por exemplo. Para alterações, o melhor é utilizar **POST** e **Request.Form**.

- **IsInt(), IsDate(), IsBool(), IsDecimal(), IsFloat()**

Estes métodos permitem verificar se o tipo dentro de uma requisição pode ser convertido em um tipo primitivo como **int**, **DateTime**, **Decimal** ou **Boolean**. Podem ser usados em conjunto com o formulário, para validações:

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

    <label for="numero">Digite um Número</label>
    <input type="text" id="numero" name="numero" />

    <label for="data">Digite uma Data</label>
    <input type="text" id="data" name="data" />

    <br/>
    <button type="submit">Enviar</button>

    @if (IsPost)
    {
        if (Request["Numero"].IsInt())
        {
            <p>Número OK</p>
        }
        else
        {
            <p>Número Inválido</p>
        }
    }

    if (Request["Data"].IsDateTime())
    {
        <p>Data OK</p>
    }
    else
    {
        <p>Data Inválida!</p>
    }
}

</form>
```

2.3.7. Helpers

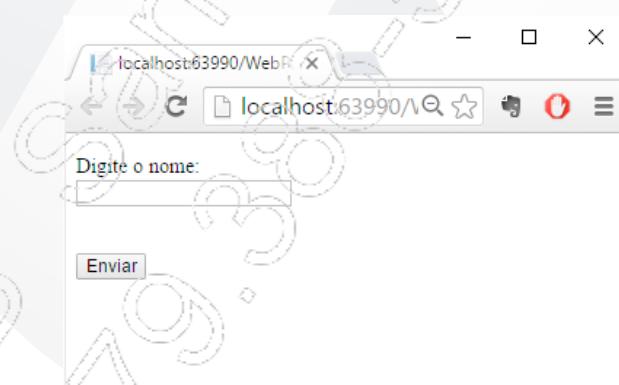
Vamos ver, nos próximos subtópicos, algumas informações sobre Helpers.

2.3.7.1. HTML

Helpers são pequenos métodos que geram código HTML. Isso facilita a construção de uma página HTML, principalmente quando há muito código com a mesma estrutura. Por exemplo, um formulário sempre vai ser constituído de diversos controles **Label** seguidos de controles **Input**. Essas tags HTML podem ser criadas automaticamente por meio dos métodos fornecidos pelo Helper. Veja o exemplo, usando o HTML:

```
@Html.Label("Digite o nome:", "nome") <br />
@Html.TextBox("nome") <br />
```

O resultado da página renderizada é este:



O código HTML gerado pelo servidor é este:

```
<label for="nome">Digite o nome:</label> <br />
<input id="nome" name="nome" type="text" value="" /> <br />
```

Os seguintes métodos geram código HTML:

CheckBox, DropDownList, Hidden, ListBox, Password, RadioButton, TextArea, ValidationMessage, Validation Summary.

2.3.7.2.Chart

O componente **Chart** renderiza imagens de gráficos. É necessário informar as sequências de valores que farão parte dos dados exibidos. O exemplo a seguir mostra um gráfico 3D com duas sequências de valores. O resultado é uma imagem, e não uma página HTML, portanto é necessário que a saída da página seja o **source** de uma imagem.

- Arquivo: **ExemploGrafico.cshtml**

```
@{  
    var grafico = new Chart(600, 400, theme: ChartTheme.Vanilla3D);  
  
    grafico.AddTitle("Vendas");  
  
    grafico.AddLegend("Tipos");  
  
    grafico.AddSeries(  
        name: "Computadores",  
        chartType: "column",  
        xValue: new[]  
            { "Jan", "Fev", "Mar", "Abr", "Mai", "Jun" },  
        yValues: new[]  
            { 20, 80, 60, 130, 120, 150 });  
  
    grafico.AddSeries(  
        name: "NoteBooks", chartType: "column",  
        xValue: new[]  
            { "Jan", "Fev", "Mar", "Abr", "Mai", "Jun" },  
        yValues: new[]  
            { 30, 90, 70, 130, 170, 200 });  
  
    grafico.Write();  
}
```

Visual Studio 2015 - ASP.NET com C# Fundamentos

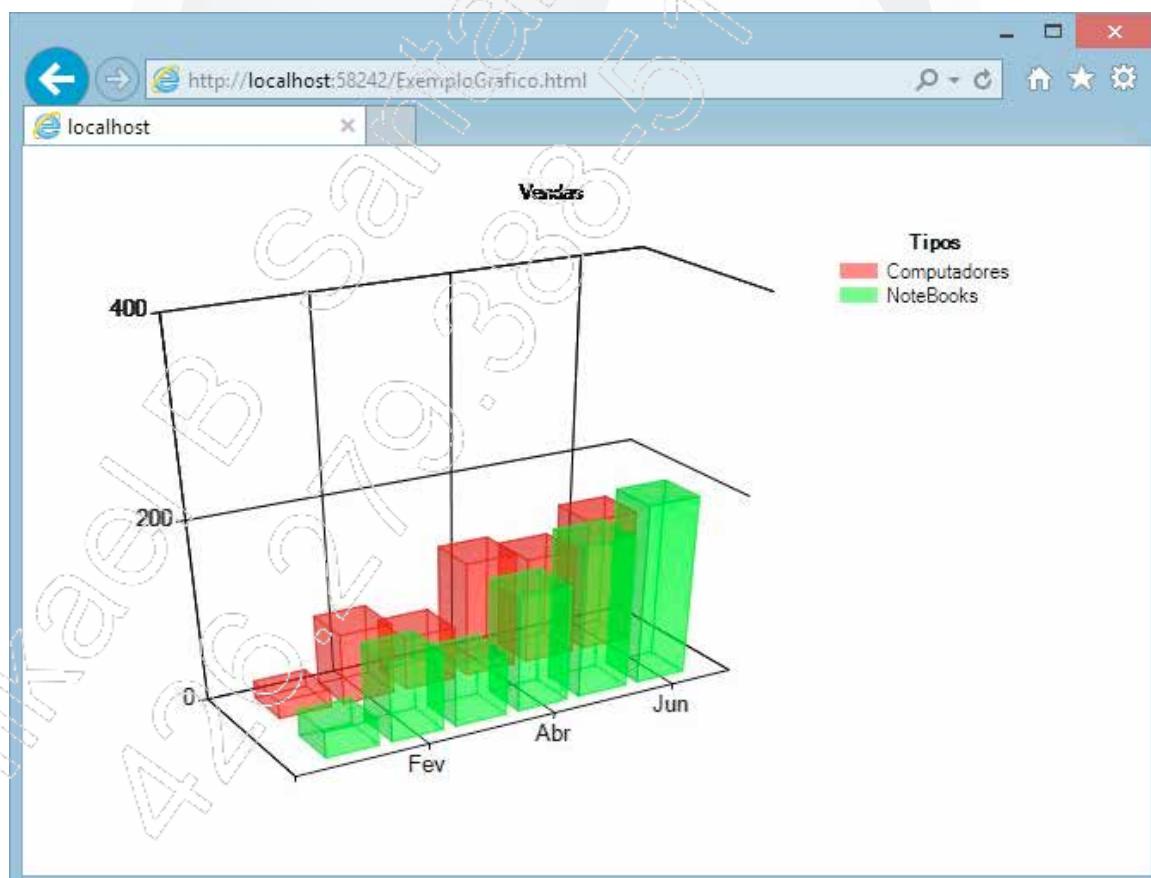
- Arquivo: **paginaSimples.html**

```
<!DOCTYPE html>
<html>
<head>
    <title></title>
</head>
<body>

</body>
</html>
```

Veja o resultado:



Vejamos, a seguir, uma explicação das linhas de comando do gráfico:

```
var grafico = new Chart(600, 400,
    theme: ChartTheme.Vanilla3D);
```

As linhas anteriores criam o gráfico com 600 de largura e 400 de altura usando o tema **Vanilla3D**. O enumerador **ChartTheme** contém os temas **blue**, **green**, **Vanilla** e **Yellow**.

```
grafico.AddTitle("Vendas");
```

A linha anterior adiciona o título do gráfico, centralizado na parte superior.

```
grafico.AddLegend("Tipos");
```

A linha anterior adiciona a legenda no canto superior direito.

```
grafico.AddSeries(
    name: "Computadores",
    chartType: "column",
    xValue: new[]
        { "Jan", "Fev", "Mar", "Abr", "Mai", "Jun" },
    yValues: new[]
        { 20, 80, 60, 130, 120, 150 });
```

As linhas anteriores adicionam a primeira série de valores. A propriedade **name** aparece na legenda. O array X são as legendas dos valores e o array Y são os valores que serão impressos. **ChartType** é o tipo do gráfico e deve ser o mesmo para todas as séries. Os principais tipos são: **Pie**, **Column**, **Area**, **Line**, **Bar** e **Point**. O mesmo vale para a segunda série de dados.

2.3.7.3. Crypto

A classe **Crypto** permite criptografar de maneira simples strings e arrays de bytes. Existem quatro métodos frequentemente utilizados:

```
Crypto.Hash("Texto")
Crypto.Hash("Texto", algoritmo)
Crypto.Hash(bytes)
Crypto.Hash(bytes, algorítmo)
```

- **Algoritmos** são fórmulas conhecidas de criptografia e podem ser **SHA256** e **SHA1**;
- **Texto** é qualquer string;
- **Bytes[]** é um array de bytes.

É importante lembrar que os métodos da classe Crypto apenas usam **hash**, ou seja, apenas criptografam e nunca descriptografam. São especialmente úteis para armazenar senhas.

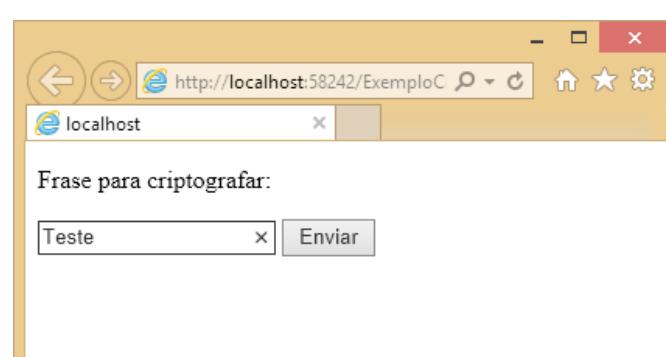
```
<html>
<head>
    <meta name="viewport" content="width=device-width" />
    <title></title>
</head>
<body>

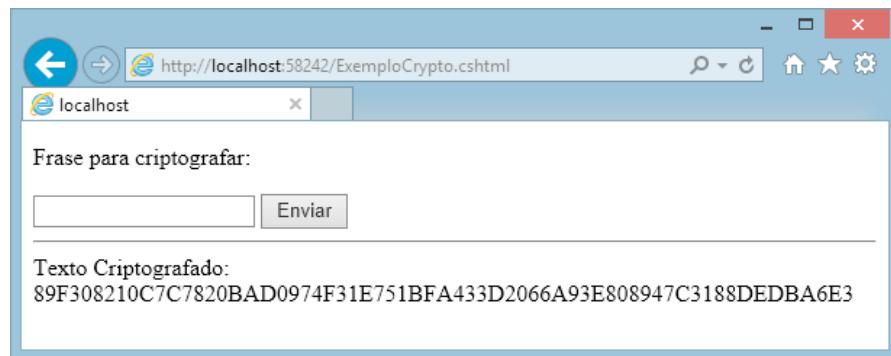
    <form action="" method="post">
        <p>Frase para criptografar:</p>
        @Html.TextBox("Texto")

        <button type="submit">Enviar</button>
    </form>

    @if (IsPostBack)
    {
        string texto = Request["texto"];
        string textoCriptografado = Crypto.Hash(texto);
        <hr/>
        <span>Texto Criptografado: @textoCriptografado</span>
    }

</body>
</html>
```





Esta é a maneira ideal de se armazenar uma senha, por exemplo. No banco de dados fica apenas a senha criptografada. Quando o usuário digita sua senha, o sistema criptografa e compara se o hash gerado é igual ao que está armazenado no banco. Se for igual, é porque a senha está correta. Usando-se hash, nunca é possível obter a informação de volta.

As classes apresentadas aqui são uma pequena amostra do que está disponível para as Web Pages. Essas classes são incluídas com o pacote básico. Usando o NuGet (instalador de pacotes do Visual Studio) é possível pesquisar e incluir pacotes com classes para os mais diversos propósitos, como autenticação no Facebook, uploads de múltiplos arquivos com barras de progresso, elementos de tela, grids, carrinhos de compra com a opção de arrastar e soltar, galeria de fotos, entre muitas outras aplicações.

Pontos principais

Atente para os tópicos a seguir. Eles devem ser estudados com muita atenção, pois representam os pontos mais importantes do capítulo.

- O navegador pode receber, inicialmente, três tipos de informação do servidor quando solicita uma página Web: o código HTML, que é responsável pela estrutura do documento; o código CSS, que determina a aparência da página; e o código JavaScript, que permite manipular eventos e elementos da página;
- Web Pages é um tipo de página que mistura código HTML com código de programação usando o mecanismo de renderização Razor;
- Helpers são arquivos que disponibilizam métodos para criar códigos HTML automaticamente.

2

Web Pages

Teste seus conhecimentos

Mikael B
Gontana
426.279.357



IMPACTA
EDITORA

1. Razor se encaixa em qual categoria de aplicativo?

- a) Plataforma
- b) Linguagem de programação
- c) View Engine
- d) Biblioteca
- e) Library

2. Qual será o resultado do comando Razor adiante?

A soma de 10 + 5 é @10+5

- a) A soma de 10 + 5 é 10+5.
- b) A soma de 10 + 5 é 15.
- c) Vai ocorrer um erro.
- d) A soma de 10 + 5 é 105.
- e) A soma de 10 + 5 é @15.

3. Qual método é usado para inserir uma página dentro de outra?

- a) InsertPage
- b) IncludePage
- c) AddPage
- d) RenderPage
- e) GetPge

4. Para que o comando RenderBody funcione, qual comando deve estar na página que será renderizada?

- a) RenderMe
- b) DOCTYPE
- c) Layout
- d) FlowLayout
- e) _Layout

5. Qual é a extensão dos arquivos Web Pages?

- a) aspx
- b) asmx
- c) vbhtml
- d) cshtml
- e) As alternativas C e D estão corretas.

2

Web Pages

Mãos à obra!

Mikael B. Sartana
426.279.357



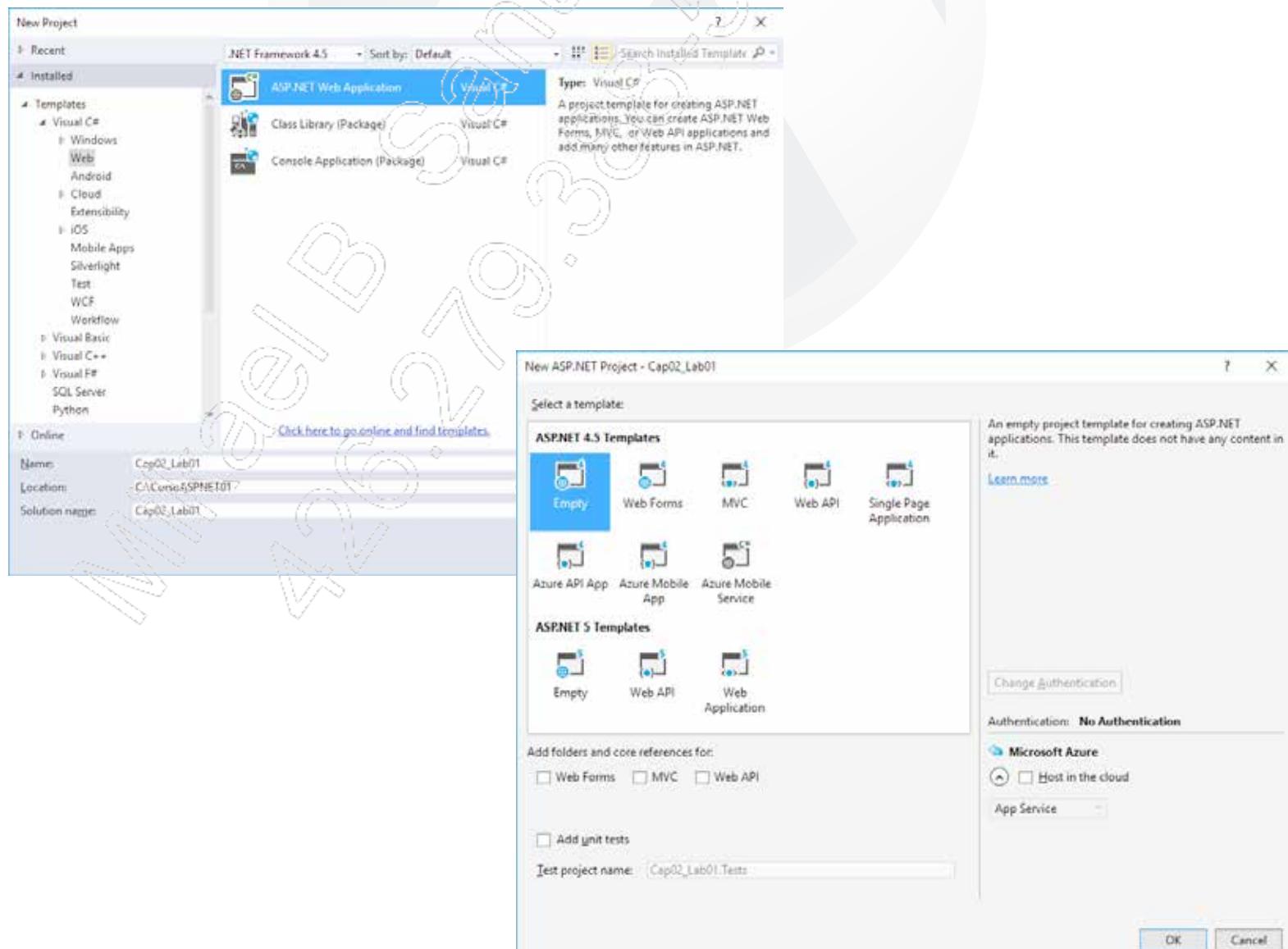
IMPACTA
EDITORA

Laboratório 1

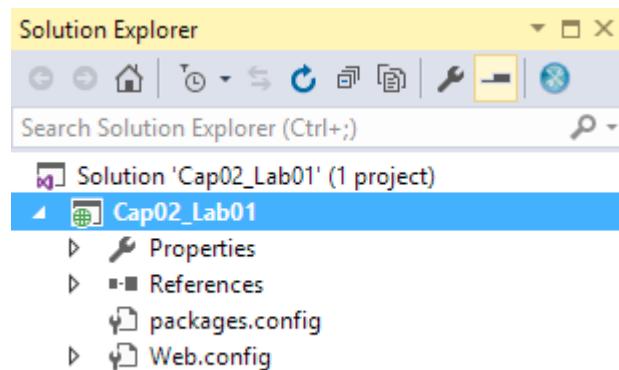
Objetivos:

- Criar um projeto ASP.NET utilizando Web Pages;
- Utilizar HTML, CSS e JavaScript;
- Utilizar o mecanismo de renderização Razor;
- Criar formulário HTML para enviar dados ao servidor e processar as informações recebidas.

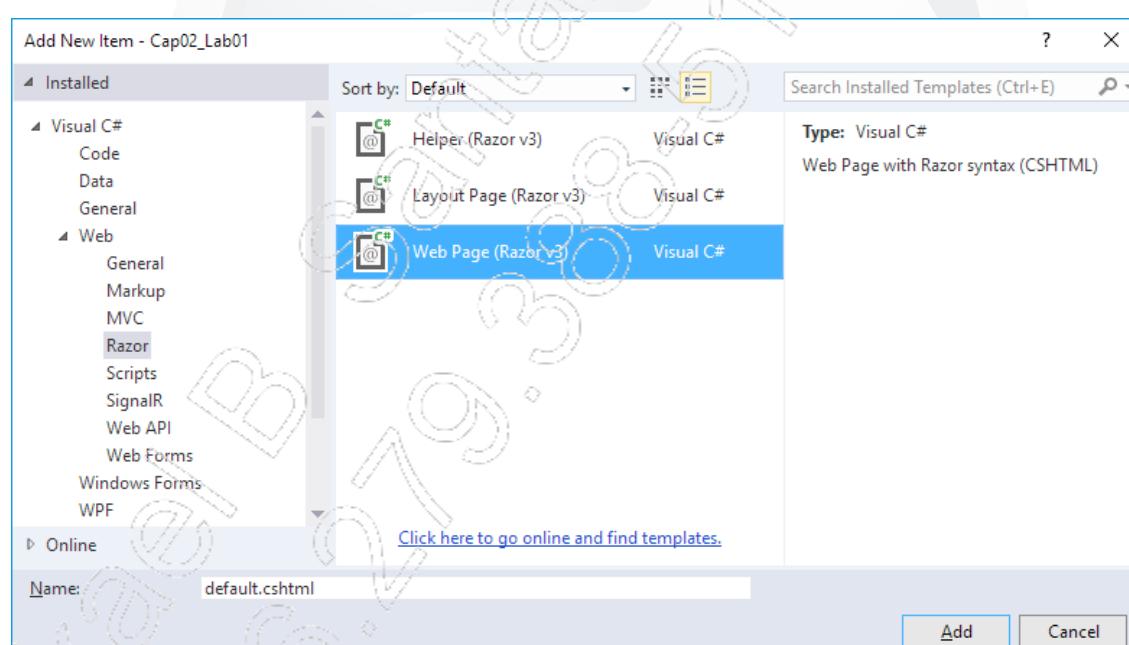
1. Crie um novo projeto ASP.NET Web Application vazio chamado **Cap02_Lab01**, usando o comando de menu **File, New Project** e escolhendo a opção **Empty** na tela de modelos;



A janela **Solution Explorer** mostra seis itens: **Solution** (com o nome da solução), **Project** (com o nome do projeto), **Properties**, **References**, **packages.config** e **Web.Config** (podem aparecer mais opções dependendo da versão e complementos instalados).



2. Adicione uma Web Page chamada **default.cshtml** por meio do comando de menu **Project, Add New Item** e escolhendo o grupo **Razor** e a opção **Web Page**:



O seguinte documento, contendo apenas HTML, será criado:

```
<!DOCTYPE html>
<html>
  <head>
    <title></title>
  </head>
  <body>

  </body>
</html>
```

Visual Studio 2015 - ASP.NET com C# Fundamentos

3. Usando as tags do HTML5, defina três regiões para o documento: **Header**, **Section** e **Footer**. Essas serão as áreas do cabeçalho, conteúdo e rodapé, respectivamente;

```
<!DOCTYPE html>
<html>
  <head>
    <title></title>
  </head>
  <body>

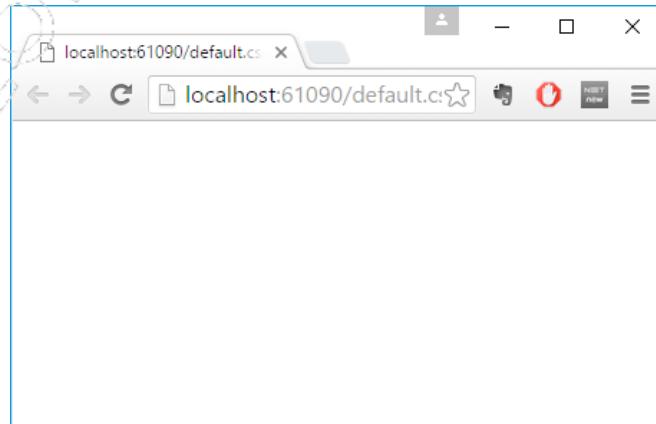
    <header>
      </header>

    <section>
      </section>

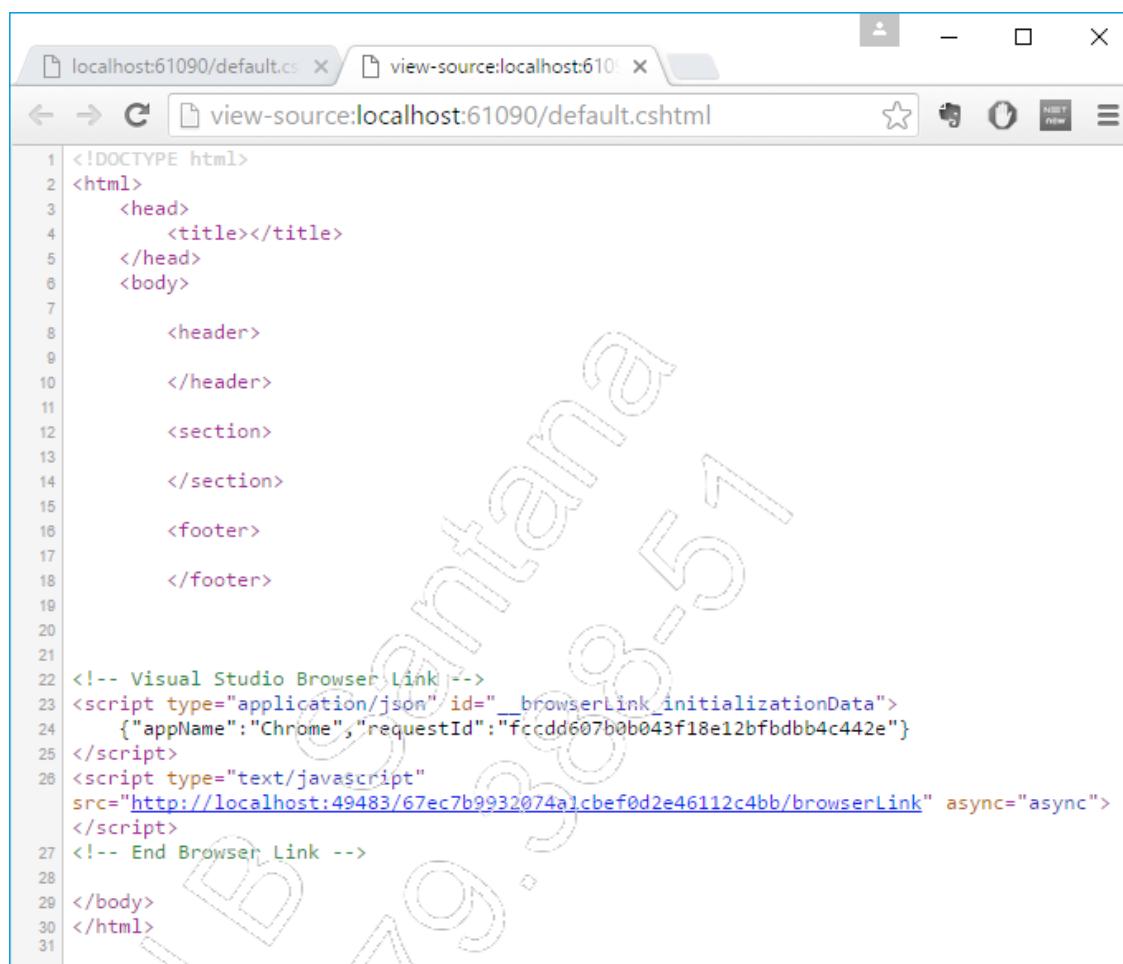
    <footer>
      </footer>

    </body>
  </html>
```

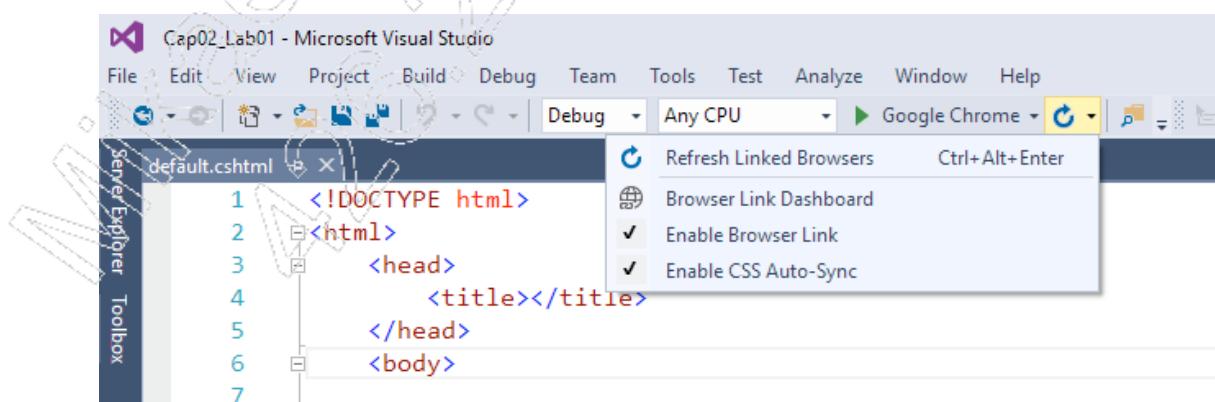
4. Todas as tags inseridas até o momento são estruturais. Nada é renderizado visualmente. Pressione CTRL + F5 e veja que a página aparece totalmente em branco:



5. Apesar de a página aparecer em branco, o conteúdo HTML foi transferido para o browser. Veja o conteúdo transmitido abrindo o menu de contexto dentro da página e escolhendo a opção **Exibir código-fonte**. Uma informação será inserida no final do documento pelo Visual Studio, caso a opção **Enable Browser Link** ou **Enable CSS Auto-Sync** esteja ligada. Essas opções permitem ver as alterações no código-fonte automaticamente em vários browsers;



```
1 <!DOCTYPE html>
2 <html>
3     <head>
4         <title></title>
5     </head>
6     <body>
7
8         <header>
9
10        </header>
11
12        <section>
13
14        </section>
15
16        <footer>
17
18        </footer>
19
20
21
22 <!-- Visual Studio Browser Link -->
23 <script type="application/json" id="__browserLink_initializationData">
24     {"appName":"Chrome","requestId":"fccdd607b0b043f18e12bfdbb4c442e"}
25 </script>
26 <script type="text/javascript"
src="http://localhost:49483/67ec7b9932074a1cbef0d2e46112c4bb/browserLink" async="async">
27 </script>
28 <!-- End Browser Link -->
29
30 </body>
31 </html>
```



Visual Studio 2015 - ASP.NET com C# Fundamentos

6. Insira o título da página e o rodapé. Visualize a página renderizada;

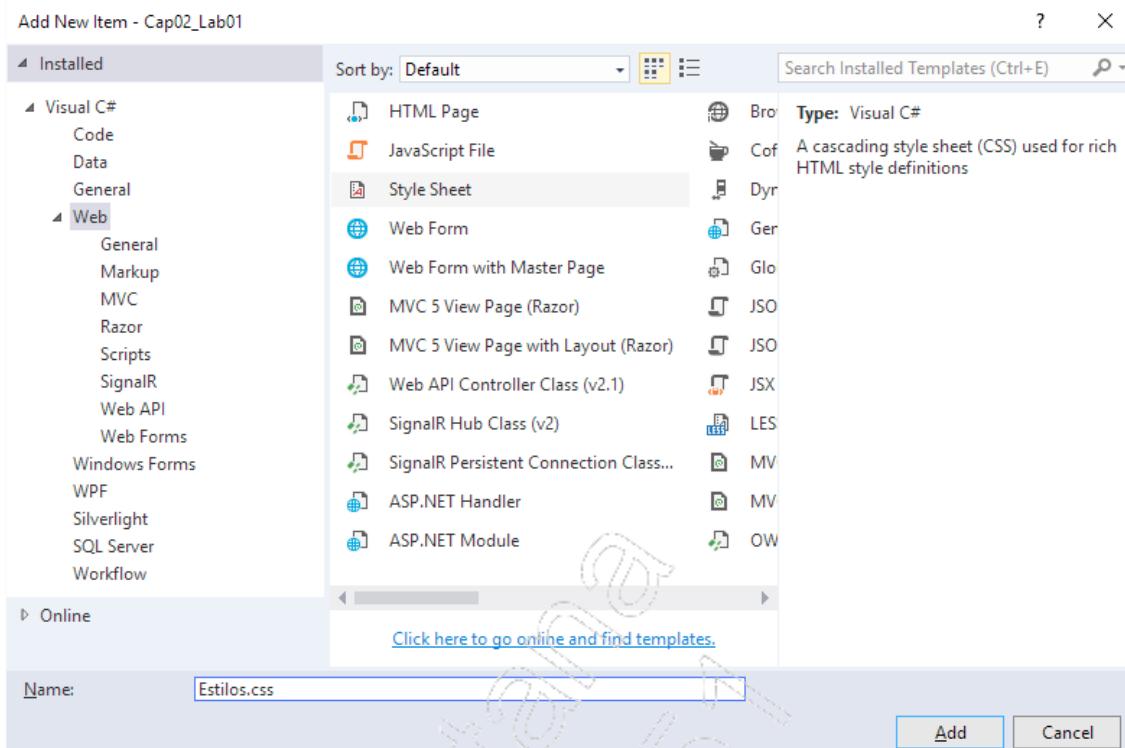
```
<header>
    <h1>Exemplo Web Page</h1>
    <h2>Tecnologias Web - Html, Javascript, Css, Razor e Web Pages</h2>
</header>

<section>
</section>

<footer>
    <hr/>
    <p>
        Desenvolvido para o curso de ASP.NET
        © @DateTime.Now.Year - Impacta Tecnologia
    </p>
</footer>
```



7. Adicione no projeto uma folha de estilo chamada **estilos.css**. Para isso, use o menu **Project, Add New Item, StyleSheet**:



8. Defina os seguintes estilos no arquivo **estilos.css**:

```
body {  
    font-family:Tahoma;  
    margin:0px;  
}
```

```
header {  
    padding:20px;  
    background-color:#749ca7;  
    color:#e0eaed;  
    letter-spacing:-1px;  
}
```

```
h1,h2{  
    margin:0px;  
}
```

```
footer {  
    position:fixed;  
    bottom:0px;  
    width:100%;  
}  
  
footer p {  
    padding:5px;  
    font-size:70%;  
}
```

```
section {  
    max-width:960px;  
    padding:20px;  
}  
  
section ul {  
    padding:0px;  
}
```

9. Arraste o arquivo **estilos.css** da janela **Solution Explorer** para dentro da página **default.cshtml**, abaixo da tag **<title>** (ou digite a referência diretamente). Visualize o resultado;

Página **default.cshtml**

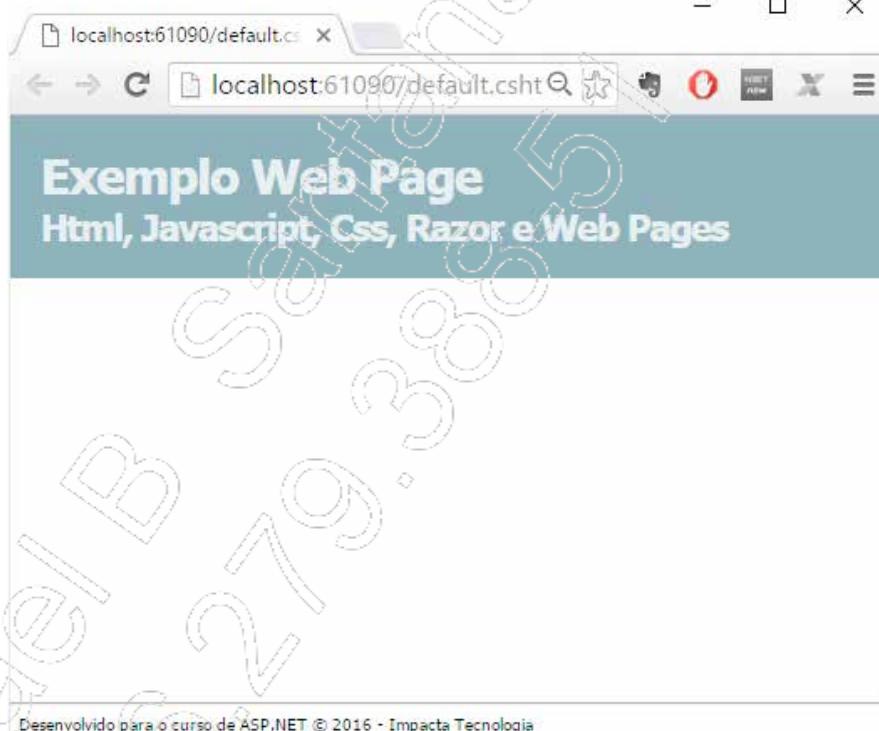
...

```
<head>
    <title></title>
```

```
        <link href="~/Estilos.css" rel="stylesheet" />
```

```
</head>
```

...



10. Na página **default.cshtml**, insira o conteúdo do item **section** e visualize a página:

```
<section>
    <p>
        No desenvolvimento Web, diversas tecnologias são utilizadas.
        Algumas são executadas do lado do Servidor(Server) e outras
        são executadas do lado do Cliente(Client).
    </p>

    <dl>
        <dt>
            <a href="#">Tecnologias: Client</a>
        </dt>
        <dd>
            <ul>
                <li>Html</li>
                <li>Css</li>
                <li>Javascript</li>
            </ul>
        </dd>

        <dt>
            <a href="#">Tecnologias: Server</a>
        </dt>
        <dd>
            <ul>
                <li>C#</li>
                <li>ASP.NET</li>
                <li>Web Pages</li>
                <li>Web Forms</li>
                <li>MVC</li>
                <li>Razor</li>
            </ul>
        </dd>
    </dl>
</section>
```



11. JavaScript é útil para realizar processamento na página em que não há a necessidade de obter dados do servidor. As descrições das tecnologias vão ficar escondidas e serão exibidas quando o usuário clicar no link. Insira o código que esconde ou exibe um elemento HTML. O script deve ficar na seção <head> do código HTML:

```
<head>
    <title></title>
    <link href="~/Estilos.css" rel="stylesheet" />

    <script>
        function exibirEsconderElemento(elementoId)
        {
            var elemento = document.getElementById(elementoId);
            if (elemento.style.display == "none") {
                elemento.style.display = "block";
            }
            else {
                elemento.style.display = "none";
            }
        }
    </script>

</head>
```

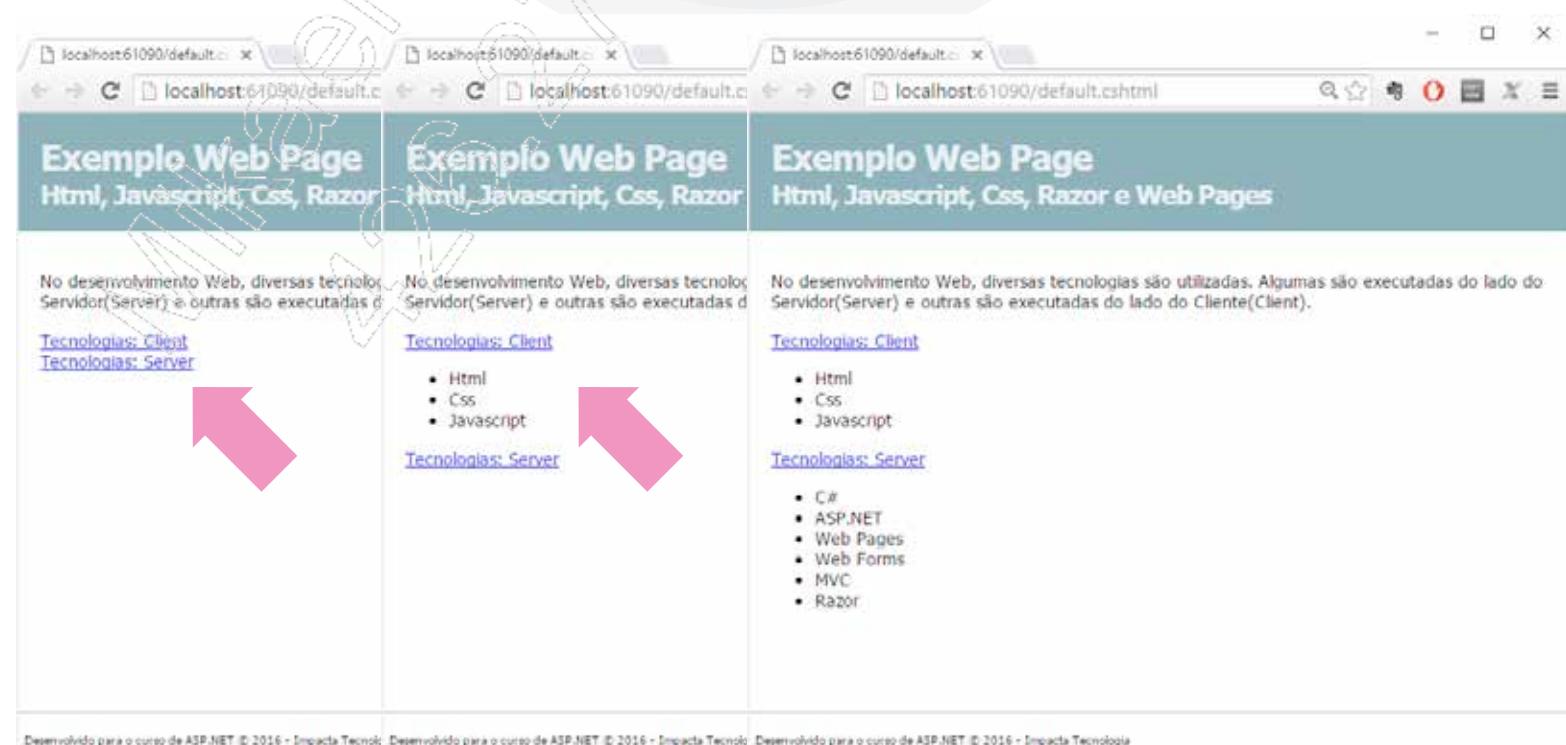
Visual Studio 2015 - ASP.NET com C# Fundamentos

12. Altere os links da página para executar a função em JavaScript e defina o atributo Id apropriado;

```
<dl>
  <dt>
    <a href="javascript:exibirEsconderElemento('ddClient')">
      Tecnologias: Client
    </a>
  </dt>
  <dd id="ddClient" style="display:none">
    <ul>
      ...
    </ul>
  </dd>

  <dt>
    <a href="javascript:exibirEsconderElemento('ddServer')">
      Tecnologias: Server
    </a>
  </dt>
  <dd id="ddServer" style="display:none">
    <ul>
      ...
    </ul>
  </dd>
</dl>
```

13. Visualize e teste a página:



Laboratório 2

Objetivos:

- Criar formulário HTML para enviar dados ao servidor e processar as informações recebidas;
- Criar uma classe estática para processar os dados.

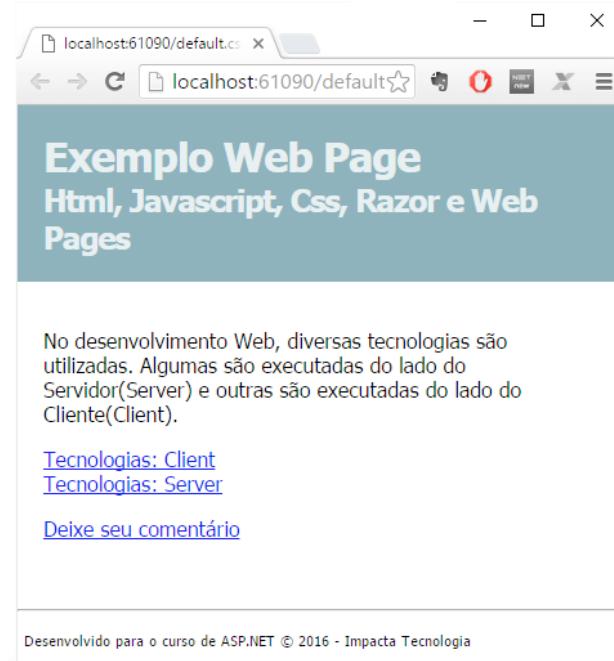
1. Partindo do final do laboratório anterior, adicione um parágrafo depois da lista de tecnologias, na página **default.cshtml** e visualize a página:

```
...
<dl>
  <dt>
    <a href="javascript:...">Tecnologias: Client</a>
  </dt>
  <dd id="ddClient" style="display:none" >
    <ul>
      <li>Html</li>
      <li>Css</li>
      <li>Javascript</li>
    </ul>
  </dd>
  <dt>
    <a href="javascript:...">Tecnologias: Server</a>
  </dt>
  <dd id="ddServer" style="display:none" >
    <ul>
      <li>C#</li>
      <li>ASP.NET</li>
      <li>Web Pages</li>
      <li>Web Forms</li>
      <li>MVC</li>
      <li>Razor</li>
    </ul>
  </dd>
</dl>

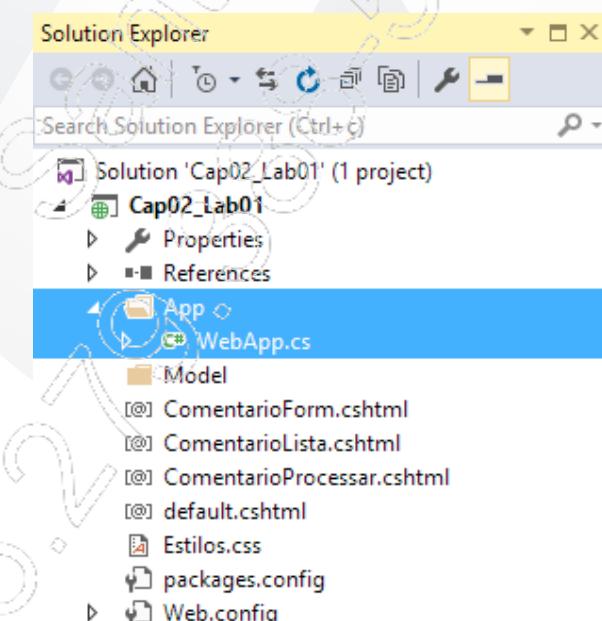
<p>
  <a href="ComentarioForm.cshtml">Deixe seu comentário</a>
</p>
...

```

Visual Studio 2015 - ASP.NET com C# Fundamentos



2. Os comentários serão gravados em um arquivo de texto. Adicione uma pasta ao projeto chamada **App** e, dentro dessa pasta, uma classe chamada **WebApp**:



3. A classe **WebApp** é estática e terá um método para gravar um comentário e outro para ler os comentários. Insira as declarações dos métodos e compile o projeto:

```
public static class WebApp
{
    public static void ComentarioIncluir(string nome, string comentario)
    {
    }

    public static string ComentariosObter()
    {
        return null;
    }
}
```

4. Na classe **WebApp**, insira uma propriedade interna para retornar o nome do arquivo:

```
private static string comentarioArquivo
{
    get{
        return HttpContext.Current.Server.MapPath("~/comentarios.txt");
    }
}
```

5. Na classe **WebApp**, crie o corpo dos métodos. Utilize a diretiva de compilação **using System.IO** para acesso às classes **StreamWriter** e **StreamReader** e a diretiva **using System.Text** para acesso à classe **Encoding**;

```
public static class WebApp
{
    private static string comentarioArquivo
    {
        get
        {
            return HttpContext.Current.Server.MapPath("~/comentarios.txt");
        }
    }

    public static void ComentarioIncluir(string nome, string comentario)
    {

        using (var writer =
            new StreamWriter(comentarioArquivo, true, Encoding.UTF8))
        {
            writer.WriteLine("{0:dd/MM/yyyy} - {1:HH:mm:ss}",
                DateTime.Now, DateTime.Now);

            writer.WriteLine("{0}: {1}\r\n", nome, comentario);
        }
    }

    public static string ComentariosObter()
    {

        string texto = string.Empty;

        if (!File.Exists(comentarioArquivo))
        {
            return texto;
        }

        using (var reader = new StreamReader(comentarioArquivo))
        {
            texto = reader.ReadToEnd();
        }

        return texto;
    }
}
```

6. Adicione, na raiz do projeto, a página **ComentárioForm.cshtml**, que contém, além dos elementos padrão, um formulário para enviar o comentário do usuário;

```
<!DOCTYPE html>
<html>
<head>
    <title></title>
    <link href="~/Estilos.css" rel="stylesheet" />
</head>
<body>

    <header>
        <a href="~/default.cshtml">
            <h1>Exemplo Web Page</h1>
            <h2>Html, Javascript, Css, Razor e Web Pages</h2>
        </a>
    </header>

    <section>

        <p>Deixe seu comentário: </p>

        <form action="~/ComentarioProcessar.cshtml" method="post">

            <div class="form-grupo">
                <label for="nome">Nome:</label>
                <input type="text" name="nome" id="nome" />
            </div>

            <div class="form-grupo">
                <label for="comentario">Comentário</label>
                <textarea id="comentario" name="comentario"></textarea>
            </div>

            <div class="form-botoes">
                <input type="submit" value="Enviar" />
            </div>
        </form>

    </section>

</body>
</html>
```

7. Adicione a página **ComentarioProcessar.cshtml**. Essa página processa os dados enviados pelo formulário;

```
@using Cap02_Lab01
<!DOCTYPE html>
<html>

    <head>
        <title></title>
        <link href="~/Estilos.css" rel="stylesheet" />
    </head>
    <body>
        <header>
            <a href="~/default.cshtml">
                <h1>Exemplo Web Page</h1>
                <h2>Html, Javascript, Css, Razor e Web Pages</h2>
            </a>
        </header>

        <section>
            @if (!IsPostBack)
            {
                Response.Redirect("~/comentarioForm.cshtml");
            }
            else
            {
                string nome = Request.Form["nome"];
                string comentario = Request.Form["comentario"];
                if (string.IsNullOrWhiteSpace(nome) ||
                    string.IsNullOrWhiteSpace(comentario))
                {
                    Response.Redirect("~/comentarioForm.cshtml");
                }
                else
                {
                    WebApp.ComentarioIncluir(nome, comentario);
                }
            }

            <p>Obrigado por seu comentário</p>

            <p>
                <a href="~/default.cshtml">Voltar</a>
                <a href="~/ComentarioLista.cshtml">Exibir Comentários</a>
            </p>
        </section>

    </body>
</html>
```

8. Adicione a página **ComentarioLista.cshtml**. Essa página exibe os comentários gravados;

```
@using Cap02_Lab01
```

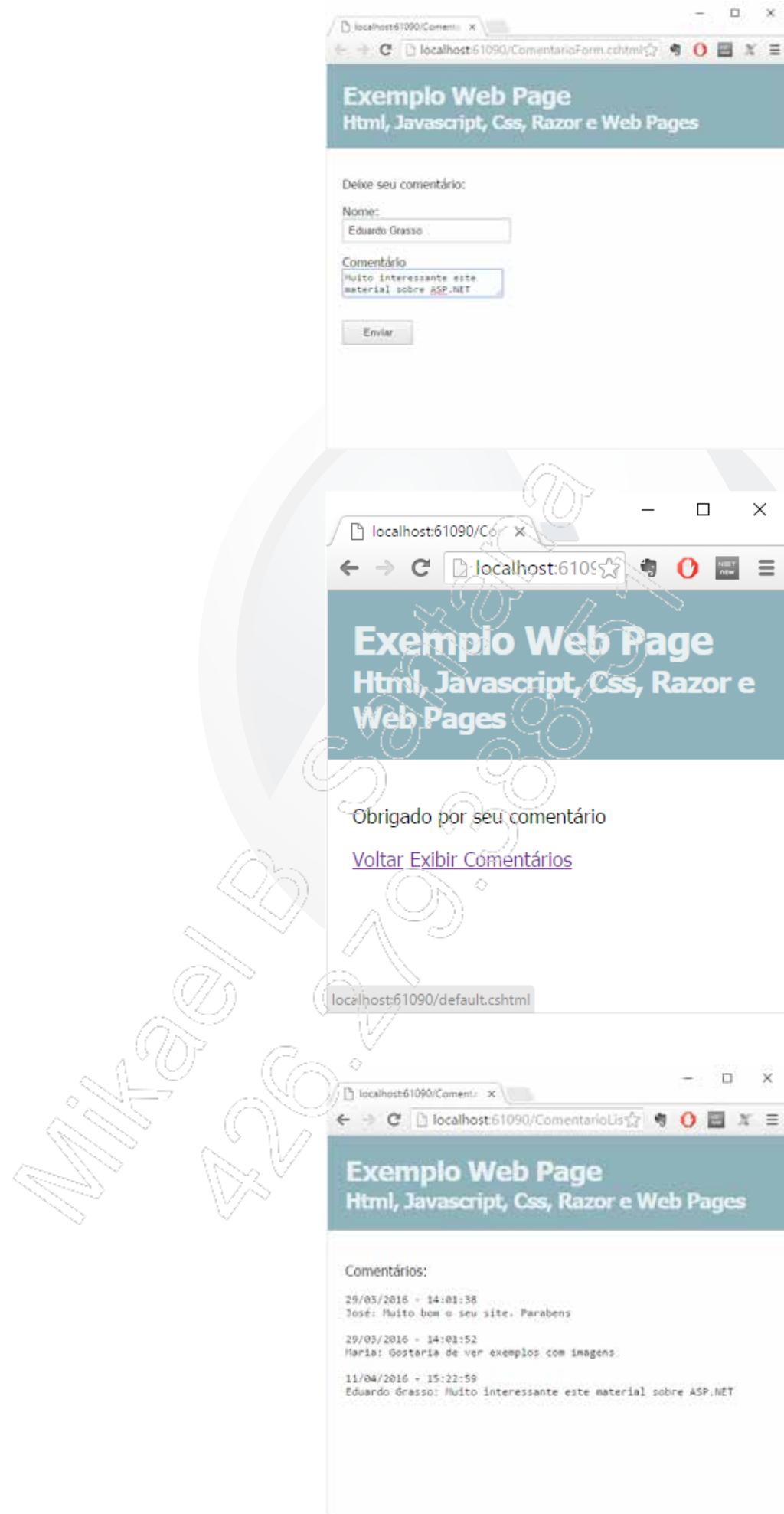
```
<!DOCTYPE html>
<html>
  <head>
    <title></title>
    <link href="~/Estilos.css" rel="stylesheet" />
  </head>
  <body>

    <header>
      <a href("~/default.cshtml")>
        <h1>Exemplo Web Page</h1>
        <h2>Html, Javascript, Css, Razor e Web Pages</h2>
      </a>
    </header>

    <section>
      <p>Comentários:</p>
      <pre>
@webApp.ComentariosObter()
</pre>
    </section>
  </body>
</html>
```

Visual Studio 2015 - ASP.NET com C# Fundamentos

9. Teste o projeto completo:



3

Web Forms

- ✓ A construção de UIs;
- ✓ Controles.



IMPACTA
EDITORA

3.1. Introdução

Os Web Forms utilizam um modelo de renderização que pode separar o código que controla a lógica da página do conteúdo HTML. Esse estilo se chama code-behind. A ligação entre os arquivos é feita por meio de tags especiais chamadas **Web Controls** e do atributo **Runat="Server"**, que pode ser definido em qualquer tag.

Este capítulo irá tratar do modelo de renderização dos Web Forms ASP.NET. Nele, trataremos, também, do processo de construção de interfaces de usuário e dos controles utilizados no ASP.NET, como controles de servidor HTML, de servidor Web, de validação e de usuário, entre outros.

3.2. A construção de UIs

A sigla **UI** significa **User Interface** (interface de usuário). Essa interface é utilizada para designar componentes ou trechos de código que têm como objetivo a interação do usuário com uma aplicação. A UI é tudo que o usuário pode ver e interagir em um aplicativo, como botões, menus, barras de rolagem e imagens, entre outros elementos.

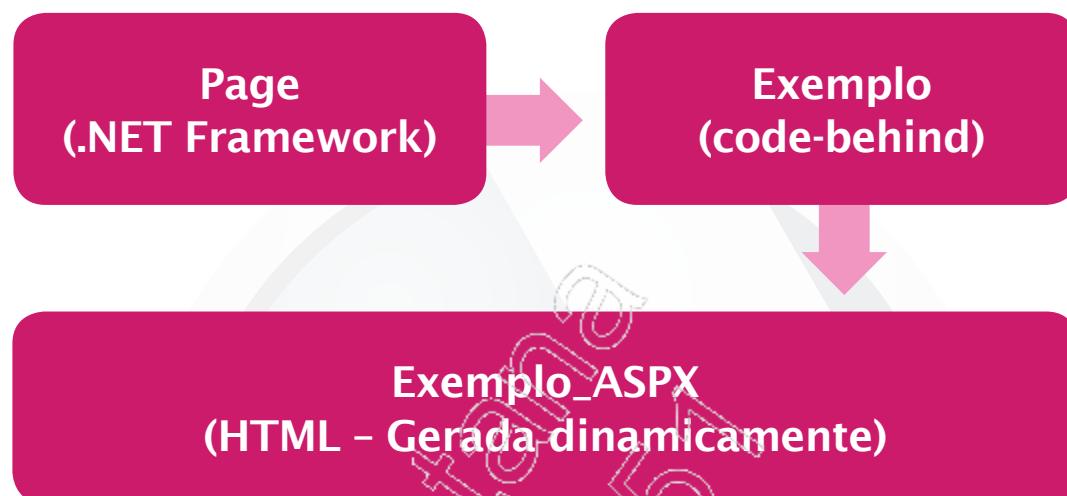
Um dos grandes desafios da construção de interfaces é fazer com que a usabilidade seja boa independente do dispositivo utilizado ou da resolução da tela. O ASP.NET utiliza vários recursos interessantes para tornar isso possível. Antes da mais nada, é preciso entender como as páginas são geradas.

3.2.1. O processo de renderização

Uma página ASP.NET no modelo Web Forms consiste de um arquivo HTML e um arquivo de código. Na declaração da página gerada pelo Visual Studio, encontramos os seguintes elementos:

```
<%@ Page Language="C#" AutoEventWireup="true" CodeFile="Exemplo.aspx.cs" Inherits="Exemplo" %>
```

O atributo **CodeFile** indica qual é o arquivo com o código-fonte relacionado a essa página HTML. Esse arquivo é chamado de **code-behind**. Nesse arquivo, deve haver uma classe derivada da classe **Page**. Quando a página for compilada, a página HTML se torna uma classe derivada dessa classe definida no code-behind. Essa herança é definida no atributo **Inherits**. Como o arquivo de code-behind pode conter mais de uma classe, o compilador precisa saber qual é a classe que será usada como classe base. O esquema das classes, depois de compilado, é o seguinte:



Finalmente, o atributo **AutoEventWireUp** define que alguns eventos serão disparados automaticamente se determinados métodos forem implementados na classe do code-behind. Os mais frequentemente utilizados são **Page_Load**, **Page_PreInit** e **Page_PreRender**. Veremos a sequência de eventos em detalhes mais adiante.

Para criar a classe final que será usada para gerar o arquivo HTML, o ASP.NET usa um recurso chamado **parse**. O parse consiste em procurar no código HTML partes que devem ser executadas no servidor. Essas partes são analisadas e transformadas em objetos. Todo código HTML que não estiver marcado para ser executado no servidor é colocado em objetos do tipo **Literal** e cujo propósito é escrever o texto HTML literalmente. Veja o exemplo a seguir:

Visual Studio 2015 - ASP.NET com C# Fundamentos

```
<%@ Page Language="C#" AutoEventWireup="true" CodeFile="Exemplo.aspx.cs" Inherits="Exemplo" %>

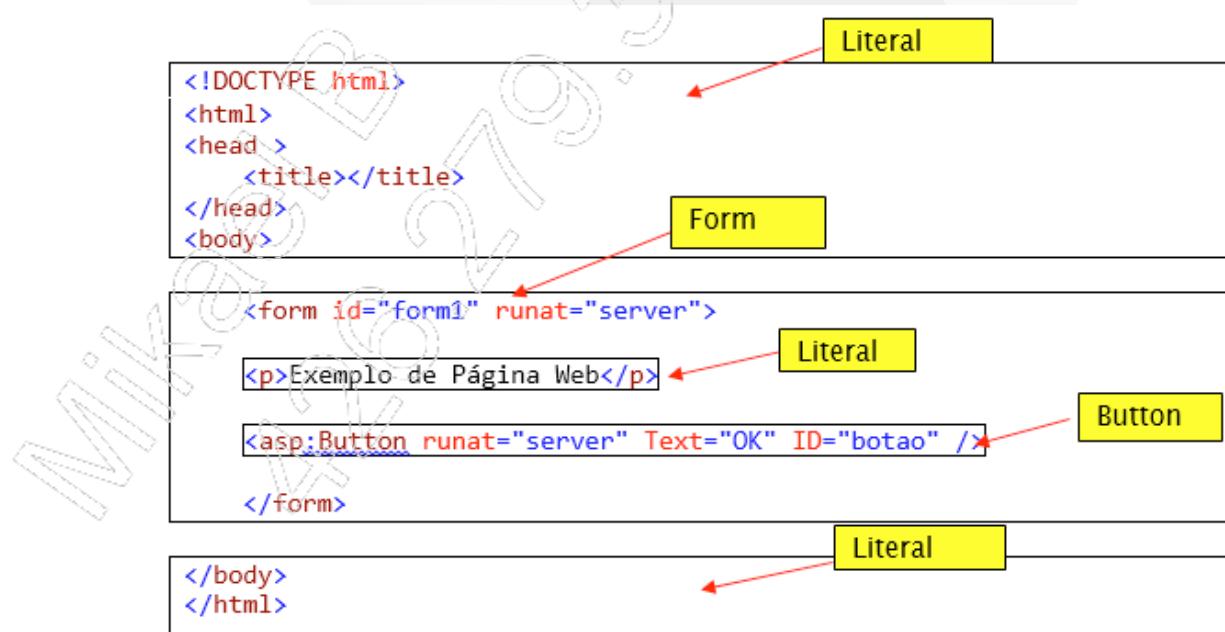
<!DOCTYPE html>
<html>
<head>
    <title></title>
</head>
<body>
    <form id="form1" runat="server">

        <p>Exemplo de Página Web</p>

        <asp:Button runat="server" Text="OK" ID="botao" />

    </form>
</body>
</html>
```

Essa página vai gerar diversos objetos:



Ligando a opção **Trace**, é possível ver a árvore gerada:

```
<%@ Page Trace="true" Language="C#" AutoEventWireup="true"
CodeFile="Exemplo.aspx.cs" Inherits="Exemplo" %>
```

UniqueID de Controle	Tipo	Bytes do Tamanho do Processamento (incluindo filhos)	Bytes de ViewState (excluindo filhos)	Bytes de ControlState (excluindo filhos)
__Page	ASP.exemplo_aspx	731	0	0
ctl00	System.Web.UI.WebControls.HiddenField	78	0	0
form1	System.Web.UI.HtmlControls.HtmlForm	633	0	0
ctl01	System.Web.UI.WebControls.Button	45	0	0
botao	System.Web.UI.WebControls.Button	58	0	0
ctl02	System.Web.UI.WebControls.HiddenField	10	0	0
ctl03	System.Web.UI.WebControls.HiddenField	20	0	0

O código-fonte enviado para o navegador contém mais elementos do que está definido no HTML. Isso ocorre porque cada objeto gera o HTML, o CSS e o JavaScript necessários para seu funcionamento. No exemplo a seguir, os caracteres em itálico foram gerados pelo ASP.NET:

```
<!DOCTYPE html>
<html>
<head>
    <title></title>
</head>
<body>
    <form method="post" action="Exemplo" id="form1">
        <div class="aspNetHidden">
            <input type="hidden" id="__VIEWSTATE" value="0wCyQK=" />
        </div>
        <p>Exemplo de Página Web</p>
        <input type="submit" name="botao" value="OK" id="botao" />
    </form>
</body>
</html>
```

3.3. Controles

Controles do lado do servidor do ASP.NET são derivados da classe **System.Web.UI.Control**. O centro de praticamente todo elemento de interface de usuário no ASP.NET, incluindo até mesmo o **System.Web.UI.Page**, é a classe **Control**.

Quando utilizamos o ASP.NET para criar páginas Web, os seguintes tipos de controles podem ser utilizados:

- **Controles de servidor HTML**: São elementos HTML programáveis, pois estão expostos ao servidor. Eles expõem um modelo objeto que mapeia de modo similar aos elementos HTML que eles renderizam;
- **Controles de servidor Web**: Estes controles possuem mais recursos embutidos do que os controles de servidor HTML e incluem, além de controles de formulários simples (como botões e caixas de texto), controles com propósitos especiais (calendários, menus, validações);
- **Controles de usuário**: Um método simples para criarmos barras de ferramentas e outros elementos reutilizáveis, embutindo controles em páginas Web ASP.NET.

3.3.1. Controles de servidor HTML

Em uma página, os elementos HTML podem ser convertidos em controles de servidor. Isso permite a manipulação da tag HTML do lado do servidor. Para isso, basta adicionarmos o atributo **runat="server"**. O framework da página ASP.NET cria ocorrências de todos os elementos contendo esse atributo durante a análise (parsing). Podemos definir um atributo **id** para o controle caso desejemos referenciar o controle como um membro dentro do código. Veja o exemplo:

```
<p id="meuParagrafo" runat="server">Exemplo</p>
```

No code-behind, é possível manipular esse parágrafo:

```
public partial class Exemplo : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
    {
        meuParagrafo.InnerText="Alterando..."
    }
}
```

As propriedades desses objetos seguem o mesmo modelo padrão de acesso a elementos (DOM - Document Object Model). Toda propriedade que pode ser definida via JavaScript ou por meio da declaração HTML pode ser manipulada pelo servidor.

3.3.2. Controles de servidor Web

Os controles de servidor Web ASP.NET são objetos encontrados em páginas Web ASP.NET que são executados quando a página é solicitada e que geram código HTML para o navegador.

Esses controles são como controles abstratos, em que o HTML gerado pelo controle pode ser bastante diferente do modelo programado. Observe:

```
<asp:Button runat="server" Text="OK" ID="botao" />
```

O controle anterior é renderizado no navegador da seguinte forma:

```
<input type="submit" name="botao" value="OK" id="botao" />
```

O Web Control **Calendar** é definido desta forma...

```
<asp:Calendar runat="server" ID="calend" />
```

...e é renderizado desta forma (parte foi omitida):

```
<table id="calend" cellspacing="0" cellpadding="2"
      title="Calendário" style="border-width:1px; border-style:solid; border-
      collapse:collapse;">
    <tr>
      <td colspan="7" style="background-color:Silver;"><table cellspacing="0"
      style="width:100%; border-collapse:collapse;">
        <tr><td style="width:15%;"><a href="javascript:_
      doPostBack('calend','V5265')" style="color:Black" title="Ir para o mês
      anterior">&lt;</a></td><td align="center" style="width:70%;">julho de
      2014</td><td align="right" style="width:15%;"><a href="javascript:_
      doPostBack('calend','V5326')" style="color:Black" title="Ir para o mês
      seguinte">&gt;</a></td></tr>
        </table></td></tr><tr><th align="center" abbr="domingo" scope="col">dom</
      th><th align="center" abbr="segunda-feira" href="javascript:_
      doPostBack('calend','5293')" style="color:Black" title="29 de junho">29</a></
      td>
      ...
      <td href="javascript:_doPostBack('calend','5334')" style="color:Black" title="9 de
      agosto">9</a></td></tr>
    </table>
```

Botões e caixas de texto, considerados controles tradicionais de formulário, estão incluídos nos controles de servidor Web, além de diversos outros controles mais complexos, que fornecem funcionalidades de formulário muito usadas, como a visualização de dados em uma grade, escolha de datas, visualização de menus etc.

Os seguintes recursos estão disponíveis nos Web Controls:

- Um modelo objeto rico, que tem a função de fornecer capacidades de programação type-safe;
- Detecção automática de navegador. Os controles podem detectar capacidades e renderizar um markup apropriado;
- A habilidade de especificar, em alguns controles, se um evento de controle causa envio imediato para o servidor, ou se é armazenado em cache e convocado quando a página é enviada;
- Suporte para temas. Isso nos permite optar por uma aparência sólida para os controles em todo o site.

A sintaxe a seguir é um exemplo do que é usado pelos controles:

```
<asp:[nome do controle] runat="server" id="[id do controle]" />
```

Nesse caso, os atributos são propriedades do controle Web, e não dos elementos HTML. Quando executamos uma página Web ASP.NET, o controle de servidor Web é renderizado na página usando markup apropriado, que vai depender, na maioria das vezes, do tipo do navegador e de outras configurações feitas para o controle. Por exemplo, de acordo com suas propriedades, um controle **TextBox** será renderizado como uma tag **input** ou uma tag **textarea**.

3.3.2.1. Controles para exibir texto

Os controles para exibir texto têm como "Text" a principal propriedade. São eles **Label** e **Literal**. O controle **Label** gera uma tag HTML **Span** e contém diversos recursos de controle de aparência, como **BackColor** e **ForeColor**. O controle **Literal** disponibiliza apenas a propriedade **Text** e não gera nenhuma tag.

```
<asp:Label ID="nomeLabel" runat="server" Text="Nome:"></asp:Label>
```

Resultado:

```
<span id="nomeLabel">Nome:</span>
```

```
<asp:Literal ID="literal" runat="server" Text="Nome:"></asp:Literal>
```

Resultado:

```
Nome:
```

Para manipular as propriedades dos controles no lado do servidor, o acesso é feito por meio de um objeto criado automaticamente pelo ASP.NET:

```
protected void Page_Load(object sender, EventArgs e)
{
    nomeLabel.Text = "Digite o nome:";
}
```

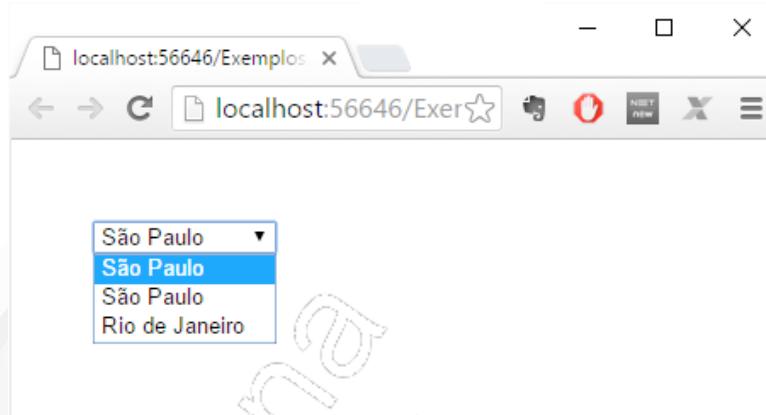
3.3.2.2. Controles para exibir listas

Os controles de lista contam com a propriedade **Item**, que é uma coleção de objetos do tipo **ListItem** que definem os itens a serem exibidos. Os controles de lista mais utilizados são: **DropDownList**, **BulletList**, **ListBox**, **CheckBoxList** e **RadioButtonList**.

Visual Studio 2015 - ASP.NET com C# Fundamentos

Veja um exemplo de DropDownList:

```
<asp:DropDownList ID="cidades" runat="server">
    <asp:ListItem>São Paulo</asp:ListItem>
    <asp:ListItem>São Paulo</asp:ListItem>
    <asp:ListItem>Rio de Janeiro</asp:ListItem>
</asp:DropDownList>
```

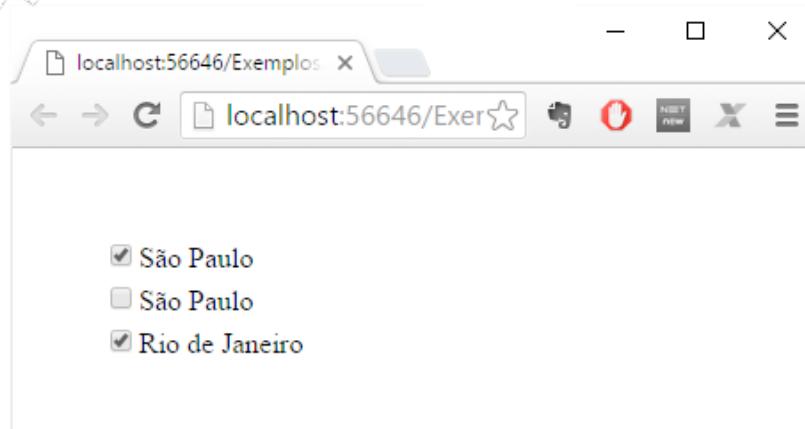


Para obter um valor selecionado da lista, utiliza-se a propriedade **selectedValue**:

```
string cidade = cidadeDropDownList.SelectedValue;
```

Veja um exemplo de CheckBoxList:

```
<asp:CheckBoxList ID="cidades" runat="server">
    <asp:ListItem>São Paulo</asp:ListItem>
    <asp:ListItem>São Paulo</asp:ListItem>
    <asp:ListItem>Rio de Janeiro</asp:ListItem>
</asp:CheckBoxList>
```

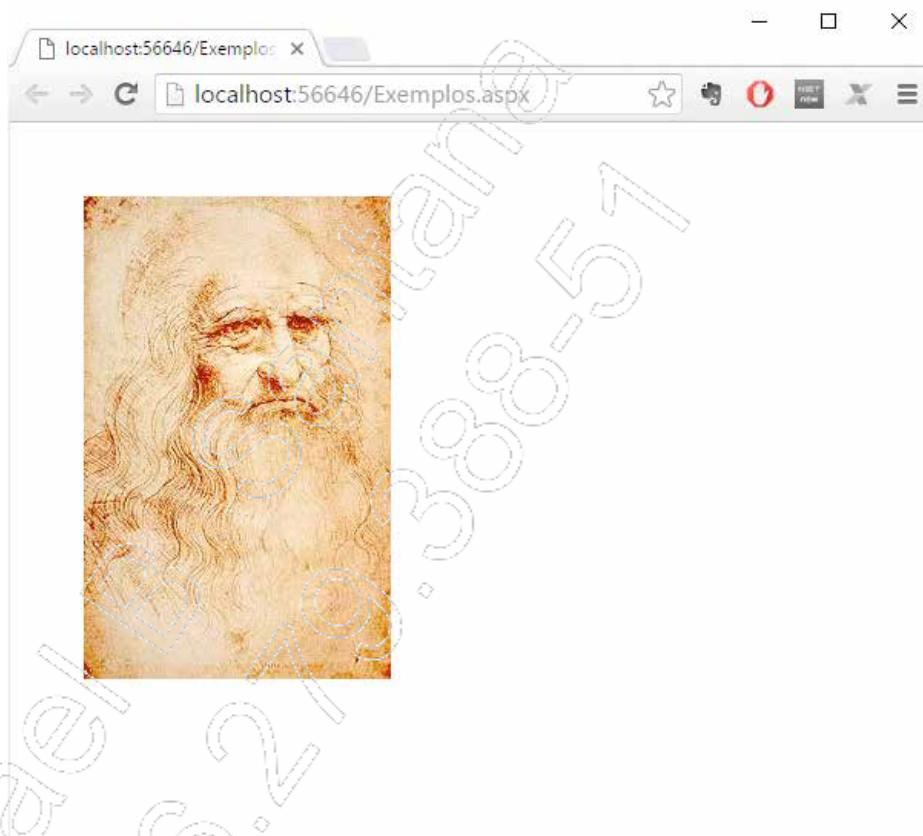


3.3.2.3. Controles para exibir imagens

Os controles para exibir imagens possuem a propriedade **ImageUrl** para definir a localização do arquivo de imagem. Os controles normalmente utilizados são **Image**, **ImageButton** e **HyperLink**.

Veja um exemplo do controle **Image**:

```
<asp:Image runat="server"  
    Title="Leonardo da Vinci"  
    id="fotoImage"  
    ImageUrl("~/Leonardo_self.jpg" />
```



3.3.2.4. Controles para agrupar outros controles

Um controle pode conter outros controles dentro dele. Isso é útil para agrupar controles relacionados e utilizar o recurso de exibir ou esconder controles. Dentre os controles utilizados para agrupar outros controles, podemos destacar **Panel**, **MultiView** e **Wizard**.

O controle **MultiView** permite, dentro dele, apenas controles do tipo View. A propriedade **ActiveViewIndex** controla qual View estará visível:

```
<asp:MultiView runat="server" ID="multiView1" ActiveViewIndex="0">

    <asp:View runat="server" >
        <p>Painel 0....</p>
    </asp:View>

    <asp:View runat="server" >
        <p>Painel 1....</p>
    </asp:View>

</asp:MultiView>
```

3.3.3. Controles de validação

O ASP.NET possui como um dos principais objetivos fornecer funcionalidades que possam abranger os cenários mais utilizados. A maioria dos sites não permite o acesso de visitantes a seus conteúdos mais importantes até que eles se autentiquem como usuários válidos. Controles de login e toda uma infraestrutura de segurança foram incluídos no ASP.NET. Os controles de validação trabalham com essa infraestrutura de segurança para gerar autorização e autenticação de modo mais simples.

Outro cenário normalmente encontrado durante a navegação em Web sites são páginas nas quais é possível que diversos tipos de dados sejam inseridos por usuários, como uma página para inserção de nome de usuário e senha, ou uma caixa de texto na qual inserimos um endereço de e-mail para recebermos mensagens, por exemplo.

O Web site que requer informações de usuários deve possuir uma lógica de programação que garanta o recebimento de informações válidas, mesmo que não sejam 100% corretas. Existem meios de se limitar ou estipular os tipos de dados que podem ser inseridos em campos específicos. Campos de dados obrigatórios, quando não preenchidos, não permitirão a continuidade do processo, por exemplo, ou certos tipos de dados de estrutura fixa, como números de telefone ou endereços de e-mail, vão se definir por uma expressão regular que impõe um determinado formato para a informação.

Um host de controle de validação acompanha os controles padrão (como um **TextBox**, por exemplo) em um Web Form no ASP.NET. Os controles de validação e os controles padrão trabalham em acordo entre si. Eles podem emitir mensagens de erro e alertas caso informações que possam ser incorretas sejam inseridas.

Vejamos, na tabela a seguir, os tipos de validadores do ASP.NET:

Tipo de validação	Controle a ser usado	Descrição
Entrada requerida	RequiredFieldValidator	Impede o usuário de omitir alguma entrada.
Comparação com um valor	CompareValidator	Compara uma entrada de usuário com um valor constante, com o valor de outro controle (usando um operador de comparação, como less than , equal ou greater than), ou com um tipo específico de informação.

Tipo de validação	Controle a ser usado	Descrição
Checagem de faixa ou intervalo	RangeValidator	Confere se uma entrada de usuário está entre limites especificados (menores e maiores). É possível checar faixas de alcance entre dois números, caracteres alfabéticos ou datas.
Combinação de padrões	RegularExpressionValidator	Analisa se a entrada se equipara a um padrão definido por uma expressão regular. Esse tipo de validação nos permite checar sequências previsíveis de caracteres, como aquelas em endereços de e-mail, números de telefone, códigos postais etc.
Definida por usuário	CustomValidator	Checa a entrada do usuário usando lógica de validação que nós mesmos escrevemos. Esse tipo de validação nos permite procurar por valores derivados em tempo de execução.

 Mais de um controle de validação pode ser anexado a um controle de entrada.

Veja um exemplo de um controle de validação:

```
<form id="form1" runat="server">

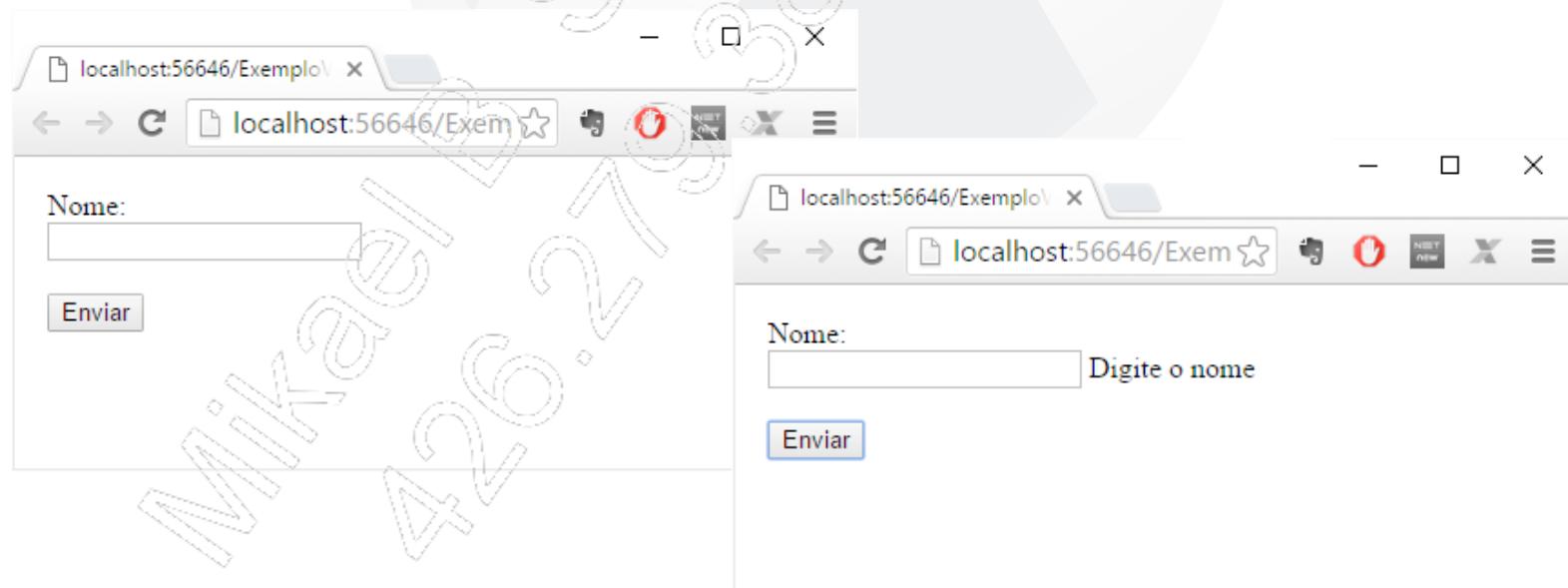
    <asp:Label ID="nomeLabel" runat="server" Text="Nome:></asp:Label>

    <asp:TextBox ID="nomeTextBox" runat="server"></asp:TextBox>

    <asp:RequiredFieldValidator ID="nomeRequiredFieldValidator"
        runat="server"
        ControlToValidate="nomeTextBox"
        ErrorMessage="Digite o nome"></asp:RequiredFieldValidator>

    <asp:Button ID="enviarButton" runat="server" Text="Enviar" />

</form>
```



3.3.3.1. Validação pelo cliente e validação pelo servidor

A validação da página ASP.NET é toda baseada na arquitetura do controle pelo servidor da página. Seu mecanismo de validação resolve a maioria dos casos de uso comum encontrados durante o desenvolvimento de um Web site.

A validação, na maioria dos sites, pode ser pelo servidor ou pelo cliente. A vantagem da validação pelo cliente é a possibilidade de evitar um trajeto de ida e volta do cliente ao servidor, para validar dados inseridos na página. Já a validação pelo servidor garante a não alteração de dados, que podem ser adulterados ou modificados durante a transmissão e também atende a navegadores que não suportam scripts executados pelo cliente.

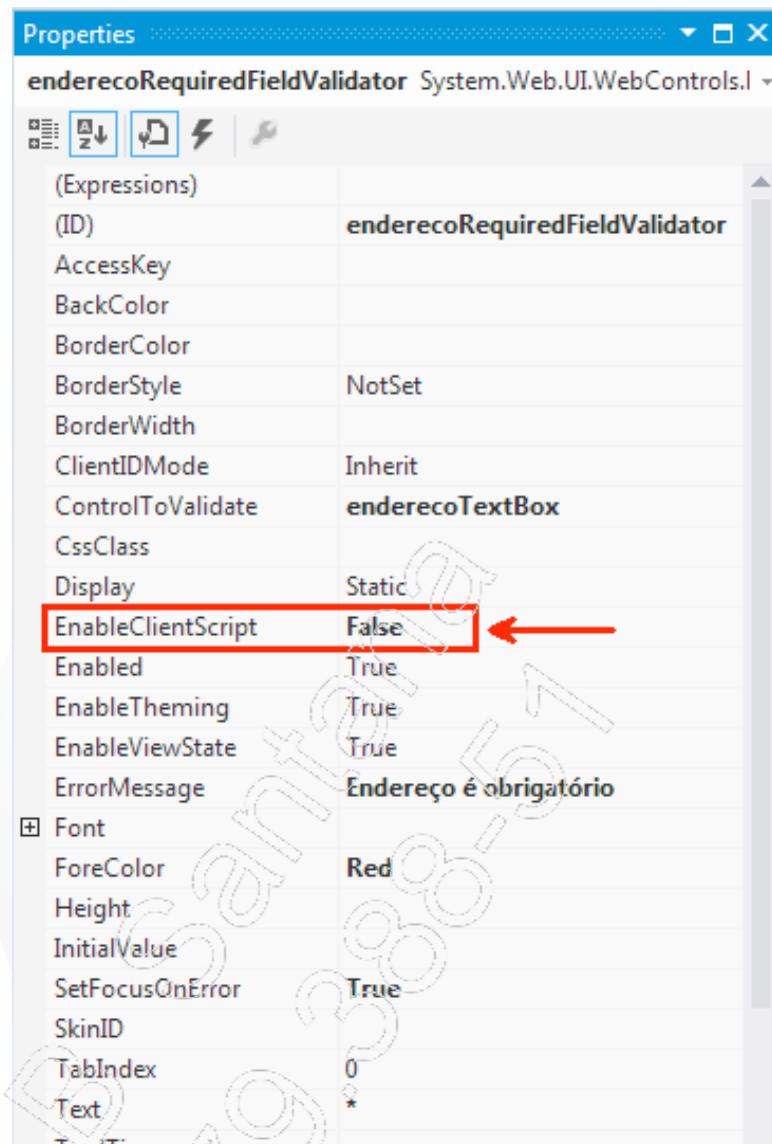
- **Validação pelo cliente**

Quando colocamos controles numa página, é comum vermos uma grande quantidade de tags no código-fonte do ASPX, que é gerado pelo Visual Studio. Isso ocorre porque cada controle validador inserido na página é correspondente a uma tag diferente. Os validadores renderizam códigos padrão que são interpretáveis pelo navegador, portanto, são como controles comuns do servidor.

Os controles validadores do ASP.NET incluem referências a utilitários JavaScript, incluídos no HTML enviado ao navegador para que a validação pelo cliente seja realizada. As funções de validação pelo cliente, essenciais para permitir tal validação, estão contidas no arquivo. Os controles de validação adicionam elementos span com atributos padrão ao HTML renderizado quando realizam a renderização para o navegador. Quando o documento HTML é carregado no navegador, os handlers de validação estão ligados.

Cientes sem suporte para JavaScript dependem diretamente da validação pelo servidor, pois não são capazes de realizar validação pelo cliente. Podemos definir a propriedade **EnableClientScript** no validador como **false** para desabilitar o script de cliente em cada controle.

Vejamos, a seguir, um exemplo de definição da propriedade **EnableClientScript** como **false** para um **RequiredFieldValidator**:



- **Validação pelo servidor**

A validação pelo servidor, gerenciada por uma infraestrutura na classe **Page**, é iniciada quando a solicitação é enviada de volta para o servidor, logo após o cliente passar nos testes de validação pelo cliente. Os controles de validação são adicionados a uma coleção de validadores que a página gerencia, enquanto os adicionamos à página. Uma interface chamada **IValidator** é implementada por cada controle de validação. Essa interface especifica um método **Validate**, uma propriedade **ErrorMessage** e uma propriedade **IsValid**. Cada validador tem sua própria lógica particular que determina a validade dos dados existentes no controle que está sendo validado.

A validação ocorre logo após o disparo (fires) do evento **Page_Load**, durante a sequência de postback para uma página, a qual verifica cada validador em contraste com seu controle associado. Os controles de validação pelo servidor podem falhar, e, nesses casos, aqueles que falharem renderizam-se como elementos span visíveis.

A propriedade **IsValid**, encontrada na própria página, pode ser marcada, aumentando a confiabilidade dos dados que o cliente envia para serem usados nos controles. O método **Validate()**, implementado pela classe **Page**, encontra-se nas listas de controle da validação para executar esse método pertencente a cada controle.

É de grande importância providenciarmos garantias de que os campos preenchidos por usuários apresentem a maior veracidade possível. Apesar de não haver como nos certificar da exatidão de uma informação, como um número de telefone, ou endereço eletrônico, podemos limitar e definir caracteres específicos, obrigando o usuário a inserir informações em um formato delimitado.

Vejamos, a seguir, um exemplo de validação efetuada pelo servidor (objeto **CustomValidator**):

```
protected void cidadeCustomValidator_ServerValidate(object source, ServerValidateEventArgs args)
{
    if (cidadeTextBox.Text.ToUpper() == "SÃO PAULO")
    {
        //Ok. Associação permitida para morador de São Paulo
        args.IsValid = true;
    }
    else
    {
        //Não válido. Associação não permitida para moradores de outra cidade
        args.IsValid = false;
    }
}
```

3.3.3.2. Propriedades dos controles de validação

Podemos encontrar propriedades padrão de outros controles padrão ASP.NET inseridas nos próprios controles de validação, como uma propriedade **Font**, uma propriedade **Text**, propriedades de cores, além de algumas propriedades que gerenciam a saída de erro que é enviada para o navegador.

Vejamos agora uma breve descrição de algumas propriedades dos controles de validação:

- **Text**: A propriedade **Text** acomoda o texto que será apresentado quando o validador identificar um conteúdo inválido para o controle associado;
- **Display**: A propriedade **Display** gerencia a renderização da mensagem de erro no lado do cliente, e seu valor pode ser tanto **Static** quanto **Dynamic**;

O elemento span altera o layout e se expande de modo dinâmico quando mostrado caso a propriedade **Display** seja definida como **Dynamic**. Em contrapartida, ao ser definido como **Static**, o span restringe-se ao limite do HTML do cliente, mesmo quando oculto.

- **ValidationGroup**: A propriedade **ValidationGroup** controla o nome do grupo no ASP.NET, que tem a capacidade de agrupar controles de validação, definindo grupos nomeados para classificá-los. Quando um controle pertence a um grupo, é necessário que um dos controles do grupo dispare para que todos os outros controles nesse grupo façam a validação, causando um efeito de múltiplos formulários em uma única página.

3.3.4. Controles de usuário

Controles de usuário são controles compostos que contêm controles filho do mesmo modo que os controles compostos. Entretanto, em vez de derivarem do **System.Web.UI.CompositeControl**, eles derivam do **System.Web.UI.UserControl**. São similares a páginas Web ASP.NET completas (um arquivo **.aspx**), com uma página de interface de usuário e um código, e possuem um componente de UI (um arquivo **.ascx**) que funciona com o Visual Studio Designer. São criados do mesmo modo que uma página ASP.NET e, então, adicionamos o markup e os controles filho necessários. Diferentemente de um **Web Form**, podemos arrastar um controle de usuário e soltá-lo dentro de um **Web Form**.

Visual Studio 2015 - ASP.NET com C# Fundamentos

Vejamos o que diferencia um controle de usuário de uma página Web ASP.NET:

- A extensão de arquivo de um controle de usuário é **.ascx**;
- Em vez de uma diretiva **@ Page**, o controle de usuário contém uma diretiva **@ Control** que define a configuração e outras propriedades;
- Controles de usuário não podem ser executados como arquivos solitários. Eles devem ser adicionados a páginas ASP.NET como qualquer controle;
- O controle de usuário não possui elementos do tipo **html**, **body** ou **form**, os quais devem encontrar-se na página de hospedagem.

! Com exceção dos elementos **html**, **body** e **form**, podemos usar, em um controle de usuário, os mesmos elementos HTML e os controles Web utilizados em uma página Web ASP.NET.

Pontos principais

Atente para os tópicos a seguir. Eles devem ser estudados com muita atenção, pois representam os pontos mais importantes do capítulo.

- Um Web Form é uma página processada no servidor com a extensão `.aspx`. O HTML da página é compilado em uma classe que fica armazenada em cache;
- O código associado à página HTML é chamado de `code-behind`;
- Existem três tipos de controles de servidor: HTML Controls, Web Controls e User Controls;
- Controles do lado do servidor do ASP.NET são derivados da classe `System.Web.UI.Control`, e os seguintes tipos de controles podem ser utilizados: controles de servidor HTML, controles de servidor Web, controles de validação e controles de usuário;
- Os controles de servidor HTML são controles HTML com o atributo `runat` definido com `server`. Dessa forma, podem ser manipulados pelo servidor;
- Controles de validação permitem a definição do que é permitido ou não quando inserimos dados, nos permitindo realizar testes em comparação com um valor específico ou um padrão de caracteres, verificar se um valor encontra-se dentro de um limite definido, entre outras limitações que podemos aplicar;
- Controles de usuário são controles compostos usados para criarmos barras de ferramentas e outros elementos reutilizáveis, embutindo controles de usuário em outras páginas Web ASP.NET.

3

Web Forms

Teste seus conhecimentos

Mikael B
426.279.3557



IMPACTA
EDITORA

1. Qual atributo é obrigatório em toda marcação HTML para que ela seja analisada e processada no servidor?

- a) ID
- b) runat="server"
- c) name
- d) asp:
- e) class

2. Como é chamado o recurso do ASP.NET que analisa a página HTML para criar árvore de controle?

- a) Compilation
- b) Request
- c) Response
- d) Debug
- e) Parse

3. Qual o controle de validação apropriado para validar se um campo do tipo e-mail foi preenchido com o formato correto?

- a) EmailValidator
- b) RequiredFieldValidator
- c) RegularExpressionValidator
- d) RangeValidator
- e) CustomValidator

4. Qual das seguintes informações a respeito de um User Control é falsa?

- a) Usa a extensão .ascx.
- b) Possui um evento Page_Load.
- c) Pode ser chamado diretamente de um navegador.
- d) Pode ter acesso aos controles da página hospedeira.
- e) Pode ter tags HTML, Web Controls e JavaScript.

5. Quais são os três tipos de controles de servidor?

- a) Client, Server e Host.
- b) HTML, CSS e JavaScript.
- c) Web Controls, HTML Controls e User Controls.
- d) Web Forms, Web Pages e Web Sites.
- e) Form Controls, Validation Controls e List Controls.

3

Web Forms

Mãos à obra!

Mikael B
Scotana
426.279.57



IMPACTA
EDITORA

Laboratório 1

Objetivos:

- Criar um formulário usando o modelo Web Forms;
- Usar os controles **Label**, **TextBox**, **DropDownList** e **Button**;
- Usar os recursos de validação para cada tipo de campo;
- Gravar um arquivo de texto com informações válidas.

1. Crie um projeto Web vazio;

2. Insira um Web Form no projeto chamado **Default.aspx**;

3. Insira o título da página:

```
<h1>Capítulo 03 - Criando Formulários</h1>
```

4. Insira, também, um multiview com dois views: um para o formulário e outro para a mensagem de cadastramento;

```
<asp:MultiView ID="formMultiView" runat="server"
    ActiveViewIndex="0">

    <asp:View ID="formView" runat="server"></asp:View>

    <asp:View ID="mensagemView" runat="server"></asp:View>

</asp:MultiView>
```

5. No primeiro view, será exibido um formulário para receber as seguintes informações (todos os campos são obrigatórios):

- **Nome, Email, Cidade e Estado** do tipo texto. **Estado** deve ser uma lista com os valores **SP, RJ e Outros**;
- **Total de Dependentes**, um número inteiro entre **0 e 20**;
- **Data de Nascimento**, do tipo data;
- **Pretensão Salarial**, do tipo moeda.

Cada campo será composto de um Label para a legenda, um TextBox, DropDownList ou CheckBox para o campo e os controles de validação, usando a tag **
** para criar linhas separadas. Nos próximos capítulos, veremos como usar folhas de estilo para formatar melhor os objetos de tela. No momento, o importante é o código HTML ficar simples para poder ser estudado detalhadamente.

Em seguida, o HTML do campo **Nome**, contendo as tags para um Label e um TextBox de um parágrafo (**<p>**). Altere o primeiro view para ter o código a seguir e teste o programa (CTRL + F5).

! Adquira o hábito de visualizar o HTML gerado pelo servidor e identificar o que cada Web Control gera.

```
<asp:View ID="formView" runat="server">

<!-- nome -->
<p>
    <asp:Label ID="nomeLabel"
        runat="server"
        AssociatedControlID="nomeTextBox"
        Text="Nome:>
    </asp:Label>

    <br />

    <asp:TextBox ID="nomeTextBox" runat="server"></asp:TextBox>

</p>

</asp:View>
```

Visual Studio 2015 - ASP.NET com C# Fundamentos

Veja, a seguir, o resultado:



O Web Control **TextBox** gerou uma tag HTML **input** e o Web Control **Label** gerou uma tag HTML **Label**. Veja o código HTML gerado:

```
view-source:localhost:57599/Default.aspx
<div>
    <h1>Capítulo 03 - Criando Formulários</h1>
<!-- nome -->
<p>
    <label for="nomeTextBox" id="nomeLabel">Nome:</label>
    <br />
    <input name="nomeTextBox" type="text" id="nomeTextBox" />
</p>
```

The screenshot shows the browser's developer tools with the source code of the page. It includes the generated HTML for the heading and the form control. The code is color-coded for syntax highlighting, with labels in purple and tags in blue.

6. O campo **Email** tem exatamente a mesma configuração do campo **Nome**: um Label e uma legenda. Insira o campo **Email** abaixo do campo **Nome** e visualize o resultado e o código-fonte gerado;

```
<asp:View ID="formView" runat="server">

<!-- nome -->
<p>
    <asp:Label ID="nomeLabel"
        runat="server"
        AssociatedControlID="nomeTextBox"
        Text="Nome:>
    </asp:Label>

    <br />

    <asp:TextBox ID="nomeTextBox" runat="server"></asp:TextBox>
</p>

<!-- Email -->
<p>
    <asp:Label ID="emailLabel"
        runat="server"
        AssociatedControlID="emailTextBox"
        Text="Email:>
    </asp:Label>

    <br />

    <asp:TextBox ID="emailTextBox" runat="server"></asp:TextBox>
</p>

</asp:View>
```

Visual Studio 2015 - ASP.NET com C# Fundamentos

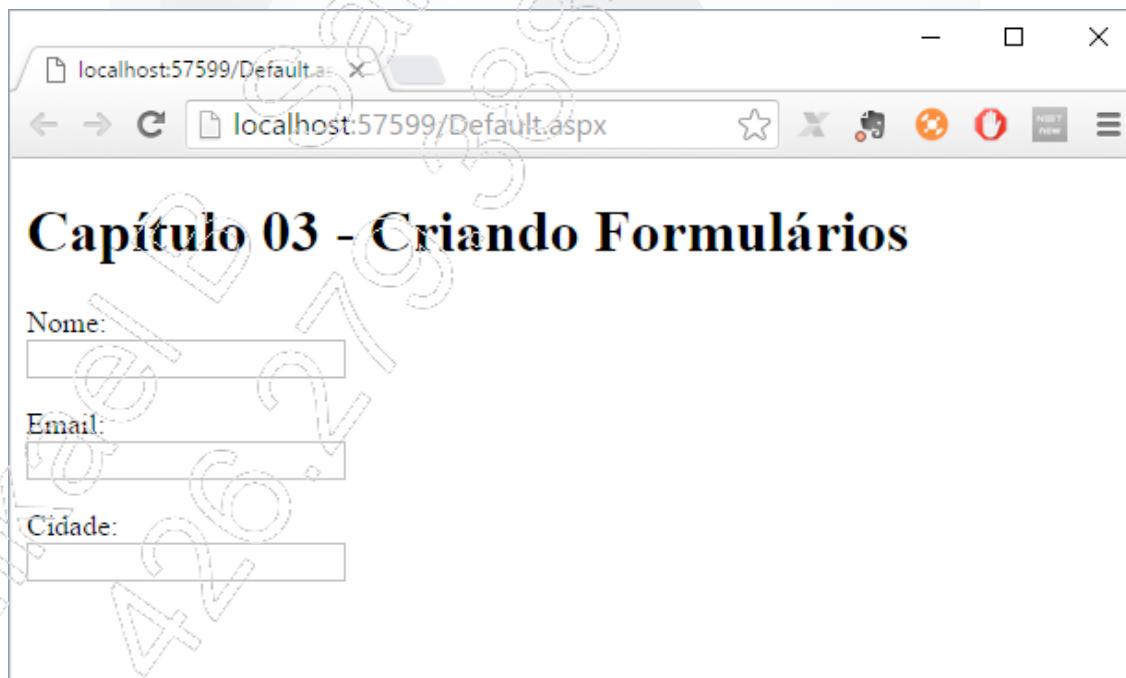
7. O campo **Cidade** segue o mesmo padrão dos campos **Nome** e **Email**. Insira o HTML do campo **Cidade** abaixo do campo **Email**:

```
<!-- Cidade -->
<p>
    <asp:Label ID="cidadeLabel"
        runat="server"
        AssociatedControlID="cidadeTextBox"
        Text="Cidade:>
    </asp:Label>

    <br />

    <asp:TextBox ID="cidadeTextBox" runat="server"></asp:TextBox>
</p>
```

Veja, a seguir, o resultado até o momento:



8. O campo **Estado** é uma lista com três itens: **SP**, **RJ** e **Outros**. O Web Control utilizado para esse caso é o **DropDownList**. A propriedade que define a lista se chama **Items** e é uma coleção de objetos do tipo **ListItem**. Cada objeto **ListItem** tem como principais propriedades **Text** (o que aparece na tela) e **Value** (o valor retornado). Insira o HTML do campo **Estado** abaixo do campo **Cidade**:

```
<!-- Estado -->
<p>
    <asp:Label ID="estadoLabel"
        runat="server"
        AssociatedControlID="estadoDropDownList"
        Text="Estado:>
    </asp:Label>

    <br />

    <asp:DropDownList ID="estadoDropDownList"
        runat="server">

        <asp:ListItem Value="SP">SP</asp:ListItem>
        <asp:ListItem Value="RJ">RJ</asp:ListItem>
        <asp:ListItem Value="Outros">Outros</asp:ListItem>

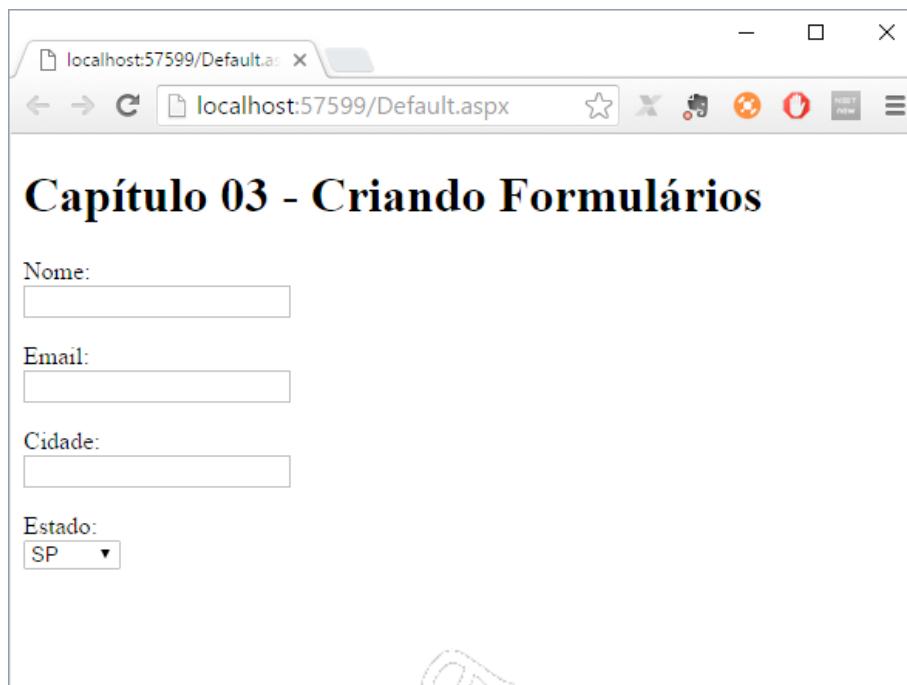
    </asp:DropDownList>
</p>
```

Quando o valor retornado é o mesmo do valor exibido (como no caso anterior), o atributo **Value** pode ser omitido. O código HTML do controle **EstadoDropDownList** pode também ser escrito assim:

```
<asp:DropDownList ID="estadoDropDownList" runat="server">
    <asp:ListItem>SP</asp:ListItem>
    <asp:ListItem>RJ</asp:ListItem>
    <asp:ListItem>Outros</asp:ListItem>
</asp:DropDownList>
```

Visual Studio 2015 - ASP.NET com C# Fundamentos

9. Visualize o resultado até o momento:



O código HTML gerado pelo DropDownList é um select. Veja o código a seguir e repare que o ASP.NET coloca o value no código gerado, mesmo que não esteja definido na tag do servidor.

```
<select name="estadoDropDownList" id="estadoDropDownList">
    <option value="SP">SP</option>
    <option value="RJ">RJ</option>
    <option value="Outros">Outros</option>
</select>
```

10. Os campos **Total de Dependentes**, **Pretensão Salarial** e **Data de Nascimento** usam Label e TextBox, igual aos primeiros do formulário. Insira o código HTML para esses três campos;

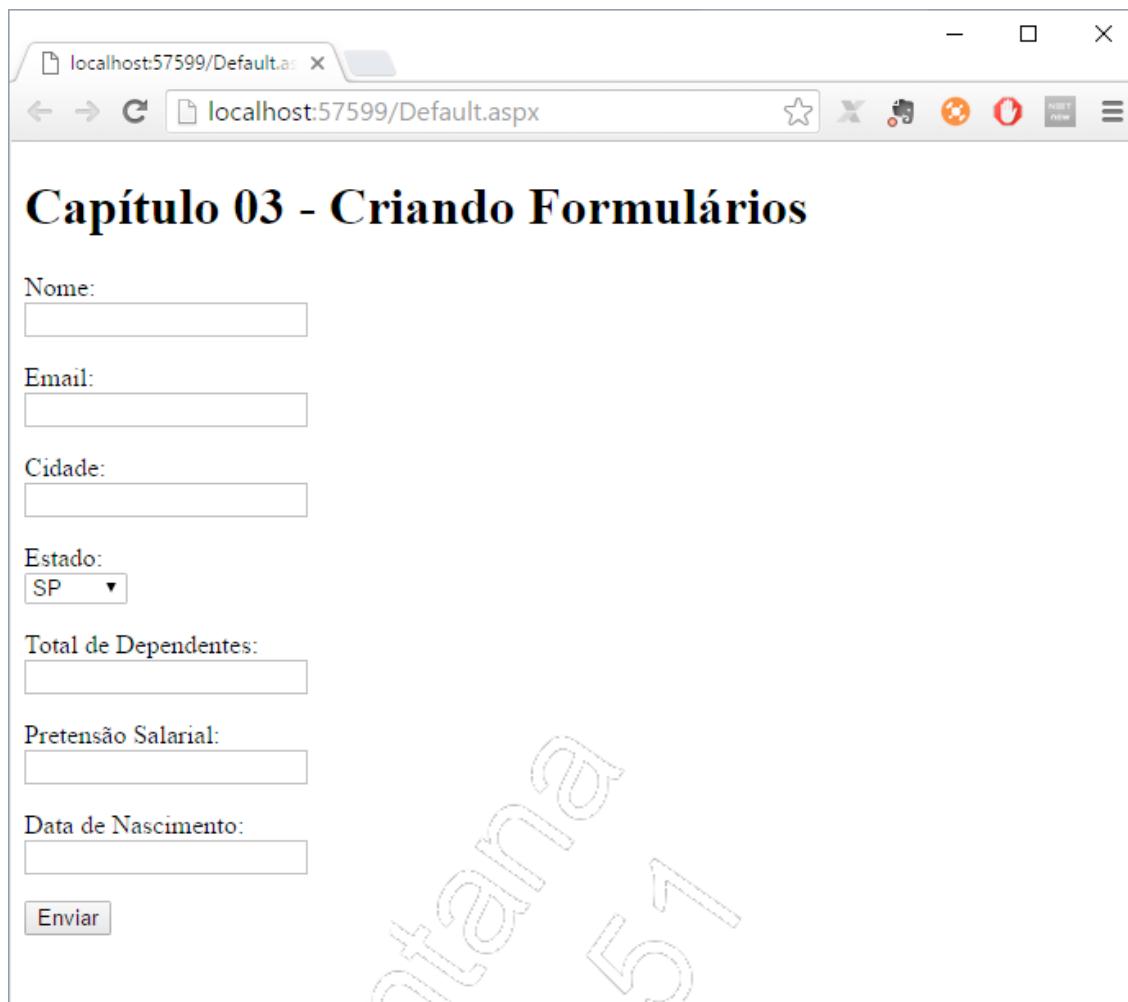
```
<!-- Total de Dependentes -->
<p>
    <asp:Label ID="totalDependentesLabel"
        runat="server"
        AssociatedControlID="totalDependentesTextBox"
        Text="Total de Dependentes:>
    </asp:Label>
    <br />
    <asp:TextBox ID="totalDependentesTextBox"
        runat="server">
    </asp:TextBox>
</p>
```

```
<!-- Pretensão Salarial -->
<p>
    <asp:Label ID="pretensaoSalarialLabel"
        runat="server"
        AssociatedControlID="pretensaoSalarialTextBox"
        Text="Pretensão Salarial:>
    </asp:Label>
    <br />
    <asp:TextBox ID="pretensaoSalarialTextBox"
        runat="server">
    </asp:TextBox>
</p>

<!-- Data de Nascimento -->
<p>
    <asp:Label ID="dataNascimentoLabel"
        runat="server"
        AssociatedControlID="dataNascimentoTextBox"
        Text="Data de Nascimento:>
    </asp:Label>
    <br />
    <asp:TextBox ID="dataNascimentoTextBox"
        runat="server">
    </asp:TextBox>
</p>
```

11. Finalmente, insira o botão **Enviar** e visualize o formulário completo;

```
<!-- Enviar -->
<p>
    <asp:Button ID="enviarButton"
        runat="server"
        Text="Enviar" />
</p>
```



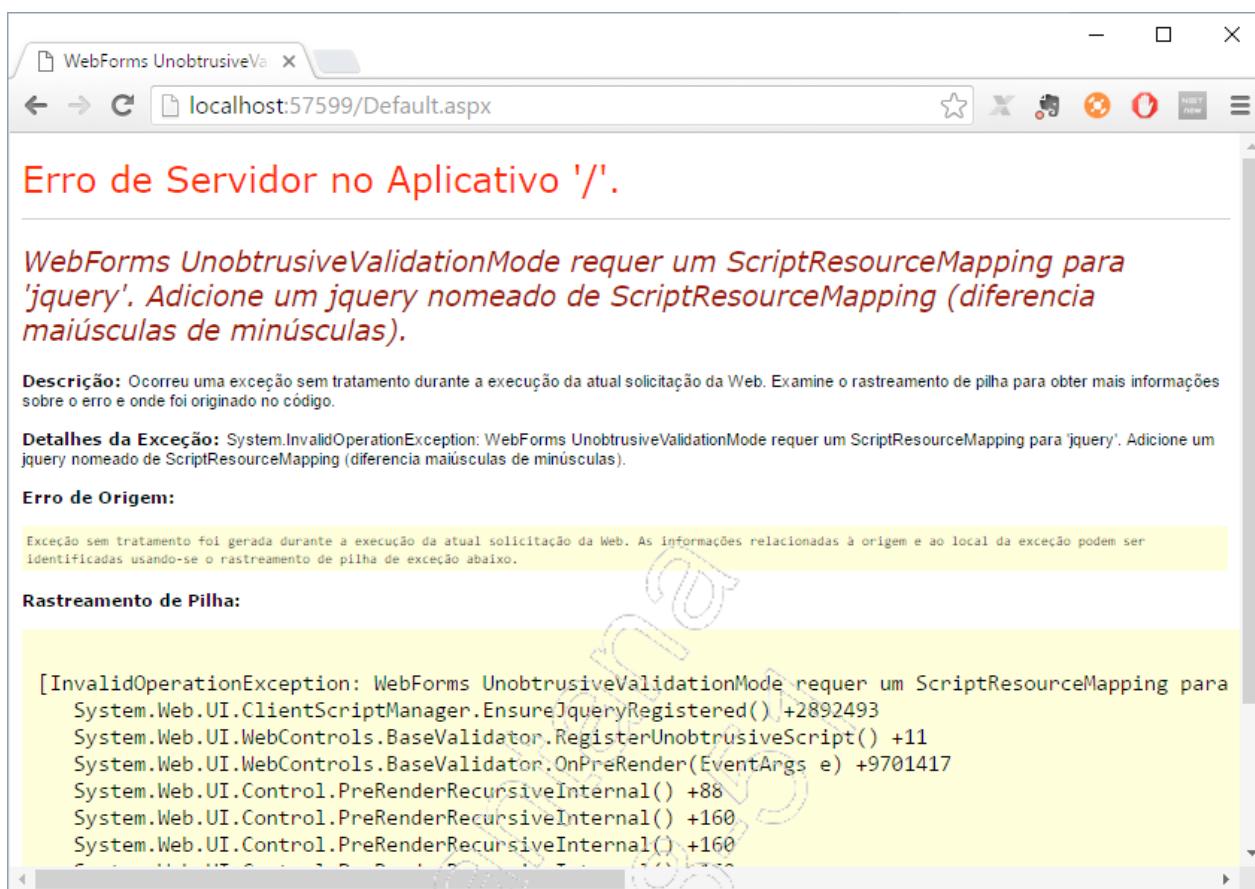
12. A próxima etapa é validar os campos. O campo **Nome** é obrigatório, portanto vamos usar o controle **RequiredFieldValidator**:

```
<!-- nome -->
<p>
    <asp:Label ID="nomeLabel" runat="server"
        AssociatedControlID="nomeTextBox" Text="Nome:>
    </asp:Label>
    <br />
    <asp:TextBox ID="nomeTextBox"
        runat="server">
    </asp:TextBox>

    <asp:RequiredFieldValidator ID="nomeRequiredFieldValidator"
        runat="server" ControlToValidate="nomeTextBox"
        Text="*"
        ErrorMessage="Digite o nome"
        ForeColor="red">
    </asp:RequiredFieldValidator>

</p>
```

13. Visualize a página. Dependendo da versão do Visual Studio e do .NET Framework, pode ocorrer um erro como este:



A partir da versão 4.0 do .NET Framework, o funcionamento dos validadores mudou. Antes, o ASP.NET gerava um código em JavaScript que funcionava para validar no cliente. Esse modo ainda pode ser usado, mas agora existe a possibilidade de usar a biblioteca jQuery (ou outra biblioteca) para realizar a validação usando um recurso chamado **Unobtrusive Validation** (validação não invasiva).

A validação não invasiva se resume a usar padrões de validação que separam o código HTML do JavaScript.

Existem duas maneiras de resolver esse problema: definir que os validadores vão funcionar como antes ou configurar o sistema para usar a biblioteca de validação não invasiva (por padrão jQuery).

Neste laboratório, vamos desligar essa funcionalidade. Nos próximos capítulos, nos quais falaremos de outras tecnologias, vamos usar a solução de configurar o jQuery.

Visual Studio 2015 - ASP.NET com C# Fundamentos

Solução 1: Desabilitando a validação não invasiva:

Inclua o seguinte item no Web.Config:

```
<configuration>
  <appSettings>
    <add
      key="ValidationSettings:UnobtrusiveValidationMode"
      value="None" />
  </appSettings>
```

```
<system.web>
  <compilation debug="true" targetFramework="4.5.1" />
  <httpRuntime targetFramework="4.5.1" />
</system.web>
</configuration>
```

14. Visualize a página, teste a validação e tente enviar o formulário sem preencher o campo **Nome**:



15. Coloque validadores para os outros campos, alterando as propriedades **ControlToValidate** e **ErrorMessage** apropriadamente e teste o formulário. Do lado de cada controle não preenchido deve aparecer um asterisco (*), porque está definido na propriedade **Text** de cada validador;

A propriedade **ErrorMessage** aparece do lado do controle apenas se a propriedade **Text** não estiver definida. Como **Text** está definido para *, a única maneira de visualizar no formulário é adicionando um controle **ValidationSummary**.

16. Após o botão, insira um validador do tipo **ValidationSummary**:

```
...
<!-- Enviar -->
<p>
    <asp:Button ID="enviarButton"
        runat="server"
        Text="Enviar" />
</p>

<asp:ValidationSummary runat="server"
    ForeColor="red" ID="formValidationSummary" />

</asp:View>
...
```

A propriedade **ErrorMessage** de cada validador será exibida no **ValidationSummary**. A propriedade **Text** é exibida no local onde o **RequiredFieldValidator** se encontra.

Visual Studio 2015 - ASP.NET com C# Fundamentos

17. Teste o formulário com as validações:

localhost:57599/Default.aspx

localhost:57599/Default.aspx

Capítulo 03 - Criando Formulários

Nome: *

Email:

Cidade: *

Estado:

Total de Dependentes:

Pretensão Salarial:

Data de Nascimento: *

• Digite o nome
• Digite a cidade
• Digite a Data de Nascimento

18. No exemplo anterior, o campo **Email** está preenchido e por isso está em um estado válido para o **RequiredFieldValidator**;

Acontece que o formato do e-mail está errado. Para validar um e-mail, adicione um **RegularExpressionValidator** (parte do código já escrito foi omitido).

A propriedade **ValidationExpression** é uma **Expressão Regular**, que é uma linguagem para validar padrões. Por enquanto vamos utilizar o Visual Studio para nos fornecer uma expressão regular válida para o e-mail.

```
<!-- email -->
<p>

<asp:Label ... ID="emailLabel" ...

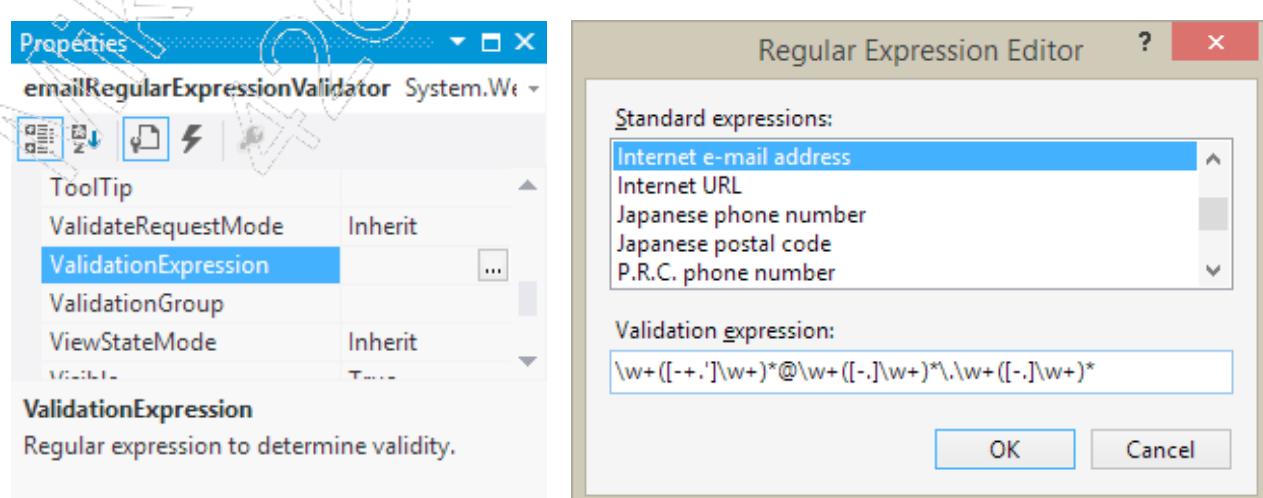
<asp:TextBox ID="emailTextBox" ...

<asp:RequiredFieldValidator runat="server" ...

<asp:RegularExpressionValidator
    ID="emailRegularExpressionValidator"
    runat="server"
    ControlToValidate="emailTextBox"
    Text ="*"
    ErrorMessage="Email Inválido"
    ValidationExpression="..."
    ForeColor="Red">
</asp:RegularExpressionValidator>

</p>
```

No modo designer, não precisa digitar, pode clicar na propriedade **ValidationExpression** do controle **emailRegularExpressionValidator** e escolher uma expressão da lista disponível.



```
<asp:RegularExpressionValidator  
    ID="emailRegularExpressionValidator"  
    runat="server"  
    ControlToValidate="emailTextBox"  
    Text="*"  
    ErrorMessage="Email Inválido"  
    ValidationExpression=  
        "\w+([-.\']\w+)*@\w+([-.\']\w+)*\.\w+([-.\']\w+)*"  
    ForeColor="Red">  
</asp:RegularExpressionValidator>
```

19. Teste o validador de e-mail:

localhost:57599/Default.aspx

Capítulo 03 - Criando Formulários

Nome: Jose

Email: jose *

Cidade: São Paulo

Estado: SP

Total de Dependentes: 1

Pretensão Salarial: 90.000,00

Data de Nascimento: 01/01/1960

Enviar

• Email Inválido

20. Quando um controle tem dois validadores, como no caso do **Email**, é necessário configurar a propriedade **Display** para ocupar o espaço apenas se estiver sendo exibida. É possível perceber que o asterisco (*) muda de lugar quando o texto é deixado vazio e depois preenchido com formato inválido. Para controlar isso, defina a propriedade **Display="Dynamic"**. O padrão é **Static**, o que significa que o espaço é reservado, mesmo estando invisível;

```
<asp:RequiredFieldValidator ID="nomeRequiredFieldValidator"
    runat="server"
    Display="Dynamic"
    ControlToValidate="nomeTextBox"
    Text="*"
    ErrorMessage="Digite o nome"
    ForeColor="red">
</asp:RequiredFieldValidator>
```

21. Para o campo **Total de Dependentes**, o valor válido é um número inteiro entre 0 e 20. O controle **RangeValidator** fornece essa validação. As propriedades **Minimum** e **Maximum** definem os limites, e a propriedade **Type** define o tipo (numérico inteiro);

```
<asp:RangeValidator
    ID="totalDependentesRangeValidator"
    runat="server" ControlToValidate="totalDependentesTextBox"
    Text="*"
    ForeColor="red"
    ErrorMessage="Digite um numero entre 0 e 20"
    MinimumValue="0"
    MaximumValue="20"
    Type="Integer">
</asp:RangeValidator>
```

22. Para o campo **Pretensão Salarial**, o valor válido é um número no formato moeda. O controle **RangeValidator** fornece essa validação também. As propriedades **Minimum** e **Maximum** definem os limites (entre **0** e **999999**, por exemplo), e a propriedade **Type** define o tipo (numérico moeda);

```
<asp:RangeValidator  
    ID="pretensaoSalarialRangeValidator"  
    runat="server" ControlToValidate="pretensaoSalarialTextBox"  
    Text="*"  
    ForeColor="red"  
    ErrorMessage="Valor inválido para salário" MinimumValue="0"  
    MaximumValue="999999" Type="Currency">  
</asp:RangeValidator>
```

23. O campo **Data de Nascimento** poderia usar, também, o controle **RangeValidator**, mas, nesse caso, vamos usar outro controle, o **CompareValidator**. Ele pode ser usado para comparar se um campo é igual, maior ou menor que outro campo ou valor e definir se um campo é de um determinado tipo. A propriedade **Type** define o tipo (Data) e a propriedade **Operator** define a operação, nesse caso **DataTypeCheck** (checar o tipo de dados);

```
<asp:CompareValidator  
    ID="dataNascimentoCompareValidator"  
    runat="server" ControlToValidate="dataNascimentoTextBox"  
    Text="*"  
    ForeColor="red"  
    ErrorMessage="Data Inválida"  
    Operator="DataTypeCheck"  
    Type="Date">  
</asp:CompareValidator>
```

24. Visualize e teste os validadores:

localhost:57599/Default.aspx

localhost:57599/Default.aspx

Capítulo 03 - Criando Formulários

Nome: José

Email: jose@teste.com.br

Cidade: São Paulo

Estado: SP

Total de Dependentes: *
-1

Pretensão Salarial: *
Não Sei

Data de Nascimento: *
30/02/1950

Enviar

- Digite um numero entre 0 e 20
- Valor inválido para salário
- Data Inválida

25. Na view que mostra o resultado, insira um **Label** para exibir o resultado e um **Button** para iniciar outro registro:

```
<asp:View ID="mensagemView" runat="server">

<p>
    <asp:Label runat="server" ID="resultadoLabel">
    </asp:Label>
</p>

<p>
    <asp:Button runat="server"
        ID="novoButton"
        Text="Cadastrar Novo Registro" />
</p>

</asp:View>
```

26. Escrevendo o código do lado do servidor: neste laboratório, vamos coletar as informações deste formulário e gravar em um arquivo de texto. Insira este código no evento do botão **Enviar**:

```
//Processa apenas se a página é válida
if (!Page.IsValid)
{
    return;
}

//Obter os Dados
string nome = nomeTextBox.Text;
string email = emailTextBox.Text;
string cidade = cidadeTextBox.Text;
string estado = estadoDropDownList.SelectedValue;
int totalDependentes =
    Convert.ToInt32(totalDependentesTextBox.Text);
decimal pretensaoSalarial =
    Convert.ToDecimal(pretensaoSalarialTextBox.Text);
DateTime dataNascimento = Convert.ToDateTime(dataNascimentoTextBox.Text);

//Preparar o Registro com os dados
//separados por ponto e vírgula

string registro=string.Format(
    "{0};{1};{2};{3};{4};{5};{6}",
    nome, email, cidade, estado,
    totalDependentes, pretensaoSalarial,
    dataNascimento.ToShortDateString()
);

//Nome Completo do arquivo
string arquivo = "cadastro.csv";
string caminho = Server.MapPath(arquivo);

//Gravar
var sw = new StreamWriter(caminho, true, Encoding.UTF8);
sw.WriteLine(registro);
sw.WriteLine();
sw.Close();

//Exibir o Resultado
formMultiView.ActiveViewIndex = 1;
resultadoLabel.Text = "Registro adicionado com sucesso";
```

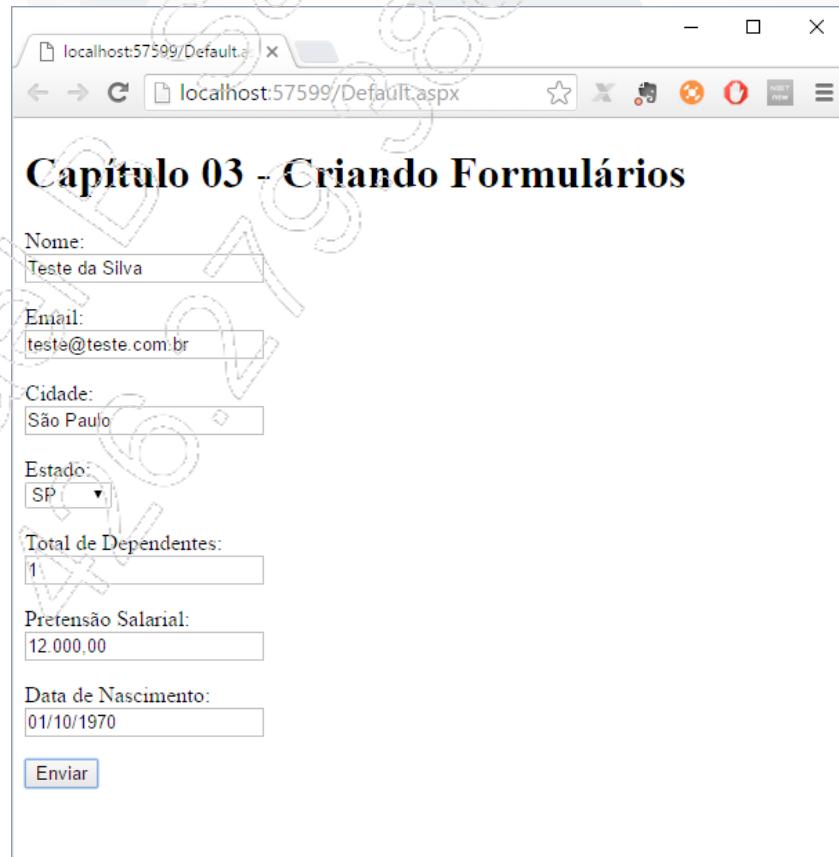
27. O código do botão **NovoRegistro** limpa os campos e exibe o primeiro painel:

```
//Limpar
nomeTextBox.Text = string.Empty;
emailTextBox.Text = string.Empty;
cidadeTextBox.Text = string.Empty;
estadoDropDownList.SelectedIndex = 0;
totalDependentesTextBox.Text = string.Empty;
pretensaoSalarialTextBox.Text = string.Empty;
dataNascimentoTextBox.Text = string.Empty;

//Primeiro Painel
formMultiView.ActiveViewIndex = 0;

//Foco no campo nome
nomeTextBox.Focus();
```

28. Teste o programa e verifique se foi criado o arquivo **cadastro.csv** na pasta em que o Web site está definido. Esse arquivo pode ser aberto no Excel.



Visual Studio 2015 - ASP.NET com C# Fundamentos

A tela a seguir é exibida quando um cadastro é realizado com sucesso:



- Abrindo a pasta do projeto:

Nome	Data de modificação	Tipo	Tamanho
bin	10/05/16 18:49	Pasta de arquivos	
obj	10/05/16 18:47	Pasta de arquivos	
Properties	10/05/16 18:47	Pasta de arquivos	
cadastro.csv	10/05/16 22:54	Arquivo de Valore...	1 KB
Cap03_Lab01.csproj	10/05/16 18:49	Visual C# Project f...	8 KB
Cap03_Lab01.csproj.user	10/05/16 18:49	Visual Studio Proj...	2 KB
Default.aspx	10/05/16 22:52	Arquivo ASPX	10 KB
Default.aspx.cs	10/05/16 22:52	Visual C# Source f...	3 KB
Default.aspx.designer.cs	10/05/16 22:50	Visual C# Source f...	11 KB
packages.config	10/05/16 18:47	XML Configuratio...	1 KB
Web.config	10/05/16 22:30	XML Configuratio...	2 KB
Web.Debug.config	10/05/16 18:47	XML Configuratio...	2 KB
Web.Release.config	10/05/16 18:47	XML Configuratio...	2 KB

- Abrindo o arquivo no Excel:

A screenshot of Microsoft Excel showing a spreadsheet titled "cadastro.csv - Excel". The data is as follows:

1	José da Silva	jose@teste.com.br	São Paulo	SP	1	25000	01/10/80
2	Teste da Silva	teste@teste.com.br	São Paulo	SP	1	12600	01/10/80
3							
4							
5							
6							
7							
8							
9							
10							
11							

4

MVC

- ✓ Modelos de desenvolvimento ASP.NET;
- ✓ ASP.NET MVC;
- ✓ Estrutura dos arquivos MVC;
- ✓ Routes;
- ✓ Controller;
- ✓ Views;
- ✓ HtmlHelper;
- ✓ Model e Views tipadas;
- ✓ GET e POST;
- ✓ Ciclo de requisição e resposta no MVC.



IMPACTA
EDITORA

4.1. Introdução

MVC é um padrão conceitual de arquitetura de software surgido em 1979 para criação de interfaces para usuários. É um conceito simples e elegante de reduzir a complexidade do desenvolvimento dividindo o código em três partes distintas, chamadas de **Model**, **View** e **Controller**, em que:

- **View**: É o que aparece para o usuário;
- **Controller**: É a parte do sistema que interage com o usuário, implementa a lógica necessária, determina qual View deve ser usada, qual informação é necessária e de que modo deve ser enviada e/ou recebida;
- **Model**: É a definição dos dados que o sistema está usando. Os dados do modelo são exibidos pelas Views e manipulados pelos Controllers.

É importante lembrar que MVC é um simples modelo conceitual e, portanto, não define nenhuma implementação específica. Existem implementações do padrão MVC em Java, Ruby, C++, C#, Python, entre outras linguagens, frameworks, aplicativos e plataformas.

ASP.NET MVC Framework é um conjunto de bibliotecas que implementa uma versão desse modelo usando recursos da plataforma .NET e tecnologias Web. É diferente da implementação em Java e Ruby, por exemplo, mas o conceito é o mesmo.

4.2. Modelos de desenvolvimento ASP.NET

Cada modelo de desenvolvimento de aplicativos Web possui características que permitem que o programador ou a equipe utilize o melhor do seu potencial e seja o mais eficiente possível de acordo com os requisitos do aplicativo.

- **Web Pages**

O modelo visto inicialmente, Web Pages, é o mais simples: páginas HTML com scripts que rodam do lado do servidor incluídos na página. Este tipo de desenvolvimento é muito parecido com PHP. Um programador que utiliza a tecnologia PHP e que conhece HTML/CSS/JavaScript consegue utilizar as Web Pages sem nenhuma dificuldade.

- **Web Forms**

O modelo baseado em eventos, Web Forms, é uma tentativa de tornar o desenvolvimento Web o mais parecido possível com o desenvolvimento Windows. Todo o funcionamento Client/Server é gerenciado pela página, ocultando do programador as chamadas via GET ou POST e a manipulação do código HTML para manter o estado da informação contida na tela. Os Web Controls que geram o código HTML facilitam muito o trabalho de criar elementos, atributos, vínculos como folhas de estilo, associação com eventos em JavaScript, entre outras tarefas que são necessárias ao escrever usando as tecnologias que o navegador entende.

- **MVC**

O MVC (Model-View-Controller) utiliza classes para criar a lógica da aplicação totalmente separado do HTML, assim como faz o code-behind dos Web Forms. Mas o MVC vai além. Não existe um vínculo de código de página relacionado direta e unicamente à página que é exibida. O mesmo código pode controlar diversas páginas e uma página pode ser exibida por diversos controladores de código. Não existe, como nos Web Forms, o conceito de Web Controls, em que o HTML é gerado automaticamente em tempo de execução. O programador tem controle total sobre o código HTML que é gerado. Isso não quer dizer que tudo deve ser escrito em detalhes.

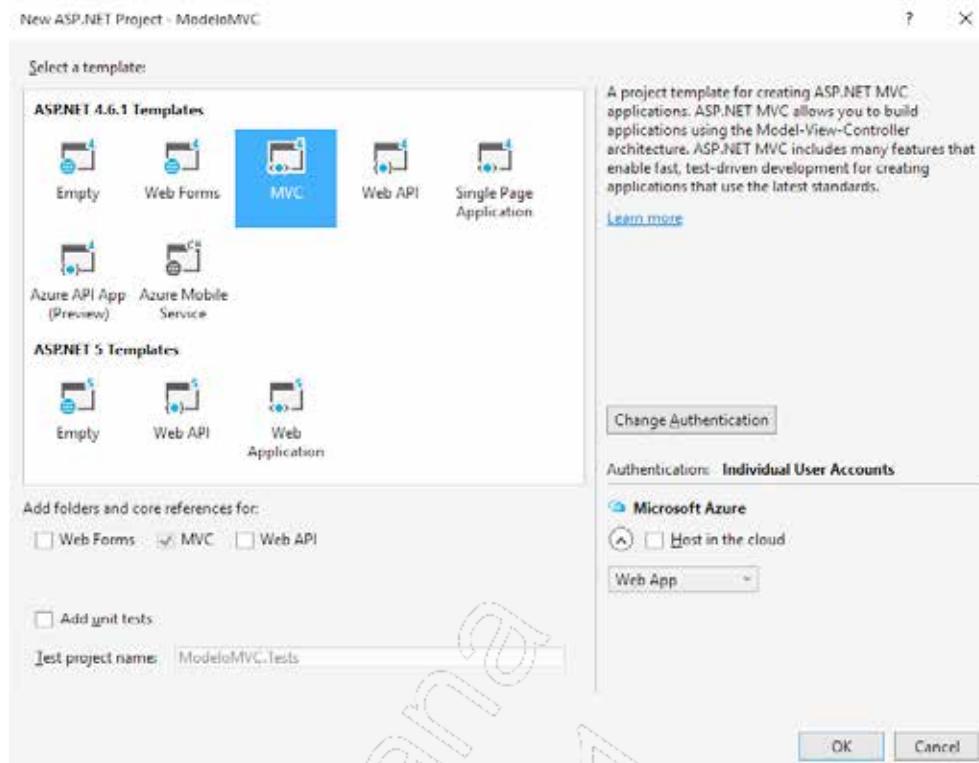
O MVC utiliza as Web Pages como renderizador de páginas e pode usar todas as facilidades do mecanismo Razor, incluindo Helpers. Além disso, assistentes especiais do Visual Studio criam páginas às tarefas tradicionais de acesso a dados.

4.3. ASP.NET MVC

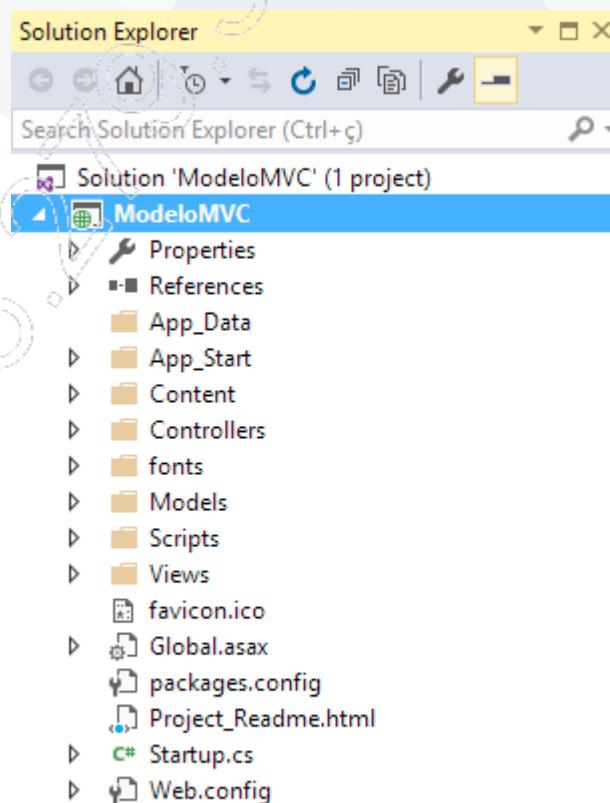
No padrão MVC, o programador deve ter controle total sobre o código HTML gerado. Não só sobre o código mas sobre as requisições, cabeçalhos HTTP e verbos (PUT, GET, DELETE, POST). É importante, para quem trabalha com MVC e ASP.NET, conhecer como funciona internamente a Internet.

Visual Studio 2015 - ASP.NET com C# Fundamentos

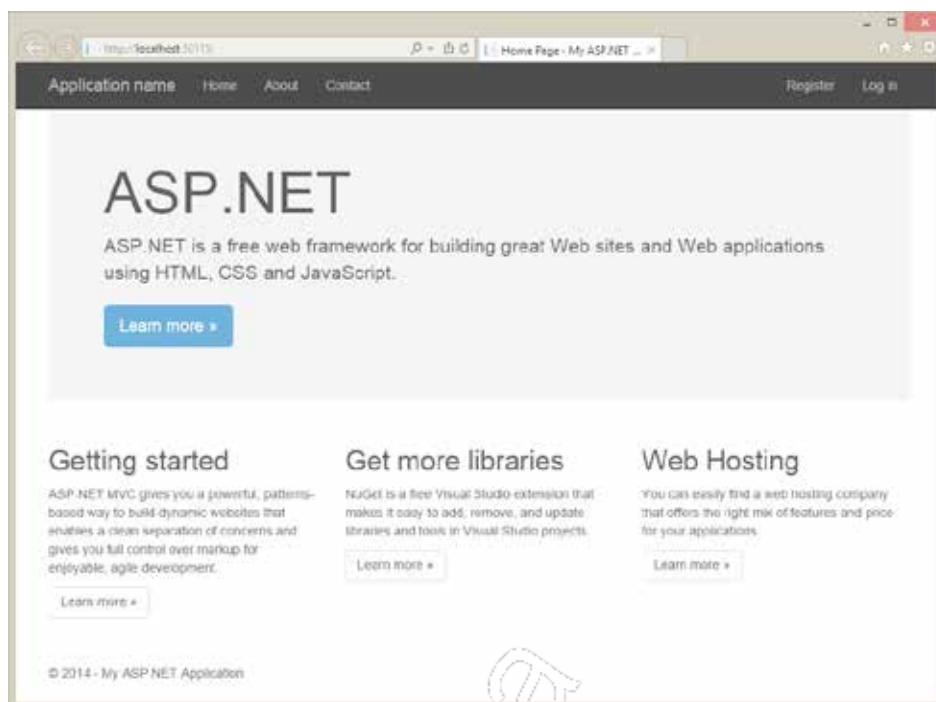
Para iniciar um novo projeto MVC, crie um novo projeto Web e escolha o modelo **MVC**:



O modelo criado contém as pastas, arquivos, referências e componentes necessários para rodar a pequena aplicação com um controle de login de usuários.



Ao executar a aplicação, o resultado é o mesmo do modelo dos Web Forms:



4.4. Estrutura dos arquivos MVC

A implementação MVC do ASP.NET define, a princípio, as seguintes distribuições de tipos de arquivos:

- **View:** São páginas com código HTML e scripts de servidor (**.cshtml**);
- **Controller:** São classes derivadas de **System.Web.Mvc.Controller** que controlam a lógica da aplicação;
- **Model:** São classes usadas para exibir ou obter dados.

Algumas importantes pastas deste modelo são as seguintes:

Pasta	Descrição
\Content	Pasta com arquivos de folhas de estilo (*.css).
\Controllers	Pasta com as classes do tipo Controller, tipo essencial do modelo MVC.
\Models	Pasta com as classes de modelos, tipo essencial do modelo MVC.
\Scripts	Pasta com diversos arquivos e bibliotecas JavaScript, usados por diferentes componentes.
\Views	Pasta com arquivos do tipo Razor Web Pages (*.cshtml), responsáveis pelas Views, tipo essencial do modelo MVC.

A implementação padrão do MVC utiliza muitos componentes, conceitos e tecnologias: Entity Framework, OAuth, Razor, Web Pages, OWIN, Data Annotations, Bundle, Minification, ASP.NET Identity, Route, Rest, SignalR, AJAX, jQuery, Bootstrap, LINQ, XML, JSON, além, é claro, de implementações de Models, Views e Controllers. Qualquer componente escolhido pela equipe da Microsoft para fazer parte do projeto pode ser substituído ou personalizado.

Para entender como funciona todo o projeto, o melhor é compreender o caminho que uma requisição percorre desde a solicitação pelo usuário até o momento em que a resposta é enviada.

4.5. Routes

O processo inicial que ocorre quando um usuário digita um endereço no browser onde está instalada uma aplicação MVC é o seguinte:

1. O navegador envia a solicitação para o endereço, por exemplo, <http://www.empresacom.br/WebApp>;
2. O IIS recebe a solicitação e passa para o aplicativo MVC;
3. O aplicativo determina qual método de qual Controller deve ser instanciado, qual método deve ser executado. O Controller é instanciado e o método definido é executado.

O passo 3 é completamente diferente do que acontece com um projeto do tipo Web Forms (usando o modelo padrão), em que o endereço é uma página. O MVC utiliza o conceito de **Routes** (rotas) para relacionar uma URL com um método de uma classe. Por exemplo:

	A URL...	...chama o método
1	Listagem">http://webApp/Produtos>Listagem	ProdutosController.Listagem()
2	http://webApp/Produtos/Pesquisar/1	ProdutosController.Pesquisar(int id)
3	http://webApp/Produtos/Incluir	ProdutosController.Incluir()
4	http://webApp/Produtos	ProdutosController.Index()
5	http://webApp/Contato	ContatoController.Index()
6	http://webApp/Contato/Index	ContatoController.Index()

Repare em algumas convenções nessa lista:

- A classe relacionada à URL sempre tem o sufixo **Controller**;
- Quando não é informado um método (itens 4 e 5), o método padrão é o **Index()**;
- Na URL, os segmentos depois do método são parâmetros (item 2).

A definição das rotas se encontra no arquivo **\App_Start\RouteConfig.cs**. A classe **RouteConfig** contém o método **RegisterRoutes**, responsável por criar as URLs que serão mapeadas para os métodos apropriados. O método **RegisterRoutes** é chamado quando a aplicação é executada pela primeira vez ou quando uma nova versão é publicada. O método de evento **Application_Start** (arquivo **Global.asax**) é responsável pela chamada.

- **\Global.asax**

```
public class MvcApplication : System.Web.HttpApplication
{
    protected void Application_Start()
    {
        AreaRegistration.RegisterAllAreas();

        FilterConfig.RegisterGlobalFilters(
            GlobalFilters.Filters);

        RouteConfig.RegisterRoutes(RouteTable.Routes);

        BundleConfig.RegisterBundles(BundleTable.Bundles);
    }
}
```

- \App_Start\RouteConfig.cs

```
public class RouteConfig
{
    public static void RegisterRoutes(
        RouteCollection routes)
    {
        routes.IgnoreRoute("{resource}.axd/{*pathInfo}");

        routes.MapRoute(
            name: "Default",
            url: "{controller}/{action}/{id}",
            defaults: new {
                controller = "Home",
                action = "Index",
                id = UrlParameter.Optional
            });
    }
}
```

O método **MapRoute** espera os seguintes parâmetros:

```
MapRoute(name, url, defaults)
```

Em que:

- **name**: Pode ser qualquer nome. Serve apenas para identificar;
- **url**: Uma string com segmentos nomeados que serão substituídos em tempo de execução. Esses segmentos devem estar entre colchetes. Por exemplo: **{dia}/{mes}/{ano}**;
- **defaults**: Um objeto (geralmente uma classe anônima) com propriedades iguais aos segmentos nomeados na URL. Por exemplo: **new {dia=1, mes=2, ano=3}**.

No MVC, alguns nomes de segmentos são especiais: **{controller}** e **{action}**. Quando o mecanismo do MVC encontra uma URL que se encaixa no padrão **{controller}/{action}/{id}**, ele procura automaticamente uma classe derivada da classe **System.Web.Mvc.Controller** que tenha o sufixo **Controller**.

Se a classe é encontrada, o MVC procura um método com o mesmo nome definido em **{action}**. O último parâmetro é opcional. Existe uma classe, chamada **UrlParameter**, que contém um único campo read-only estático chamado **Optional** e que serve para definir se um segmento de uma rota é opcional.

```
routes.MapRoute(  
    name: "Default",  
    url: "{controller}/{action}/{id}",  
    defaults: new {  
        controller = "Home",  
        action = "Index",  
  
        id = UrlParameter.Optional  
    }  
)
```

Se o último parâmetro foi informado, será passado como parâmetro para o método. Por exemplo, nesta URL:

```
http://webApp/Produtos/Pesquisar/2
```

O item 2 do exemplo anterior será passado como parâmetro ao método **Pesquisar** da classe **ProdutosController**:

```
class ProdutosController:Controller  
{  
    public string Pesquisar(int id)  
    {  
        Return "pesquisando:" + id;  
    }  
}
```

Visual Studio 2015 - ASP.NET com C# Fundamentos

Outra característica é que o **{action}** tem um valor padrão, que é **Index**, portanto, se ele não for passado, será como se tivesse sido passado o parâmetro **Index**. Veja o exemplo:

```
http://webApp/Produtos
```

A URL anterior chama o método adiante:

```
class ProdutosController:Controller
{
    public string Index()
    {
        Return "método index";
    }
}
```

Agora que já foi identificado qual Controller será chamado, é possível continuar a trajetória da requisição do usuário:

1. O navegador envia a solicitação para o endereço, por exemplo, <http://www.empresacom.br/WebApp>;
2. O IIS recebe a solicitação e passa para o aplicativo MVC;
3. O aplicativo determina qual método de qual Controller deve ser instanciado, qual método deve ser executado. O Controller é instanciado e o método definido é executado;
4. O Controller recebe a requisição, processa as informações, consulta e/ou altera dados (Models) e define qual View será chamada.

Por exemplo, o endereço <http://webApp/Home/About> faz o método **About** da classe **HomeController** ser executado:

- **/Controllers/HomeController.cs**

```
public class HomeController : Controller
{
    public ActionResult Index()
    {
        return View();
    }

    public ActionResult About()
    {
        ViewBag.Message = "Your application description page.";
        return View();
    }

    public ActionResult Contact()
    {
        ViewBag.Message = "Your contact page.";
        return View();
    }
}
```

4.6. Controller

As classes do tipo Controller do modelo MVC são derivadas da classe abstrata **System.Web.Mvc.Controller**. Essa classe fornece a infraestrutura e os métodos necessários para responder a solicitações HTTP feitas a uma aplicação Web MVC.

```
public class HomeController : Controller
{
    ...
}
```

O retorno dos métodos é uma chamada ao método **View()**, que está definido na classe base **Controller**, sendo, portanto, herdado pela classe **HomeController**. Esse retorno é do tipo **ViewResult**, um dos tipos derivados de **ActionResult**. Essa classe (**ActionResult**) representa o resultado de um método disparado por uma URL. Esse tipo de método é chamado de **método de ação**.

```
public class HomeController : Controller
{
    ...
    public ActionResult About()
    {
        ...
        return View();
    }
}
```

O retorno deve ser do tipo **ActionResult**.

Esta propriedade é do tipo **ViewResult**, classe derivada de **ActionResult**.

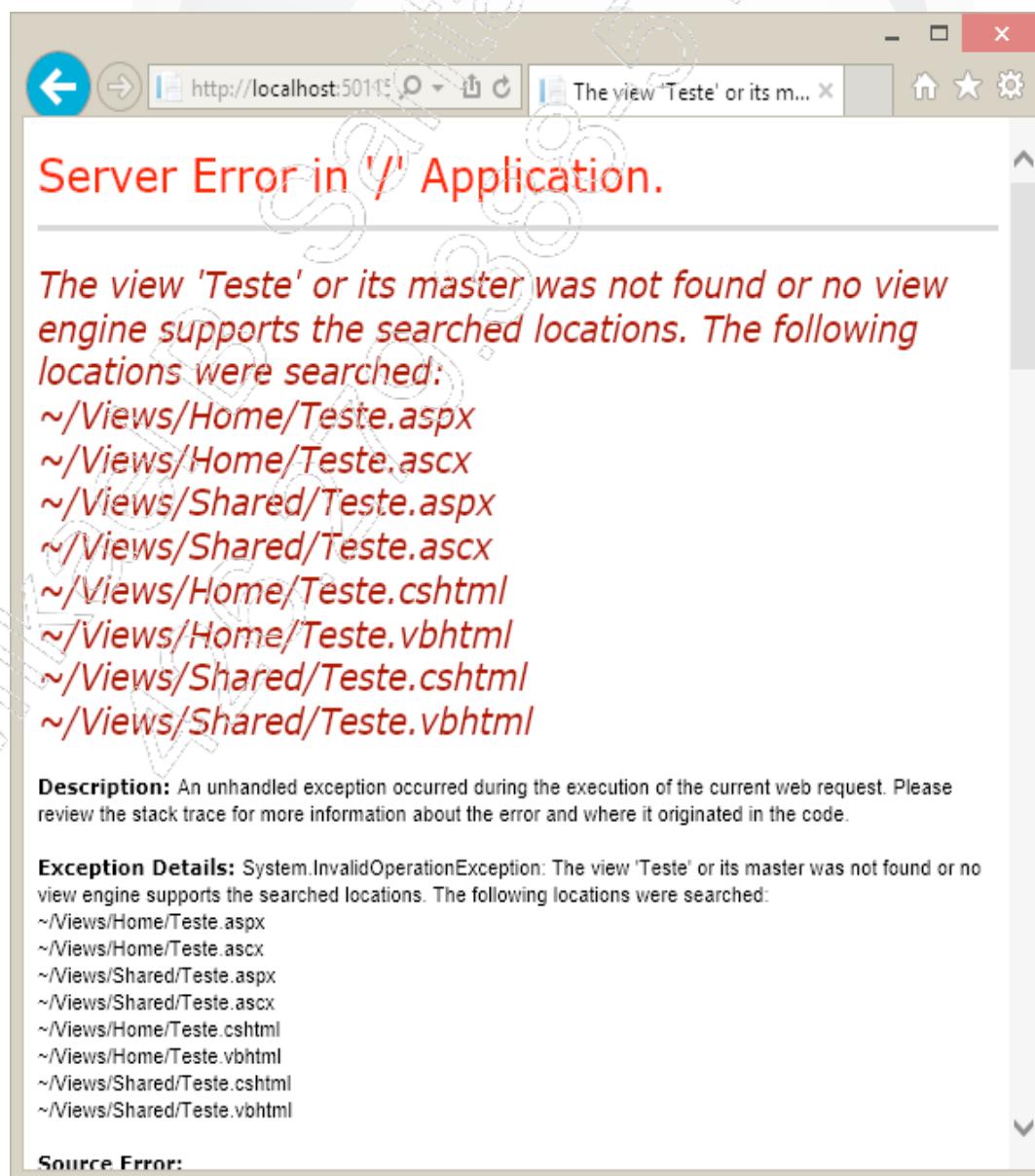
A classe **ActionResult** é muito importante no ASP.NET MVC, pois é a classe base para muitas outras classes que retornam dados:

- **EmptyResult**: Uma resposta vazia, sem conteúdo, é retornada;
- **PartialViewResult**: Retorna uma renderização parcial de uma View;
- **ViewResult**: O retorno mais comum. Uma View é enviada na resposta da solicitação HTTP;
- **RedirectResult**: Realiza um redirecionamento padrão do protocolo HTTP;
- **JsonResult**: Retorna dados no formato JSON;
- **JavascriptResult**: Retorna dados no formato JavaScript;

- **ContentResult:** Escreve o conteúdo diretamente no retorno da solicitação sem que o mecanismo MVC procure por uma View;
- **FileContentResult:** Retorna um arquivo para a solicitação HTTP;
- **FileStreamResult:** Retorna um fluxo de dados (stream) para a solicitação (cliente);
- **FilePathResult:** Retorna um arquivo para a solicitação recebida.

4.7. Views

O retorno tradicional é um **ViewResult** e o mecanismo do MVC procura um arquivo com o nome da ação (se nenhum outro for informado) e diversos outros arquivos com combinações de nomes. Se nenhum nome for encontrado, um erro é disparado:



O local padrão das **Views** é a pasta **/Views**. Dentro dessa pasta, o mecanismo do MVC procura arquivos contendo o nome do Controller e o nome do método. São procurados arquivos de code-behind (**.aspx**), user controls (**.ascx**) e Web pages (**.cshtml** e **.vbhtml**). Além da pasta com o nome do Controller, são procurados arquivos na pasta **View/Shared**, que é uma pasta com arquivos compartilhados.

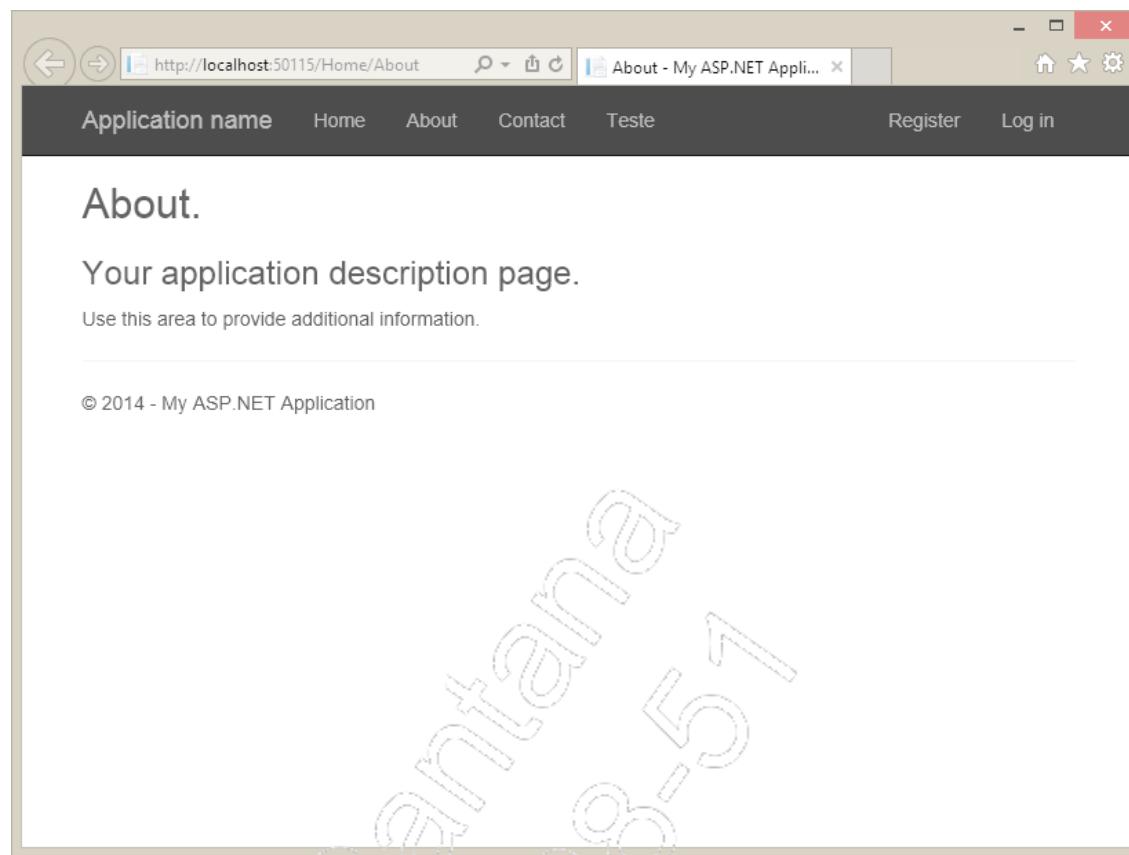
Agora que o mecanismo de definir Views é conhecido, é possível atualizar a sequência de eventos de uma requisição no MVC:

1. O navegador envia a solicitação para o endereço, por exemplo, **http://www.empresacom.br/WebApp**;
2. O IIS recebe a solicitação e passa para o aplicativo MVC;
3. O aplicativo determina qual método de qual Controller deve ser instanciado, qual método deve ser executado. O Controller é instanciado e o método definido é executado;
4. O Controller recebe a requisição, processa as informações, consulta e/ou altera dados (Models) e define qual View será chamada;
5. A View é encontrada (geralmente na pasta **/Views**) e é renderizada, usando o mecanismo **Razor**.

O conteúdo a seguir é da página **/Views/Home/About.cshtml**:

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

Ao ver a página **http://servidor/home/about** sendo exibida, percebe-se que ela tem muito mais conteúdo do que o HTML apresentado na listagem anterior. Isso porque esta View usa o **Razor View Engine** com um arquivo de layout.



Os arquivos envolvidos na página final que será gerada são os seguintes:

- **_ViewStart.cshtml**: Este arquivo, que reside na raiz do site, define um arquivo de layout que é usado como padrão;
- **\Views\Shared_Layout.cshtml**: Este é o arquivo definido pelo arquivo `_ViewStart`;
- **\Views_Login_Partial.cshtml**: Parte de uma página (área de login e mensagens de usuário);
- **\Views\Errors.cshtml**: Página de erro compartilhada;
- **\Views\Lockout.cshtml**: Mensagem exibida quando a conta é bloqueada.

4.7.1. ViewBag

As páginas **Razor** recebem, de diversas formas, informações do Controller. Uma dessas maneiras é usar a propriedade **ViewBag**. Esta classe fornece propriedades dinâmicas (são resolvidas em tempo de execução) que podem armazenar qualquer tipo de informação.

```
public class HomeController : Controller
{
    public ActionResult MeuMetodo
    {
        ViewBag.Mensagem = "Bem Vindo";
        ViewBag.ValorDesconto = 0.00;

        return View();
    }
}
```

- Para recuperar os dados:

```
<h2>MeuMetodo</h2>

<p>@ViewBag.Mensagem</p>
<p>Desconto de hoje:@ViewBag.ValorDesconto</p>
```

Os métodos gerados automaticamente usam esta propriedade (entre outras formas) para passar informações do Controller para as Views. Por exemplo, o método **About**, mostrado adiante:

```
public ActionResult About()
{
    ViewBag.Message = "Your application description page.";
    return View();
}

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

4.8. HtmlHelper

A classe **System.Web.Mvc.HtmlHelper** fornece a base para a renderização das páginas HTML. A maioria dos métodos usados são extensões adaptadas a esta classe de acordo com as configurações do aplicativo. Vejamos, nos próximos subtópicos, alguns membros importantes desta classe ou estendidos por outras classes.

4.8.1. ActionLink

Cria uma âncora HTML com destino em uma ação (não em uma View). As duas formas mais frequentemente usadas são as seguintes:

- `@html.ActionLink("texto", "acao")`
- `@html.ActionLink("texto", "acao", "controller")`

Observe o comando a seguir:

```
@Html.ActionLink("Entre em Contato", "contato", "home");
```

O comando anterior cria o seguinte código HTML:

```
<a href="/home/contato">Entre em Contato</a>
```

Se não for informado o nome do Controller, será usado o nome da pasta atual, como o comando adjacente:

```
@Html.ActionLink("Entre em Contato", "contato");
```

Esse comando cria o mesmo código HTML do exemplo anterior, em que foi passado o nome do Controller (no caso, **home**).

4.8.2. Partial

O método **Partial** renderiza um arquivo **.cshtml** e retorna uma string que pode ser armazenada em uma variável ou incluída na página. É usado para incluir Telas de Logins/Encerrar Sessão, Galeria de Fotos, Menus de Paginação dentro da página principal. Veja um exemplo:

```
<div>  
@Html.Partial("_NoticiaDoDia")  
</div>
```

O conteúdo do arquivo **_NoticiaDoDia.cshtml** informado será inserido na posição do comando (no caso anterior, dentro da **div**).

4.8.3. BeginForm

Renderiza um formulário HTML. Por padrão, usa o método POST e a mesma View onde está declarado.

```
@using (Html.BeginForm())  
{  
    <input id="nome" name="nome" type="text" value="" />  
    <input type="submit" value="Enviar" />  
}
```

Ao ser renderizado, o método **BeginForm** cria a tag **HTML form** e as chaves servem para determinar onde termina o formulário.

```
<form action="/Exemplos/HtmlForm" method="post">  
  
    <input id="nome" name="nome" type="text" value="" />  
  
    <input type="submit" value="Enviar" />  
  
</form>
```

Cada controle de formulário, como caixas de texto, seletores e botões, pode ser escrito por meio dos métodos que facilitam a criação de componentes HTML. A listagem adiante mostra o uso dos mais comuns:

- **TextBox**: Cria uma tag **HTML input** para a renderização de uma caixa de texto para entrada de dados;
- **Label**: Cria uma tag **HTML Label**, geralmente usada em conjunto com a caixa de texto, para a criação de legenda;
- **Password**: Cria uma caixa de texto (input) para entrada de senhas;
- **CheckBox**: Cria uma caixa de checagem para entrada de dados booleanos.

O exemplo a seguir cria um pequeno formulário, com os elementos organizados na sequência em que geralmente são utilizados. O código HTML relativo à estrutura do documento foi omitido para maior clareza:

```
@using (Html.BeginForm())
{
    @Html.Label("nome", "Digite o Nome:");
    @Html.TextBox("nome");

    @Html.Label("senha", "Senha:")
    @Html.Password("senha")

    @Html.Label("lembrar", "Lembrar Nome e Senha")
    @Html.CheckBox("lembrar", true);

    <input type="submit" value="Enviar" />
}
```

Visual Studio 2015 - ASP.NET com C# Fundamentos

Ao ser renderizado, esse código gera a seguinte marcação HTML:

```
<form action="/Exemplos/HtmlForm" method="post">

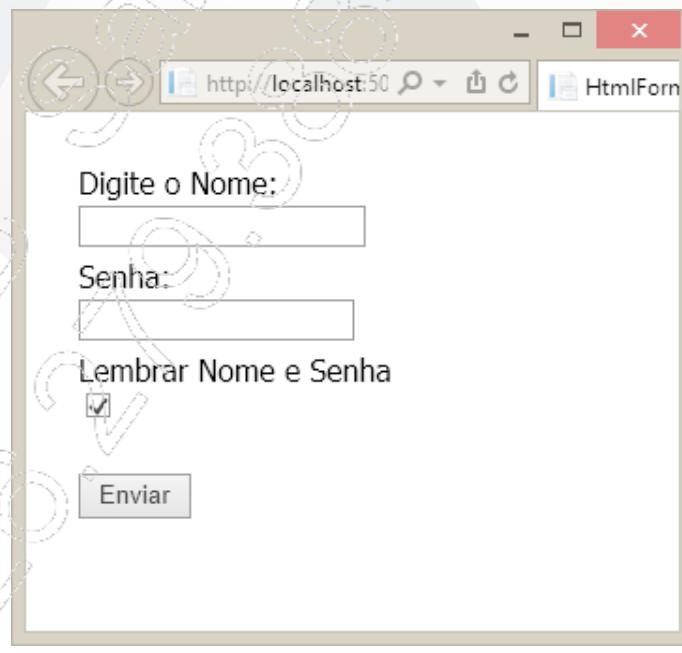
<label for="nome">Digite o Nome:</label>
<input id="nome" name="nome" type="text" value="" />

<label for="senha">Senha:</label>
<input id="senha" name="senha" type="password" />

<label for="lembrar">Lembrar Nome e Senha</label>
<input checked="checked" id="lembrar" name="lembrar" type="checkbox"
value="true" />

<input name="lembrar" type="hidden" value="false" />
<input type="submit" value="Enviar" />

</form>
```



4.9. Model e Views tipadas

Até agora, as informações da classe **Controller** para a classe **View** foram passadas por meio da propriedade **ViewBag**, que utiliza o dicionário interno de dados do ASP.NET. Para situações simples, esta estrutura funciona, mas a necessidade real das aplicações sempre envolve estruturas complexas de dados que devem ser organizadas, manipuladas, compartilhadas e entendidas por todos os envolvidos em um projeto. É neste ponto que a modelagem de dados entra em ação e é parte integrante da estrutura do MVC.

Os modelos de dados podem vir de uma biblioteca externa ou podem ser classes definidas dentro da aplicação. Se formos criar as classes dentro da aplicação Web, a pasta padrão é a pasta **Models**. Considere a seguinte classe dentro da pasta **Models**:

- **MinhaApp/Models/Produto.cs**

```
public class Produto
{
    public int ProdutoId { get; set; }

    public string Nome { get; set; }

    public decimal Preco { get; set; }
}
```

Considere que um método da classe **ProdutoController** retorne uma instância desta classe:

- **MinhaApp/Controllers/ProdutoController.cs**

```
public class ProdutoController : Controller
{
    public ActionResult PromocaoDoDia()
    {
        var p = new Produto()
        {
            Nome = "Cadeira de Madeira",
            Preco = 150,
            ProdutoId = 32
        };
        return View(p);
    }
}
```

Repare na instância da classe **Produto** sendo passada para a View.

Na View, usa-se a declaração **Model** para informar o tipo que está sendo passado do Controller. A View contém uma propriedade herdada chamada **Model** que é preenchida com a instância passada pelo Controller.

A classe **HtmlHelper** contém os métodos **DisplayNameFor** e **DisplayFor** que exibem o nome de um campo e seu valor, respectivamente.

O argumento passado para ambos os métodos é uma **Expressão Lambda** representando a propriedade da classe definida em **@model**.

Veja o exemplo:

- **MinhaApp/Views/PromocaoDoDia.cshtml**

```
@model MinhaApp.Models.Produto
```

Repare na declaração do tipo
recebido por esta View.

```
<h4>Promoção do Dia</h4>
@Html.DisplayNameFor(model => model.Nome) :
@Html.DisplayFor(model => model.Nome)
<br />

@Html.DisplayNameFor(model => model.Preco) :
@Html.DisplayFor(model => model.Preco)
<br />
```



No exemplo, foi usado o método **DisplayFor** e passada como parâmetro uma expressão para definir o campo:

```
@Html.DisplayFor(model => model.Preco)
```

É possível, também, usar a propriedade **Model**, que é automaticamente preenchida com o objeto recebido pela View. O resultado é o mesmo.

```
@Model.Preco
```

4.10. GET e POST

As Views podem receber e exibir ou enviar dados. Na grande maioria das vezes, é utilizado o método GET de uma requisição HTTP para exibir dados ou enviar parâmetros de consulta e o método POST para enviar dados, principalmente quando a finalidade do envio é a inclusão ou alteração de informações. Quando é utilizado o conceito de REST, os outros verbos HTTP são utilizados, como DEL para excluir e PUT para alterar.

Por exemplo, ao clicar no botão **Gravar** do formulário a seguir, uma requisição com POST é feita ao servidor e o método **NovoProduto** é disparado. Este método é o mesmo usado para visualizar o formulário com GET.

```
<form action="/Produto/NovoProduto" method="post">

    <label>Nome</label>
    <input name="Nome" type="text" value="" />

    <label>Preço</label>
    <input name="Preco" type="text" value="" />

    <input type="submit" value="Gravar" />

</form>
```

Para processar o POST, é necessário usar um atributo chamado **HttpPost**.

```
// GET (Implícito)
public ActionResult NovoProduto()
{
    return View();
}

[HttpPost]
public ActionResult NovoProduto(Produto p)
{
    return View();
}
```

O segundo método é chamado quando um formulário é enviado, porque está marcado para ser executado apenas quando a requisição enviar o método POST. Isso só ocorre quando é clicado um botão do tipo **Submit**, que estiver dentro de um **form**, com o método de ação (**action**) definido para POST, como é o caso neste exemplo:

```
<form action="/Produto/NovoProduto" method="post">  
...  
<input type="submit" value="Gravar" />  
</form>
```

A instância de **Produto** que é recebida como um parâmetro pelo método **NovoProduto** na variável **p** é automaticamente criada e preenchida com os valores dos controles inseridos no formulário.

Neste exemplo, a propriedade **Produtoid** conterá o valor **0**, porque não tem um controle chamado **Produtoid**. Este campo será necessário para alterar ou excluir um produto.

```
<form ...  
...  
<input name="Nome" type="text" value="" />  
<input name="Preco" type="text" value="" />  
...  
</form>
```

```
[HttpPost]  
public ActionResult NovoProduto(Produto p)  
{  
    // Código para Incluir ...  
  
    return View();  
}
```

A classe **HtmlHelper** disponibiliza uma série de métodos que podem ser usados para auxiliar a criação dos elementos do formulário:

- **Html.BeginForm**: Inicia um formulário;
- **@Html.AntiForgeryToken**: Cria um campo oculto para validar o request e evitar ataques;
- **@Html.ValidationSummary**: Cria uma tag `` para exibir mensagens de validação;
- **@Html.LabelFor**: Cria uma legenda para um campo;
- **@Html.TextBoxFor**: Cria uma caixa de texto;
- **@Html.PasswordFor**: Cria uma caixa de texto do tipo senha;
- **@Html.ValidationMessageFor**: Cria um span para exibir mensagens de validação;
- **@Html.CheckBoxFor**: Cria uma caixa do tipo **CheckBox**;
- **@Html.ActionLink**: Cria links para métodos definidos nas classes do tipo **Controller**;
- **@Html.Hidden**: Cria um campo de formulário oculto;
- **@Html.DropDownListFor**: Cria um **DropDownList**.

Esses métodos não são de uso obrigatório e servem apenas para facilitar a escrita do código HTML e para evitar a revisão do código quando algum modelo mudar. Ao contrário dos WebControls, eles não são objetos compilados na memória com eventos e gerenciamento de estado por meio de ViewState. São apenas métodos cuja saída é uma string que será colocada na página ou armazenada em uma variável.

O exemplo a seguir foi extraído do arquivo **Login.cshtml**, gerado pelo Visual Studio em um novo projeto MVC com a autenticação padrão (por usuários), e mostra o método para criar um TextBox e o HTML gerado:

- Comando

```
@Html.TextBoxFor(m => m.Email, new { @class = "form-control" })
```

- HTML

```
<input name="Email" class="form-control" id="Email" type="text" value="" data-val-required="The Email field is required." data-val-email="The Email field is not a valid e-mail address." data-val="true">
```

A sintaxe para este e muitos outros métodos disponíveis é a seguinte:

```
@Html.TextBoxFor( expressão , objeto )
```

Em que:

- **Expressão** é uma expressão lambda e representa uma função que recebe um objeto do tipo definido na View e retorna uma propriedade ou campo;
- **Objeto** é uma propriedade ou campo de uma instância de uma classe do mesmo tipo informado na diretiva **@model**.

Portanto, observe a declaração a seguir:

```
@Html.TextBoxFor(m => m.Email, new { @class = "form-control" })
```

Na declaração anterior, a expressão é a seguinte:

```
m=>m.Email
```

E o objeto que contém as propriedades para serem transformadas em atributos na tag HTML é o seguinte:

```
new { @class = "form-control" }
```

O uso do arroba (@) na palavra **class** é necessário porque esta é uma palavra reservada da linguagem. Outros atributos não precisam deste prefixo:

```
new { @class = "form-control", maxLength = "40" }
```

4.11. Ciclo de requisição e resposta no MVC

Vejamos, a seguir, o ciclo completo da requisição e resposta no MVC:

1. O navegador envia a solicitação para o endereço, por exemplo, <http://www.empresa.com.br/WebApp>;
2. O IIS recebe a solicitação e passa para o aplicativo MVC;
3. O aplicativo determina qual método de qual Controller deve ser executado e chama este método;
4. O Controller recebe a requisição, processa as informações, consulta e/ou altera dados (Models) e define qual View será chamada;
5. A View é encontrada (geralmente na pasta `/Views`) e é renderizada, usando o mecanismo Razor;
6. A View apresenta os dados para o usuário, podendo usar um tipo passado pelo Controller para criar a tela. Esse tipo é chamado de **Modelo**;
7. A View exibe o modelo de dados e pode, também, enviar dados preenchidos pelo usuário para o Controller por meio de protocolos, como POST;
8. O Controller recebe objetos do modelo enviados pela View para processamento e retorna os resultados para a mesma ou outra View.

Pontos principais

Atente para os tópicos a seguir. Eles devem ser estudados com muita atenção, pois representam os pontos mais importantes do capítulo.

- MVC significa **Model-View-Controller** e é um padrão de arquitetura no desenvolvimento de interfaces de usuários;
- **Controller** é a parte do MVC responsável pela lógica da aplicação;
- **View** é a parte visível da aplicação;
- **Model** é o modelo de dados relacionado com a aplicação;
- O **ASP.NET MVC Framework** é um conjunto de bibliotecas para auxiliar no desenvolvimento de aplicações compatíveis com o modelo MVC usando o Visual Studio e a plataforma .NET;
- Cada comando no ASP.NET MVC é representado por uma URL que é mapeada para disparar métodos específicos;
- Os métodos do ASP.NET relacionados aos comandos são chamados **Ações** e são fornecidos pelas classes derivadas de Controller;
- A saída dos métodos de ação são do tipo **ActionResult**;
- **ViewBag** é uma propriedade que encapsula a propriedade **ViewData** e facilita a troca de informações entre o Controller e a View;
- A diretriz de compilação **@model** cria uma View fortemente tipada;
- **Razor** é o mecanismo usado para renderizar as páginas, por padrão, no MVC;
- A classe **HtmlHelper** fornece métodos para criar código HTML usando os modelos de dados.

MVC

Teste seus conhecimentos

4

Mikael B Santana
426.279.3857



IMPACTA
EDITORA

1. O início do processo em um projeto MVC é a chamada de uma URL por parte do usuário. Esse processo leva primeiramente até uma classe de qual tipo?

- a) Model
- b) View
- c) Controller
- d) Page
- e) CSS

2. Como se chama o mecanismo de mapear as URLs para ações a serem executadas?

- a) Debug
- b) ORM
- c) Route
- d) UML
- e) Optimization

3. Por padrão, qual é o tipo de retorno de um método de uma classe derivada de Controller?

- a) HTML
- b) String
- c) Stream
- d) ActionResult
- e) HtmlResult

4. Qual é o nome da propriedade que permite enviar informações do Controller para a View?

- a) Transfer
- b) ViewState
- c) ViewBag
- d) ViewMaster
- e) ViewHtml

5. Como se chama a classe utilizada para facilitar a escrita de código HTML e como ela é disponibilizada?

- a) HtmlHelper - @Helper
- b) Html - @Helper
- c) HtmlHelper - @Html
- d) HtmlContext - <%Html
- e) HtmlHelper - @Render

4

MVC

Mãos à obra!

Mikael B Santana
426.279.3857



IMPACTA
EDITORA

Laboratório 1

A - Criando um site pessoal com portfólio

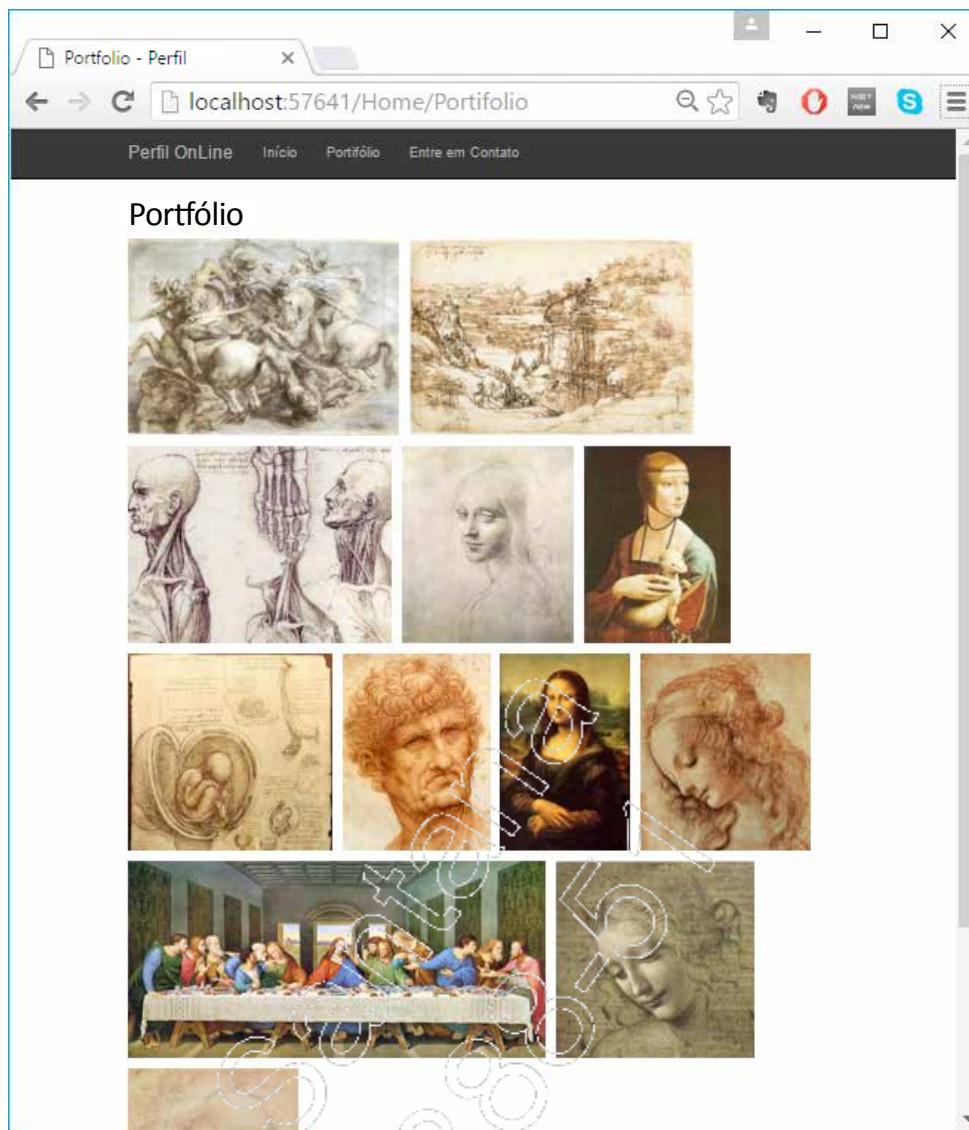
Neste laboratório, você elaborará um Web site pessoal para divulgação de um portfólio e informações de contato.

As telas são as seguintes:

- **Tela inicial**



- Portfólio



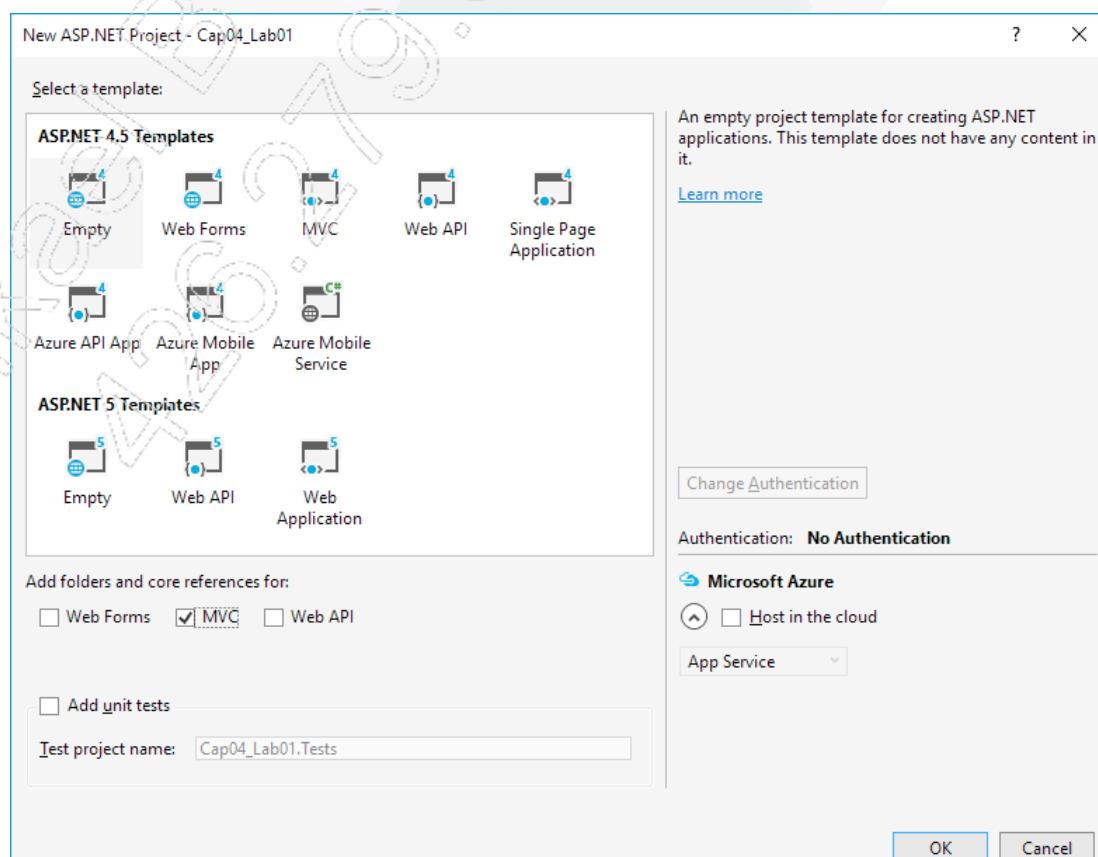
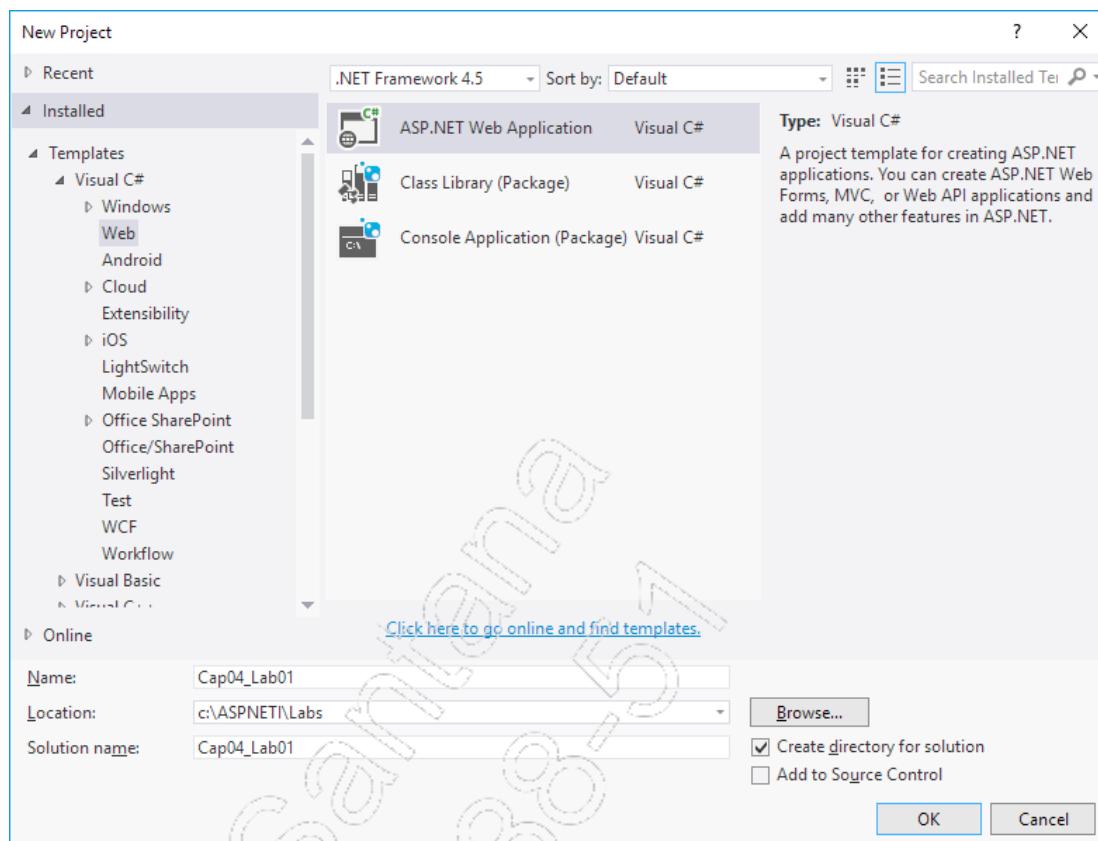
- Tela de contato

A screenshot of a web browser window titled "Contato - Perfil". The address bar shows "localhost:57641/Home/Contato". The page content is titled "Contato" and contains the text "Por favor, preencha o formulário para entrar em contato comigo:". Below this, there are three input fields labeled "Nome:", "Email:", and "Mensagem:", each with a corresponding placeholder text area. At the bottom of the form is a "Enviar" button. At the very bottom of the page, a copyright notice reads "© 2016 - Perfil OnLine".

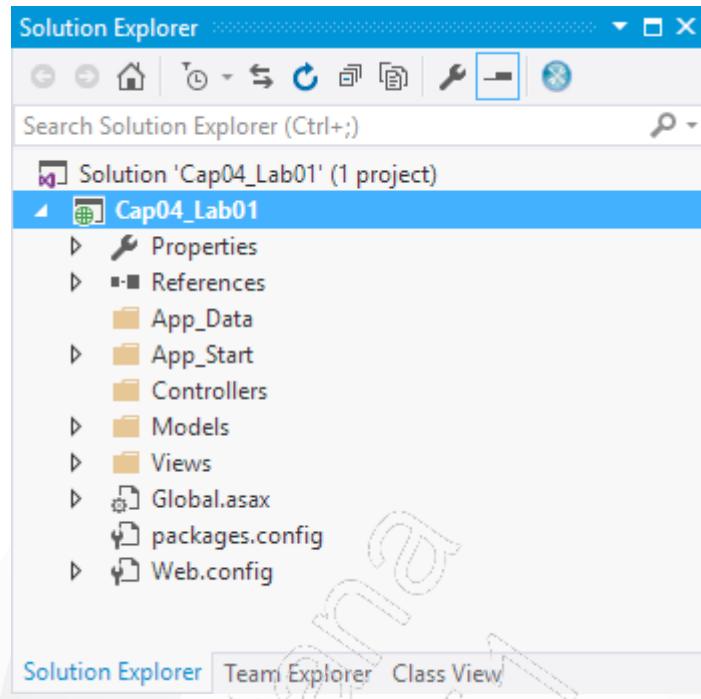
Visual Studio 2015 - ASP.NET com C# Fundamentos

Para criar esse programa, siga os passos adiante:

1. Crie um novo projeto ASP.NET (Empty), marcando a biblioteca MVC, chamado **Cap04_Lab01**;



2. A janela **Solution Explorer** exibe as três pastas básicas: **Models**, **Views** e **Controllers**. Crie uma classe chamada **ContatoViewModel** dentro da pasta **Models**:

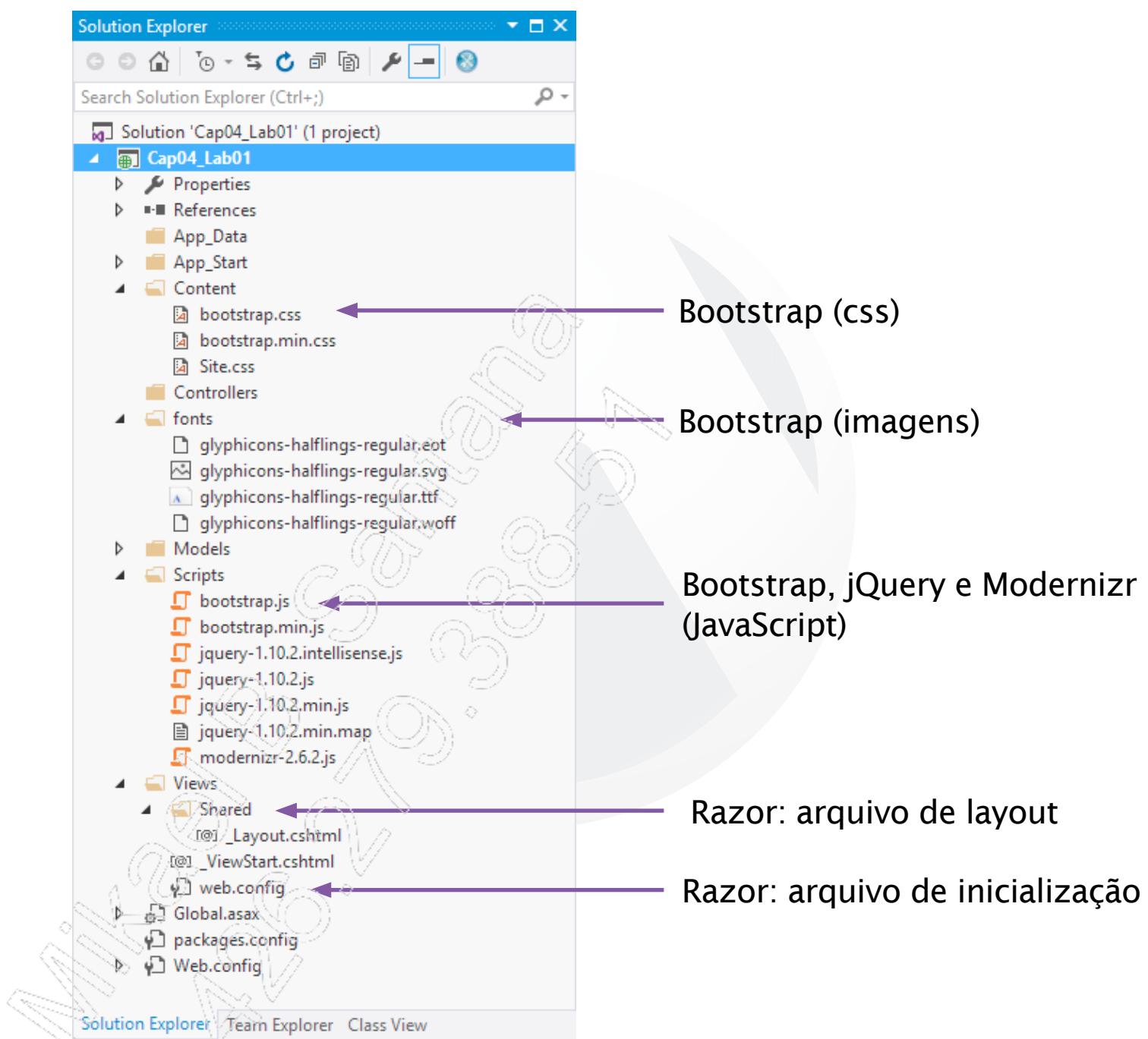


```
public class ContatoViewModel
{
    public string Nome { get; set; }

    public string Email { get; set; }

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

3. As Views serão baseadas em uma página de layout. Na pasta **View**, escolha o menu de contexto **Add View** e escolha o nome **_viewstart**. Isso vai acionar uma série de comandos para incluir a página inicial de layout e a folha de estilos usando o componente Bootstrap. Esse componente utiliza as bibliotecas JavaScript Modernizr e jQuery. Veja como ficam os arquivos do projeto:



4. Altere a View _Layout, dentro da pasta Shared:

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>@ViewBag.Title - Perfil</title>

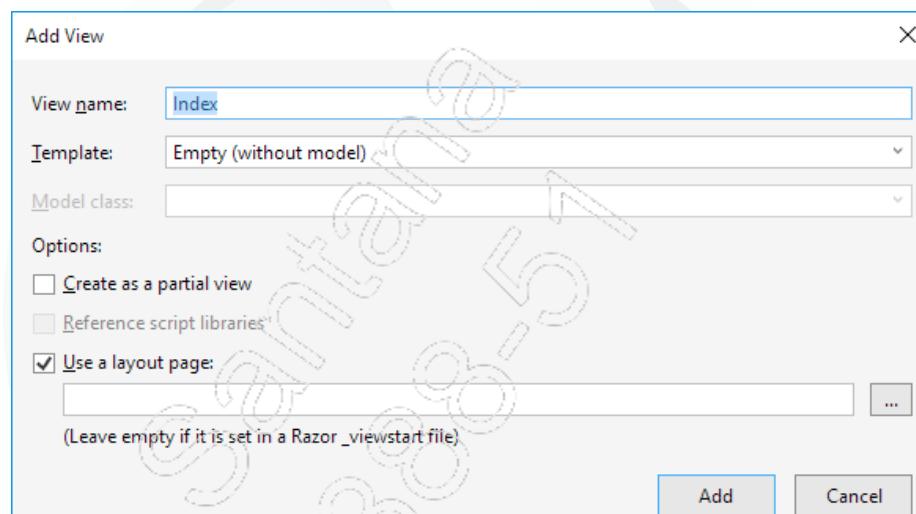
    <link href="~/Content/bootstrap.min.css" rel="stylesheet" type="text/css" />
    <link href="~/Content/Site.css" rel="stylesheet" type="text/css" />
    <script src="~/Scripts/modernizr-2.6.2.js"></script>
</head>
<body>
    <div class="navbar navbar-inverse navbar-fixed-top">
        <div class="container">
            <div class="navbar-header">
                <button type="button" class="navbar-toggle"
                    data-toggle="collapse" data-target=".navbar-collapse">
                    <span class="icon-bar"></span>
                    <span class="icon-bar"></span>
                    <span class="icon-bar"></span>
                </button>
                @Html.ActionLink("Perfil OnLine", "Index", "Home",
                    new { area = "" }, new { @class = "navbar-brand" })
            </div>
            <div class="navbar-collapse collapse">
                <ul class="nav navbar-nav">
                    <li> @Html.ActionLink("Início", "Index", "Home") </li>
                    <li> @Html.ActionLink("Portfólio", "Portfolio", "Home") </li>
                    <li> @Html.ActionLink("Entre em Contato", "Contato", "Home") </li>
                </ul>
            </div>
        </div>
        <div class="container body-content">
            @RenderBody()
            <hr />
            <footer>
                <p>&copy; @DateTime.Now.Year - Perfil OnLine</p>
            </footer>
        </div>

        <script src="~/Scripts/jquery-1.10.2.min.js"></script>
        <script src="~/Scripts/bootstrap.min.js"></script>
    </body>
</html>
```

5. Na pasta **Controllers**, insira um Controller chamado **Home**. Para isso, use o menu de contexto e a opção **Add Controller**:

```
public class HomeController : Controller
{
    // GET: Home
    public ActionResult Index()
    {
        return View();
    }
}
```

6. Com o cursor dentro do método **Index**, abra o menu de contexto, escolha **Add View** e deixe os itens no padrão:



7. Na View criada, insira o código HTML (pode adaptar o texto livremente) e visualize a página **Home/Index**:

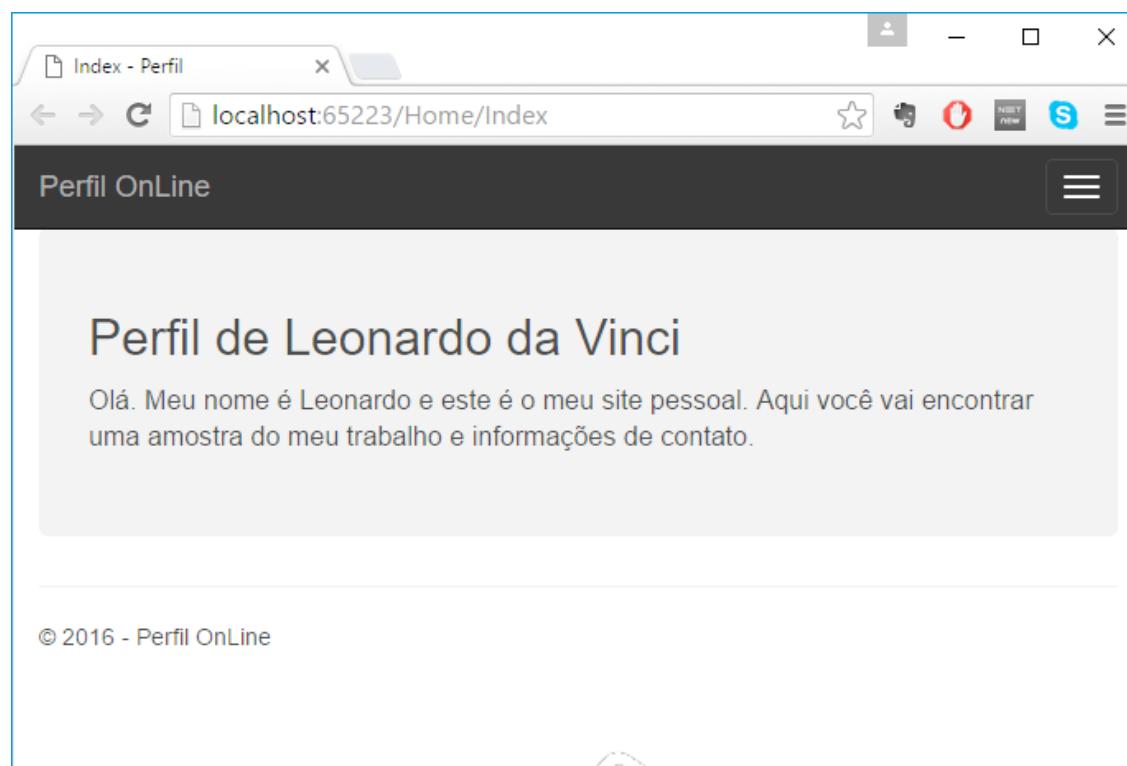
```
@{
    ViewBag.Title = "Index";
}

<div class="jumbotron">

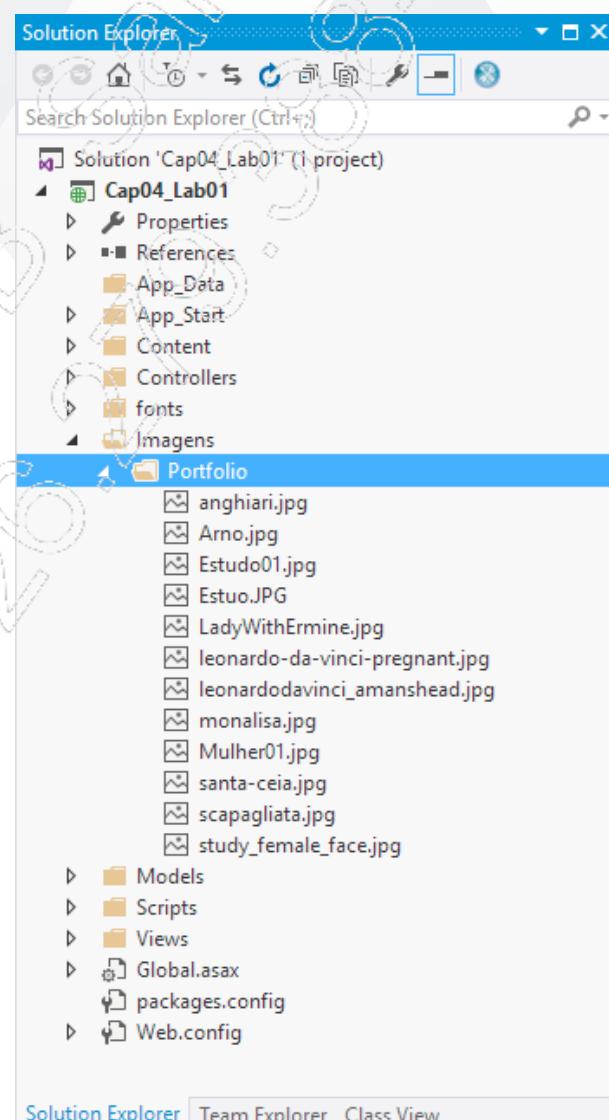
    <h2>Perfil de Leonardo da Vinci</h2>

    <p class="lead">
        Olá. Meu nome é Leonardo e este é o meu site pessoal.
        Aqui você vai encontrar uma amostra do meu trabalho e informações de contato.
    </p>

</div>
```



8. Crie uma pasta chamada **Imagens** e insira uma subpasta chamada **Portfolio**. Pode adicionar dentro da pasta **Portfolio** quaisquer arquivos de imagens que representam o trabalho ou projetos que se deseja mostrar;



Visual Studio 2015 - ASP.NET com C# Fundamentos

9. Dentro do Controller **Home**, crie o método **Portfolio**:

```
using System.IO;

namespace EmptyMVC.Controllers
{
    public class HomeController : Controller
    {
        // GET: Home
        public ActionResult Index()
        {
            return View();
        }

        public ActionResult Portfolio()
        {
            string pathVirtual = "~/imagens/Portfolio/";

            string pathFisico = Server.MapPath(pathVirtual);

            string[] arquivos = Directory.GetFiles(pathFisico);

            List<string> imageUrlList = new List<string>();

            foreach(string arquivo in arquivos)
            {
                string nomeArquivo = Path.GetFileName(arquivo);

                string imageUrl = Url.Content(pathVirtual + nomeArquivo);

                imageUrlList.Add(imageUrl);
            }

            ViewBag.ImageUrlList = imageUrlList;

            return View();
        }
    }
}
```

10. Dentro do Controller **Home** e do método **Portfolio**, escolha, no menu de contexto, **Add View** e deixe as opções padrão. Dentro da View, defina o código HTML:

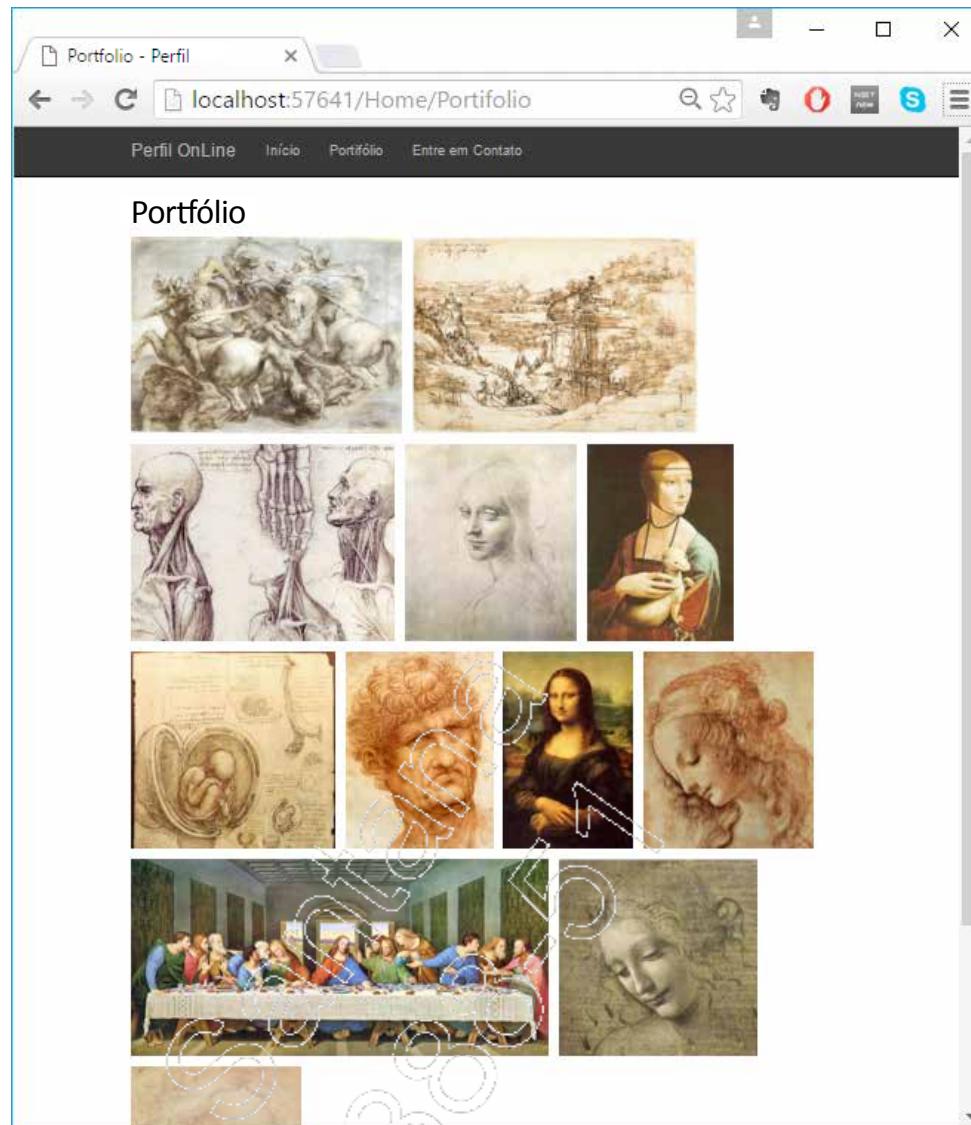
```
@{  
    ViewBag.Title = "Portfolio";  
}  
  
<h2>Portfólio</h2>  
  
@foreach (string arquivo in ViewBag.ImageUrlList)  
{  
      
}  
  
<div class="clearfix"></div>
```

11. Repare que o código HTML da View **Portfolio** define uma tag de imagem (img) com o atributo class definido para img-portfolio. Essa classe de estilo CSS deve ser definido no arquivo **site.css** dentro da pasta **Content**:

```
body {  
    padding-top: 50px;  
    padding-bottom: 20px;  
}  
  
/* Set padding to keep content from hitting the edges */  
.body-content {  
    padding-left: 15px;  
    padding-right: 15px;  
}  
  
/* Set width on the form input elements since they're 100% wide by  
default */  
input,  
select,  
textarea {  
    max-width: 280px;  
}  
  
.img-portfolio {  
    float: left;  
    height: 200px;  
    margin-right: 10px;  
    margin-bottom: 10px;  
}
```

Visual Studio 2015 - ASP.NET com C# Fundamentos

12. Visualize o portfólio:



13. No Controller **Home**, insira o método **Contato**. São dois métodos: um para exibir o formulário (GET) e outro para receber os dados (POST):

```
public class HomeController : Controller
{
    public ActionResult Index()
    {
        ...
    }

    public ActionResult Portfolio()
    {
        ...
    }
}
```

```
public ActionResult Contato()
{
    ViewBag.Concluido = false;

    return View();
}
```

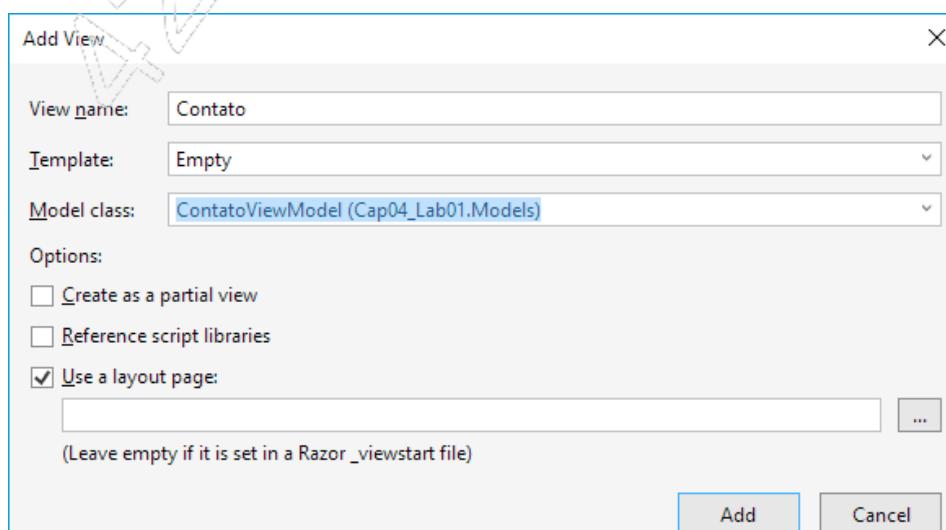
```
[HttpPost]
public ActionResult Contato(ContatoViewModel contato)
{
    string pathVirtual = "~/contatos.txt";

    string pathFisico = Server.MapPath(pathVirtual);

    using (var sw = new StreamWriter(pathFisico, true))
    {
        sw.WriteLine(contato.Nome);
        sw.WriteLine(contato.Email);
        sw.WriteLine(contato.Mensagem);
        sw.WriteLine(DateTime.Now);
        sw.WriteLine();
    }
    ViewBag.Concluido = true;
}

}
```

14. Adicione uma View, ao método **Contato**, desta vez escolhendo um modelo (**ContatoViewModel**) e altere o HTML:



Visual Studio 2015 - ASP.NET com C# Fundamentos

```
@model Cap04_Lab01.Models.ContatoViewModel

@{
    ViewBag.Title = "Contato";
}



## Contato



@if (ViewBag.Concluido)
{
    <p>Obrigado por seu cadastro. Em breve entrarei em contato.</p>
}

else
{
    <p>Por favor, preencha o formulário para entrar em contato comigo:</p>
    <hr />
    <form method="post">
        <div class="form-group">
            <label for="Nome" class="control-label">Nome:</label>
            <input type="text" id="Nome" name="Nome"
                   class="form-control" />
        </div>

        <div class="form-group">
            <label for="Email" class="control-label">Email:</label>
            <input type="text" id="Email" name="Email"
                   class="form-control" />
        </div>

        <div class="form-group">
            <label for="Mensagem" class="control-label">Mensagem:</label>
            <input type="text" id="Mensagem" name="Mensagem"
                   class="form-control" />
        </div>
        <div class="form-group">
            <input type="submit" class="btn btn-default" value="Enviar" />
        </div>
    </form>
}
```

15. Teste o cadastro (verifique se o arquivo **cadastro.txt** (na raiz do site) está registrando os dados).

Index - Perfil Contato - Perfil
localhost:57641/Home/Contato

Perfil OnLine Início Portifólio Entre em Contato

Contato

Por favor, preencha o formulário para entrar em contato comigo:

Nome:
Eduardo Grassso

Email:
eduardo.grasso@live.com

Mensagem:
Gostaria de aulas de desenho....

Enviar

© 2016 - Perfil OnLine

Contato - Perfil localhost:57641/Home/Contato

Perfil OnLine Inicio Portfólio Entre em Contato

Contato

Obrigado por seu cadastro. Em breve entrarei em contato.

© 2016 - Perfil OnLine

Sugestão para estudo:

- Criar uma página para visualizar os contatos feitos pelo site;
- Exibir uma legenda no portfólio;
- Incluir uma foto de perfil na página inicial.

5

Gerenciamento do estado da sessão

- ✓ Manipulando cookies;
- ✓ ViewState (Web Forms).

5.1. Introdução

Uma das características principais da Internet é a comunicação sem armazenamento de estado. Isso significa que, após o servidor responder a uma solicitação, a comunicação é encerrada, e qualquer nova requisição de um aplicativo cliente gera uma nova comunicação e um novo processo de resposta pelo servidor.

Esse modelo funciona muito bem, pois permite que o servidor consiga atender a um grande número de solicitações, mas, por outro lado, dificulta o armazenamento das informações entre as chamadas e de continuidade de processos.

O processo implantado no servidor para identificar e armazenar informações sobre as requisições é chamado de **gerenciamento de estado**.

5.1.1. Cookies

Um cookie é o principal recurso disponível para uma aplicação manter o estado do aplicativo entre as chamadas. Os cookies sempre são transmitidos em cada solicitação e são únicos para cada comunicação entre o browser e o servidor. Mas são limitados e não são seguros. A principal limitação é o formato de texto. Por padrão, não podem ser armazenados objetos em cookies, a não ser que sejam transformados em texto, no processo conhecido como **serialização**.

5.1.2. Session

O IIS usa um recurso interessante para gerenciar o estado. A primeira vez que um browser solicita uma página, o IIS gera um cookie especial, chamado **cookie de sessão**. Esse cookie é um código gerado dinamicamente. Ao mesmo tempo, o IIS reserva um espaço na memória do servidor e associa esse espaço ao cookie gerado. Da próxima vez que o aplicativo cliente chamar o servidor, esse espaço de memória pode ser recuperado, utilizando esse cookie como chave. Esse processo se chama **sessão**. Em uma sessão, é possível armazenar qualquer tipo de objeto, não apenas texto.

Como o servidor não tem memória infinita, existe a necessidade de limpar o espaço reservado na memória se esta não foi usada. Quando um aplicativo cliente fica inativo por mais de 20 minutos, o IIS automaticamente encerra a sessão. Esse tempo pode ser configurado.

O objeto **Session**, disponível em todos os modelos de programação ASP.NET, consiste em um dicionário, no qual encontramos pares chave/valor. Podemos estabelecer uma associação entre quaisquer objetos baseados em CLR (Common Language Runtime) e uma chave que escolhermos, armazenando-os no objeto **Session** para que estejam lá quando chegar alguma solicitação pertencente a essa sessão.

O exemplo a seguir mostra como incluir uma informação no objeto **Session**:

```
Session["data"] = DateTime.Now;
```

Para recuperar essa informação, é necessário fazer o CAST:

```
DateTime dataArmazenada = (DateTime)Session["data"];
```

Quando existe a necessidade de recuperar uma informação da sessão ativa, deve-se tomar o cuidado de verificar antes se o objeto existe, pois, entre uma requisição e outra, o tempo da sessão pode ter expirado.

```
if(Session["data"]!=null)
{
    DateTime dataArmazenada = (DateTime)Session["data"];
}
```

Para encerrar uma sessão, é utilizado o método **Abandon**. A chamada desse método exclui todas as informações gravadas na sessão atual e libera a área de memória reservada para essa sessão.

É uma boa prática sempre fornecer uma maneira de o usuário encerrar a sessão. Veja o exemplo:

```
Session.Abandon();
```

O tempo padrão de uma sessão é de 20 minutos. Para definir um tempo diferente, é possível usar a propriedade **TimeOut**. É possível, também, definir o tempo da sessão por meio do arquivo de configuração.

```
//Esta sessão será finalizada em 5 minutos
```

```
Session.Timeout = 5;
```

- Usando o arquivo **web.config**:

```
<system.web>

    <sessionState timeout="5"></sessionState>
    ...
<system.web>
```

5.2. Manipulando cookies

O ASP.NET permite criar e recuperar cookies por meio dos objetos **Response** e **Request**, respectivamente. Os cookies, em relação ao objeto **Session**, têm a vantagem de que podem ficar gravados fisicamente no computador que faz a requisição e, por isso, podem ter um tempo ilimitado de expiração, mesmo que o usuário feche o navegador. O objeto **Session**, por sua vez, existe apenas enquanto o navegador está aberto.

Veja um exemplo de como criar um cookie:

```
Response.Cookies["versao"].Value = "1.0";
```

Agora, veja como ler um cookie:

```
string versao = Request.Cookies["versao"].Value;
```

Como acontece com a **Session**, é uma boa prática verificar se o cookie existe antes de obter o valor:

```
if (Request.Cookies["versao"] != null)
{
    string versao = Request.Cookies["versao"].Value;
}
```

Os cookies gerados dessa forma são temporários e existem apenas enquanto o navegador está aberto. Para criar um cookie que fica gravado fisicamente no computador, é necessário definir a propriedade **Expires**. No exemplo a seguir, um cookie é criado, tendo o tempo de expiração em 10 dias:

```
Response.Cookies["versao"].Value = "1.0";
Response.Cookies["versao"].Expires = DateTime.Now.AddDays(10);
```

É possível, também, gerar um cookie criando uma instância da classe **HttpCookie**. O construtor dessa classe exige o fornecimento do nome do cookie:

```
HttpCookie meuCookie = new HttpCookie("nome");
meuCookie.Value = "Maria da Silva";
meuCookie.Expires = DateTime.Now.AddDays(10);
Response.Cookies.Add(meuCookie);
```

Um cookie pode ter múltiplos valores. Nesse caso, a propriedade que deve ser definida no lugar da propriedade **Value** é a propriedade **Values**, que é uma coleção do tipo chave/valor:

```
HttpCookie meuCookie = new HttpCookie("cliente");
meuCookie.Values.Add("nome", "Maria da Silva");
meuCookie.Values.Add("email", "maria@servidor.com");
meuCookie.Values.Add("clientId", "23133");

meuCookie.Expires = DateTime.Now.AddDays(10);
Response.Cookies.Add(meuCookie)
```

Para recuperar as informações, utiliza-se o objeto **Request**, sempre tomando o cuidado de verificar se o cookie existe. Como valores em um cookie apenas armazenam strings, é necessário converter para o tipo correto. No exemplo a seguir, o valor **clientId** é do tipo **System.Int32**:

```
if (Request.Cookies["cliente"] != null)
{
    string nome = Request.Cookies["cliente"].Values["nome"];
    string email = Request.Cookies["cliente"].Values["email"];
    int clientId = Convert.ToInt32(
        Request.Cookies["cliente"].Values["clientId"]);
}
```

5.3. ViewState (Web Forms)

ViewState é o mecanismo usado pelo ASP.NET apenas no modelo Web Forms, para manter o estado da página. Todo Web Control, por padrão, armazena o valor de suas propriedades em um objeto chamado **ViewState**. Quando a página é renderizada, essa informação é incluída em um controle HTML oculto. Quando ocorre um postback, ou seja, quando a página é reenviada para o servidor para processamento, os valores contidos no ViewState são obtidos e os controles são atualizados. É possível visualizar o controle **ViewState** olhando o código-fonte enviado ao navegador:

```
<!DOCTYPE html>

<html lang="en">
<head>
...
</head>
<body>
    <form method="post" action="Default" id="ctl01">
        <div class="aspNetHidden">
            <input type="hidden" name="__VIEWSTATE" id="__VIEWSTATE" value="-
DaKPTWlKk0vQiphAbOSNLNW0wlyG8Agl517kzbRee4Jkj5dijWeoTaTawt7wSNFAk2/vT2oi-
8FkX8BvPvqxKPR3pZZtYeWMibIDWRCA13T1MxDIdxP3bOON1X0Rq9jM5WwjWhxnq6sD2p05169wa-
AiIVw54UPiwFa0lsSEw1lXms8GSTjuopkLvnXdrP3titJTxaf07qhBd0y4vvMJqcw==" />
        ...
    </div>
...
</body>
</html>
```

Os dados armazenados no **ViewState** estão serializados em um formato chamado **base64**. Esse formato economiza largura de banda porque reduz o número de caracteres que trafegam pela rede a cada requisição. Como está no formulário, o **ViewState** não utiliza nem cookies e nem memória do servidor.

Para armazenar dados no **ViewState**, basta usar uma chave de acesso e definir o valor, que pode ser de qualquer tipo, porque o **ViewState** aceita objetos. Veja o exemplo a seguir:

```
ViewState[ "numero" ] = 10;  
ViewState[ "data" ] = DateTime.Now;  
ViewState[ "texto" ] = "Isto é um texto.";
```

Para recuperar os dados, é necessário fazer o cast.

```
int numero = (int)ViewState[ "numero" ];  
DateTime data = (DateTime)ViewState[ "data" ];  
string texto = (string)ViewState[ "texto" ];
```

O **ViewState** tem a limitação de funcionar por página ou por POST e entre páginas. Isso significa que o **ViewState** só pode ser mantido se houver um postback ou se uma página chamar outra diretamente. No **ViewState**, não há expiração e nem a necessidade de cookies.

Todo Web Control tem uma propriedade chamada **EnableViewState**, que permite ligar ou desligar a gravação automática das propriedades no ViewState da página. Isso é útil em determinadas circunstâncias. Por exemplo, se uma página ASP.NET exibe o resultado de um relatório para visualização por meio de uma GridView, o EnabledViewState pode ser desligado para economizar largura de banda. O único efeito colateral é que, se houver um botão ou qualquer controle que faça um postback, ao ser acionado, fará sumir da tela o GridView. Por isso o desligamento do ViewState, por via de regra, é feito apenas nas páginas que não fazem uso do postback.

Visual Studio 2015 - ASP.NET com C# Fundamentos

O quadro a seguir compara os objetos ASP.NET que fornecem controle de estado:

Session	Cookies	ViewState
Gravado no servidor.	Gravados no cliente.	Gravado na página HTML em controle oculto.
Aceita objetos.	Aceitam apenas texto.	Aceita objetos.
Expiração definida em minutos.	Expiração definida por data, sem limite de tempo.	Expiração por navegação fora do contexto da página.
Usado geralmente em intranets ou sites com poucos usuários.	Usados em portais e sites com milhões de usuários.	Usado em qualquer tipo de Web site.
Bastante seguro: Não há como ter acesso a dados da sessão estando fora dela.	Pouco seguros: Os dados ficam no cliente e são facilmente interceptados.	Razoavelmente seguro: Os dados armazenados podem ser criptografados.
Pode ser usado em qualquer modelo.	Pode ser usado em qualquer modelo.	Por padrão, apenas em Web Forms.

Pontos principais

Atente para os tópicos a seguir. Eles devem ser estudados com muita atenção, pois representam os pontos mais importantes do capítulo.

- Normalmente, as aplicações Web são distribuídas e o protocolo HTTP não possui estado, razão pela qual manter o rastreamento dos clientes é uma tarefa que deve ser realizada na maioria dos casos. O controle dessas informações é chamado **gerenciamento de estado**;
- Ao habilitarmos o estado da sessão, o ASP.NET cria um novo objeto **Session** para cada uma das novas solicitações. Esse objeto permite armazenar dados por tempo limitado;
- Os cookies são informações no formato texto armazenadas pelo navegador e transmitidas a toda requisição para um determinado domínio;
- O **ViewState** é um mecanismo que armazena informações em controles ocultos dentro da página. Esses dados são transmitidos quando é feito um POST para um servidor;
- O ASP.NET ainda oferece suporte a algumas configurações adicionais que não são disponibilizadas pelo IIS. Tais configurações consistem em valores a serem digitados diretamente no arquivo **web.config**;
- O controle **Wizard** oferece um template que realiza as tarefas de navegação básicas ao longo das páginas, de forma que só precisamos realizar tarefas específicas do site em que estivermos trabalhando. Esse controle possui uma lógica que trabalha com passos e oferece recursos para que possamos gerenciar esses passos, além de oferecer, também, suporte tanto à navegação linear quanto à não linear.

5

Gerenciamento do estado da sessão

Teste seus conhecimentos

Mikael
Santana
426.2
B
P
57



IMPACTA
EDITORA

1. Onde ficam armazenados os dados do objeto Session?

- a) No navegador.
- b) Na memória do computador cliente.
- c) Na memória do servidor.
- d) No arquivo web.config.
- e) No arquivo machine.config.

2. Como uma sessão pode ser finalizada?

- a) Session.Finalize();
- b) Session.End();
- c) Session.Timeout=0;
- d) Session.Clear();
- e) Session.Abandon();

3. Que tipo de informação um cookie armazena?

- a) Objetos de qualquer tipo.
- b) Strings.
- c) Objetos da memória do servidor.
- d) Configurações do cliente.
- e) Instâncias da classe CookieRequest().

4. Um sistema precisa que o nome do usuário seja lembrado quando este voltar a usar o sistema algum dia. Que estratégia deve ser usada?

- a) Session
- b) Cookies
- c) web.config
- d) WebControls
- e) HtmlControls

5. Qual objeto armazena informações no formulário em campos ocultos?

- a) Session
- b) Cookies
- c) ViewState
- d) web.config
- e) Page

5

Gerenciamento do estado da sessão

Mãos à obra!

Mikael
B
Santana
57
2000
426.219
Mikael B
Santana
57
2000
426.219



IMPACTA
EDITORA

Laboratório 1

Objetivos:

- Criar um Web site para fornecer informações do licenciamento;
- Armazenar o estado em Session e Cookies;
- Postback para outra página.

1. Inicie um novo Web site do tipo **ASP.NET Empty Web Site** e nomeie-o como **Cap05Lab01**;

2. Adicione uma Master Page;

3. Adicione a página **Default.aspx** baseada na Master Page;

4. Adicione o arquivo **estilos.css**:

```
* {  
    padding: 0px;  
    margin: 0px;  
}  
  
body {  
    padding:0px;  
    margin:0px;  
}  
  
html {  
    padding:0px;  
    margin:0px;  
}  
  
#pagina {  
    font-family: Tahoma, Arial, Helvetica;  
    color:#313131;  
}
```

```
#cabecalho {  
    background-color:#6872a3;  
    color:#d6ddff;  
    padding:20px;  
}  
  
#cabecalho h1 {  
    font-size:150%;  
    letter-spacing:-1px;  
}  
  
#menu {  
    background-color:#313131;  
    padding:10px;  
}  
#menu a {  
    color:white;  
    text-decoration:none;  
    display:inline-block;  
    margin-right:5px;  
}  
  
@media screen and (max-width:400px) {  
    #menu a {  
        display:block;  
        margin-bottom:2px;  
        font-size:90%;  
    }  
}  
  
#conteudo {  
    max-width:900px;  
    margin:auto;  
    padding:20px;  
    margin-bottom:100px;  
}  
  
#conteudo h2 {  
    color:#6872a3;  
    letter-spacing:-1px;  
    margin-bottom:20px;  
}  
#conteudo p {  
    margin-bottom:10px;  
    line-height:150%;  
}
```

```
#rodape {  
    border-top:1px solid #51597d;  
  
    position:fixed;  
    bottom:0px;  
    left:0px;  
    width:100%;  
    overflow:hidden;  
    background-color:#bfcaf8;  
  
}  
  
#rodape p {  
    text-align:center;  
    font-size:80%;  
    padding:5px;  
}  
  
@media screen and (max-width:400px) {  
    #rodape {  
        position:relative;  
    }  
  
}  
  
.erro {  
    color:red;  
    display:block;  
    margin-top:10px;  
    margin-bottom:10px;  
    font-weight:bold;  
}
```

Gerenciamento do estado da sessão

5. Adicione o código HTML da Master Page com as áreas comuns **página, cabeçalho, conteúdo e rodapé**:

```
<%@ Master Language="C#" AutoEventWireup="true" CodeFile="MasterPage.master.cs" Inherits="MasterPage" %>

<!DOCTYPE html>

<html xmlns="http://www.w3.org/1999/xhtml">

<head runat="server">
    <title></title>

        <link href="estilos.css" rel="stylesheet" />

    <asp:ContentPlaceHolder id="head" runat="server">
    </asp:ContentPlaceHolder>
</head>

<body>
    <form id="form1" runat="server">
        <div id="pagina">

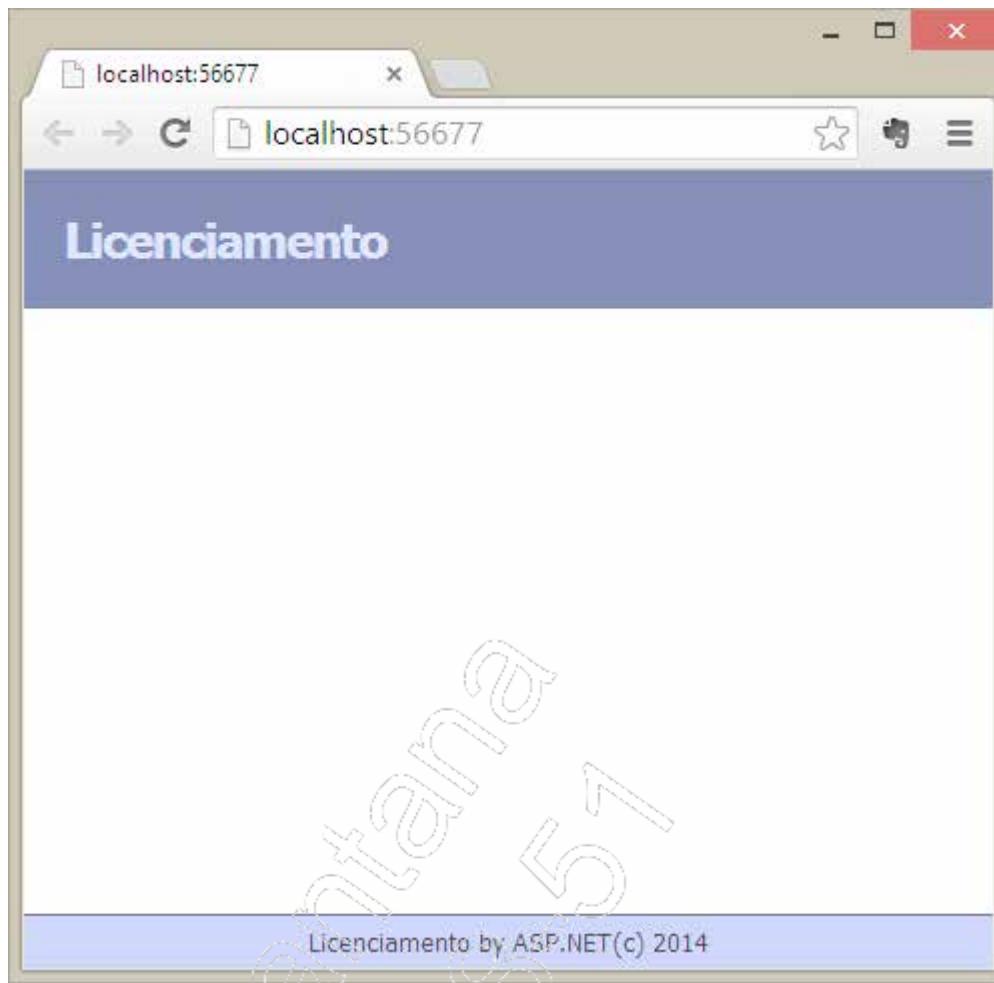
            <div id="cabecalho">
                <h1>Licenciamento</h1>
            </div>

            <div id="conteudo">
                <asp:ContentPlaceHolder
                    id="ContentPlaceHolder1"
                    runat="server">
                </asp:ContentPlaceHolder>
            </div>

            <div id="rodape">
                <p>Licenciamento by ASP.NET(c) 2014</p>
            </div>

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

6. Visualize a página Default.aspx:



7. Na página Default.aspx, crie um formulário para obter o nome e a placa. Use um **LinkButton** com o texto **Continuar** para enviar os dados. Lembre-se que o código fica dentro do segundo controle **Content**:

```
<h2>Dados do Proprietário e Veículo</h2>

<asp:Label runat="server"
    CssClass="erro" ID="erroLabel" ></asp:Label>
<p>
    <asp:Label runat="server"
        ID="nomeLabel"
        AssociatedControlID="nomeTextBox"
        Text="Nome do Proprietário:"></asp:Label>
    <br />
    <asp:TextBox runat="server"
        ID="nomeTextBox"
        Width="250px"></asp:TextBox>
</p>
```

```
<p>
    <asp:Label runat="server"
        ID="placaLabel" AssociatedControlID="placaTextBox"
        Text="Placa do Carro:"></asp:Label>
    <br />
    <asp:TextBox runat="server"
        ID="placaTextBox"
        Width="90px"
        MaxLength="7"></asp:TextBox>
</p>

<p style="margin-top:40px">
    <asp:LinkButton runat="server" id="continuarLinButton"
        Text="Continuar"></asp:LinkButton>
</p>
```

8. Adicione o código do botão Continuar:

```
if (string.IsNullOrEmpty(nomeTextBox.Text))
{
    erroLabel.Text = "&bull; Digite o nome";
    erroLabel.Visible = true;
    return;
}

if(string.IsNullOrEmpty(placaTextBox.Text))
{
    erroLabel.Text = "&bull; Digite a placa";
    erroLabel.Visible = true;
    return;
}

Session["nome"] = nomeTextBox.Text;
Session["placa"] = placaTextBox.Text;
Response.Redirect("servicos.aspx");
```

Visual Studio 2015 - ASP.NET com C# Fundamentos

9. Teste o formulário:

The image displays two screenshots of a web browser window titled "Licenciamento". Both windows show a form titled "Dados do Proprietário e Veículo".
The top window has a red error message: "• Digite o nome" above a text input field labeled "Nome do Proprietário".
The bottom window shows the same form, but the "Nome do Proprietário" field contains the value "Maria".
Both windows have a "Continuar" button at the bottom and a footer bar that says "Licenciamento by ASP.NET(c) 2014".

Gerenciamento do estado da sessão

10. Adicione um Web Form, baseado na Master Page, chamado **Servicos.aspx**. A página deve exibir os dados do proprietário e a placa do automóvel. Ainda deve ter um link para informações dos serviços e outro para voltar. Insira o HTML a seguir na página **Serviços** e visualize:

```
<h2>Serviços</h2>

<p>
    Bem-vindo, Sr(a)
    <asp:Label runat="server" ID="nomeLabel"></asp:Label>
</p>

<p>
    Clique

    <asp:HyperLink runat="server"
        NavigateUrl "~/Licenciamento.aspx"
        Text="aqui"></asp:HyperLink>

    para mais informações sobre o veículo de placa

    <asp:Label runat="server" ID="placaLabel"></asp:Label>
</p>

<p style="margin-top:40px">
    <asp:HyperLink runat="server" NavigateUrl "~/Default.aspx">Voltar
    </asp:HyperLink>
</p>
```

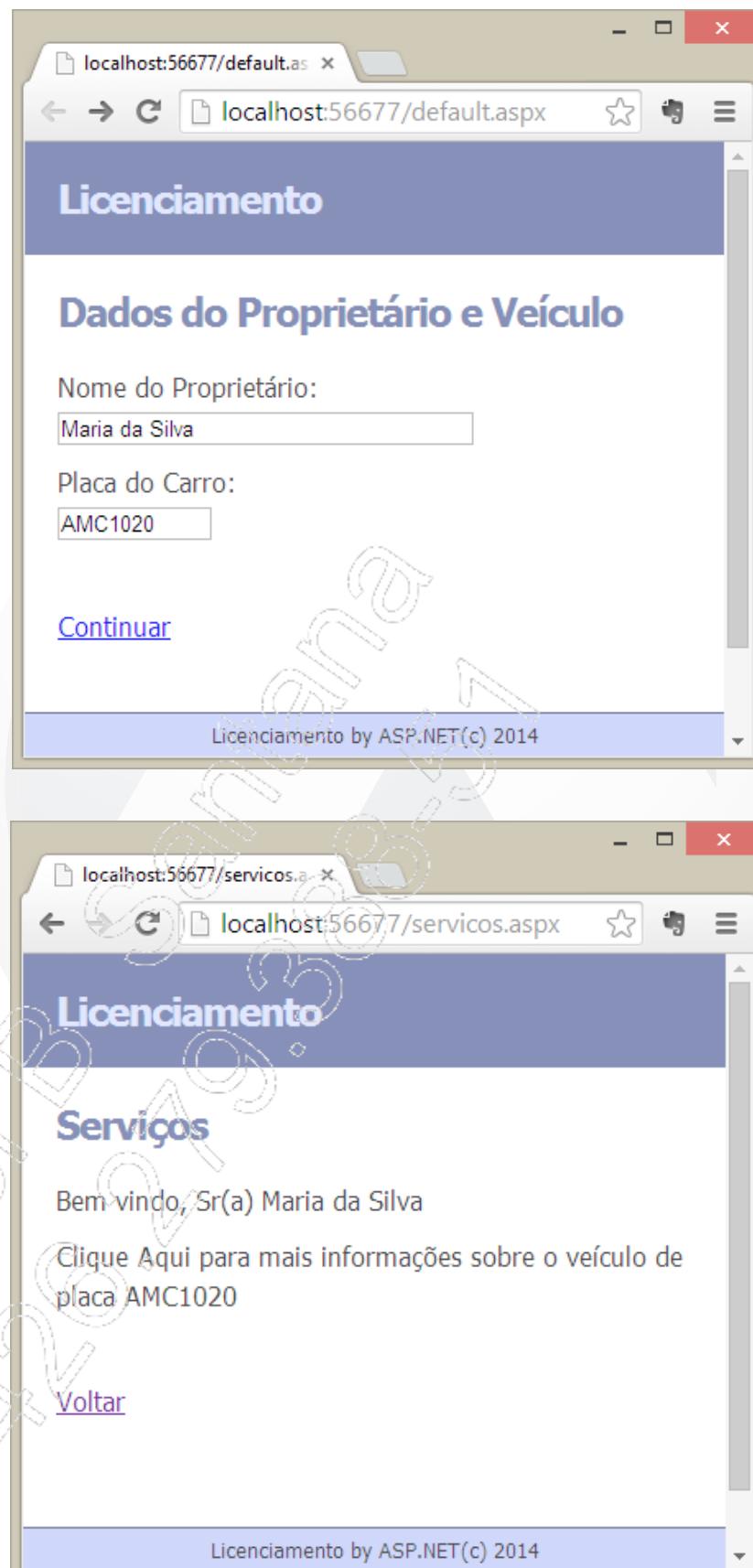
11. Na página **Serviços**, insira o código que exibe os dados da sessão, no evento **Page_Load**:

```
//Validar: Só pode entrar nesta página
//Se os dados do Requisitante estiverem na sessão
if (Session["nome"] == null || Session["placa"]==null)
{
    Response.Redirect("default.aspx");
}

//Exibe na tela
nomeLabel.Text = Session["nome"].ToString();
placaLabel.Text = Session["placa"].ToString();
```

Visual Studio 2015 - ASP.NET com C# Fundamentos

12. Teste o programa:



13. Crie um Web Form, baseado na Master Page, chamado **Licenciamento.aspx**. Monte o formulário para mostrar os dados do carro e do licenciamento:

```
<h2>Calendário do Licenciamento</h2>

<p>
    Nome: <asp:Label ID="nomeLabel" runat="server"></asp:Label>
</p>

<p>
    Placa:
    <asp:Label ID="placaLabel" runat="server"></asp:Label>
</p>

<p>
    Final:
    <asp:Label ID="finalLabel" runat="server"></asp:Label>
</p>

<p>
    Mês do licenciamento:
    <asp:Label ID="mesLabel" runat="server"></asp:Label>
</p>

<p style="margin-top:40px">
    <asp:HyperLink runat="server" NavigateUrl="~/Servicos.aspx">
        Voltar para a página de serviços
    </asp:HyperLink>
</p>

<p>
    <asp:HyperLink runat="server" NavigateUrl="~/Default.aspx">
        Informar outro veículo
    </asp:HyperLink>
</p>
```

14. Insira o código do evento **Page_Load**:

```
protected void Page_Load(object sender, EventArgs e)
{
    //Validação: os Dados devem estar na sessão
    if (Session["nome"] == null ||
        Session["placa"] == null ||
        Session["placa"].ToString().Length<7)
    {
        Response.Redirect("default.aspx");
    }

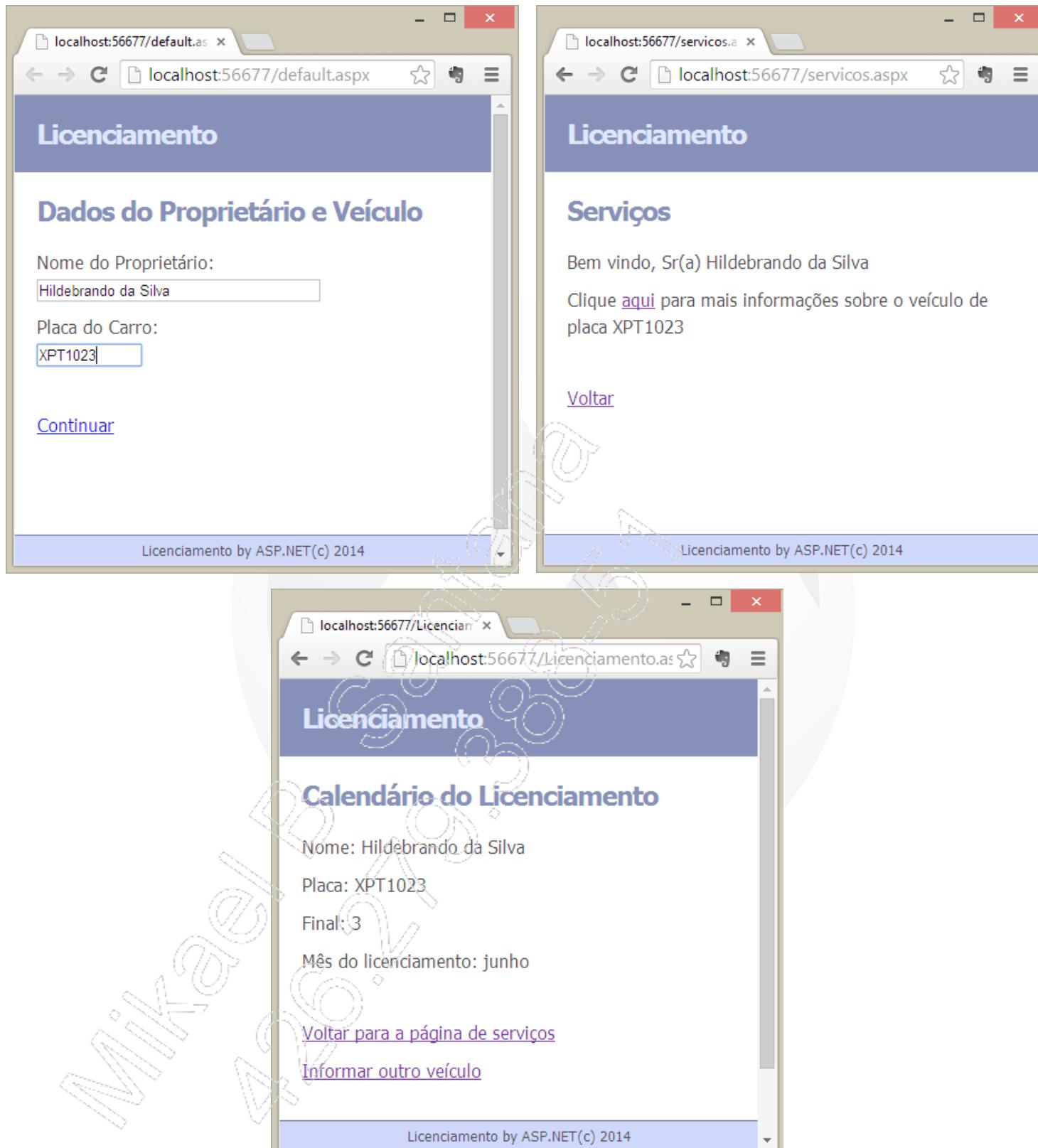
    //Obtém os dados de entrada
    string placa = Session["placa"].ToString().Trim();
    int final =
        Convert.ToInt32(placa.Substring(placa.Length - 1));
    string nome = Session["nome"].ToString();

    //processamento: Obtém o mês
    string[] meses =
    {
        "dezembro", "abril", "maio",
        "junho", "julho", "julho", "agosto",
        "setembro", "outubro", "novembro" };

    string mes = meses[final];

    //Exibe
    nomeLabel.Text = nome;
    placaLabel.Text = placa;
    finalLabel.Text = final.ToString();
    mesLabel.Text = mes;
}
```

15. Teste o Web site:



Sugestão de estudo:

- Reescrever o laboratório utilizando o modelo MVC;
- Reescrever o código para usar apenas cookies, sem sessão.

6

Componentes

- ✓ NuGet;
- ✓ Bundle/Minification;
- ✓ Bootstrap;
- ✓ FriendlyUrls.

6.1. Introdução

Os modelos de projeto disponíveis no Visual Studio incluem diversos componentes. Alguns desses componentes são desenvolvidos pela própria Microsoft enquanto outros (jQuery, por exemplo) foram incorporados. Neste capítulo, serão vistos alguns componentes que fazem parte tanto do modelo para MVC como para o modelo Web Forms. Alguns destes recursos estão resumidos na lista a seguir:

- **Bundle**: Recurso que permite agrupar diversos arquivos em um pacote, agilizando o tempo de carregamento das páginas;
- **Minification**: Recurso que permite reduzir o tamanho dos arquivos removendo espaços em branco, comentários e alterando o nome de variáveis;
- **Bootstrap**: Framework para gerenciamento de folhas de estilos;
- **FriendlyUrls**: Uma biblioteca que permite mapear uma URL para uma ou mais páginas da aplicação Web.

6.2. NuGet

Os componentes de terceiros, para serem incluídos em uma aplicação, devem ser referenciados por meio das propriedades do projeto. Além disso, os arquivos binários (DLLs) e outros que foram necessários devem ser copiados para as pastas apropriadas. Ainda assim existem componentes que necessitam de inserções dentro do arquivo de configuração ASP.NET (**Web.config**) e do arquivo de eventos (**Global.asax**).

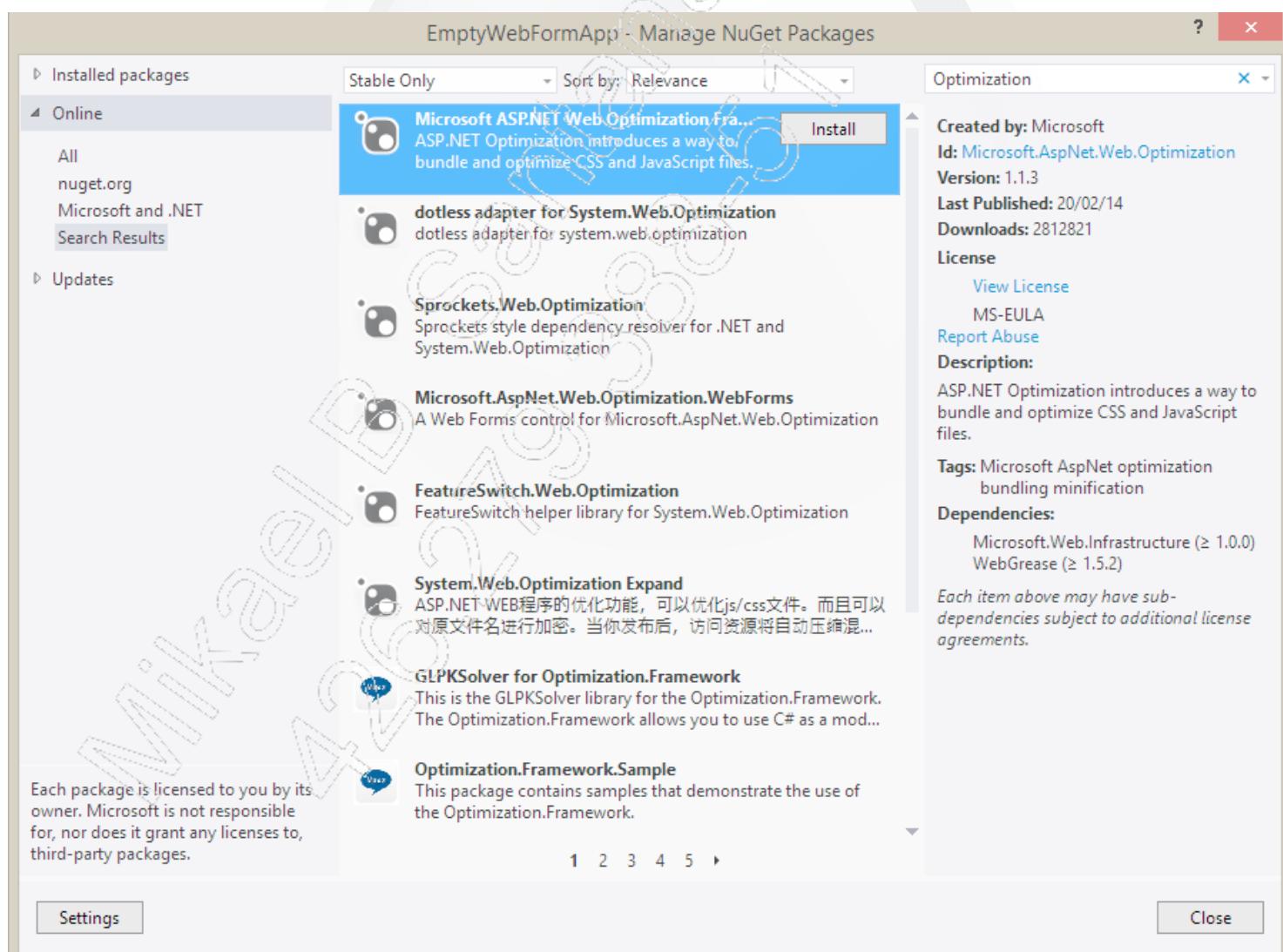
A maneira mais fácil de incluir um componente dentro da aplicação é utilizar o gerenciador de pacotes do Visual Studio, NuGet. Com esse recurso, a inclusão e a manutenção de bibliotecas são facilitadas pelo uso de scripts que automatizam o processo de configuração do componente.

No exemplo a seguir, o componente Bundle será instalado, assim como todas as bibliotecas relacionadas.

6.3. Bundle/Minification

O recurso **Bundle** permite reunir vários arquivos em um pacote para que o navegador possa realizar uma única chamada. A maioria dos navegadores impõe um limite de seis conexões simultâneas. Quando todas as conexões estão sendo usadas, o navegador bloqueia as novas até que uma fique liberada. Esse processo de bloqueio/liberação se repete, mantendo sempre o limite de conexões. O fato de reunir vários arquivos em apenas um ajuda o navegador a manter o máximo de conexões abertas e prontas para o uso.

O recurso **Bundle** está incluído no componente **Web.Optimizations**. Usando NuGet (menu **Tools / NuGet Package Manager / Manage NuGet Packages for Solution**), basta procurar **Optimization** na caixa de pesquisa. O pacote oficial se chama **Microsoft ASP.NET Web Optimization Framework**.



Visual Studio 2015 - ASP.NET com C# Fundamentos

O framework **Web.Optimization** depende de outros componentes. O NuGet baixa automaticamente a versão exigida de todos os componentes necessários. São eles:

- **Antlr**

É uma biblioteca para analisar a sintaxe de um código-fonte de uma linguagem, em um processo chamado **Parser**. Neste caso, é usado para analisar a sintaxe das folhas de estilo (arquivos **.css**).

- **WebGrease**

Biblioteca para otimizar CSS, JavaScript e imagens.

- **Newtonsoft.Json**

Biblioteca para gerar e manipular arquivos com o formato JSON.

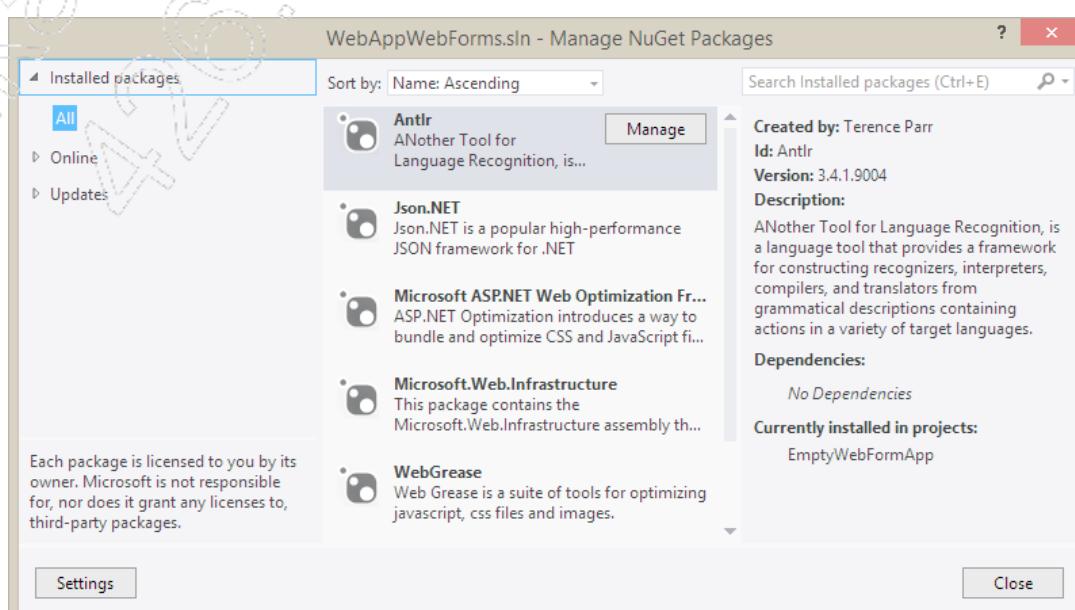
- **Microsoft.Web.Infrastructure**

Framework para registrar módulos HTTP dinamicamente.

- **Microsoft.AspNet.Web.Optimization**

O framework principal que contém o recurso **Bundle**.

É possível observar todos os componentes por meio do **NuGet**:



O componente **Web.Optimization** disponibiliza uma classe chamada **BundleTable** que serve para armazenar os conjuntos de arquivos. Cada conjunto de arquivo deve ser uma classe derivada da classe **Bundle**. Existem duas classes derivadas que são constantemente utilizadas: **ScriptBundle** e **StyleBundle**. A primeira é utilizada para criar grupos de arquivos de scripts (*.js) e a segunda para criar grupos de estilos (*.css).

6.3.1. Implementando o Bundle

Para implementar o Bundle é necessário criar uma instância de StyleBundle ou ScriptBundle. Em seguida a essa instância, devem ser adicionados os paths dos arquivos. O último passo é adicionar à instância a coleção do tipo **BundleCollection**, que está disponível na propriedade **Bundles** da classe estática **BundleTable**.

O ideal é disponibilizar isso no início do aplicativo. O melhor lugar é o evento **Application_Start**, disponível no arquivo **Global.Asax**:

```
public class Global : System.Web.HttpApplication
{
    protected void Application_Start(....)
    {
        var pathVirtual = "~/estilos";
        var sb =
            new System.Web.Optimization.StyleBundle(pathVirtual);
        sb.Include("~/css/estilo1.css", "~/css/estilo2.css");
        System.Web.Optimization.BundleTable.Bundles.Add(sb);
    }
    ...
}
```

Ou, para escrever o código de forma mais concisa, pode-se usar o encadeamento de métodos, conforme o exemplo adiante. É importante se acostumar com esta forma de escrever, porque praticamente todos os modelos da Microsoft a utilizam.

```
using System.Web.Optimization;  
  
...  
  
public class Global : System.Web.HttpApplication  
{  
  
    protected void Application_Start(...);  
    {  
  
        BundleTable.Bundles.Add(new StyleBundle("~/estilos")  
            .Include("~/css/estilo1.css",  
                    "~/css/estilo2.css"));  
    }  
}
```

6.3.2. Utilizando o Bundle

Existem diversas maneiras de utilizar o recurso **Bundle**. A primeira delas é utilizando uma chamada ao lado do servidor e usando a classe **Styles**:

```
<%@ Page Language="C#" ...>  
<%@ Import Namespace="System.Web.Optimization" %>  
  
<html>  
  <head runat="server">  
    <%: Styles.Render("~/estilos") %>  
  </head>  
  <body>  
    <form id="form1" runat="server">  
      <h1>Exemplo Bundle</h1>  
      <p>Esta página usa 2 folhas de estilos combinadas</p>  
    </form>  
  </body>  
</html>
```

Observando o código-fonte da página gerado, percebe-se que os arquivos incluídos no **global.asax** também foram gerados:

```
<!DOCTYPE html>
<html>
<head>
    <title></title>

    <link href="/css/Estilo1.css" rel="stylesheet" />
    <link href="/css/Estilo2.css" rel="stylesheet" />

</head>
<body>
    ...
</body>
</html>
```

Isso ainda não resolve o problema, porque o navegador terá que chamar cada folha de estilo em conexões diferentes. Isso aconteceu porque, por padrão, a otimização está desligada. Para ligar a otimização, usa-se a propriedade **EnableOptimization** da classe **BundleTable**:

```
public class Global : System.Web.HttpApplication
{
    protected void Application_Start(object sender, EventArgs e)
    {
        BundleTable.Bundles.Add(
            new StyleBundle("~/estilos")
                .Include("~/css/Estilo1.css",
                    "~/css/Estilo2.css"));

        BundleTable.EnableOptimizations = true;
    }
}
```

Visual Studio 2015 - ASP.NET com C# Fundamentos

Agora, o código-fonte gerado agrupa os dois arquivos em um só:

```
<!DOCTYPE html>
<html>
<head>
    <title></title>

        <link href="/estilos?v=nVwG09IMZVsg..." rel="stylesheet" />

</head>
<body>
...
</body>
</html>
```

O conteúdo desse arquivo é a união dos dois arquivos sem espaços e sem comentários. Veja, na lista a seguir, os arquivos originais e o arquivo gerado:

- Arquivos originais:

```
Estilo1.css
/*
    Formato do Corpo da Página
*/
body {
    font-family:'Courier New';
}
```

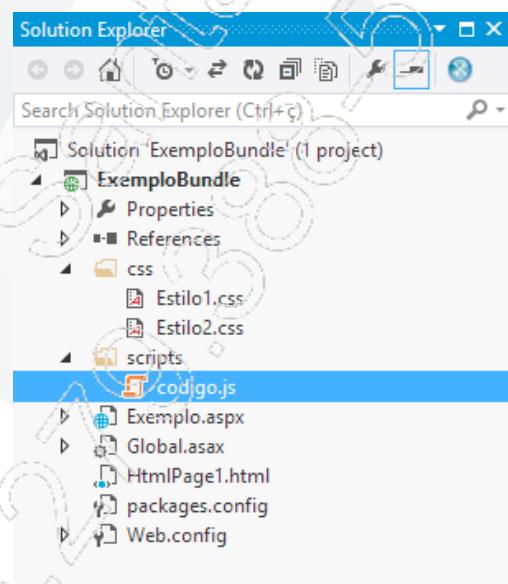
```
Estilo2.css
/*
    Formato do Título Principal
*/
h1{
    border-bottom:1px solid black;
}
```

- Arquivo gerado no processo **Bundle**:



Repare que os arquivos não só foram agrupados mas otimizados. Os espaços e comentários foram retirados e a cor **black** foi trocada para **#000**. Esse processo de otimização do arquivo se chama **Minification**.

No caso de arquivos JavaScript, não apenas os espaços e comentários são retirados mas os nomes de variáveis também. Considere o seguinte arquivo dentro da pasta **scripts**:



- **codigo.js**

```
// função para retornar
// o nome completo
function obterNomeCompleto(nome, sobrenome)
{
    var espaco = ' ';
    var nomeCompleto = nome + espaco + sobrenome;
    return nomeCompleto;
}
```

Visual Studio 2015 - ASP.NET com C# Fundamentos

Para incluir este arquivo em um **Bundle**, usa-se a classe **ScriptBundle**.

- Arquivo **Global.asax**

```
protected void Application_Start(object sender, EventArgs e)
{
    BundleTable.Bundles.Add(
        new StyleBundle("~/estilos")
            .Include("~/css/Estilo1.css",
                    "~/css/Estilo2.css"));

    BundleTable.Bundles.Add(
        new ScriptBundle("~/js")
            .Include("~/scripts/codigo.js"));

    BundleTable.EnableOptimizations = true;
}
```

- Na página, usa-se a classe **Scripts**:

```
<%@ Page Language="C....>
<%@ Import Namespace="System.Web.Optimization" %>
<html>
<head runat="server">
    <%:
        Styles.Render("~/estilos")
    %>

    <%:
        Scripts.Render("~/js")
    %>

</head>
<body>
    <form id="form1" runat="server">
        <h1>Exemplo Bundle</h1>
        <p>Esta página usa 2 folhas de estilos combinadas</p>
    </form>
</body>
</html>
```

- No código gerado, o script otimizado é vinculado:

```
<head>
    <title></title>
    <link href="/estilos?v=nVwG09IM..." rel="stylesheet" />

    <script src="/js?v=e57uXPNtCY..."></script>

</head>
```

O conteúdo do JavaScript é este:



- Comparando com o original:

```
// função para retornar
// o nome completo
function obterNomeCompleto(nome, sobrenome)
{
    var espaco = ' ';
    var nomeCompleto = nome + espaco + sobrenome;
    return nomeCompleto;
}
```

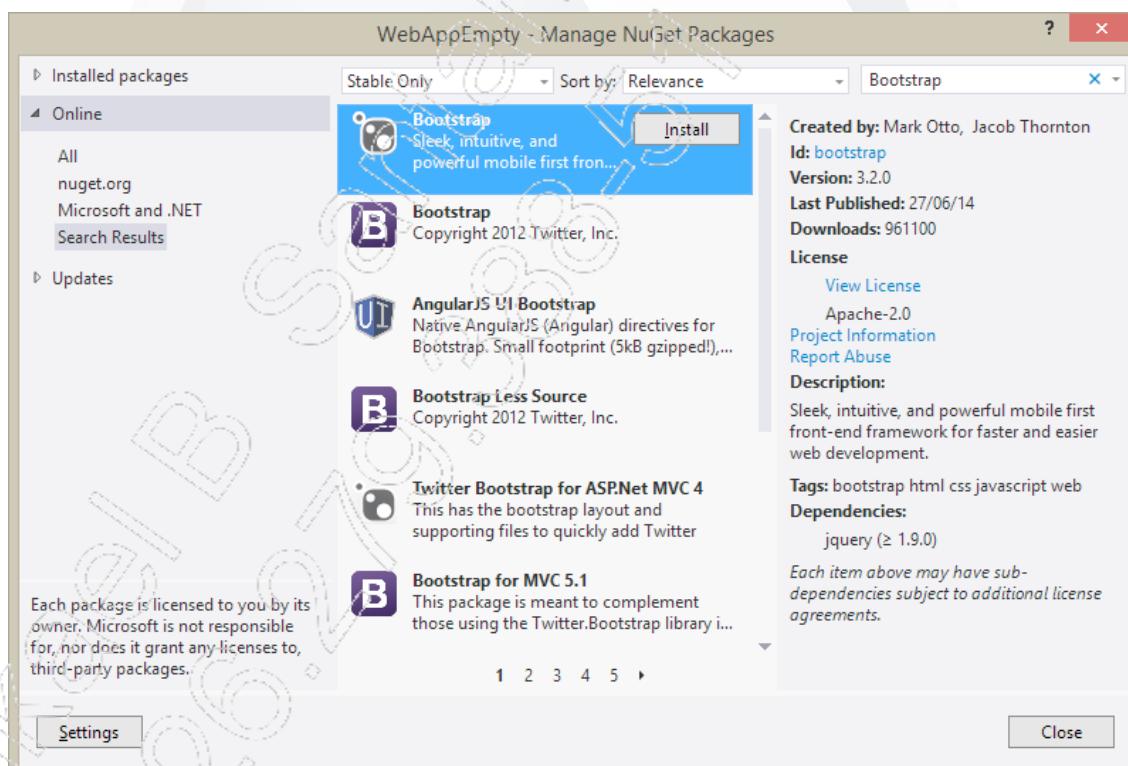
Os recursos de **Bundle** e **Minification** tornam o carregamento da página mais eficiente. Os novos modelos do ASP.NET para Web Forms e MVC usam esses recursos como padrão.

6.4. Bootstrap

Bootstrap é um framework JavaScript, desenvolvido para criar páginas responsivas. Essas páginas se adaptam ao dispositivo e às dimensões da tela onde o aplicativo é executado.

Fisicamente, o Bootstrap é composto de arquivos CSS, JavaScript e de imagens (incorporadas em fontes true type). Com poucas regras, o Bootstrap é facilmente incorporado em qualquer site que utilize HTML5. No Visual Studio, na versão 2013, ele já vem como parte dos modelos básicos.

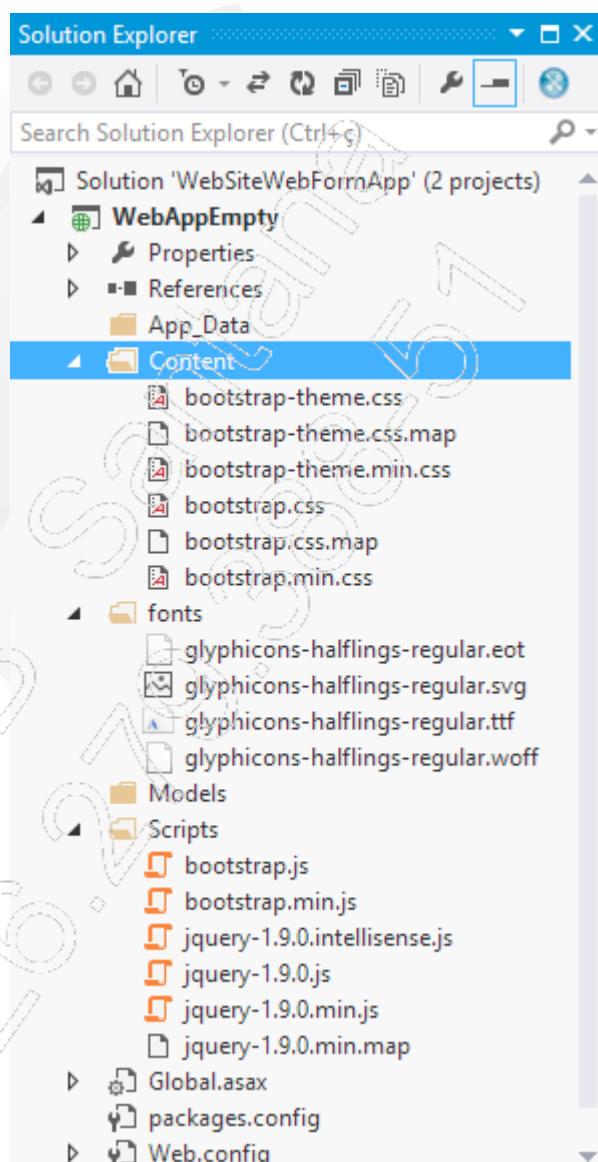
O exemplo a seguir utiliza o Bootstrap em um projeto vazio, para facilitar a localização dos arquivos e elementos. Em um projeto vazio, entre no NuGet e pesquise **Bootstrap**:



O conteúdo adiante é inserido no arquivo **packages.config**. Repare que o Bootstrap precisa do framework **jQuery** para funcionar:

Packages.config

```
<?xml version="1.0" encoding="utf-8"?>
<packages>
    <package id="bootstrap" version="3.2.0" targetFramework="net451" />
    <package id="jQuery" version="1.9.0" targetFramework="net451" />
</packages>
```



Visual Studio 2015 - ASP.NET com C# Fundamentos

Os principais arquivos são os seguintes:

- Na pasta **Content**:
 - **bootstrap-theme.css**: Folha de estilo contendo classes que definem um tema;
 - **bootstrap.css**: Folha de estilo contendo as classes CSS básicas para definição de colunas, tabelas, título etc.
- Na pasta **Script**:
 - **bootstrap.js**: Arquivo JavaScript com o código do Bootstrap;
 - **jquery-(versão).js**: Biblioteca jQuery, necessária para rodar o Bootstrap.
- Na pasta **Foods**:
 - **Glyphicon.***: Arquivos de fonte true type contendo ícones que podem ser usados em qualquer elemento.

Para a maioria dos arquivos, existe uma versão com o sufixo **.min**. Esse arquivo é uma versão compactada do arquivo original para ser colocado em ambiente de produção.

No modo de desenvolvimento do projeto, é interessante usar a versão completa dos componentes para poder utilizar o recurso de debug, mas, em produção, os arquivos compactados oferecem um ganho de performance.

Outros arquivos que compõem o pacote são os que terminam com a extensão **.map**. Esses arquivos são usados por pré-processadores de arquivos CSS.

Esses são componentes que permitem incluir variáveis e realizar operações lógicas simples em folhas de estilo. Alguns dos mais conhecidos são **Less**, **Sass** e **Stylus**. Não são necessários para a operação básica do Bootstrap.

Para que o Bootstrap funcione com as opções básicas, são necessárias as seguintes inclusões na página:

```
<head runat="server">
  <title></title>

  <link href="Content/bootstrap.css" rel="stylesheet" />
  <link href="Content/bootstrap-theme.css" rel="stylesheet" />

  <script src="Scripts/jquery-1.9.0.js"></script>
  <script src="Scripts/bootstrap.js"></script>

</head>
```

O princípio básico do Bootstrap é padronizar o nome das classes usadas para criar o layout de uma aplicação Web. Algumas das classes que podem ser usadas são:

btn	btn-default	btn-danger	btn-link
navbar	navbar-header	navbar-fixed-top	navbar-inverse
container	jumbotron	table	table-condensed
label	label-sucess	label-info	label-danger
dropdown	dropdown-menu	alert	alert-warning
col	row	gryphicon	form

Observando os nomes, é fácil perceber que são classes para formatar elementos da tela: botões (**btn**), barras de navegação (**navbar**), tabelas (**table**), mensagens (**alert**), legendas (**label**), divisões (**container**) e assim por diante. O site do Bootstrap (<http://getbootstrap.com>) contém centenas de exemplos e modelos de uso de cada classe.

Visual Studio 2015 - ASP.NET com C# Fundamentos

Neste exemplo simples, as classes **jumbotron**, **btn** e **btn-info** foram usadas:

```
<%@ Page Language="C#" ...  
<!DOCTYPE html>  
  
<html>  
<head runat="server">  
    <link href="Content/bootstrap.css" rel="stylesheet" />  
    <link href="Content/bootstrap-theme.css" rel="stylesheet" />  
  
    <script src="Scripts/jquery-1.9.0.js"></script>  
    <script src="Scripts/bootstrap.js"></script>  
</head>  
<body>  
    <form id="form1" runat="server">  
  
        <div class="jumbotron">  
            <h1>Bem-vindo ao bootstrap </h1>  
  
            <asp:Button runat="server"  
                CssClass="btn btn-info"  
                Text="Clique aqui" />  
  
        </div>  
    </form>  
</body>  
</html>
```



Vejamos a adição de uma navegação usando as classes **navbar**, **navbar-header**, **navbar-brand**, **navbar-nav** e **container** (partes do código HTML foram omitidas para maior clareza):

```
<%@ Page Language="C#" ...>

<!DOCTYPE html>
<html>
<head runat="server">
    ...
</head>

<body>
    <form id="form1" runat="server">

        <div class="navbar">

            <div class="navbar-header">

                <a class="navbar-brand" href="#">Meu WebSite</a>

            </div>

            <div class="container">

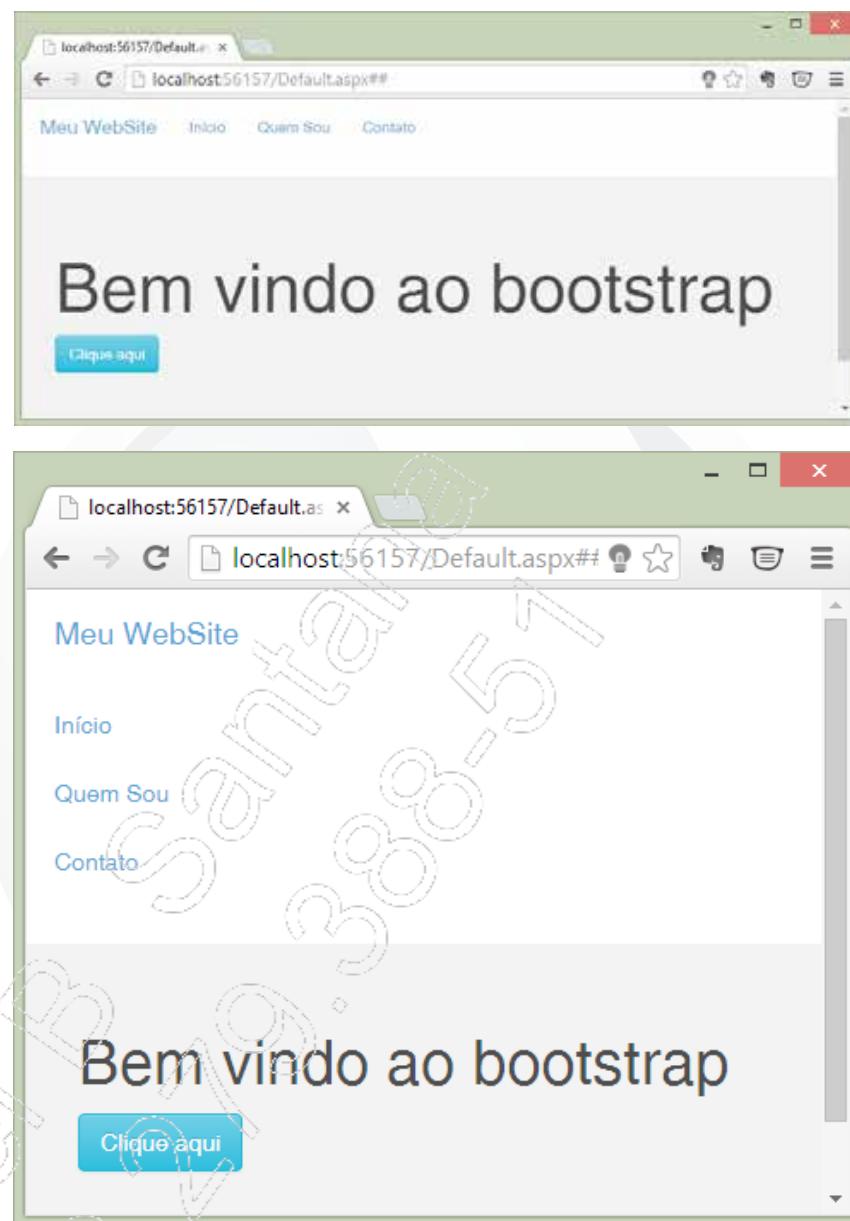
                <ul class="nav navbar-nav">

                    <li><a href="#">Início</a></li>
                    <li><a href="#">Quem Sou</a></li>
                    <li><a href="#">Contato</a></li>
                </ul>
            </div>
        </div>

        <div class="jumbotron">
            <h1>Bem-vindo ao bootstrap </h1>
            <asp:Button runat="server" . . . />
        </div>
    </form>
</body>
</html>
```

Visual Studio 2015 - ASP.NET com C# Fundamentos

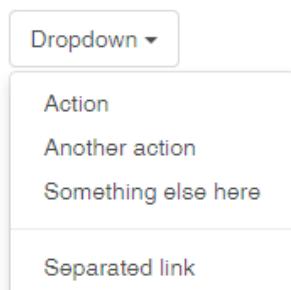
Ao ser renderizado, o link principal é gerado, bem como o menu horizontal, se houver espaço. Se não houver espaço para o menu, este é automaticamente convertido para a versão vertical. Assim, o site pode ser facilmente visualizado, tanto em computadores desktop como em tablets ou celulares.



Este gerenciamento automático de layout, compatível com todos os browsers e todos os possíveis dispositivos (computador, tablet, celular), demonstra o poder deste componente. Levaria muito tempo (geralmente na base da tentativa e erro) para adaptar as folhas de estilo a todas as possíveis variações de ambiente em que o site vai ser visualizado. O Bootstrap é resultado de muito esforço e testes de diversos profissionais de design, e permanece em contínuo desenvolvimento pela comunidade que o criou, adaptando-se aos novos ambientes. Não foi à toa que a Microsoft resolveu incorporar oficialmente o Bootstrap ao Visual Studio e aos modelos de projetos.

Vejamos, a seguir, uma lista de alguns estilos prontos para uso (as imagens foram obtidas nos exemplos oficiais do site):

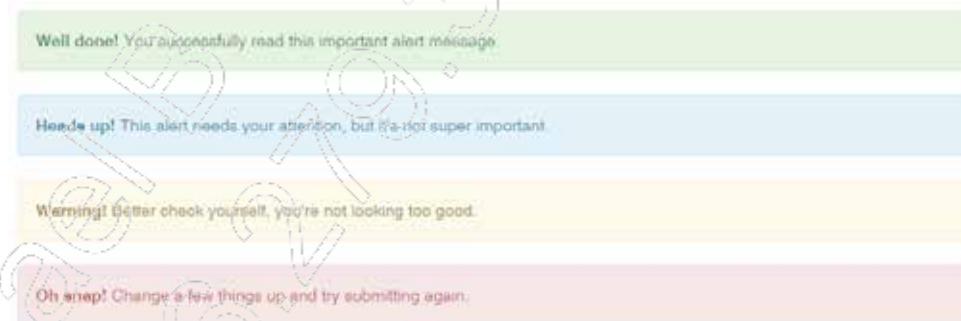
- **Caixas de seleção**



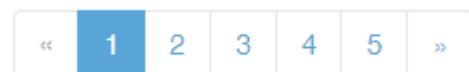
- **Botões**



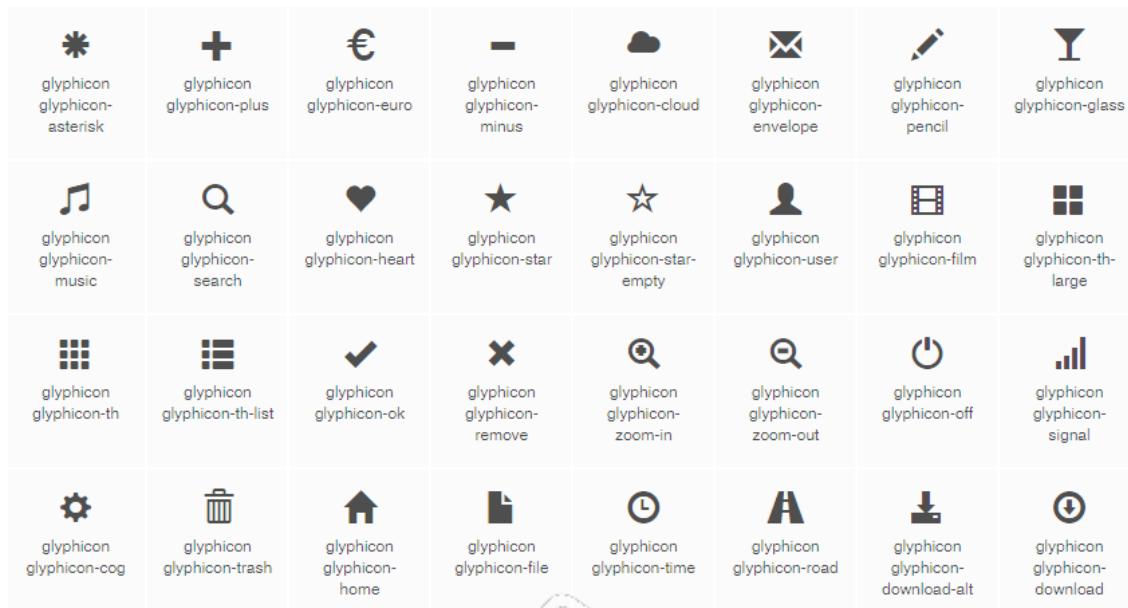
- **Alertas**



- **Paginação**



- Ícones



- Barras de progresso



- Painéis



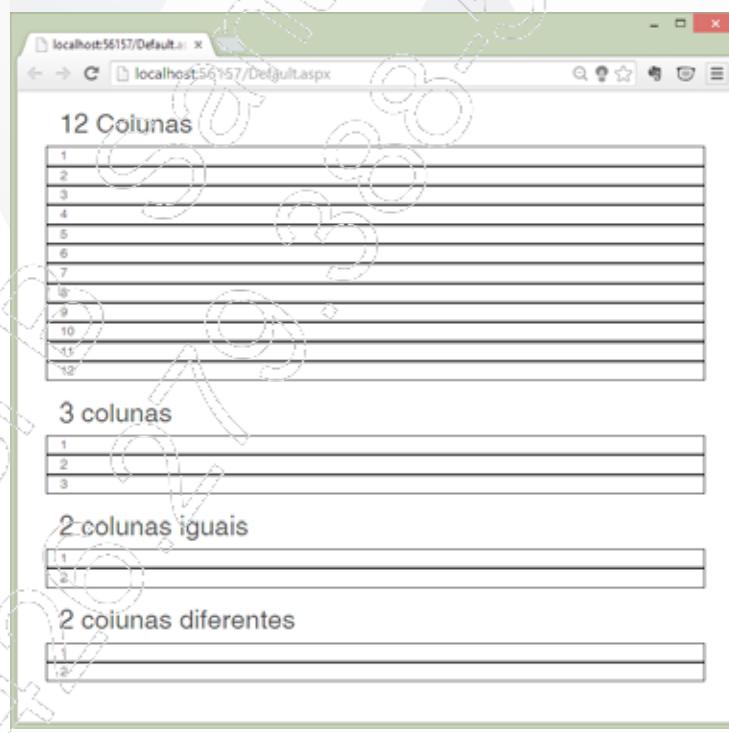
Um dos conceitos mais interessantes do Bootstrap é a divisão do layout em linhas e 12 colunas. O número 12 foi escolhido porque é divisível por 2, 3, 4 e 6. Se quiser 2 colunas, a largura de cada coluna será 6. Se quiser 3 colunas, cada coluna terá a largura de 4. Se quiser uma coluna grande e outra pequena, uma coluna pode ter a largura 8 e a segunda coluna 4. A soma deve ser sempre 12.

Quando as colunas em uma linha não cabem na tela, automaticamente cada coluna é exibida embaixo da anterior. Vejamos alguns exemplos:

- Resolução normal (> 900 px):



- Resolução pequena (< 900 px):



Adiante, o código HTML do exemplo anterior define as colunas:

```
<h2>12 Colunas</h2>
<div class="row">
    <div class="col-md-1"> 1 </div>
    <div class="col-md-1"> 2 </div>
    <div class="col-md-1"> 3 </div>
    <div class="col-md-1"> 4 </div>
    <div class="col-md-1"> 5 </div>
    <div class="col-md-1"> 6 </div>
    <div class="col-md-1"> 7 </div>
    <div class="col-md-1"> 8 </div>
    <div class="col-md-1"> 9 </div>
    <div class="col-md-1"> 10 </div>
    <div class="col-md-1"> 11 </div>
    <div class="col-md-1"> 12 </div>
</div>
```

```
<h2>3 colunas</h2>
<div class="row">
    <div class="col-md-4"> 1 </div>
    <div class="col-md-4"> 2 </div>
    <div class="col-md-4"> 3 </div>
</div>
```

```
<h2>2 colunas iguais</h2>
<div class="row">
    <div class="col-md-6"> 1 </div>
    <div class="col-md-6"> 2 </div>
</div>
```

```
<h2>2 colunas diferentes</h2>
<div class="row ">
    <div class="col-md-10"> 1 </div>
    <div class="col-md-2"> 2 </div>
</div>
```

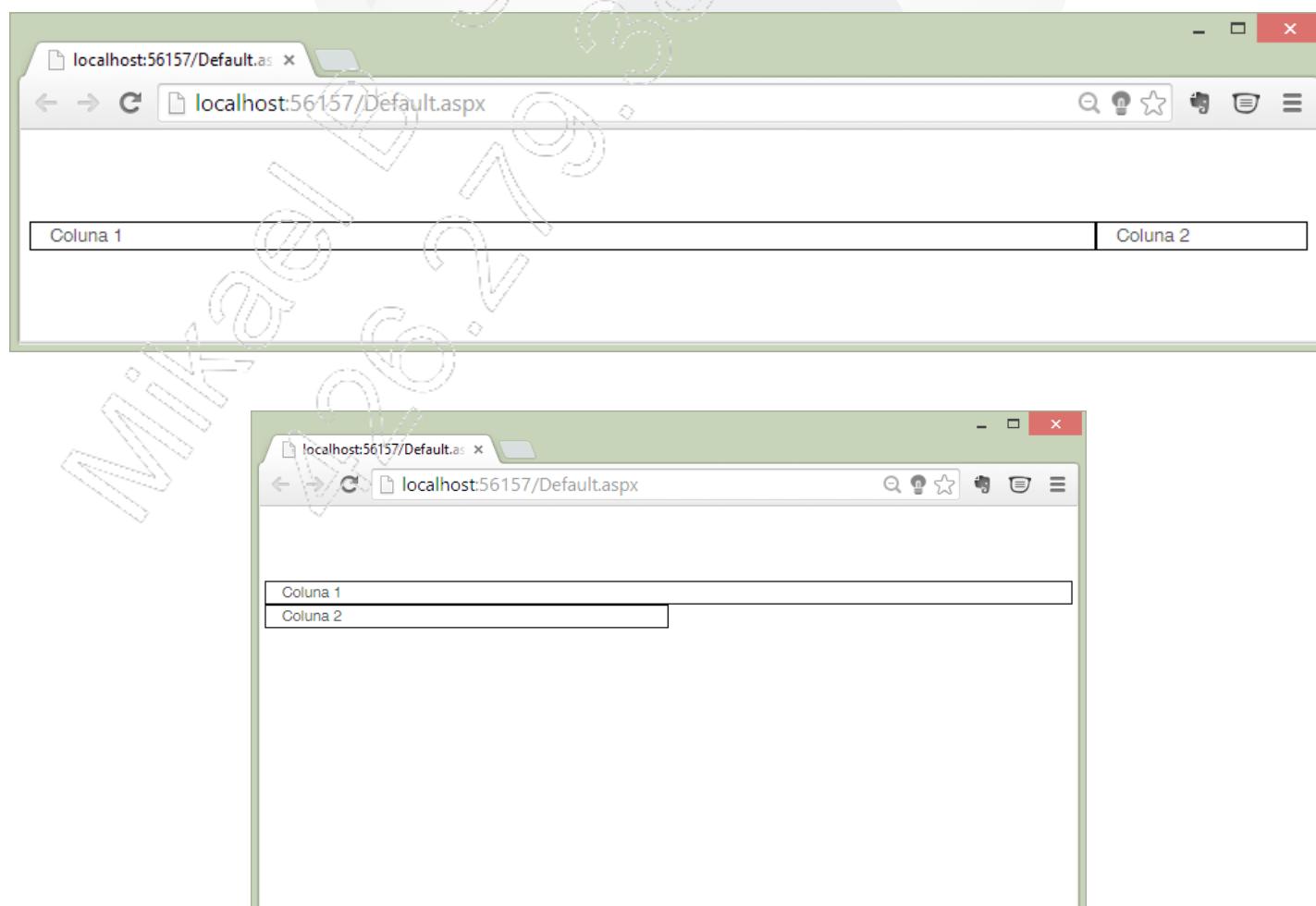
As letras **md** no nome da classe se referem ao tipo de dispositivo:

- **md**: Dispositivos médios (desktop): Resolução ≥ 992 px;
- **sm**: Dispositivos pequenos (tablets): Resolução ≥ 768 px;
- **xl**: Dispositivos extra-pequenos (celulares): Resolução < 768 px;
- **lg**: Dispositivos grandes (desktop): Resolução ≥ 1200 px.

A classe **col-sm-5** significa "cinco colunas quando a tela for de um tablet", enquanto que a classe **col-md-2** significa "duas colunas quando for um monitor normal".

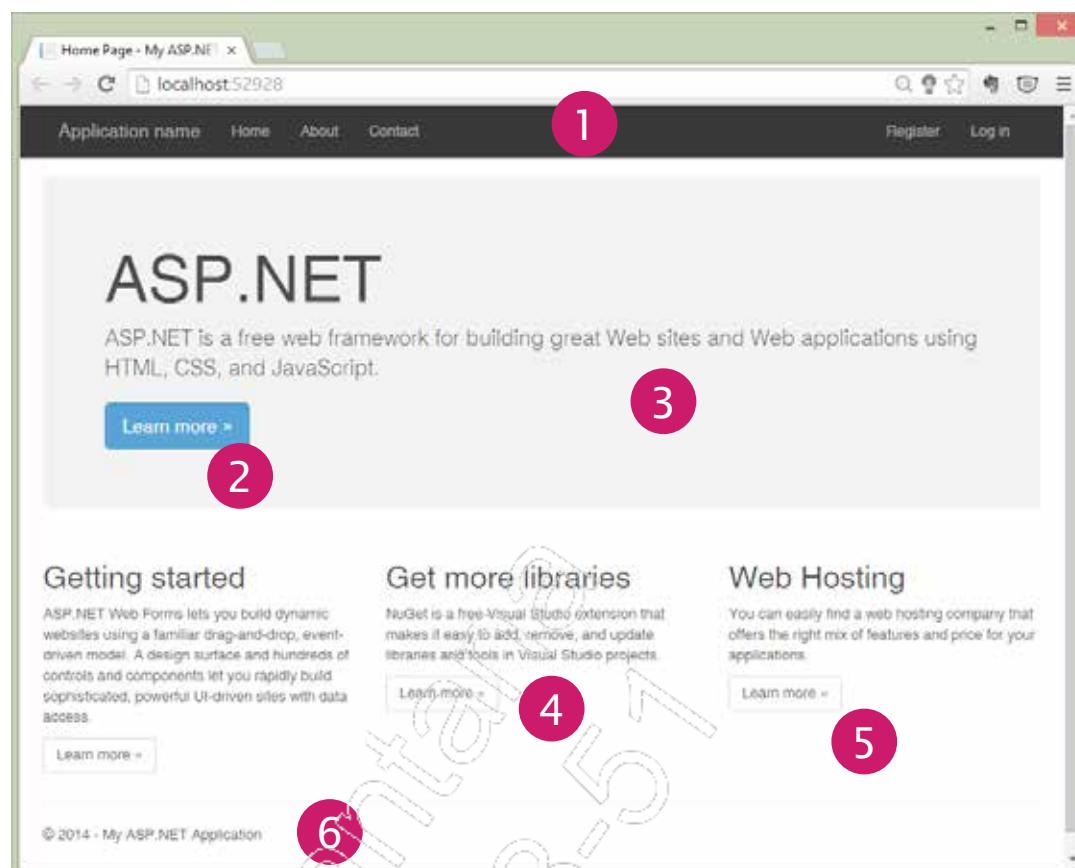
O exemplo a seguir mostra duas colunas, uma ao lado da outra, quando for um desktop e uma embaixo da outra quando for celular:

```
<div class="row ">
  <div class="col-xs-12 col-md-10"> Coluna 1</div>
  <div class="col-xs-6 col-md-2"> Coluna 2</div>
</div>
```



Visual Studio 2015 - ASP.NET com C# Fundamentos

O projeto padrão de um Web Form do Visual Studio 2013 utiliza as seguintes classes do Bootstrap:



- 1 – navbar, navbar-inverse, navbar-fixed-top, navbar-header, navbar-brand;
- 2 – btn, btn-primary, btn-lg;
- 3 – jumbotron;
- 4 – col-md-4;
- 5 – btn, btn-default;
- 6 – <footer> (não foi usada a classe e sim a definição da tag).

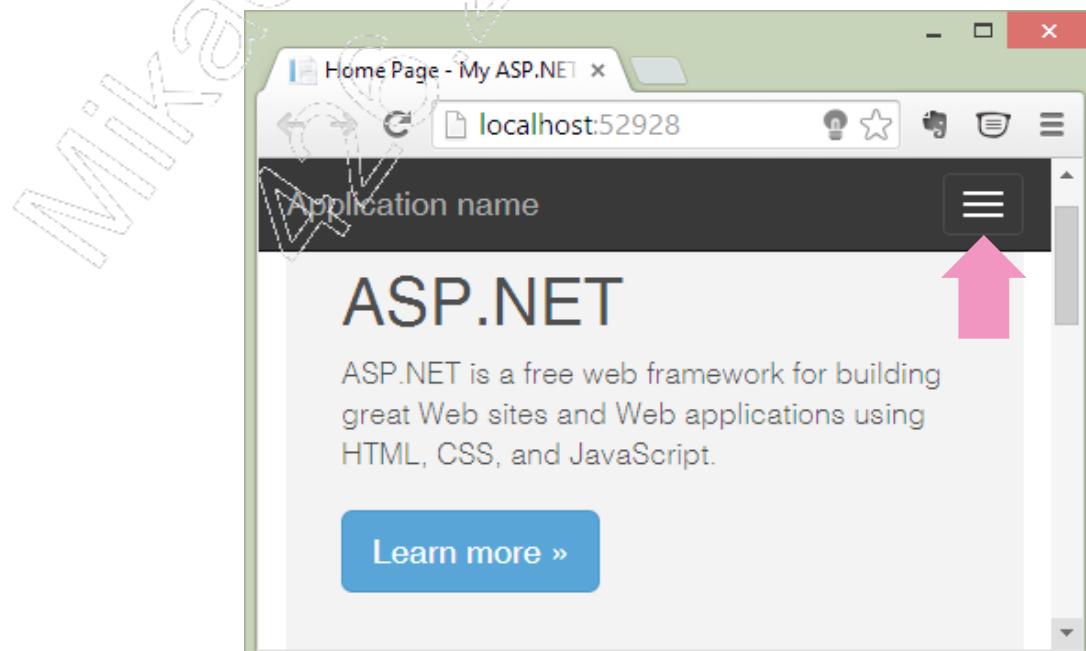
Ao analisar o HTML do navegador, é possível perceber itens que não estão sendo exibidos na visualização anterior. O item **icon-bar** não aparece porque apenas será visível em um celular ou dispositivo com resolução baixa. As imagens a seguir mostram o código HTML exibido em diferentes resoluções de tela:

- Dados exibidos em um celular:

```
<div class="navbar navbar-inverse navbar-fixed-top">
  <div class="container">
    <div class="navbar-header">

      <button type="button" class="navbar-toggle"
        data-toggle="collapse"
        data-target=".navbar-collapse">
        <span class="icon-bar"></span>
        <span class="icon-bar"></span>
        <span class="icon-bar"></span>
      </button>

      <a href="" class="navbar-brand">App Namer</a>
    </div>
    <div class="navbar-collapse collapse">
      <ul class="nav navbar-nav">
        <li><a href="">Home</a></li>
        <li><a href="">About</a></li>
        <li><a href="">Contact</a></li>
      </ul>
      <ul class="nav navbar-nav navbar-right">
        <li><a href="">Register</a></li>
        <li><a href="">Log in </a></li>
      </ul>
    </div>
  </div>
</div>
```



Visual Studio 2015 - ASP.NET com C# Fundamentos

- Dados exibidos em um computador desktop:

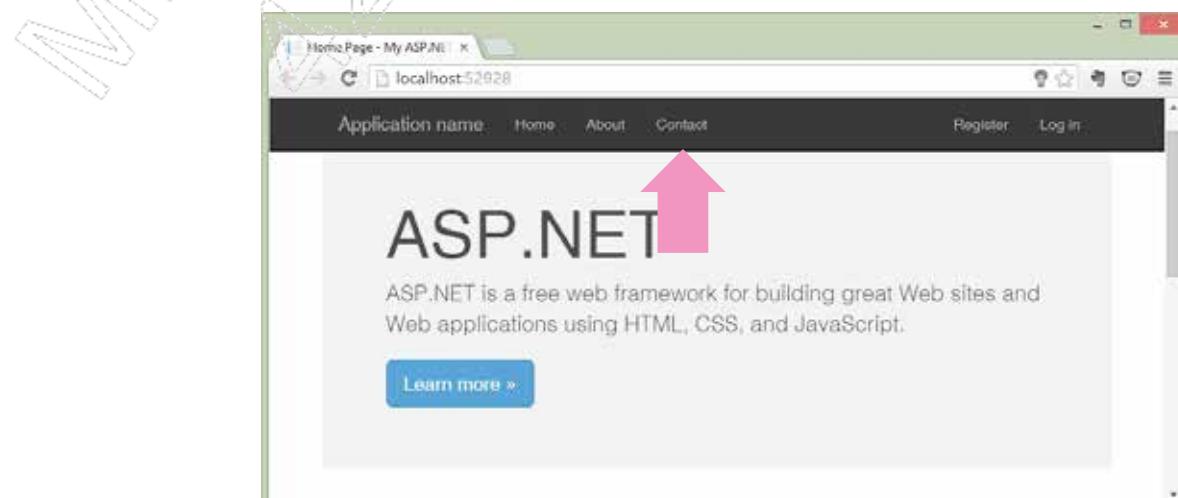
```
<div class="navbar navbar-inverse navbar-fixed-top">
    <div class="container">
        <div class="navbar-header">
            <button type="button" class="navbar-toggle"
                data-toggle="collapse"
                data-target=".navbar-collapse">
                <span class="icon-bar"></span>
                <span class="icon-bar"></span>
                <span class="icon-bar"></span>
            </button>

            <a href="" class="navbar-brand">App Namer</a>
        </div>

        <div class="navbar-collapse collapse">

            <ul class="nav navbar-nav">
                <li><a href="">Home</a></li>
                <li><a href="">About</a></li>
                <li><a href="">Contact</a></li>
            </ul>

            <ul class="nav navbar-nav navbar-right">
                <li><a href="">Register</a></li>
                <li><a href="">Log in </a></li>
            </ul>
        </div>
    </div>
</div>
```

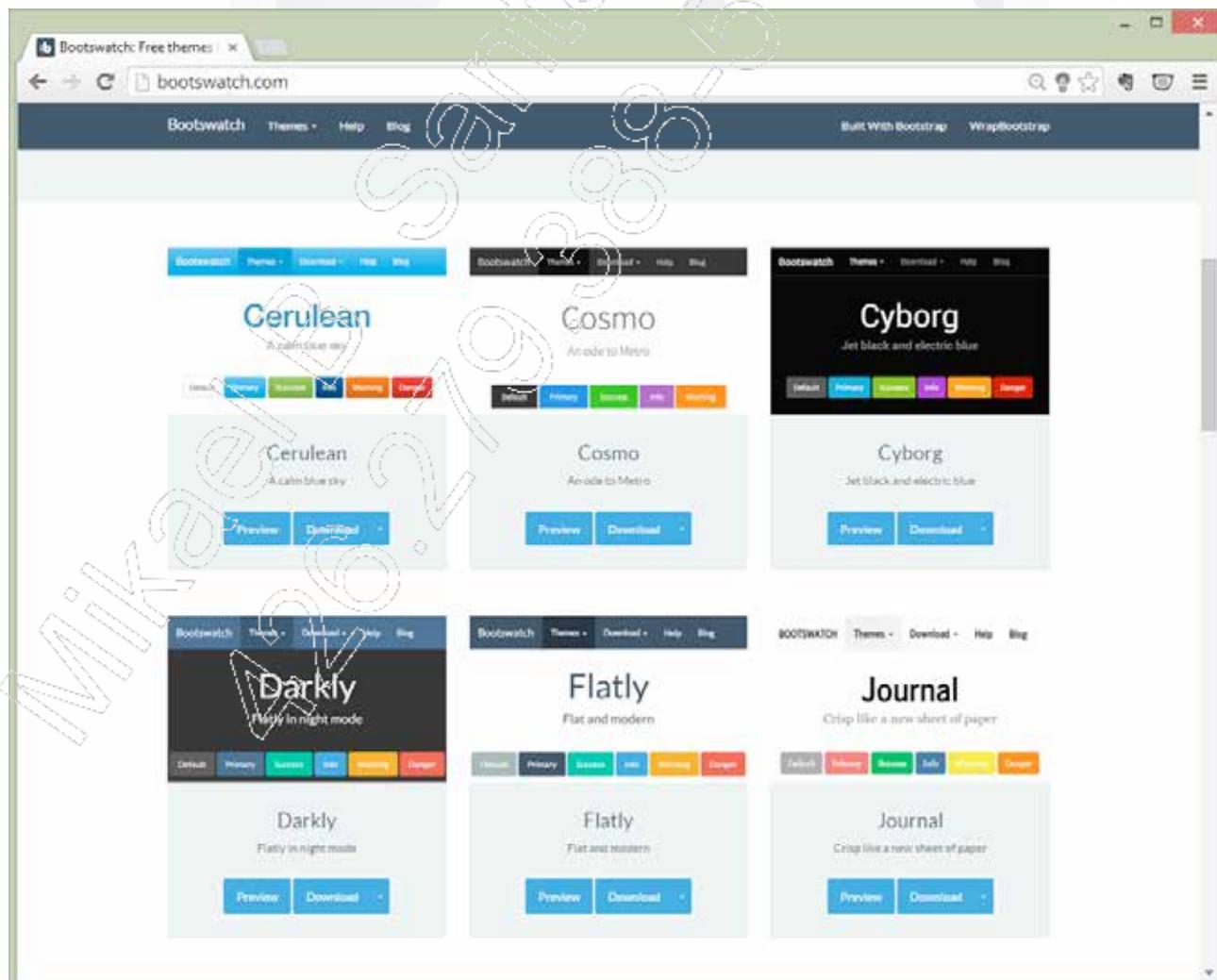


Isso é conseguido porque, nas definições da folha de estilo do arquivo **bootstrap.css**, está definido para não ser exibido se a largura da tela for maior que 768 px:

```
@media (min-width: 768px) {  
    .navbar-toggle {  
        display: none;  
    }  
}
```

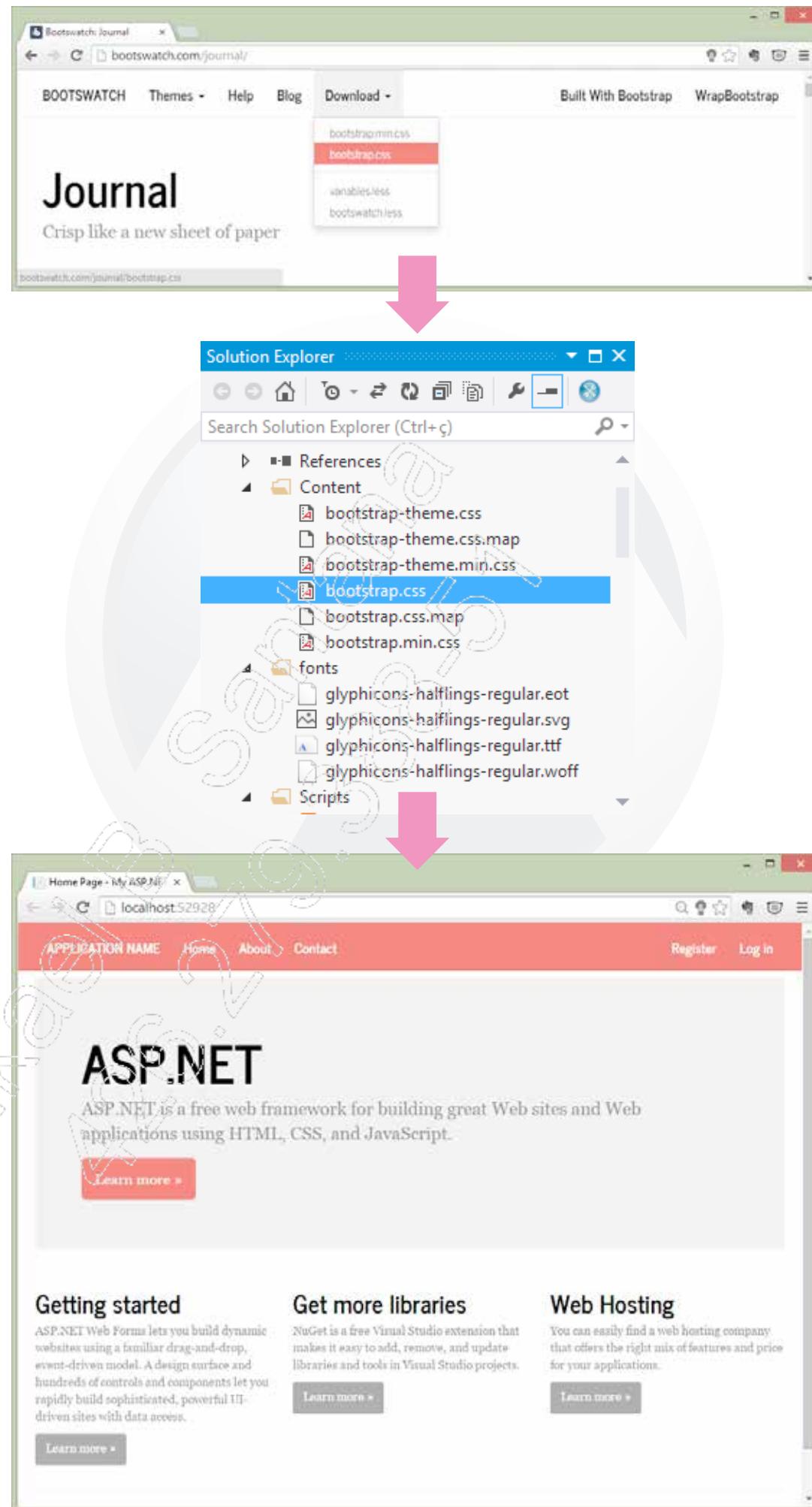
Observando a folha de estilos padrão do Bootstrap, é possível encontrar dezenas de referências a classes que são exibidas ou ocultas dependendo da largura da tela.

No site <http://bootswatch.com> é possível encontrar diversos temas para usar no lugar do tema padrão que é disponibilizado no modelo do Visual Studio:



Visual Studio 2015 - ASP.NET com C# Fundamentos

Existem várias maneiras de definir temas. A mais simples é fazer o download do arquivo **bootstrap.css** e substituir o arquivo original do Visual Studio:



6.5. FriendlyUrls

ASP.NET FriendlyUrls é um componente que fornece uma maneira simples de remover da URL a extensão de arquivos registrados no ASP.NET. Na prática, isso torna as URLs mais significativas e fáceis de serem lidas e memorizadas.

Vejamos o exemplo de URL a seguir:

http://servidor.com.br/produtos.aspx?id=1

Ela ficaria da seguinte forma:

http://servidor.com/produtos/1

O componente **FriendlyUrls** utiliza um recurso chamado **Routing**. Esse recurso permite definir URLs que podem ser usadas para fornecer acesso a um recurso do aplicativo (não necessariamente uma página).

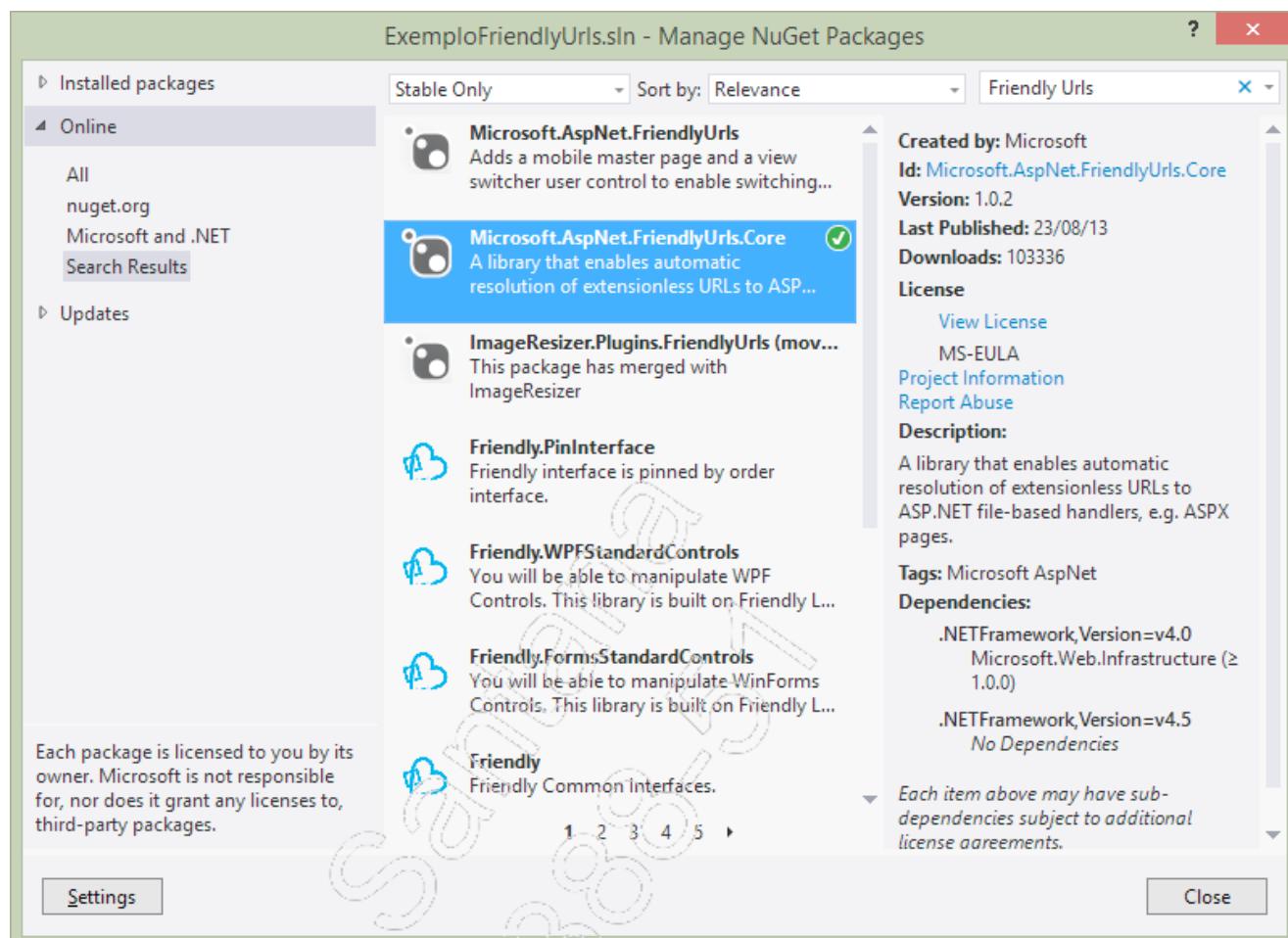
O recurso **Routing** é amplamente usado no modelo MVC, em que não há uma relação direta entre uma URL e uma página. O modelo Web Form, diferente do modelo MVC, executa um código que sempre está associado a um arquivo ASPX (code-behind).

Esse padrão dos Web Forms pode ser sobreposto, gerenciando diretamente as rotas, por meio das classes do namespace **System.Web.Routing**. Acontece que esse gerenciamento exige uma atenção especial a vários detalhes, como tratamento de parâmetros, verificação da existência ou não de arquivos, conflito nos nomes, permissões etc.

O componente **FriendlyUrls** serve exatamente para realizar esse gerenciamento sem a necessidade de usar diretamente as classes para gerenciamento de rotas, facilitando muito a implantação dessa funcionalidade em um projeto Web Forms.

Visual Studio 2015 - ASP.NET com C# Fundamentos

Para usar o recurso **FriendlyUrls** é necessário adicionar as referências ao componente. A melhor maneira de fazer isso é usar o **NuGet**. O componente principal se chama **Microsoft.AspNet.FriendlyUrls.Core**.



Para habilitar o recurso de mapear os arquivos de um aplicativo Web Form para usar URLs "amigáveis", é necessário chamar o método de extensão **EnableFriendlyUrls()** da classe **RouteCollections**, que é disponibilizada na propriedade **Routes** da classe estática **RouteTable**.

```
using System.Web.Routing;
using Microsoft.AspNet.FriendlyUrls;

public class Global : System.Web.HttpApplication
{
    protected void Application_Start()
    {
        RouteTable.Routes.EnableFriendlyUrls();
    }
}
```

A partir deste ponto, qualquer página pode ser chamada sem a extensão:



O sistema também responde se forem passados outros segmentos na URL. No exemplo adiante, a URL **servidor/Contato** teve adicionados os segmentos **Modo** e **Telefone**. O componente é inteligente o suficiente para verificar se existe uma pasta chamada **Contato/Modo/Telefone** e um arquivo **default.aspx** dentro dessa pasta. Se houver, esse arquivo (**default.aspx**) é retornado. Se não houver, o arquivo **Contato.aspx** é processado.



Para obter os segmentos, o componente **FriendlyUrls** disponibiliza o método de extensão **GetFriendlyUrlsSegments**, que retorna uma lista de strings com os segmentos. Por exemplo, se a URL for **Produto/Pesquisar/23**, o método **GetFriendlySegments** retorna uma lista com duas strings: **Pesquisar** e **23**.

Visual Studio 2015 - ASP.NET com C# Fundamentos

O exemplo a seguir espera que a URL seja **Contato/Modo/xxxx**, em que **xxxx** é o modo de contato (telefone, e-mail, fax, carta etc.). É feita uma verificação para ver se existem apenas dois segmentos e se o primeiro segmento é **Modo**:

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

using Microsoft.AspNet.FriendlyUrls;

public partial class Contato : System.Web.UI.Page
{
    void Page_Load(object sender, EventArgs e)
    {

        var segmentos = Request.GetFriendlyUrlSegments();

        if (segmentos.Count == 2 && segmentos[0] == "Modo")
        {
            mensagemLabel.Text = "Modo de contato:" +
                segmentos[1];
        }
    }
}
```



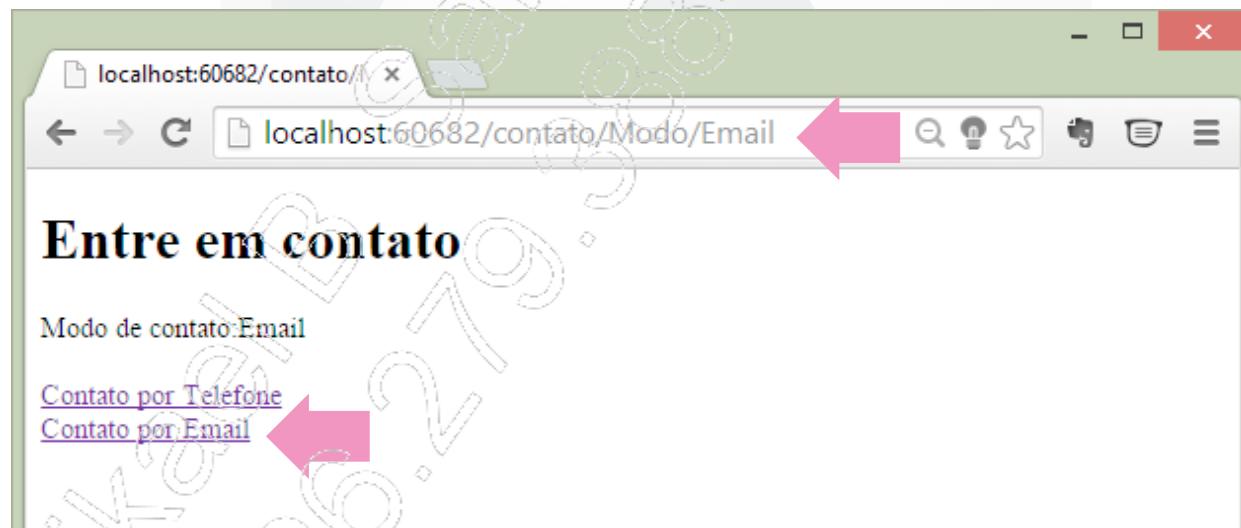
A classe **FriendlyUrl** contém o método estático **Href**, que concatena o caminho inicial com os segmentos. A sintaxe é:

```
FriendlyUrl.Href(Inicio, segmento1, segmento2, segmentos...)
```

No exemplo adiante, dois **HyperLinks** são preenchidos usando este método:

```
protected void Page_Load(object sender, EventArgs e)
{
    contatoHyperLink1.NavigateUrl =
        FriendlyUrl.Href("~/contato", "Modo", "Telefone");
    contatoHyperLink1.Text = "Contato por Telefone";

    contatoHyperLink2.NavigateUrl =
        FriendlyUrl.Href("~/contato", "Modo", "Email");
    contatoHyperLink2.Text = "Contato por Email";
}
```



Pontos principais

Atente para os tópicos a seguir. Eles devem ser estudados com muita atenção, pois representam os pontos mais importantes do capítulo.

- Dentre os recursos e bibliotecas dos modelos do .NET Framework incluídos no Visual Studio destacam-se Bundle, Minification, Bootstrap e FriendlyUrls;
- **Bundle** é o processo de agrupar diversos arquivos para serem disponibilizados como um único pacote, agilizando o tempo de carregamento da página;
- **Minification** é o processo de eliminar espaços, comentários e alterar nomes de variáveis com o intuito de diminuir o tamanho de arquivos de script e de folhas de estilos CSS;
- **Bootstrap** é uma biblioteca usada para criar aplicações Web responsivas, que se adaptem à resolução da tela e ao dispositivo que está sendo usado para a visualização;
- **FriendlyUrls** é um componente que permite usar o recurso de **Routes** para criar URLs com um formato melhorado, retirando a extensão dos arquivos ASPX.

6

Componentes

Teste seus conhecimentos

Mikael B
426.279.3557



IMPACTA
EDITORA

1. Em qual namespace se encontram as classes que realizam o processo de bundling e minimization?

- a) System.Web
- b) System.interop.runtime
- c) Microsoft.Web
- d) Microsoft.AspNet.Web.Optimization
- e) System.Web.Optimization

2. O bootstrap é dependente de quais itens?

- a) jQuery
- b) Bundle
- c) jQuery e Bubling
- d) jQuery e jSon
- e) HTML5, CSS3 e JavaScript

3. Como é renderizado um bundle que agrupa folhas de estilo?

- a) @Styles.Render
- b) @Css.Render
- c) @Script.Render
- d) @BundleTable.Render
- e) @BootStrap.Render

4. Na URL a seguir, qual método pode ser utilizado para obter o id do produto, usando o componente FriendlyUrls?

<http://servidor/produto/consultar/3>

- a) getId
- b) GetFriendlyUrlSegments
- c) GetFriendlyUrlId
- d) QueryString
- e) GET

5. Qual recurso do Visual Studio permite facilmente incluir referências a componentes externos em uma aplicação Web?

- a) Deploy
- b) Bundle
- c) NuGet
- d) Publish
- e) Build

6

Componentes

Mãos à obra!

Mikael B. Sartana
426.279.357



IMPACTA
EDITORA

Laboratório 1

A – Criando um Web site institucional simples para apresentar uma empresa, usando o modelo MVC

Neste laboratório, você criará um Web site institucional simples para apresentar uma empresa, usando o modelo MVC sem autenticação. Esse Web site terá três páginas: uma página inicial com um resumo da empresa, uma página com um perfil mais detalhado e uma página de contato, na qual o visitante preenche um formulário e os dados são gravados em um arquivo de texto.

As telas são as seguintes:

- Tela inicial



- Tela Quem Somos

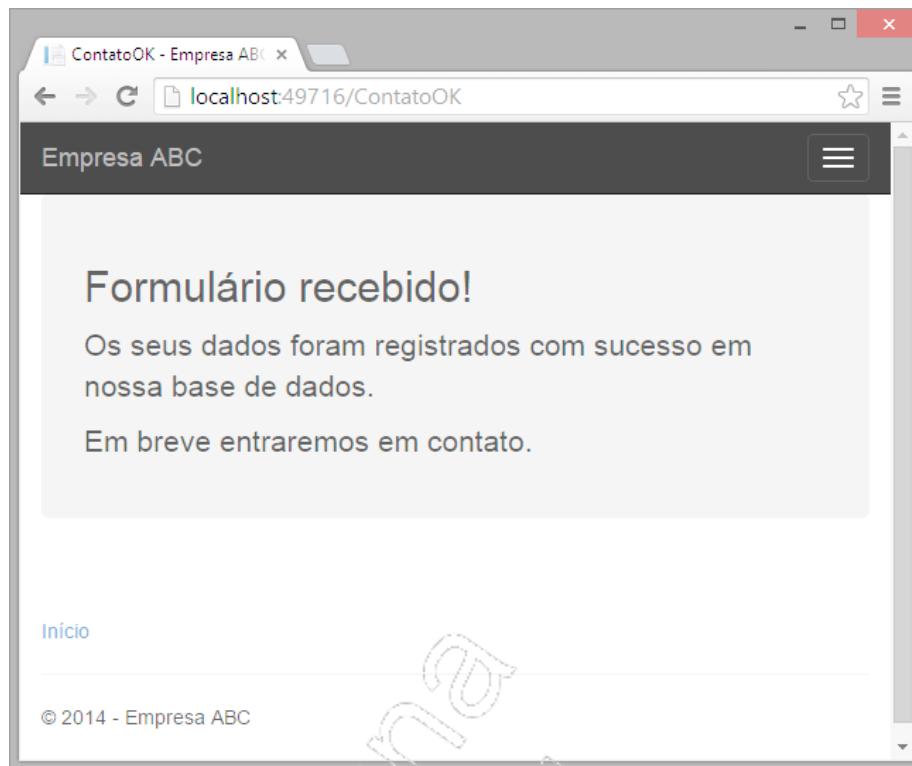


- Formulário de contato e resposta

The screenshot shows a web browser window titled 'Entre em Contato - Empresa ABC'. The URL is 'localhost:49716/Contato'. The page has a dark header with the company name 'Empresa ABC'. The main content area features a heading 'Entre em Contato' and a sub-instruction 'Por favor, preencha o formulário abaixo para entrar em contato'. Below this, there are input fields for 'Seu Nome' (with 'Harison Ford' entered) and 'Seu Email' (with 'hans.solo@star.was'). There is also a dropdown menu for 'Assunto' (selected 'Informações') and a text area for 'Mensagem' containing the text 'Quem, Quando, Como e Porque?'. At the bottom is a 'Enviar' (Send) button.

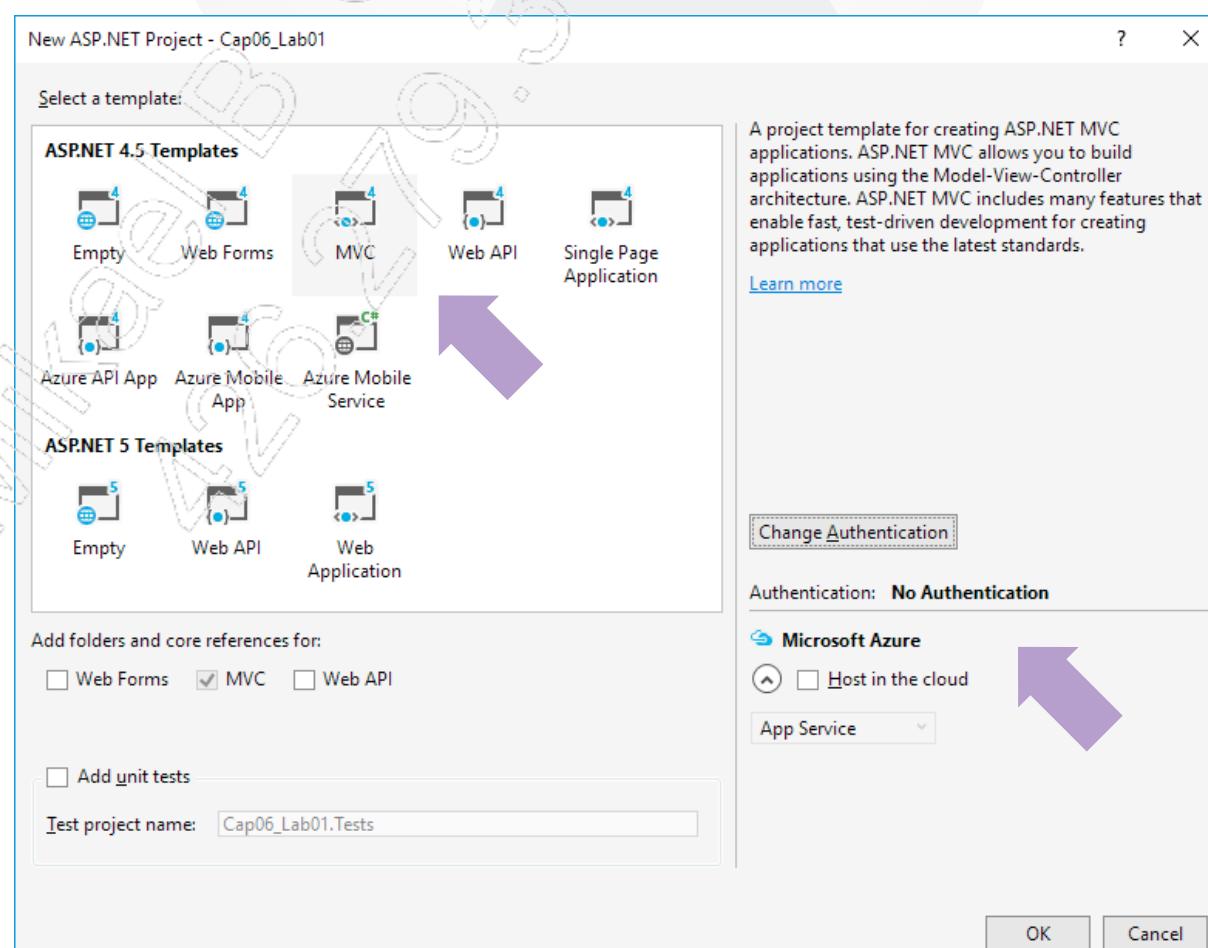
Visual Studio 2015 - ASP.NET com C# Fundamentos

- Quando os dados são digitados corretamente

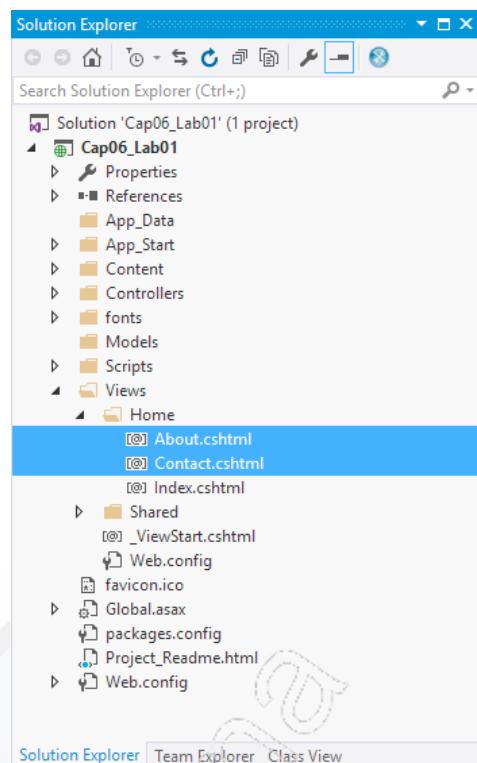


Para isso, siga os passos adiante:

1. O novo Web site terá como base o template do Visual Studio para MVC. Crie um novo projeto MVC sem autenticação, chamado **Cap06_Lab01**:



2. Exclua as Views About e Contact:



3. Exclua os métodos About e Contact do Controller Home:

```
public class HomeController : Controller
{
    public ActionResult Index()
    {
        return View();
    }

    public ActionResult About()
    {
        ViewBag.Message = "Your application description page.";

        return View();
    }

    public ActionResult Contact()
    {
        ViewBag.Message = "Your contact page.";

        return View();
    }
}
```

4. O primeiro passo é identificar cada elemento do layout e descobrir sua finalidade. Faremos isso colocando um comentário HTML(`<!-- -->`) em cima de cada elemento identificado. Abra o arquivo de layout em **Views / Shared / _Layout.cshtml**. No cabeçalho `<head>`, comente a tag do viewport. Essa tag contém informação para o bootstrap renderizar corretamente as páginas de celulares. O comando é para definir o zoom para 1.0, ou seja, sem zoom:

```
<!-- Bootstrap -->
<meta name="viewport"
      content="width=device-width, initial-scale=1.0" />
```

5. Comente o título da página. Repare que o título é a concatenação do título que é definido em cada página derivada desse layout com o nome da empresa:

```
<!-- título da Página-->
<title>@ViewBag.Title - Empresa ABC</title>
```

6. Comente o bundle de CSS e de scripts:

```
<!-- Bundle css e Modernizr -->
@Styles.Render("~/Content/css")
@Scripts.Render("~/bundles/modernizr")
```

7. Dentro da tag **Body**, comente a tag `div` que cria a navegação, crie os novos links e altere os links:

```
<!--Navegação -->

<div class="navbar navbar-inverse navbar-fixed-top">
  <div class="container">
    <div class="navbar-header">

      <button type="button" class="navbar-toggle"
             data-toggle="collapse"
             data-target=".navbar-collapse">
        <span class="icon-bar"></span>
        <span class="icon-bar"></span>
        <span class="icon-bar"></span>
      </button>
```

```
@Html.ActionLink("Empresa ABC",
    "Index", "Home",
    new { area = "" },
    new { @class = "navbar-brand" })
```

```
</div>
<div class="navbar-collapse collapse">
    <ul class="nav navbar-nav">

        <li>@Html.ActionLink("Início",
            "Index", "Home")</li>

        <li>@Html.ActionLink("Quem Somos",
            "QuemSomos", "Home")</li>

        <li>@Html.ActionLink("Entre em Contato",
            "Contato", "Home")</li>

    </ul>
    </div>

</div>
</div>
```

8. Comente a tag que guarda lugar para o corpo da página:

```
<!-- conteúdo da página-->
@RenderBody()
```

9. Comente a tag do rodapé:

```
<!-- Rodapé-->
<hr />
<footer>
    <p>&copy; @DateTime.Now.Year - Empresa ABC</p>
</footer>
```

Visual Studio 2015 - ASP.NET com C# Fundamentos

10. Comente os bundles:

```
<!-- Bundles -->
@Scripts.Render("~/bundles/jquery")
@Scripts.Render("~/bundles/bootstrap")
```

11. Comente a seção opcional:

```
<!-- seção opcional -->
@RenderSection("scripts", required: false)
```

12. Altere a View **Home/Index** para exibir dados da empresa:

```
@{
    ViewBag.Title = "Início";
}

<!-- Quadro Principal -->
<div class="jumbotron">

    <!-- Título -->
    <h1>Empresa ABC</h1>

    <!-- SubTítulo -->
    <p class="lead">
        A Empresa ABC tem como compromisso pesquisar novas tecnologias que
        facilitem o desenvolvimento de aplicações WEB
    </p>

    <!-- Saiba Mais -->
    <p>
        <a href="QuemSomos"
            class="btn btn-primary btn-lg">Saiba mais &raquo;</a>
    </p>
</div>
```

13. Execute e veja a página inicial:



14. Crie o Action **QuemSomos** no controller **Home** e, em seguida, a view com o mesmo nome. O texto pode ser livre. A seguir, temos apenas uma sugestão:

- **Controller**

```
public class HomeController : Controller
{
    public ActionResult Index()
    {
        return View();
    }

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

- **View**

```
@{  
    ViewBag.Title = "QuemSomos";  
}  
  
<h2> Empresa ABC</h2>  
<h3> Indo onde ninguém jamais esteve...</h3>  
<p> A empresa ABC tem como missão pesquisar e descobrir  
    novas tecnologias que tornem a experiência do programador  
    que cria aplicações web mais interessante e instigante.  
</p>  
<p> O número de bibliotecas, frameworks, padrões, ferramentas, protocolos  
    e modelos disponíveis para o desenvolvimento de aplicações aumenta a  
    cada ano.  
</p>  
  
<p> Nosso trabalho é testar e validar cada ferramenta que surge no mercado.  
    Cada uma é analisada quanto a sua relevância, facilidade de uso,  
    integração com padrões existentes e possibilidade de extensão e  
    adaptação.  
</p>  
  
<p> Alguns de nossos clientes:</p>  
<ul>  
    <li> Nasa Like inc.</li>  
    <li> Richard & Lumers Laws</li>  
    <li> News Tech</li>  
    <li> George Luque Inc</li>  
    <li></li>  
</ul>  
  
<p> Solicite a visita de um representante de nossa empresa sem compromisso  
    em uma visita real ou em video conferencia.</p>
```

15. Visualize a página:



16. Na página de contato, o usuário vai preencher um formulário. Para isso, vamos usar um modelo. Crie, na pasta **Models**, a classe **ContatoViewModel**:

```
public class ContatoViewModel
{
    public string Nome { get; set; }
    public string Email { get; set; }
    public string Assunto { get; set; }
    public string Mensagem { get; set; }
}
```

Visual Studio 2015 - ASP.NET com C# Fundamentos

17. Para gravar os dados, crie uma classe estática dentro de uma pasta chamada **Classes**. A classe vai se chamar **RotinasWeb**:

```
using Cap06_Lab01.Models;
using System;
using System.IO;
using System.Linq;
using System.Text;
using System.Web;

namespace Cap06_Lab01.Classes
{
    public static class RotinasWeb
    {
        public static void ContatoGravar(
            ContatoViewModel contato)
        {
            string arquivo = HttpContext
                .Current
                .Server
                .MapPath("~/App_Data/Contatos.txt");

            using (var sw = new
                StreamWriter(arquivo,
                    true, Encoding.UTF8))
            {
                sw.WriteLine(DateTime.Now);
                sw.WriteLine(contato.Nome);
                sw.WriteLine(contato.Email);
                sw.WriteLine(contato.Assunto);
                sw.WriteLine(contato.Mensagem);
                sw.WriteLine(new string('-', 30));
            }
        }
    }
}
```

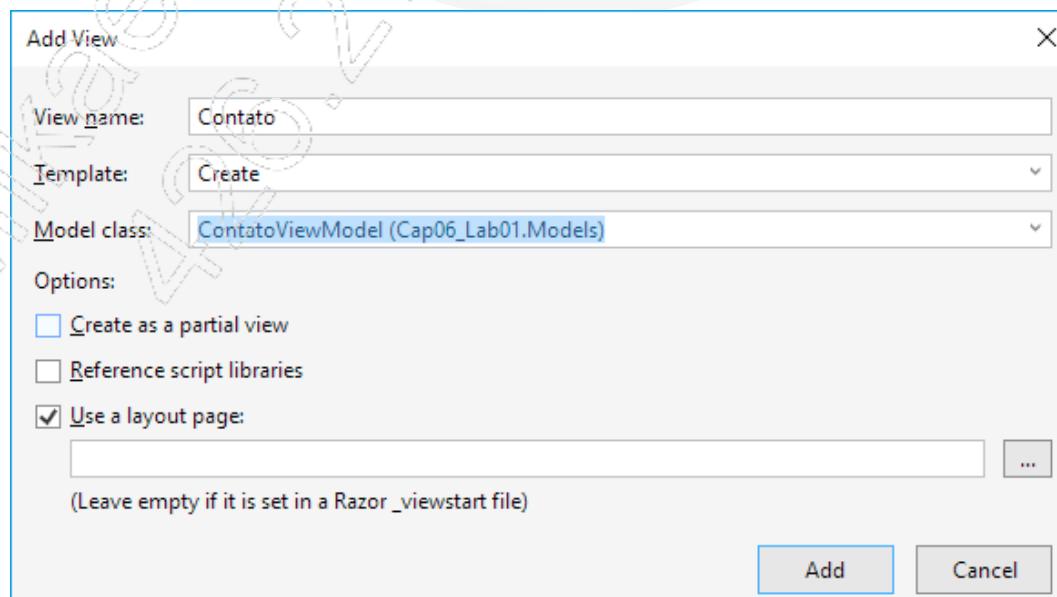
18. Crie a action (Método) **Contato** no controlador **Home**:

```
public class HomeController : Controller
{
    public ActionResult Index()
    {
        return View();
    }

    public ActionResult QuemSomos()
    {
        return View();
    }

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

19. Crie a view do método **Contato**, usando o menu de contexto de dentro do método e escolhendo o template **Create**. Não se esqueça de informar o modelo (**ContatoViewModel**);



20. Pouca coisa deve ser alterada no template criado pelo assistente. Apenas altere o título da página, o título do botão e exclua o link no final. Os métodos utilizados para criar o formulário são: `@Html.Label`, `@Html.TextBox`, `@Html.ValidationSummary`, `@HtmlBeginForm`, `@Html.AntiForgeryToken` e `@Html.ValidationMessageFor`. Esses métodos serão estudados em detalhes nos próximos módulos;

```
@model Cap06_Lab01.Models.ContatoViewModel

{@
    ViewBag.Title = "Contato";
}
@if (ViewBag.Concluido)
{
    <h2>Obrigado por seu cadastro </h2>
}

else
{
    <h2>Entre em Contato</h2>

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

        <div class="form-horizontal">

            <hr />
            @Html.ValidationSummary(...)

            <div class="form-group">
                @Html.LabelFor(model => model.Nome...)
                <div class="col-md-10">
                    @Html.EditorFor(model => model.Nome, ...)
                    @Html.ValidationMessageFor(model => model.Nome, ...)

                </div>
            </div>
    }
}
```

```
... (outros campos)

<div class="form-group">
    <div class="col-md-offset-2 col-md-10">

        <input type="submit" value="Enviar"
            class="btn btn-default" />

    </div>
</div>
</div>
}

}
```

21. Agora, escreva o método **Contato** que será executado quando houver um POST e altere o método **Contato** que será executado quando o método de chamada é GET para incluir o campo **Concluído** no objeto **ViewBag**:

```
public class HomeController : Controller
{
    public ActionResult Index()
    {
        return View();
    }

    public ActionResult QuemSomos()
    {
        return View();
    }

    public ActionResult Contato()
    {
        ViewBag.Concluido = false;
        return View();
    }

    [HttpPost]
    public ActionResult Contato(Models.ContatoViewModel contato)
    {
        try
        {
            Classes.RotinasWeb.ContatoGravar(contato);
            ViewBag.Concluido = true;
        }
        catch (Exception ex)
        {
            ModelState.AddModelError("", ex.Message);
        }
        return View();
    }
}
```

Visual Studio 2015 - ASP.NET com C# Fundamentos

22. Teste o programa. Verifique se o arquivo de texto foi gerado corretamente.

The screenshot shows a web browser window titled "Contato - Empresa ABC". The address bar displays "localhost:56503/Home/Contato". The page header includes the company name "Empresa ABC" and navigation links for "Início", "Quem Somos", and "Entre em Contato". The main content area is titled "Entre em Contato" and "ContatoViewModel". It contains four input fields labeled "Nome", "Email", "Assunto", and "Mensagem", followed by a "Enviar" button. At the bottom, there is a copyright notice: "© 2016 - Empresa ABC". A large watermark reading "Santana 57" is diagonally across the page.



Sugestão para estudo:

- Criar o mesmo projeto usando Web Forms;
- Criar uma página com uma lista de produtos;
- Gerar um erro (throw...) se houver tentativa de inserir uma informação inválida.

7

Acesso a dados ADO.NET

- ✓ Providers;
- ✓ Principais classes de acesso a dados;
- ✓ Classes desconectadas;
- ✓ Conectando;
- ✓ Enviando comandos ao banco de dados;
- ✓ Lendo informações do banco;
- ✓ Exibindo informações.



IMPACTA
EDITORA

7.1. Introdução

O ASP.NET oferece diversas maneiras de conectar, consultar e manipular informações de banco de dados. Todas elas usando, na sua estrutura básica, as classes do .NET Framework de acesso a dados.

O conjunto de recursos disponíveis para manipulação de quaisquer fontes de dados é chamado ADO.NET. Neste capítulo, vamos ver como utilizar as classes do ADO.NET em páginas ASP.NET.

A arquitetura básica do ADO.NET é composta de **providers**, que são classes que fornecem acesso a fontes de dados, e de **consumers**, que são classes que utilizam as informações fornecidas pelos providers.

Uma fonte de dados pode ser um banco de dados SQL Server, Oracle, MySQL, arquivos XML, arquivos CSV, planilhas do Excel ou qualquer tipo de informação organizada.

7.2. Providers

O .NET Framework oferece quatro providers dentro da biblioteca de classes padrão:

Provider	Namespace	Destino
SQL Server	System.Data.SqlClient	Banco de dados SQL Server.
Oracle	System.Data.OracleClient	Banco de dados Oracle.
OLE DB	System.Data.OleDbClient	Qualquer banco com driver para o componente OLE DB.
ODBC	System.Data.ODBC	Qualquer banco com driver para o componente ODBC.

7.3. Principais classes de acesso a dados

Todos os providers implementam classes que herdam as classes ou interfaces do namespace **System.Data.Common**. Isso significa que a classe utilizada de qualquer provider terá as mesmas propriedades, métodos e eventos básicos de todos os outros. Não há diferença na programação de um provider ou de outro.

Classe base	Finalidade
DbConnection	Conexão com o banco de dados.
DbCommand	Execução de comandos.
DbDataReader	Leitura de dados.
DbDataAdapter	Transferência e sincronização de dados.

O .NET Provider para SQL Server, no namespace **System.Data.SqlClient**, fornece as seguintes classes derivadas das classes bases:

Classe	Classe base
SqlConnection	DbConnection
SqlCommand	DbCommand
SqlDataReader	DbDataReader
SqlDataAdapter	DbDataAdapter

O .NET Provider para OLE DB, no namespace **System.Data.SqlClient**, fornece as seguintes classes derivadas das classes base:

Classe	Classe base
OleDbConnection	DbConnection
OleDbCommand	DbCommand
OleDbDataReader	DbDataReader
OleDbDataAdapter	DbDataAdapter

O mesmo processo serve para qualquer outra classe de dados, mesmo que não faça parte do .NET Framework. A classe para conexão com MySQL, por exemplo, pode ser obtida pelo site do banco de dados MySQL. A implementação é exatamente a mesma: **MySqlConnection**, **MySqlCommand**, **MySqlDataReader** e **MySqlDataAdapter**.

7.4. Classes desconectadas

O namespace **System.Data** contém um conjunto de classes desenvolvidas para armazenar, em memória, informações dos bancos de dados. Essas classes podem ser usadas para visualizar ou modificar dados no banco de dados original. Elas representam exatamente a estrutura de um banco de dados relacional.

As principais classes desse namespace são **DataSet**, **DataTable**, **DataRow** e **DataColumn**, que representam, respectivamente, um banco de dados, uma tabela, uma linha e uma coluna desse banco. A classe **XXXDataAdapter** tem a finalidade de servir de ponte entre as classes que se conectam no banco e as desconectadas.

Classe	Objetivo
DataSet	Armazenar informações sobre tabelas e relacionamentos.
DataTable	Armazenar informações sobre uma tabela.
DataColumn	Armazenar informações sobre um campo.
DataRow	Armazenar informações sobre um registro.

As classes desconectadas apresentadas aqui não são as únicas disponíveis para armazenar informações de banco de dados. Os dados de um aplicativo podem ser armazenados em classes especialmente criadas para cada situação. Esse processo se chama **modelagem** e pode ser feito manualmente ou automaticamente usando outras tecnologias, conceitos e ferramentas, como LINQ ou Entity Framework.

7.5. Conectando

A classe **Connection** contém uma propriedade chamada **ConnectionString** para definir a conexão com o banco. Uma vez criada a conexão, é possível abrir e fechar a conexão com os métodos **Open()** e **Close()**, respectivamente.

Vejamos, a seguir, um exemplo no qual é estabelecida uma conexão com um banco de dados do tipo SQL Server:

```
//String de conexão
string conexao = @"Data Source=localhost\sqlexpress;
                    Initial Catalog=loja;
                    Integrated Security=true";

//Conexão
var cn = new SqlConnection();
cn.ConnectionString = conexao;

//Conectar
cn.Open();

//Fechar a conexão
cn.Close();
```

7.6. Enviando comandos ao banco de dados

O envio de comandos ao banco de dados pode ser feito assim que a conexão a este for estabelecida. A classe **Command** precisa de duas informações: o comando a ser executado e a conexão a ser estabelecida. A propriedade **Connection** define a conexão, e a propriedade **CommandText** define o comando a ser executado. Se o comando contiver parâmetros, a propriedade **Parameters**, que é uma coleção, deverá ser preenchida. O método **AddWithValue** é uma maneira rápida de inserir parâmetros apenas passando o nome do parâmetro (uma string) e o valor (objeto do tipo correto).

Visual Studio 2015 - ASP.NET com C# Fundamentos

Veja, a seguir, um exemplo de inclusão de dados:

```
private void Incluir(string nome, string email)
{
    //String de conexão
    string conexao = @"Data Source=localhost\sqlexpress;
                        Initial Catalog=loja;
                        Integrated Security=true";
    //Comando a ser executado
    string sql = @"INSERT INTO Clientes(Nome,Email)
                    Values(@Nome,@Email)";

    //Conexão
    var cn = new SqlConnection();
    cn.ConnectionString = conexao;

    //Comando
    var cmd = new SqlCommand();
    cmd.CommandText = sql;
    cmd.Connection = cn;
    cmd.Parameters.AddWithValue("@nome", nome);
    cmd.Parameters.AddWithValue("@email", email);

    //Executar
    cn.Open();
    cmd.ExecuteNonQuery();
    cn.Close();
}
```

Veja, agora, um exemplo de alteração de dados:

```
private void Alterar(int clienteId, string nome, string email)
{
    //String de conexão
    string conexao = @"Data Source=localhost\sqlexpress;
                        Initial Catalog=loja;
                        Integrated Security=true";
    //Comando a ser executado
    string sql = @"UPDATE Clientes
                    SET(Nome=@Nome,Email=@Email)
                    WHERE ClienteId=@ClienteId";

    //Conexão
    var cn = new SqlConnection();
    cn.ConnectionString = conexao;

    //Comando
    var cmd = new SqlCommand();
    cmd.CommandText = sql;
    cmd.Connection = cn;
    cmd.Parameters.AddWithValue("@nome", nome);
    cmd.Parameters.AddWithValue("@email", email);
    cmd.Parameters.AddWithValue("@clienteId", clienteId);

    //Executar
    cn.Open();
    cmd.ExecuteNonQuery();
    cn.Close();
}
```

Veja, também, um exemplo de exclusão de dados:

```
private void Excluir(int clienteId)
{
    //String de conexão
    string conexao = @"Data Source=localhost\sqlexpress;
                      Initial Catalog=loja;
                      Integrated Security=true";

    //Comando a ser executado
    string sql = @"DELETE FROM Clientes
                   WHERE ClienteId=@ClienteId";

    //Conexão
    var cn = new SqlConnection();
    cn.ConnectionString = conexao;

    //Comando
    var cmd = new SqlCommand();
    cmd.CommandText = sql;
    cmd.Connection = cn;
    cmd.Parameters.AddWithValue("@clienteId", clienteId);

    //Executar
    cn.Open();
    cmd.ExecuteNonQuery();
    cn.Close();
}
```

7.7. Lendo informações do banco

Para que possamos manipular os resultados obtidos, o ADO.NET dispõe de dois modos, a saber:

- A classe **DbDataReader**;
- A classe **DataSet**.

7.7.1. DbDataReader

O acesso aos dados é realizado uma linha por vez, sem a possibilidade de retorno à linha anterior e sem a possibilidade de alteração. O banco de dados envia os dados usando um stream de dados, de maneira muito rápida e eficiente. No exemplo adiante, os dados lidos de um DbDataReader são armazenados em uma coleção de strings:

```
public List<string> CidadeLista()
{
    //Retorno
    var lista = new List<string>();

    //Comando a ser executado
    string sql = "SELECT Nome FROM Cidades ";

    //string de Conexão
    string conexao = @"Data Source=localhost; Initial Catalog=EmpresaDb;
        Integrated Security=true";
    //Connection
    var cn = new SqlConnection();
    cn.ConnectionString = conexao;

    //Command
    var cmd = new SqlCommand();
    cmd.CommandText = sql;
    cmd.Connection = cn;

    //Abre a conexão;
    cn.Open();

    //Obtém uma instância do DataReader
    SqlDataReader reader = cmd.ExecuteReader();

    //Método Read() lê um registro por vez e retorna true se existem dados
    while (reader.Read())
    {
        //Para obter os dados, a propriedade [this] do DataReader é utilizada
        string nomeCidade = reader["Nome"].ToString();

        //adiciona na lista
        lista.Add(nomeCidade);
    }

    //Fecha
    reader.Close();
    cn.Close();

    //Retorna a lista
    return lista;
}
```

7.7.2. DataSet

A classe **DataSet** armazena informações na memória utilizando um modelo parecido com o que é utilizado por um banco de dados relacional, ou seja, utiliza tabelas, linhas, colunas, chaves primárias, relacionamentos e visualizações. As classes para armazenar esse tipo de informação estão no namespace **System.Data**. A lista a seguir apresenta as principais dessas classes:

Classe	Objetivo
DataSet	Representa uma coleção de tabelas e relacionamentos.
DataTable	Representa uma tabela.
 DataColumn	Representa uma coluna de uma tabela.
 DataRow	Representa uma linha de dados de uma tabela.
 DataView	Representa uma visualização de uma tabela filtrada e ordenada.

As principais propriedades e métodos das classes **DataSet** e **DataTable** são as seguintes:

- **DataSet**
 - **Tables[]**: Coleção de **DataTable**;
 - **WriteXml()**: Método para gravar dados em formato XML;
 - **ReadXml()**: Método para ler dados de um arquivo XML.
- **DataTable**
 - **Columns[]**: Coleção de colunas;
 - **Rows[]**: Coleção de linhas.

É possível preencher um DataSet ou DataTable manualmente ou importar de um banco de dados. Para importar de um banco de dados existe a classe do ADO.NET chamada **DbDataAdapter**, que utiliza internamente as classes tradicionais para acesso a dados (**DbConnection**, **DbCommand** e **DbDataReader**), mas o processo é todo encapsulado. O exemplo adiante preenche um DataTable com informações do banco:

```
public DataTable ProdutoLista()
{
    //Retorno
    var tabela = new DataTable();

    //Comando a ser executado
    string sql = "SELECT Id, Nome FROM Produto ";

    //string de Conexão
    string conexao = @"Data Source=localhost;
        Initial Catalog=EmpresaDb;
        Integrated Security=true;";

    //DbDataAdapter
    var da = new SqlDataAdapter(sql, conexao);

    //Abre a conexão com o banco, percorre os
    //dados, preenche a tabela e fecha a conexão
    da.Fill(tabela);

    //Retorna a lista
    return tabela;
}
```

Internamente, o **DbDataAdapter** utiliza a propriedade **SelectCommand** para preencher os dados. Essa propriedade é definida quando são informadas, no construtor, a expressão SQL e a string de conexão. Uma vez criada essa instância da classe **DbCommand**, é possível utilizar o método **Fill** para preencher um DataTable ou DataSet.

7.8. Exibindo informações

Os modelos ASP.NET utilizam diferentes técnicas para vincular dados de uma fonte com o componente que exibe as informações.

Modelo	Forma de vínculo
Web Forms	Os Web Forms utilizam propriedades dos Web Controls. A propriedade DataSource define a fonte de dados, e o método DataBind() cria o HTML que exibe os dados. A fonte de dados pode ser uma instância da classe DataReader , uma instância da classe DataTable ou uma coleção fortemente tipada, ou seja, uma coleção de objetos com propriedades públicas definidas.
MVC	O MVC não utiliza Web Controls. As Views podem receber uma instância de uma coleção e utilizar esses dados para criar o HTML.
Web Pages	As Web Pages contêm uma classe com métodos estáticos chamada DataBase que permite executar comandos diretamente no banco de dados. Os dados podem ser construídos manualmente ou utilizar alguns componentes, como Grid() , que geram HTML automaticamente.

7.8.1. Web Forms: Usando Web Controls para exibir dados

Nos próximos subtópicos, vamos ver como exibir dados usando os controles **GridView** e **DropDownList**.

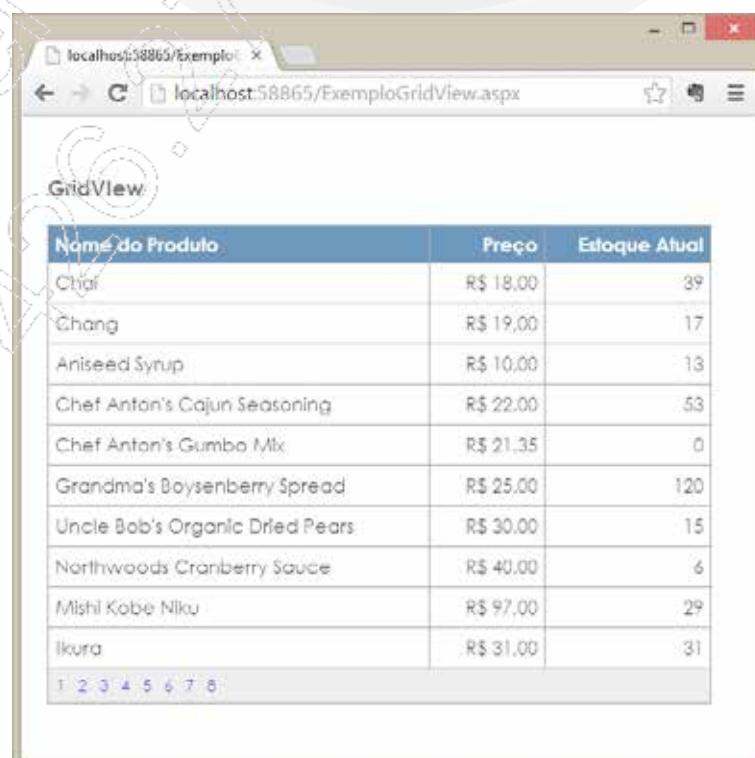
7.8.1.1. GridView

O controle **GridView** exibe dados em formato de colunas e linhas.

Veja quais são as principais propriedades e métodos deste controle:

- **DataSource**: A fonte de dados;
- **DataBind()**: Inicia o processo de vínculo de dados;
- **AllowPage**: Ativa o recurso de paginação;
- **PageSize**: Número de registros por página.
- **Code-behind**:

```
GridView1.AllowPage= true;  
GridView1.DataSource = ObterDados();  
GridView1.DataBind();
```



A screenshot of a Microsoft Internet Explorer browser window. The address bar shows 'localhost:58865/ExemploGridView.aspx'. The main content area is titled 'GridView' and displays a table with three columns: 'Nome do Produto', 'Preço', and 'Estoque Atual'. The table contains 12 rows of data. At the bottom of the table, there is a navigation bar with links labeled 1, 2, 3, 4, 5, 6, 7, 8.

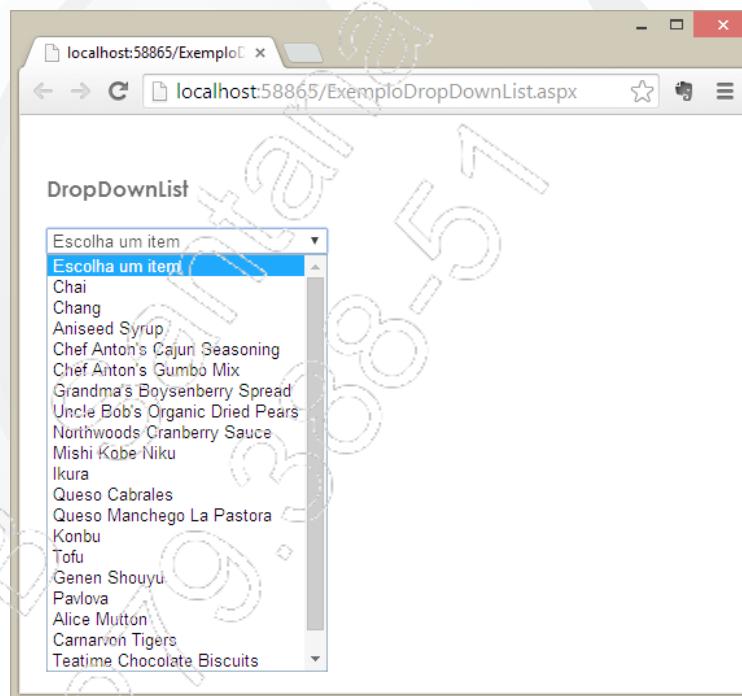
Nome do Produto	Preço	Estoque Atual
Chai	R\$ 18,00	39
Chang	R\$ 19,00	17
Aniseed Syrup	R\$ 10,00	13
Chef Anton's Cajun Seasoning	R\$ 22,00	53
Chef Anton's Gumbo Mix	R\$ 21,35	0
Grandma's Boysenberry Spread	R\$ 25,00	120
Uncle Bob's Organic Dried Pears	R\$ 30,00	15
Northwoods Cranberry Sauce	R\$ 40,00	6
Mishi Kobe Niku	R\$ 97,00	29
Ikura	R\$ 31,00	31

7.8.1.2.DropDownList

Exibe um item e uma lista ao ser clicado. Neste tipo de Web Control, é necessário informar o campo que será exibido (**DataTextField**) e o campo que será retornado (**DataValueField**).

Veja um exemplo:

```
DownList.DataTextField="Nome";
DownList.DataValueField="Codigo";
DownList.DataSource = ObterDados();
DownList.DataBind();
```



- Para obter o valor retornado:

```
string valor=DropDownList1.SelectedValue;
```

7.8.2. MVC: Usando Views para exibir dados

No formato MVC, uma View recebe um tipo que pode ser um item ou uma coleção de dados e os dados são vinculados usando esta instância:

- **Model:**

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

- **Controller:**

```
public class HomeController : Controller
{
    public ActionResult Cidades()
    {
        var lista=new List<Cidade>();
        lista.Add(new Cidade() { Id = 1, Nome = "São Paulo" });
        lista.Add(new Cidade() { Id = 2, Nome = "Rio de Janeiro" });
        return View(lista);
    }
}
```

- **View:**

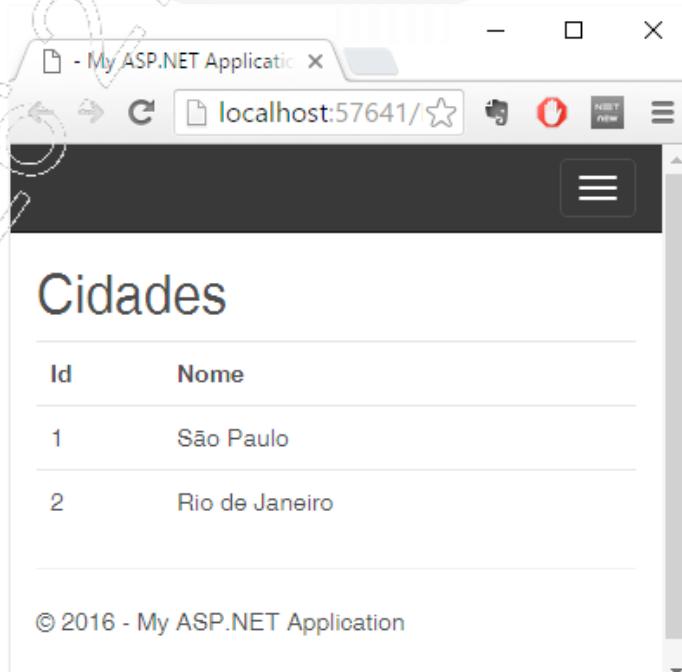
```
@model List<EmptyMVC.Models.Cidade>

<h2>Cidades</h2>



| Id       | Nome       |
|----------|------------|
| @item.Id | @item.Nome |


```

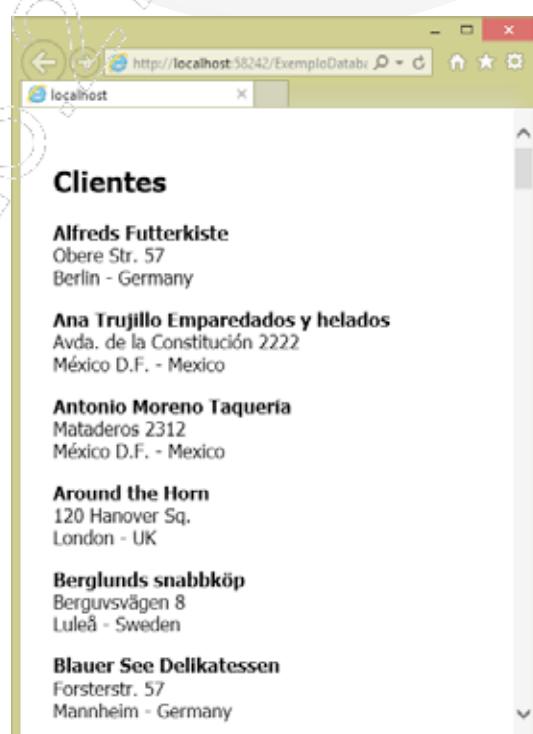


7.8.3. Web Pages: Usando Database para exibir dados

A classe **Database** está disponibilizada na biblioteca **WebMatrix.Data** e fornece comandos para conectar um banco de dados, executar comandos e ler os resultados. Veja um exemplo a seguir:

```
@using WebMatrix.Data  
{@  
    var db = Database.Open("northwind");  
    var sql = @"SELECT CompanyName, Address, City, Country  
              FROM CUSTOMERS";  
    var clientes = db.Query(sql).ToList();  
}  
>  
<html>  
  
    <body>  
        <h2>Clientes</h2>  
        @foreach (var cliente in clientes)  
        {  
            <p>  
                <strong>@cliente.CompanyName</strong><br />  
                @cliente.Address <br />  
                @cliente.City  
                -  
                @cliente.Country  
            </p>  
        }  
    </body>  
</html>
```

Ao executá-lo, os dados são extraídos do banco e exibidos:



Pontos principais

Atente para os tópicos a seguir. Eles devem ser estudados com muita atenção, pois representam os pontos mais importantes do capítulo.

- O ADO.NET é o nome dado às classes do .NET Framework usadas para manipular dados;
- As classes principais conectadas são **DbConnection**, **DbCommand**, **DbDataReader** e **DbDataAdapter**;
- As classes principais desconectadas são **DataSet**, **DataTable**, **DataRow** e **DataColumn**;
- As ações realizadas consistem em estabelecer conexão com o banco de dados, enviar comandos ao banco de dados e manipular os resultados obtidos;
- Para vincular controles que apresentam colunas em Web Forms, a propriedade **DataSource** é a única que deve ser informada. Para controles que mostram uma única coluna, as propriedades **TextField** e **ValueField** devem ser informadas;
- O Web Control só renderiza o HTML quando é executado o método **DataBind()**;
- O MVC utiliza o modelo de View tipada para exibir dados;
- As Web Pages contêm a classe **DataBase**, que permite conectar e obter dados diretamente de um provider.

7

Acesso a dados ADO.NET

Teste seus conhecimentos

Mikael
Baptana
426.2
57



IMPACTA
EDITORA

1. Em que namespace são encontradas as classes de acesso a dados para o banco de dados SQL Server?

- a) System.Data
- b) System.Common
- c) System.Data.SqlServer
- d) System.Data.SqlClient
- e) Todas as alternativas anteriores estão corretas.

2. Qual classe é usada para ler informações do banco?

- a) SqlConnection
- b) SqlDataReader
- c) SqlParameter
- d) DataSet
- e) SqlConnection

3. Qual é o método que deve ser chamado, em um Web Form, para que um controle vinculado gere o código HTML?

- a) Render()
- b) Go()
- c) DataSet()
- d) Open()
- e) DataBind()

4. No modelo MVC, como se processa a vinculação de dados?

- a) DataList
- b) DataBind
- c) View tipada
- d) Web Control tipado
- e) Não é possível exibir dados no modelo MVC.

5. Em uma listagem de milhões de registros, qual componente ADO.NET é recomendado?

- a) DataSet
- b) DataTable
- c) DbDataReader
- d) DbCommand
- e) DbList

7

Acesso a dados ADO.NET

Mãos à obra!

Mikael Braga
426.271-5767



IMPACTA
EDITORA

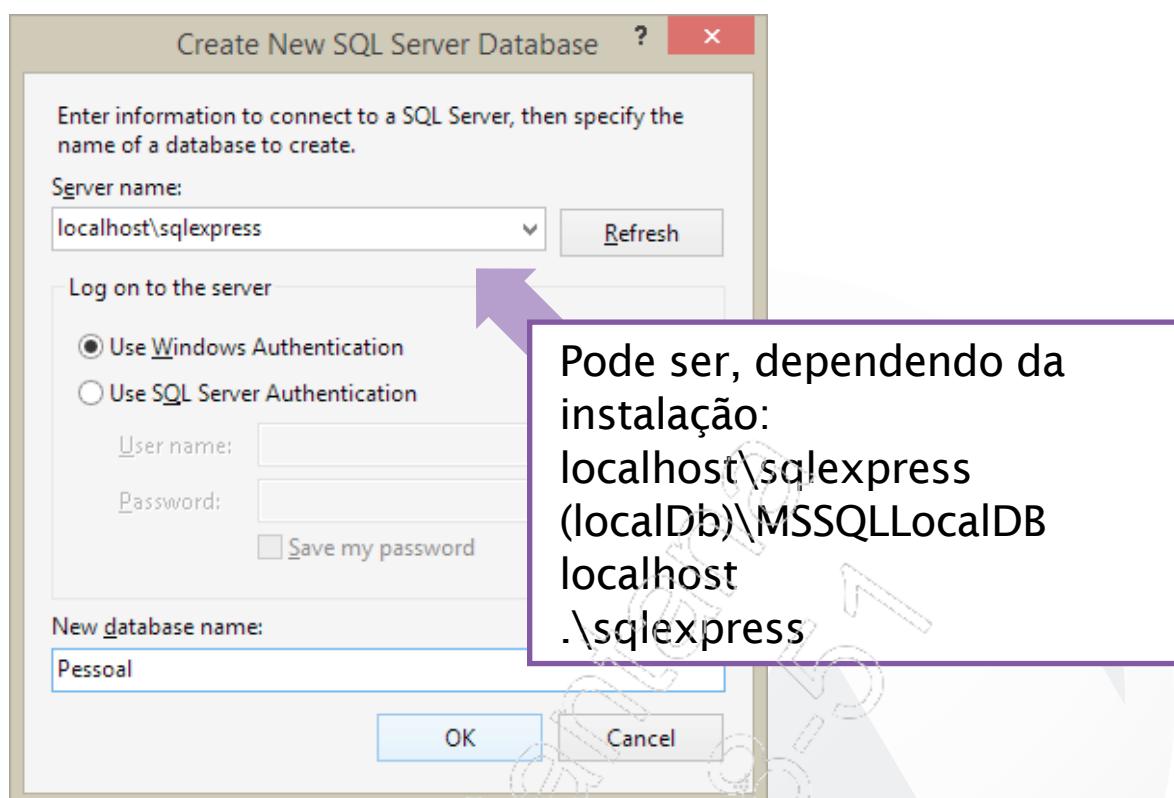
Laboratório 1

Objetivos:

- Criar um programa que pesquise, insira, exclua e altere dados em uma tabela de um banco SQL Server;
- Criar um banco de dados usando o Visual Studio;
- Criar, alterar e visualizar tabela de um banco de dados usando as ferramentas do Visual Studio;
- Usar classes do ADO.NET para conectar o banco de dados, visualizar, alterar e excluir informações;
- Usar GridView para visualizar dados de uma tabela;
- Editar dados usando formulários.

1. Crie um novo projeto Web vazio chamado **Cap07Lab01**;
2. No Server Explorer, abra o menu de contexto, escolha o item **Data Connections** e a opção **Create New Sql Server Database**. O nome do servidor é **(LocalDB)\MSSQLLocalDB** e o banco de dados se chama **Pessoal**. Se o nome do servidor não for reconhecido é porque o seu computador está com uma versão anterior do SQL Server e, então, o nome será **localhost\sqlexpress**;

Se outra pessoa instalou o SQL Server ou se você está trabalhando em um ambiente empresarial, é aconselhável entrar em contato com o administrador da rede para saber qual banco de dados pode ser usado para testes e qual o endereço na rede.

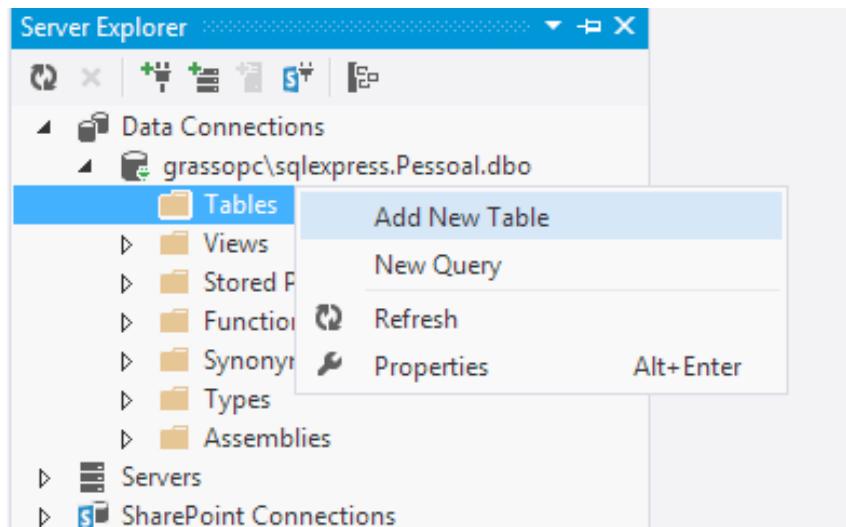


Esse banco de dados vai armazenar coisas como e-mails de conhecidos, contas a pagar, tarefas, coleção de MP3, filmes assistidos ou qualquer informação para uso pessoal.

Neste laboratório, vai ser criada uma **Lista de tarefas**. Essa lista vai registrar o nome da tarefa, a prioridade de 1 a 3 (sendo 1 a mais importante), se foi concluída e observações.

Visual Studio 2015 - ASP.NET com C# Fundamentos

3. No menu de contexto do banco criado, dentro da pasta **Tables**, selecione **Add New Table**:



4. Defina os campos, conforme a imagem a seguir. Os campos são os seguintes:

- **Id**: Identificador numérico do tipo **identity**;
- **Nome**: A descrição da tarefa, por exemplo, **Estudar ASP.NET**;
- **Prioridade**: Número de 1 a 3;
- **Concluída**: Campo **Sim** ou **Não**;
- **Observações**: Texto opcional.

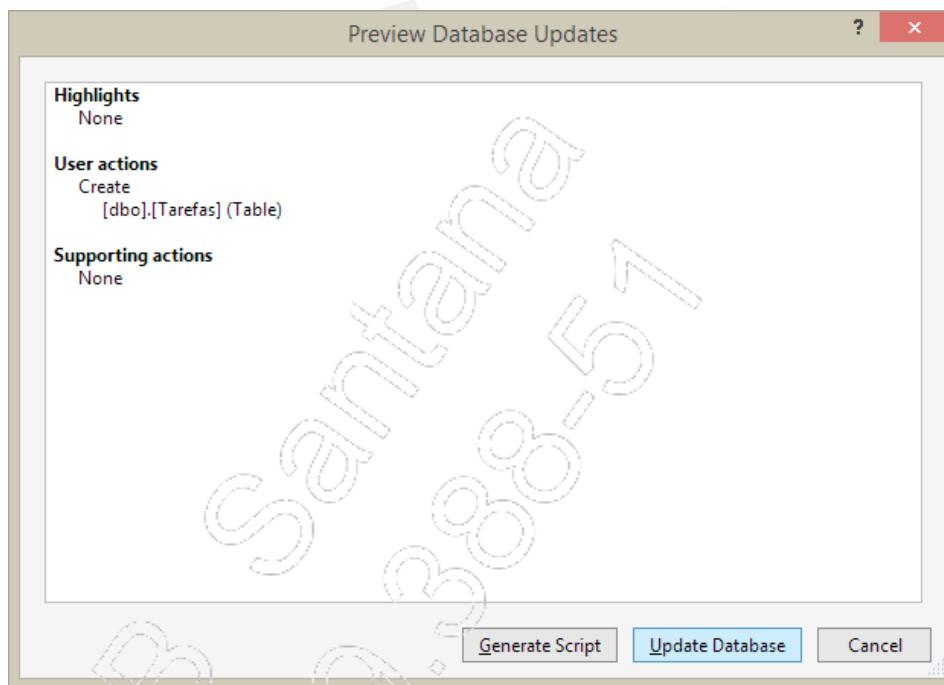
A screenshot of the 'dbo.Tarefas (Design)' table in SQL Server Management Studio. The 'Name' column lists the fields: Id, Nome, Prioridade, Concluida, and Observacoes. The 'Data Type' column specifies: int, varchar(100), int, bit, and varchar(MAX) respectively. The 'Allow Nulls' column has checkboxes for all fields except Observacoes, which is checked. On the right, there are sections for 'Keys (1)', 'Check Constraints (0)', 'Indexes (0)', 'Foreign Keys (0)', and 'Triggers (0)'. Below the table definition, the T-SQL code for creating the table is shown:

```
CREATE TABLE Tarefas
(
    [Id] INT NOT NULL IDENTITY PRIMARY KEY,
    Nome VARCHAR(100) NOT NULL,
    Prioridade INT NOT NULL,
    Concluida BIT NOT NULL,
    Observacoes VARCHAR(MAX)
)
```

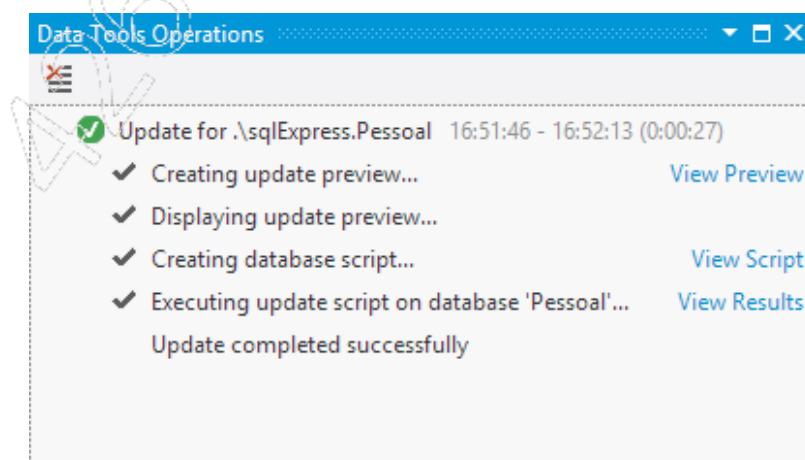
5. Clique em **Update** para criar a tabela:

Name	Data Type	Allow Nulls	Key
Id	int	<input type="checkbox"/>	Primary
Nome	varchar(100)	<input type="checkbox"/>	Candidate
Prioridade	int	<input type="checkbox"/>	Candidate

6. Clique em **Update Database** para confirmar a criação da tabela;

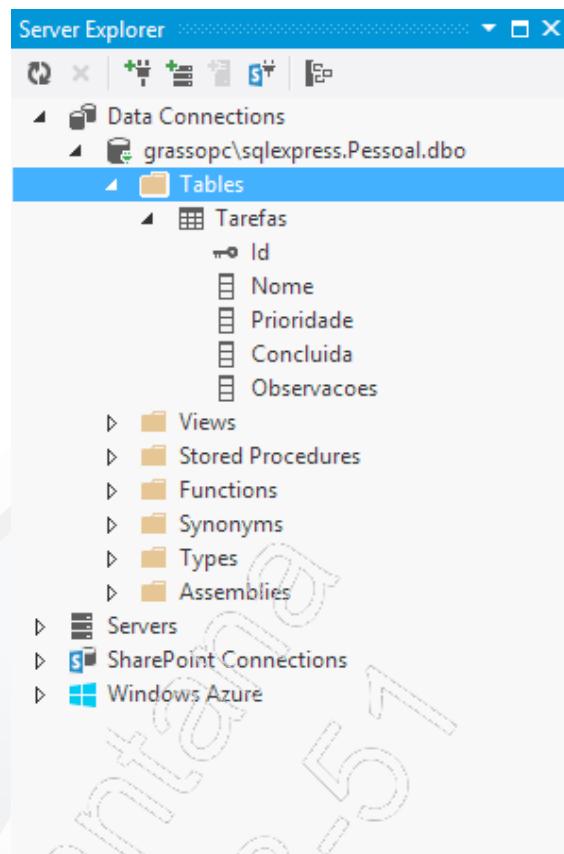


7. Verifique se foi gravado com sucesso:



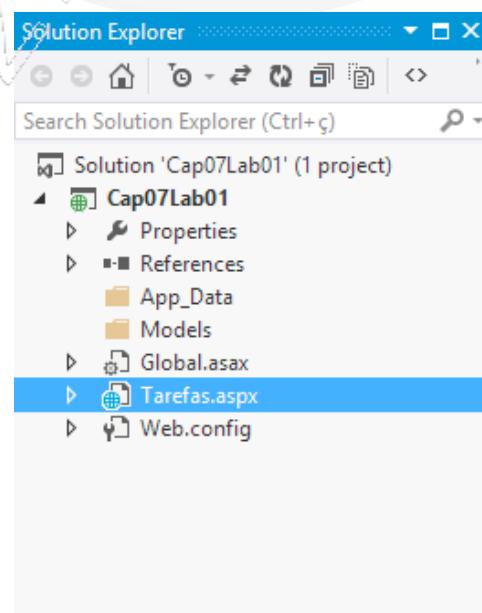
Visual Studio 2015 - ASP.NET com C# Fundamentos

8. Verifique se a tabela aparece no **Server Explorer**. Se não aparecer, use o menu de contexto na pasta **Tables** e escolha **Refresh**;



- Criando a interface de usuário:

9. Uma única tela vai listar as tarefas e atualizar os dados. Não é necessária uma Master Page. Insira no projeto um Web Form chamado **Tarefas.aspx**;



10. O layout HTML tem as divisões **Página**, **Cabeçalho**, **Conteúdo** e **Rodapé**. Insira-os na página;

```
<div id="pagina">

    <div id="cabecalho">
        <h1>Tarefas</h1>
    </div>

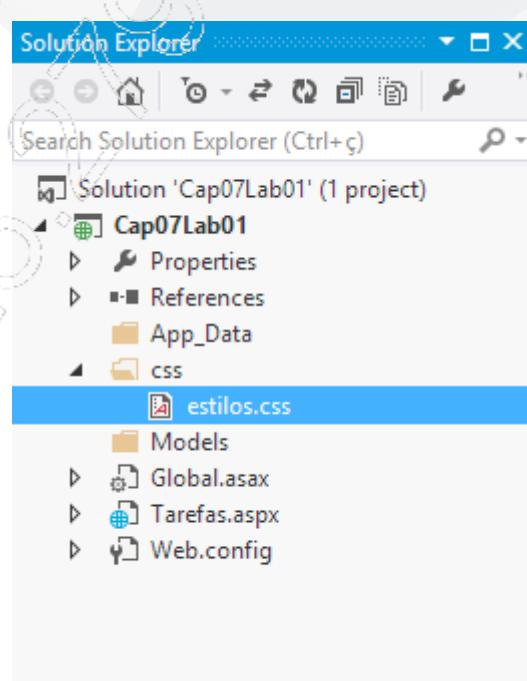
    <div id="conteudo">

    </div>

    <div id="rodape">
        <p>
            Desenvolvido para estudar ASP.NET (c)2014 - Me
        </p>
    </div>

</div>
```

11. Crie uma pasta (**Add / New Folder**) chamada **css** e, dentro dela, uma folha de estilos chamada **estilos.css**. Associe essa página à página **Tarefas.aspx**;



Visual Studio 2015 - ASP.NET com C# Fundamentos

Folhas de estilo têm suporte para mobile. Usar design responsivo.

```
* {  
    margin:0px;  
}  
  
body {  
    padding:0px;  
    margin:0px;  
    background-color:#2e5069;  
    font-family:'Century Gothic', Tahoma, Arial;  
}  
  
#pagina  
{  
    width:700px;  
    max-width:80%;  
    margin: 20px auto;  
}  
  
#cabecalho {  
    background-color:#fff;  
    border-radius:10px;  
}  
#cabecalho h1 {  
    color:#2e5069;  
    padding:10px;  
}  
  
#conteudo {  
    background-color:#fff;  
    min-height:200px;  
    margin-top:10px;  
    border-radius:10px;  
    padding:20px;  
}  
  
.gridView {  
    margin-bottom:20px;  
    font-size:90%;  
    width:100%;  
}  
  
.gridView tr:hover {  
    background-color:#dde6f3;  
}
```

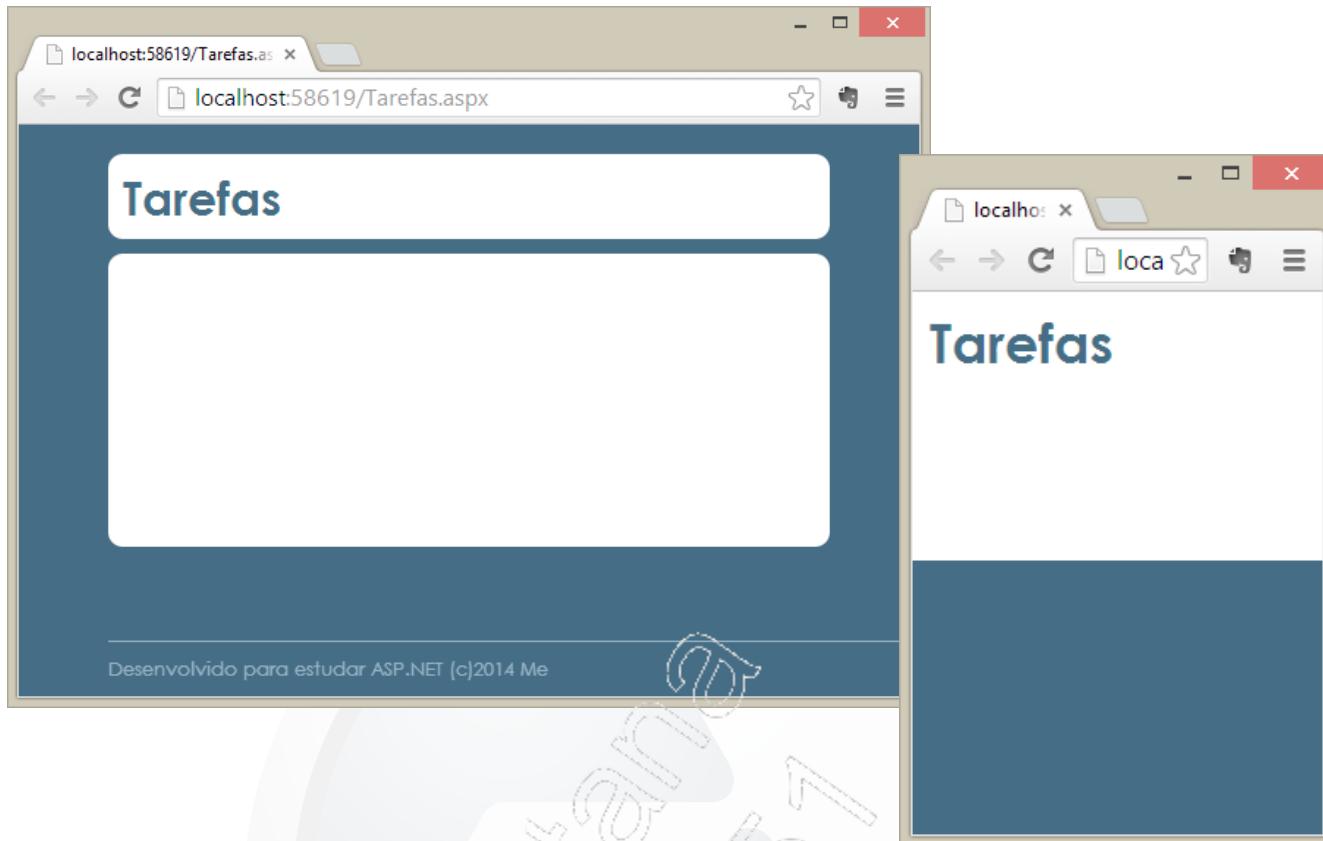
```
.gridView td {  
    padding:5px;  
    border-color:#ccc;  
}  
  
.gridView td a {  
    text-decoration:none;  
}  
  
.gridView th {  
    padding:5px;  
    border-color:#ccc;  
    color:#272944;  
}  
  
}  
  
.colConcluidaItem{  
    text-align:center;  
    width:3%;  
}  
.  
.colConcluidaHeader{  
    width:3%;  
}  
.  
.colPrioridadeItem{  
    text-align:center;  
    width:3%;  
}  
.  
.colPrioridadeHeader{  
    text-align:center;  
    width:3%;  
}  
.  
.colTarefaItem{  
    text-align:left;  
}  
.  
.colTarefaHeader{  
    text-align:left;  
}
```

```
#rodape {  
    font-size:80%;  
    position:fixed;  
    bottom:0px;  
    color:#ceeaff;  
    width:700px;  
    border-top:1px solid #ceeaff;  
    opacity:0.5;  
  
}  
#rodape p {  
    padding:10px 0px 10px 0px;  
    text-align:left;  
}  
fieldset {  
    padding:20px;  
    border:1px solid #ccc;  
    border-radius:10px;  
    margin-bottom:10px;  
}  
    fieldset legend {  
        padding:5px;  
    }  
fieldset .legenda, .fieldset .campo {  
    display:block;  
}  
    fieldset .legenda {  
        cursor:pointer;  
    }  
fieldset .campo {  
    margin-bottom:12px;  
    width:100%;  
}  
fieldset .obs {  
    min-height:200px;  
}  
.larguraMax80 { max-width:80px; }  
  
@media (max-width:400px) {  
#pagina {  
    width:inherit;  
    max-width:100%;  
    margin:0px;  
    border-radius:0px;  
    border:none;  
}
```

```
#cabecalho {  
    border-radius: 0px;  
    margin:0px;  
}  
  
#conteudo {  
border-radius:0px;  
margin-top:0px;  
min-height:100px;  
}  
  
#rodape {  
    display:none;  
}  
  
.colConcluidaItem{  
    display:none;  
}  
.colConcluidaHeader{  
    display:none;  
}  
  
.colPrioridadeItem{  
    display:none;  
}  
.colPrioridadeHeader{  
    display:none;  
}  
}  
}  
  
@media (max-height:550px) {  
    #rodape { display:none; }  
}  
  
.mensagemErro {  
    font-size:110%;  
    font-weight:bold;  
    display:block;  
    Color:red;  
}  
  
.mensagemOK {  
    font-size:110%;  
    font-weight:bold;  
    display:block;  
    Color:#dde6f3;  
}
```

Visual Studio 2015 - ASP.NET com C# Fundamentos

12. Teste o programa na resolução para desktop e celular:



13. Existem duas telas: a listagem e o formulário. O controle **MultiView** vai alternar entre essas duas telas. Insira o MultiView e o Label de mensagem. Insira o código HTML para a criação do MultiView dentro da div **conteúdo**:

```
<asp:Label runat="server" ID="mensagemLabel" CssClass="mensagem"  
    Visible="false"></asp:Label>  
<asp:MultiView ID="tarefasMultiView"  
    runat="server"  
    ActiveViewIndex="0">  
  
    <asp:View ID="listagemView" runat="server">  
  
    </asp:View>  
  
    <asp:View ID="formView" runat="server">  
  
    </asp:View>  
</asp:MultiView>
```

14. No primeiro View, insira um **GridView** para a listagem e um **LinkButton** para o comando **Incluir**:

```
<asp:View ID="listagemView" runat="server">

    <asp:GridView ID="tarefasGridView" runat="server">
        </asp:GridView>

        <asp:LinkButton ID="novaTarefaLinkButton" runat="server">
            Nova Tarefa
        </asp:LinkButton>

    </asp:View>
```

15. É bom iniciar com a inclusão, para ter dados na listagem. O formulário de inclusão está definido no **FormView**:

```
<asp:View ID="formView" runat="server">

    <fieldset class="tarefaCampos">
        <legend>Tarefa</legend>

        <!-- prioridade -->
        <asp:Label CssClass="legenda"
            ID="prioridadeLabel" AssociatedControlID="prioridadeDropDownList"
            runat="server" Text="Prioridade:"></asp:Label>

        <asp:DropDownList CssClass="campo larguraMax80"
            ID="prioridadeDropDownList" runat="server">
            <asp:ListItem></asp:ListItem>
            <asp:ListItem>1</asp:ListItem>
            <asp:ListItem>2</asp:ListItem>
            <asp:ListItem>3</asp:ListItem>
        </asp:DropDownList>

        <!-- Concluída -->
        <asp:Label CssClass="legenda"
            ID="concluidaLabel" AssociatedControlID="concluidaDropDownList"
            runat="server"
            Text="Concluída:"></asp:Label>

        <asp:DropDownList CssClass="campo larguraMax80"
            ID="concluidaDropDownList"
            runat="server">
            <asp:ListItem></asp:ListItem>
            <asp:ListItem>Sim</asp:ListItem>
            <asp:ListItem>Não</asp:ListItem>
        </asp:DropDownList>
```

Visual Studio 2015 - ASP.NET com C# Fundamentos

```
<!-- nome -->
<asp:Label CssClass="legenda"
    AssociatedControlID="nomeTextBox"
    ID="nomeLabel" runat="server"
    Text="Nome:></asp:Label>

<asp:TextBox CssClass="campo"
    ID="nomeTextBox" runat="server"></asp:TextBox>

<!-- observações -->
<asp:Label CssClass="legenda"
    AssociatedControlID="observacoesTextBox"
    ID="observacoesLabel"
    runat="server"
    Text="Observações:></asp:Label>

<asp:TextBox CssClass="campo obs"
    ID="observacoesTextBox"
    runat="server" TextMode="MultiLine"></asp:TextBox>

<!-- comandos -->
<asp:LinkButton ID="gravarLinkButton"
    runat="server"
    Text="Gravar"
    CssClass="botaoLink"></asp:LinkButton>

<asp:LinkButton ID="excluirLinkButton"
    runat="server"
    Text="Excluir"
    CssClass="botaoLink"
    Visible="false">
</asp:LinkButton>

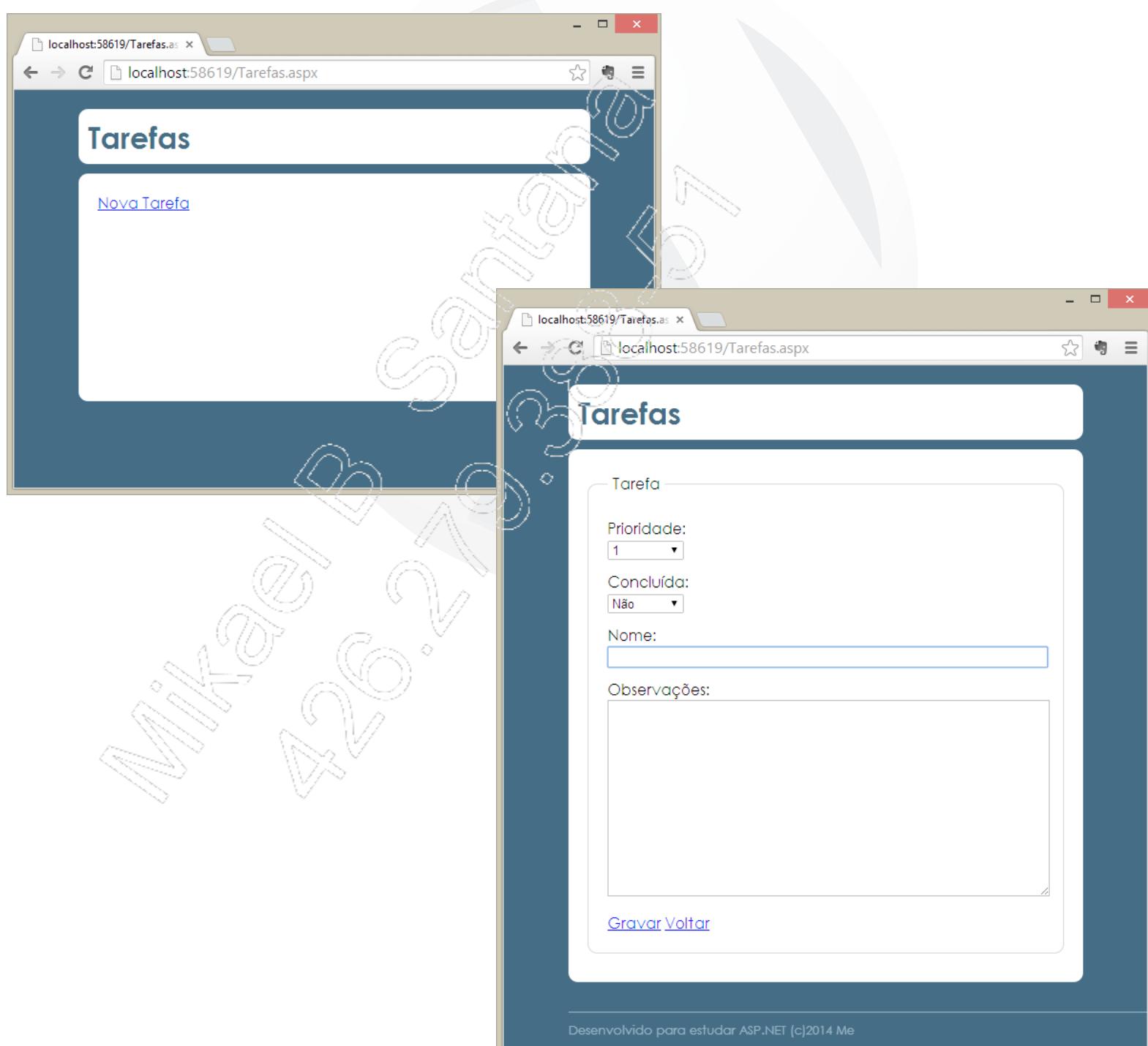
<asp:LinkButton ID="voltarLinkButton"
    runat="server"
    Text="Voltar"
    CssClass="botaoLink"></asp:LinkButton>

</fieldset>
</asp:View>
```

16. Crie o evento (temporário) do botão **Nova Tarefa**, da **ListagemFormView**:

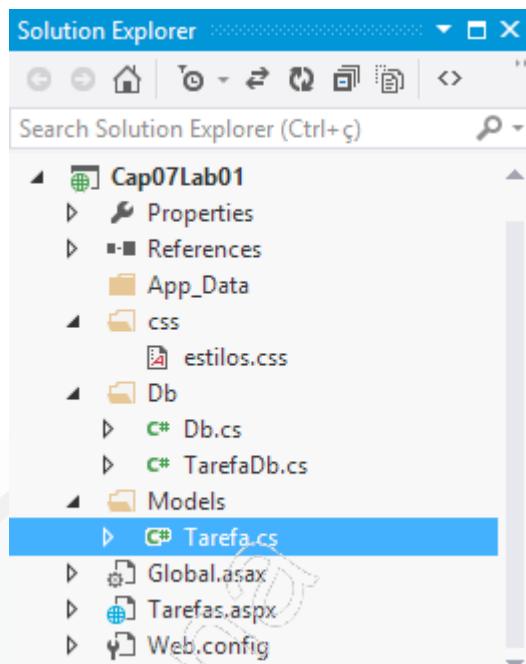
```
//  
// Nova Tarefa - Evento  
  
protected void novaTarefaLinkButton_Click  
    (object sender, EventArgs e)  
{  
    tarefasMultiView.ActiveViewIndex = 1;  
}
```

17. Teste o programa:



Visual Studio 2015 - ASP.NET com C# Fundamentos

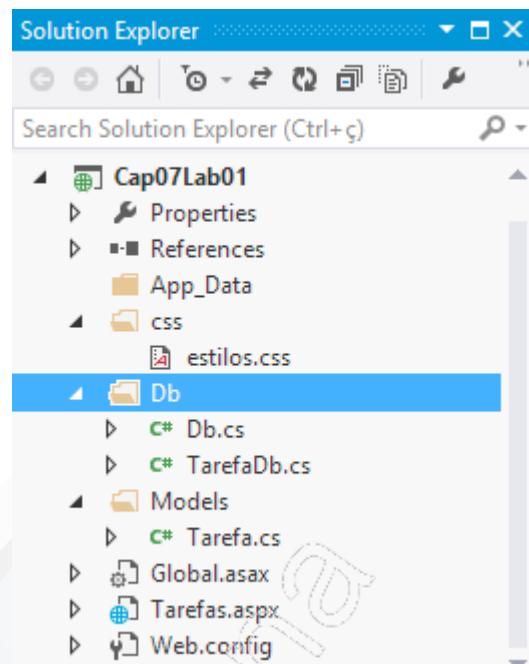
18. Para manipular os dados, crie uma classe chamada **Tarefa**, dentro de uma pasta chamada **Models**:



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

namespace Cap07Lab01.Models
{
    public class Tarefa
    {
        public int Id { get; set; }
        public string Nome { get; set; }
        public int Prioridade { get; set; }
        public bool Concluida { get; set; }
        public string Observacoes { get; set; }
    }
}
```

19. Para acesso ao banco de dados, crie uma pasta chamada **Db** com duas classes: uma chamada **Db** e outra chamada **TarefaDb**:



20. A classe **Db** fornece uma propriedade chamada **Conexao**, para exibir a string de conexão:

```
namespace Cap07Lab01.Db
{
    public static class Db
    {
        public static string Conexao
        {
            get
            {
                return @"Data Source=.\sqlExpress;
Initial Catalog=Pessoal;
Integrated Security=True";
            }
        }
    }
}
```

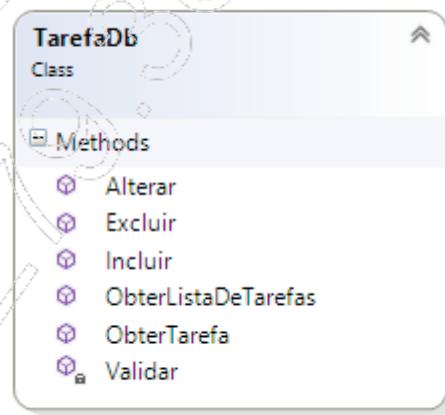
Visual Studio 2015 - ASP.NET com C# Fundamentos

21. A classe **TarefaDb** faz as operações **CRUD** (Create, Read, Update, Delete – criar, ler, atualizar e excluir):

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Data.SqlClient;
using Cap07Lab01.Models;

namespace Cap07Lab01.Db
{
    public class TarefaDb
    {
        }
}
```

Serão criados os seguintes métodos: **Incluir**, **Alterar**, **Excluir**, **Validar**, **ObterListaDeTarefas** e **ObterTarefa**.



22. Implemente o método **Validar**. Compile o programa sempre que terminar um método para ter certeza de que não tem nenhum erro de sintaxe;

```

//  

// Validação  

//  

private static void Validar(Tarefa tarefa)  

{  

    //Nome é obrigatório  

    if (string.IsNullOrEmpty(tarefa.Nome) ||  

        tarefa.Nome.Trim().Length == 0)  

    {  

        throw new Exception(  

            "Informe o nome da tarefa");  

    }  

    //Prioridade é um número entre 1 e 3  

    if (tarefa.Prioridade < 1 ||  

        tarefa.Prioridade > 3)  

    {  

        throw new Exception(  

            "A prioridade deve ser um número entre 1 e 3" );  

    }  

    // Formata observações para ficar Null  

    // Isso facilita algumas Conversões de Dados  

    if (tarefa.Observacoes == null)  

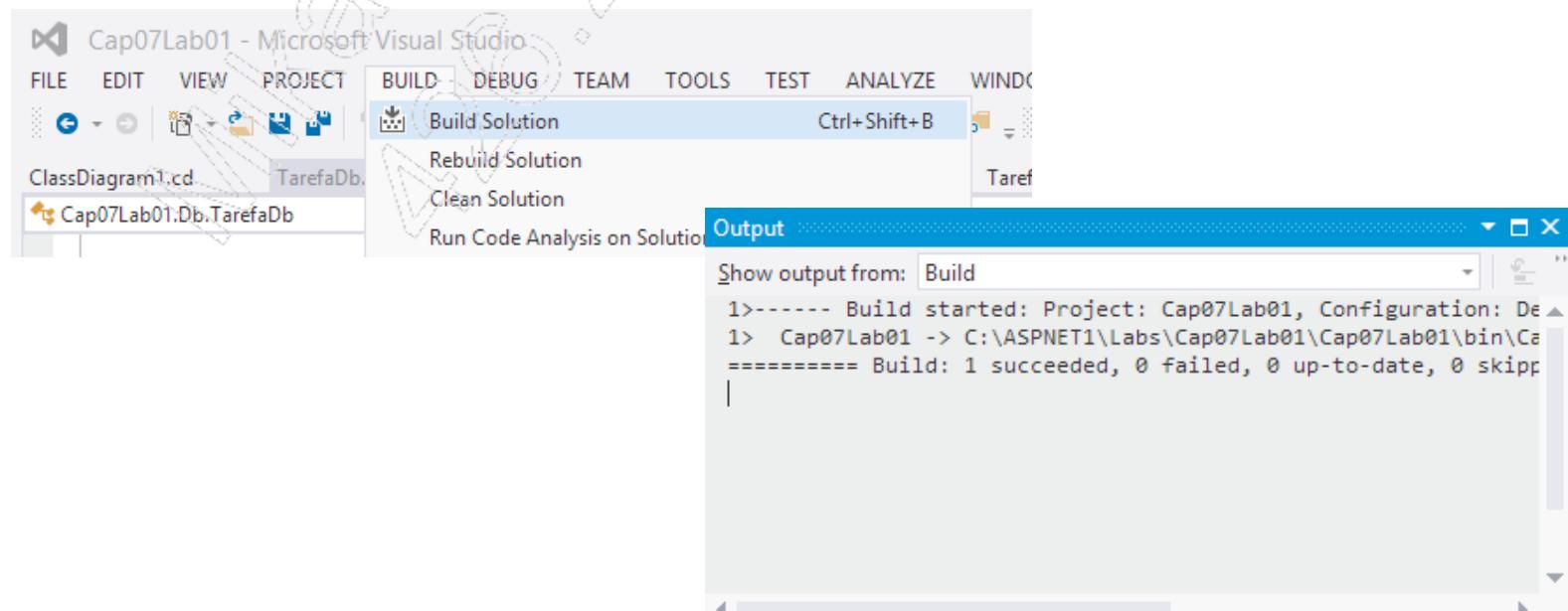
    {  

        tarefa.Observacoes = string.Empty;  

    }  

}

```



23. Implemente o método **Incluir**. Lembre-se sempre de compilar para detectar qualquer erro de sintaxe:

```
//  
// Incluir  
//  
public static int Incluir(Tarefa tarefa)  
{  
    Validar(tarefa);  
  
    string sql =  
        @"INSERT INTO Tarefas  
            (Nome, Prioridade, Concluida, Observacoes)  
        VALUES (@Nome, @Prioridade, @Concluida, @Observacoes);  
        SELECT @@IDENTITY";  
  
    int novoId = 0;  
  
    using (var cn = new SqlConnection(Db.Conexao))  
    {  
        var cmd = new SqlCommand(sql, cn);  
  
        cmd.Parameters.AddWithValue("@Nome", tarefa.Nome);  
  
        cmd.Parameters.AddWithValue("@Prioridade",  
            tarefa.Prioridade);  
  
        cmd.Parameters.AddWithValue("@Concluida",  
            tarefa.Concluida);  
  
        cmd.Parameters.AddWithValue("@Observacoes",  
            tarefa.Observacoes);  
  
        cn.Open();  
        novoId = Convert.ToInt32(cmd.ExecuteScalar());  
    }  
  
    return novoId;  
}
```

24. Implemente o método **ObterListaDeTarefas**. Esse método retorna uma coleção fortemente tipada de objetos do tipo **Tarefa**:

```
//  
// Obter Lista  
//  
public static List<Tarefa> ObterListaDeTarefas()  
{  
    var lista = new List<Tarefa>();  
    string sql =  
        @"SELECT Id, Nome, Prioridade, Concluida, Observacoes  
        FROM Tarefas  
        ORDER BY Concluida, Prioridade";  
    using (var cn = new SqlConnection(Db.Conexao))  
    {  
        using (var cmd = new SqlCommand(sql, cn))  
        {  
            cn.Open();  
            using (var dr = cmd.ExecuteReader())  
            {  
                while (dr.Read())  
                {  
                    var tarefa = new Tarefa();  
                    tarefa.Id = Convert.ToInt32(dr["Id"]);  
  
                    tarefa.Nome =  
                        Convert.ToString(dr["Nome"].ToString());  
  
                    tarefa.Prioridade =  
                        Convert.ToInt32(dr["Prioridade"]);  
  
                    tarefa.Concluida =  
                        Convert.ToBoolean(dr["Concluida"]);  
  
                    tarefa.Observacoes =  
                        Convert.ToString(dr["Observacoes"]);  
  
                    lista.Add(tarefa);  
                }  
            }  
        }  
    }  
    return lista;  
}
```

25. Com esses três métodos, já é possível construir a interface para incluir e exibir os dados. No code-behind da página **Tarefas.aspx**, crie um método chamado **TarefasCarregar** e chame esse método na primeira vez que a página é carregada:

```
using Cap07Lab01.Db;
using Cap07Lab01.Models;
using System;
using System.Collections.Generic;
using System.Diagnostics;
using System.Linq;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;

namespace Cap07Lab01
{
    public partial class Tarefas : System.Web.UI.Page
    {
        //
        // PageLoad
        //
        protected void Page_Load(object sender, EventArgs e)
        {
            if (!Page.IsPostBack)
            {
                TarefasCarregar();
            }
        }

        //
        // Preenche a grid com a lista de tarefas
        //
        private void TarefasCarregar()
        {

            tarefasMultiView.ActiveViewIndex = 0;
            tarefasGridView.DataSource = TarefaDb.ObterListaDeTarefas();
            tarefasGridView.DataBind();

        }
    }
}
```

 Sempre compile depois de criar um método.

26. Crie o(s) método(s) para manipular mensagens:

```
//  
// Mensagem Limpar  
//  
private void MensagemLimpar()  
{  
    mensagemLabel.Visible = false;  
}  
  
//  
// Mensagem Erro  
//  
private void MensagemErro(string mensagem)  
{  
    mensagemLabel.Text = mensagem;  
    mensagemLabel.Visible = true;  
    mensagemLabel.CssClass = "mensagemErro";  
}  
  
//  
// Mensagem OK  
//  
private void MensagemOK(string mensagem)  
{  
    mensagemLabel.Text = mensagem;  
    mensagemLabel.Visible = true;  
    mensagemLabel.CssClass = "mensagemOK";  
}
```

27. Crie o método para limpar o formulário:

```
//  
// Limpar Form  
//  
private void LimparForm()  
{  
    MensagemLimpar();  
    this.nomeTextBox.Text = string.Empty;  
    this.prioridadeDropDownList.SelectedIndex = 0;  
    this.concluidaDropDownList.SelectedIndex = 0;  
    this.observacoesTextBox.Text = string.Empty;  
    ViewState["Id"] = 0;  
}
```

Visual Studio 2015 - ASP.NET com C# Fundamentos

28. Crie o método para preparar a inclusão:

```
//  
// Botões Preparar Inclusão  
//  
private void BotoesPrepararInclusao()  
{  
    this.gravarLinkButton.Visible = true;  
    this.excluirLinkButton.Visible = false;  
}  
  
//  
// Preparar Inclusão  
//  
private void PrepararInclusao()  
{  
    tarefasMultiView.ActiveViewIndex = 1;  
    LimparForm();  
    BotoesPrepararInclusao();  
    prioridadeDropDownList.SelectedIndex = 1;  
    concluidaDropDownList.SelectedIndex = 2;  
    nomeTextBox.Focus();  
}
```

29. Crie o evento **Click** do botão **Nova Tarefa**:

```
//  
// Nova Tarefa - Evento  
//  
protected void novaTarefaLinkButton_Click(...)  
{  
    PrepararInclusao();  
}  
  
• No HTML:  
  
<asp:LinkButton  
    ID="novaTarefaLinkButton"  
    runat="server"  
    OnClick="novaTarefaLinkButton_Click">Nova Tarefa</asp:LinkButton>
```

Compile e teste!

30. Crie o método que obtém a tarefa da tela:

```
//  
// Obter Tarefa  
//  
private Tarefa TarefaObterDaTela()  
{  
    var tarefa = new Tarefa();  
    tarefa.Id = Convert.ToInt32(ViewState["Id"]);  
    tarefa.Nome = nomeTextBox.Text;  
  
    tarefa.Prioridade =  
        Convert.ToInt32(prioridadeDropDownList.SelectedValue);  
  
    tarefa.Concluida = concluidaDropDownList.SelectedIndex == 1;  
  
    tarefa.Observacoes =  
        Convert.ToString(observacoesTextBox.Text);  
  
    return tarefa;  
}
```

31. Crie o método que grava:

```
//  
// Gravar  
//  
private void Gravar()  
{  

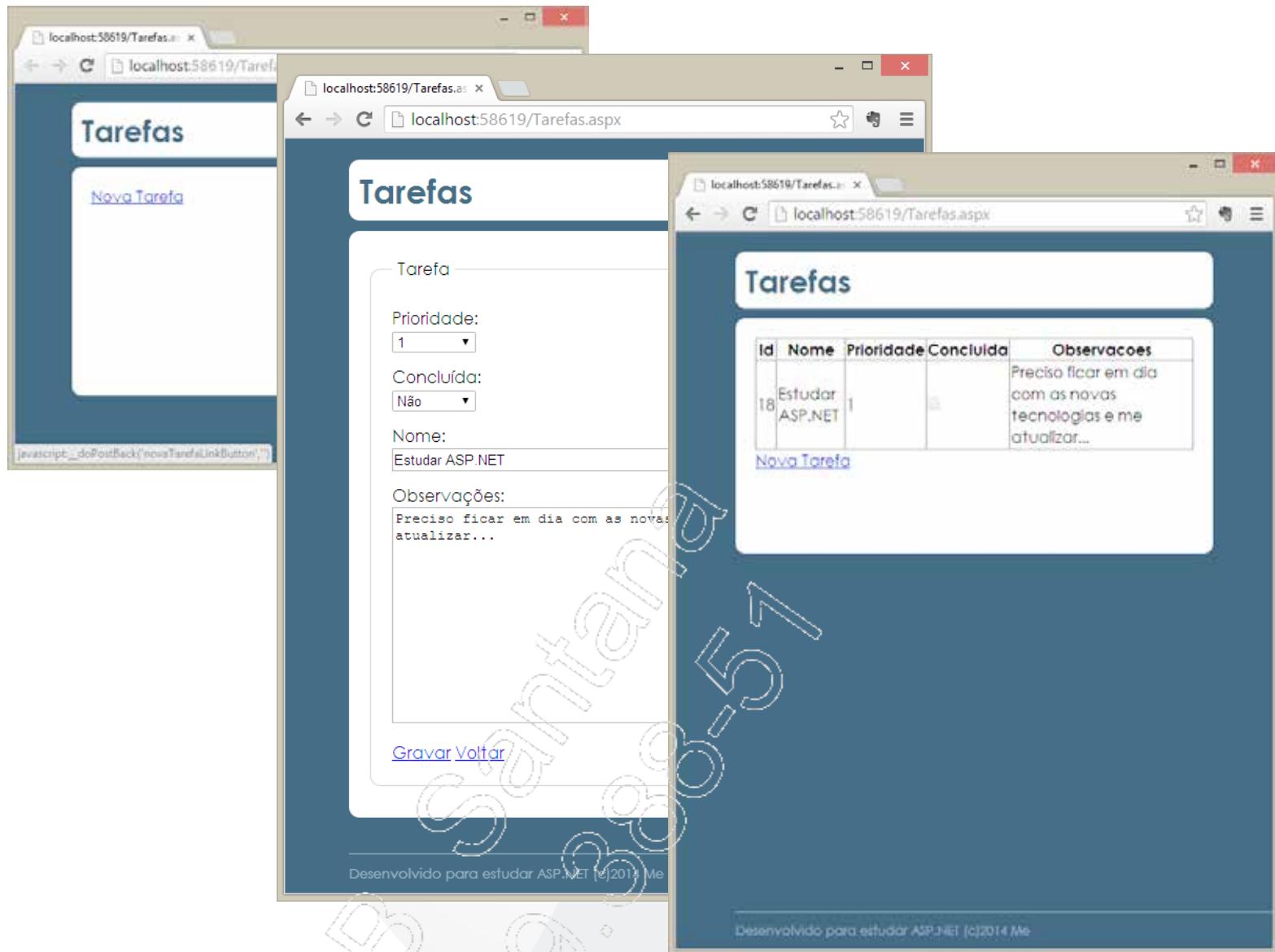
```

32. Crie o método do evento **Click** do botão **Gravar**:

```
<asp:LinkButton  
    ID="gravarLinkButton"  
    runat="server"  
    Text="Gravar"  
    CssClass="botaoLink"  
    OnClick="gravarLinkButton_Click"></asp:LinkButton>
```

```
//  
// Gravar  
//  
protected void gravarLinkButton_Click(object sender,  
    EventArgs e)  
{  
    try  
    {  
        Gravar();  
    }  
    catch (Exception ex)  
    {  
        MensagemErro(ex.Message);  
    }  
}
```

33. Teste a inclusão:



34. É possível melhorar a aparência da grid com algumas configurações:

```
<asp:GridView CssClass="gridView"
    ID="tarefasGridView"
    AutoGenerateColumns="False"
    DataKeyNames = "Id"
    runat="server" >

    <Columns>

        <asp:CheckBoxField
            ItemStyle-CssClass="colConcluidaItem"
            HeaderStyle-CssClass="colConcluidaHeader"
            DataField="Concluida"
            HeaderText="Concluída" />
```

Visual Studio 2015 - ASP.NET com C# Fundamentos

```
<asp:BoundField DataField="prioridade"
    ItemStyle-CssClass="colPrioridadeItem"
    HeaderText="Prioridade"
    HeaderStyle-CssClass="colPrioridadeHeader" />

<asp:ButtonField DataTextField="Nome"
    ItemStyle-CssClass="colTarefaItem"
    HeaderStyle-CssClass="colTarefaHeader" CommandName="alterar"
    HeaderText="Tarefa" />

</Columns>
</asp:GridView>
```



35. Completando a classe **TarefaDb**, crie o método Alterar:

```
//  
// Alterar  
//  
public static int Alterar(Tarefa tarefa)  
{  
    string sql = @"UPDATE Tarefas  
                  SET Nome=@Nome,  
                      Prioridade=@Prioridade,  
                      Concluida=@Concluida,  
                      Observacoes=@Observacoes  
                  WHERE Id=@Id";  
    int total = 0;  
    using (var cn = new SqlConnection(Db.Conexao))  
    {  
        using (var cmd = new SqlCommand(sql, cn))  
        {  
            cmd.Parameters.AddWithValue("@Nome", tarefa.Nome);  
  
            cmd.Parameters.AddWithValue("@Prioridade",  
                tarefa.Prioridade);  
  
            cmd.Parameters.AddWithValue("@Concluida",  
                tarefa.Concluida);  
  
            cmd.Parameters.AddWithValue("@Observacoes",  
                tarefa.Observacoes);  
  
            cmd.Parameters.AddWithValue("@Id", tarefa.Id);  
  
            cn.Open();  
  
            total = Convert.ToInt32(cmd.ExecuteScalar());  
        }  
    }  
    return total;  
}
```

36. Crie o método ObterTarefa:

```
//  
// Obter Tarefa  
//  
public static Tarefa ObterTarefa(int id)  
{  
    string sql = @"SELECT Id,  
                  Nome,  
                  Prioridade,  
                  Concluida,  
                  Observacoes  
            FROM Tarefas  
            WHERE ID=@ID";  
  
    Tarefa tarefa = null;  
    using (var cn = new SqlConnection(Db.Conexao))  
    {  
        using (var cmd = new SqlCommand(sql, cn))  
        {  
            cmd.Parameters.AddWithValue("@ID", id);  
            cn.Open();  
  
            using (var dr = cmd.ExecuteReader())  
            {  
  
                if (dr.Read())  
                {  
                    tarefa = new Tarefa()  
                    {  
                        Id = Convert.ToInt32(dr["Id"]),  
                        Nome = Convert.ToString(dr["Nome"]),  
                        Concluida = Convert.ToBoolean(dr["Concluida"]),  
                        Prioridade = Convert.ToInt32(dr["Prioridade"]),  
                        Observacoes = Convert.ToString(dr["Observacoes"])  
                    };  
                }  
            }  
        }  
    }  
    return tarefa;  
}
```

37. Crie o método Excluir:

```
//  
// Excluir  
  
public static int Excluir(int id)  
{  
    string sql = @"DELETE TAREFAS WHERE Id=@Id";  
    int total = 0;  
    using (var cn = new SqlConnection(Db.Conexao))  
    {  
  
        using (var cmd = new SqlCommand(sql, cn))  
        {  
            cmd.Parameters.AddWithValue("@Id", id);  
            cn.Open();  
            total = Convert.ToInt32(cmd.ExecuteNonQuery());  
        }  
    }  
    return total;  
}
```

38. De volta à página tarefas.aspx, crie o método TarefaExibir():

```
//  
// Tarefa Exibir  
  
private void TarefaExibir(Tarefa tarefa)  
{  
  
    ViewState["Id"] = tarefa.Id;  
  
    nomeTextBox.Text = tarefa.Nome;  
  
    concluidaDropDownList.SelectedIndex =  
        tarefa.Concluida ? 1 : 0;  
  
    prioridadeDropDownList.SelectedValue =  
        tarefa.Prioridade.ToString();  
  
    observacoesTextBox.Text = tarefa.Observacoes;  
}
```

39. Crie o método **PrepararAlteracaoExclusao()**:

```
//  
// Preparar Alteração Exclusão  
//  
private void PrepararAlteracaoExclusao()  
{  
    excluirLinkButton.Visible = true;  
    tarefasMultiView.ActiveViewIndex = 1;  
    nomeTextBox.Focus();  
}
```

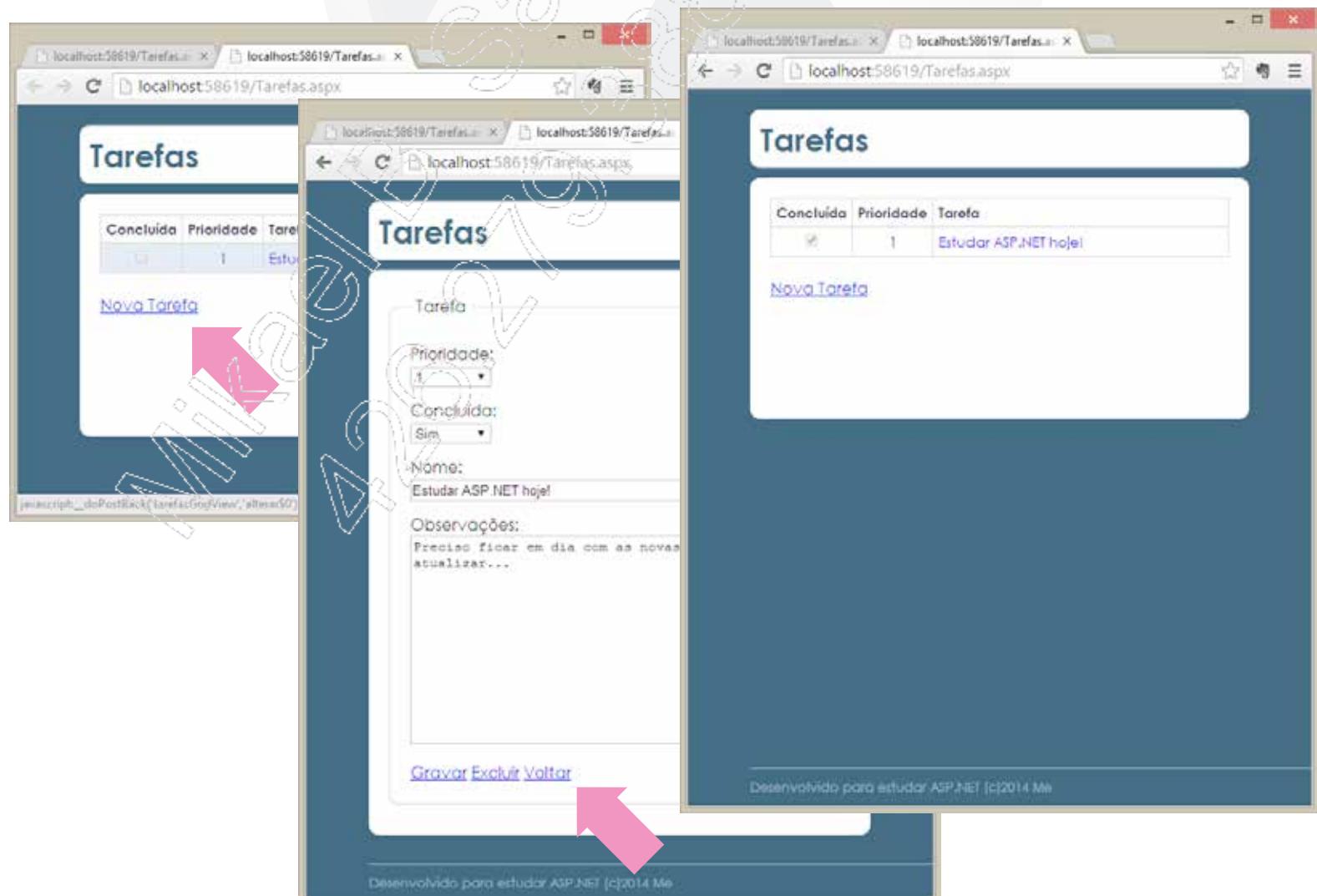
40. Crie o evento **RowCommand** da GridView:

```
<asp:GridView  CssClass="gridView"  
             ID="tarefasGridView"  
             AutoGenerateColumns="False"  
             DataKeyNames ="Id"  
             runat="server" OnRowCommand="tarefasGridView_RowCommand">  
  
//  
// Row Command  
//  
protected void tarefasGridView_RowCommand(...  
{  
    try  
    {  
        MensagemLimpar();  
  
        var gv = (GridView)sender;  
  
        int rowIndex = Convert.ToInt32(e.CommandArgument);  
  
        int id = (int)gv.DataKeys[rowIndex].Value;  
  
        var tarefa = TarefaDb.ObterTarefa(id);  
  
        if (tarefa != null)  
        {  
            TarefaExibir(tarefa);  
            PrepararAlteracaoExclusao();  
        }  
    }  
    catch (Exception ex)  
    {  
        MensagemErro(ex.Message);  
    }  
}
```

41. Altere o evento do botão Gravar:

```
//  
// Gravar  
//  
private void Gravar()  
{  
    MensagemLimpar();  
    var tarefa = TarefaObterDaTela();  
  
    if (tarefa.Id == 0)  
    {  
        tarefa.Id = TarefaDb.Incluir(tarefa);  
    }  
    else  
    {  
        TarefaDb.Alterar(tarefa);  
    }  
  
    TarefasCarregar();  
}
```

42. Teste a alteração de dados:



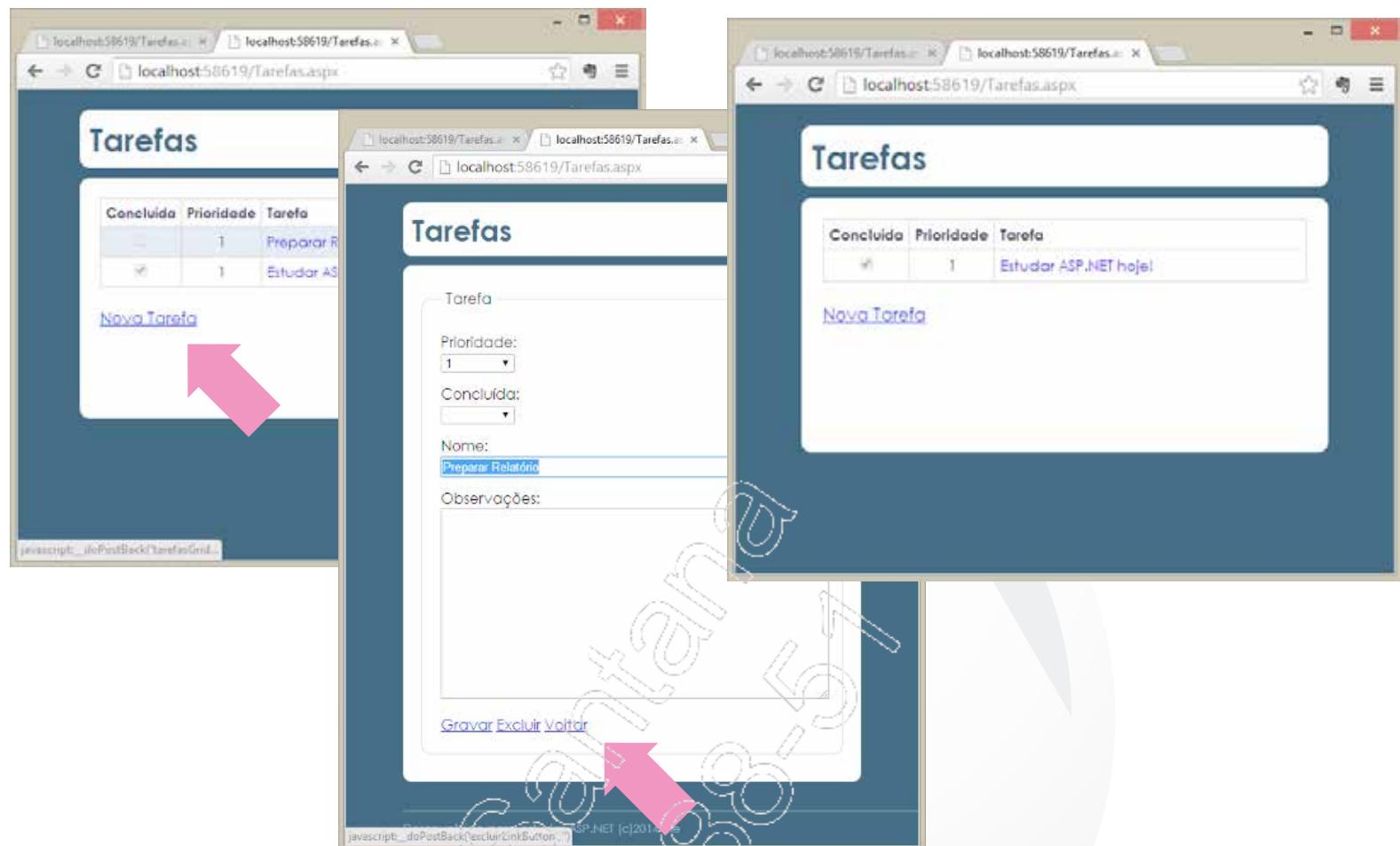
43. Crie o método **Excluir**:

```
//  
// Excluir  
//  
private void Excluir()  
{  
    MensagemLimpar();  
    var tarefa = TarefaObterDaTela();  
    TarefaDb.Excluir(tarefa.Id);  
    TarefasCarregar();  
  
}
```

44. Crie o evento do método **Excluir**:

```
//  
// Excluir  
//  
protected void excluirLinkButton_Click(object sender,  
                                         EventArgs e)  
{  
    try  
    {  
        Excluir();  
    }  
  
    catch (Exception ex)  
    {  
        MensagemErro(ex.Message);  
    }  
}
```

45. Sistema completo! Teste a exclusão:



Sugestões para estudo:

- Criar uma confirmação para a exclusão;
- Incluir um campo **Porcentagem Concluída**. É um campo numérico com valores entre 0 e 100. Se for definido 100, o campo concluído é automaticamente marcado com **true**;
- Incluir um botão **Exportar**, para gerar uma listagem no formato CSV, que pode ser importada facilmente para outros aplicativos, como Excel.

8

Boas práticas em ASP.NET

- ✓ Web Controls (Web Forms) e HTML (MVC);
- ✓ Performance;
- ✓ Código.



IMPACTA
EDITORA

8.1. Introdução

As ferramentas disponíveis no ASP.NET permitem criar qualquer tipo de aplicação Web que atenda a qualquer especificação exigida. Mas é preciso usá-las de maneira prática e eficiente. Este capítulo resume algumas dicas de como utilizar a ferramenta certa para algumas situações e como evitar alguns problemas conhecidos.

Diferente de aplicações Windows, que utilizam uma biblioteca específica para o sistema operacional em que são executadas, as aplicações Web utilizam diferentes tecnologias e são executadas em navegadores de diversos fabricantes que se comunicam com servidores que também utilizam diferentes tecnologias de outros fabricantes. É um ambiente muito mais complicado para desenvolver, integrar, testar e implementar soluções.

É fundamental seguir algumas recomendações e princípios ao decidir a forma de escrever o programa. Um dos princípios mais importantes é o **SoC** (Separation of Concerns, ou separação das responsabilidades). Segundo ele, um software deve ser separado em partes, cada uma com uma única responsabilidade. Ou seja, a parte que define a aparência do aplicativo deve ser separada da lógica de regras de negócio, da parte de acesso a dados, da parte de comunicação com os serviços e assim por diante.

Vamos ver algumas dessas recomendações e princípios.

8.2. Web Controls (Web Forms) e HTML (MVC)

Os próximos tópicos apresentam recomendações e princípios relacionados a Web Controls no modelo Web Form.

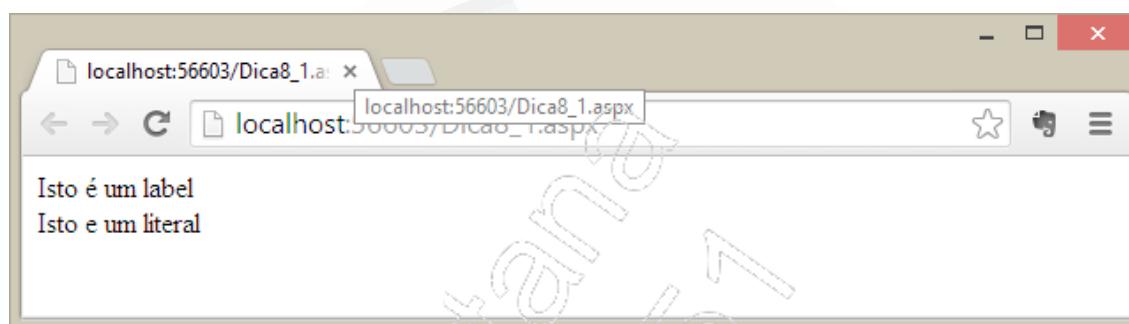
8.2.1. Use Literal no lugar de Label

Se o objetivo é apenas exibir dados da página sem formatação, use o controle **Literal** em vez do controle **Label**. O controle **Label** é uma classe mais pesada, consome mais memória porque tem recursos como fonte, cor e visibilidade. Além disso, o **Label** sempre gera a tag **span**. O controle **Literal** não tem nenhuma propriedade de formatação e não gera nenhuma tag HTML. É mais rápido e eficiente.

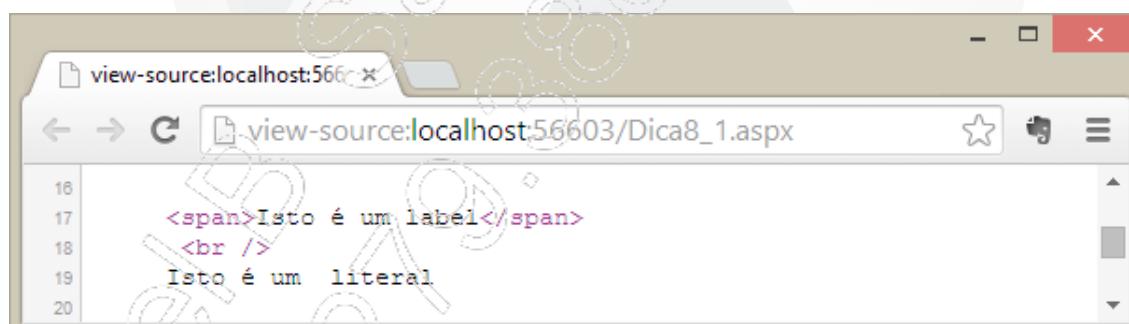
- Veja como é a tag HTML desses controles em uma página .aspx:

```
<asp:Label runat="server" Text="Label">  
Isto é um label</asp:Label>  
  
<br />  
  
<asp:Literal runat="server" Text="Literal">  
Isto é um literal</asp:Literal>
```

- Visualize no browser:



- Veja como o código foi gerado:



8.2.2. Use HtmlEncoding

Quando exibir uma informação na tela usando **Label** ou **Literal**, é bom sempre usar o **HtmlEncoding**, principalmente se há o risco de ter símbolos como < ou > dentro do texto. Isso evita, também, o problema de algum browser com acentuação.

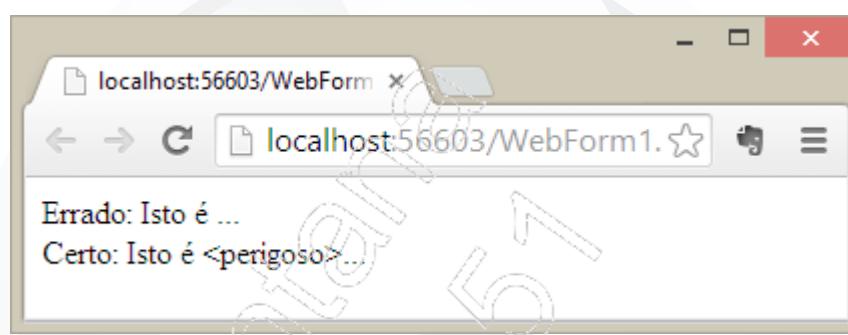
- Página .aspx:

```
<asp:Literal ID="Literal1" runat="server"></asp:Literal><br />
<asp:Literal ID="Literal2" runat="server"></asp:Literal>
```

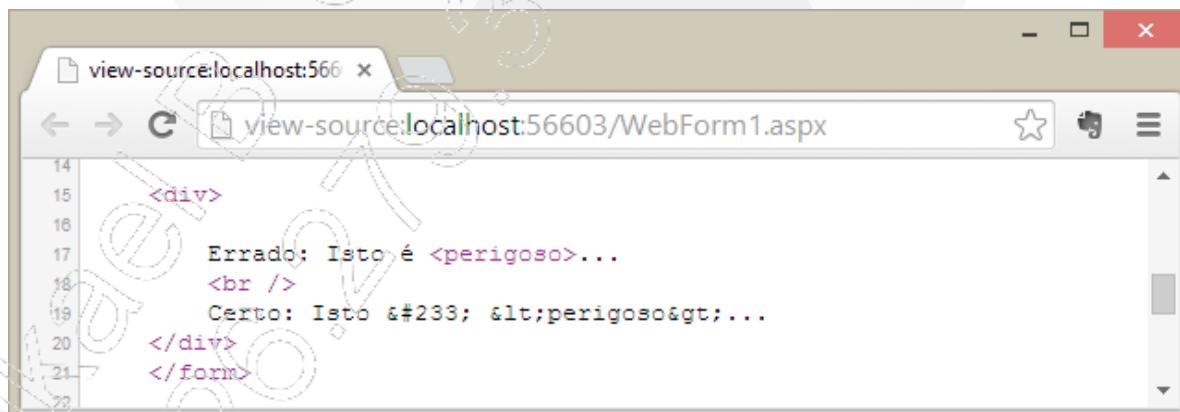
- Code-behind:

```
Literal1.Text = "Errado: Isto é <perigoso>...";  
Literal2.Text = Server.HtmlEncode("Certo: Isto é <perigoso>...");
```

- Visualize no browser:



- Veja como o código foi gerado:



8.2.3. Associe a legenda com o campo

Sempre que criar uma legenda para um campo, use a propriedade **AssociatedControlId** para identificar o TextBox relacionado. Isso facilita a leitura por pessoas com deficiência visual, a indexação do Google e, ainda, tem o efeito de clicar na legenda e o cursor se posicionar no TextBox automaticamente. No caso do MVC, utilize a tag **<label for="...">** para associar o label com o controle HTML de entrada de dados.

- Web Controls (Web Forms):

```
<asp:Label ID="nomeLabel"  
    runat="server"  
    Text="Nome:"  
    AssociatedControlID="nomeTextBox">  
</asp:Label>  
  
<asp:TextBox ID="nomeTextBox" runat="server"></asp:TextBox>
```

- HTML (MVC e Web Pages):

```
<label for="nome">Nome:</label>  
<input name="nome" type="text" />
```

8.2.4. Não use tabela para formulário

Quando desenvolver um formulário com legendas e campos, evite gerar tabelas para criar o layout com o campo na frente da legenda. Se a sua página for acessada por um celular, o ideal é que o campo se ajuste para caber na tela. Isso se chama **design responsivo** e é cada vez mais importante porque, a cada ano, cresce o número de pessoas que acessam o conteúdo Web via celulares ou tablets. Existem várias maneiras de resolver isso. A mais frequente é criar estilos CSS para os Labels e para os TextBoxes.

- Folha de estilo:

```
<style type="text/css">  
    .campo {  
        display:inline-block;  
        width:200px;  
        margin-top:5px;  
    }  
  
    .legenda {  
        display: inline-block;  
        width:50px;  
        margin-top:5px;  
    }  
</style>
```

Visual Studio 2015 - ASP.NET com C# Fundamentos

- Web Forms (propriedade `CssClass`):

```
<asp:Label ID="nomeLabel" runat="server" Text="Nome :"  
CssClass="legenda"></asp:Label>
```

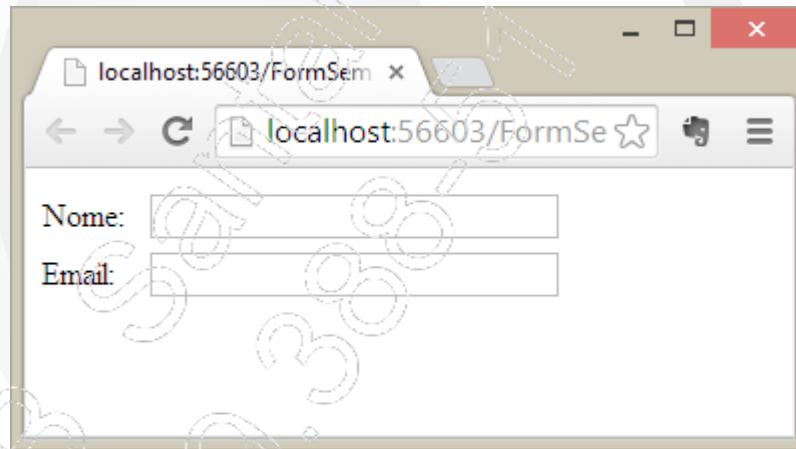
```
<asp:TextBox ID="nomeTextBox" runat="server"  
CssClass="campo"></asp:TextBox>
```

- HTML (MVC e Web Pages):

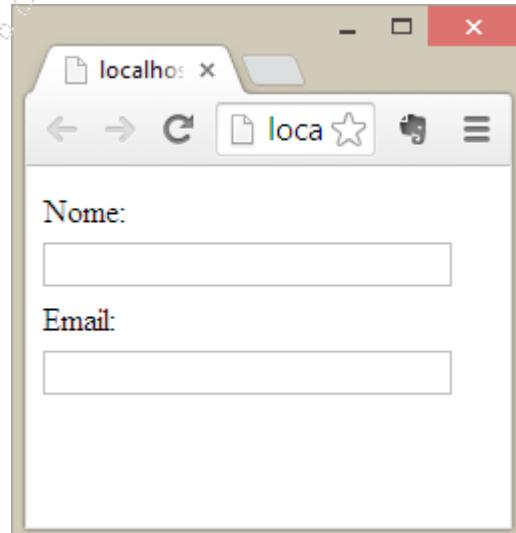
```
<label for="nome" class="Legenda" >Nome:</label>
```

```
<input name="nome" type="text" class="Campo" />
```

- Na tela grande, o **TextBox** fica ao lado:



- Na tela pequena, o **TextBox** fica embaixo:

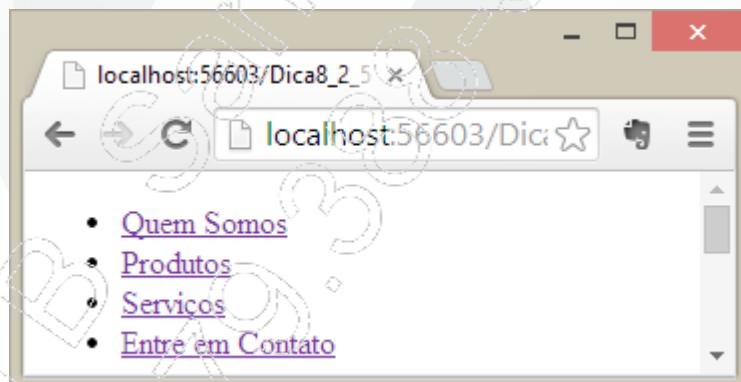


8.2.5. Use listas para menus

Quando criar um menu, utilize as tags de listas (**ul**) incluindo os HyperLinks dentro dos itens (**li**) da lista. Usando folhas de estilo, é possível formatar a lista na horizontal, na vertical e definir com precisão as margens. Evite usar a tag de linha em branco (**br**) ou tabelas para definir o layout. Veja o exemplo:

- Sem aplicar nenhum estilo:

```
<div>
  <ul>
    <li><a href="quemSomos.aspx">Quem Somos</a></li>
    <li><a href="produtos.aspx">Produtos</a></li>
    <li><a href="servicos.aspx">Serviços</a></li>
    <li><a href="contato.aspx">Entre em Contato</a></li>
  </ul>
</div>
```



- Aplicando estilos CSS para um menu horizontal:

- Arquivo **.aspx**:

```
<div class="menuHorizontal">
  <ul>
    <li><a href="#">Quem Somos</a></li>
    <li><a href="#">Produtos</a></li>
    <li><a href="#">Serviços</a></li>
    <li><a href="#">Entre em Contato</a></li>
  </ul>
</div>
```

- Arquivo **estilos.css**:

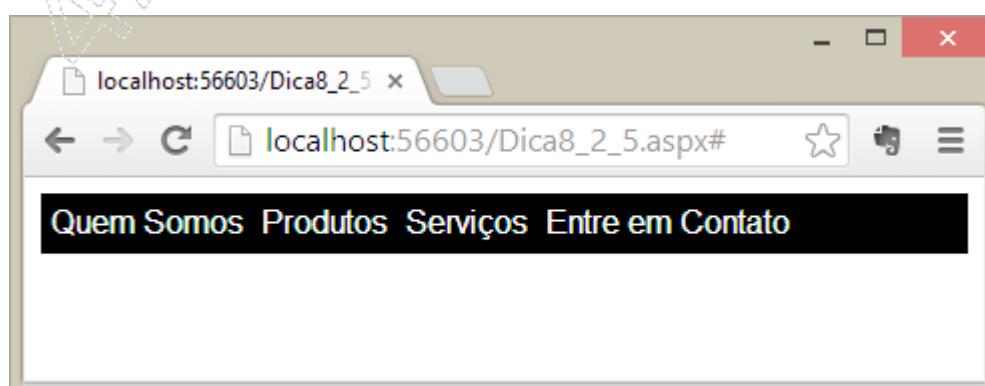
```
/* Fundo preto; 5px de espaço interno*/
.menuHorizontal{
    background-color:#000000;
    padding:5px;
}

/* listas sem o símbolo de bullet e sem margens */
.menuHorizontal ul{
    list-style-type:none;
    margin:0px;
    padding:0px;
}

/* lista exibida horizontalmente */
.menuHorizontal ul > li {
    display:inline-block;
    margin-right:5px;
}

/* HyperLink branco, sem sublinhado na fonte Arial*/
.menuHorizontal ul>li>a {
    text-decoration:none;
    color:#ffffff;
    font-family:Arial;
}
```

- Menu exibido no browser:



- Aplicando estilos CSS para um menu vertical:

- Arquivo **.aspx**:

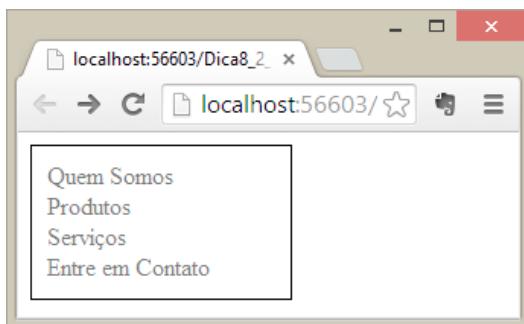
```
<div class="menuVertical">
    <ul>
        <li><a href="#">Quem Somos</a></li>
        <li><a href="#">Produtos</a></li>
        <li><a href="#">Serviços</a></li>
        <li><a href="#">Entre em Contato</a></li>
    </ul>
</div>
```

- Arquivo **estilos.css**:

```
/* menu com borda, 150px de largura, margem interna de 10 pixels
   e margem inferior externa de 10px */
.menuVertical {
    border:1px solid #000000;
    width:150px;
    padding:10px;
    margin-bottom:10px;
}

/* listas sem o símbolo de bullet e sem margens */
.menuVertical ul{
    list-style-type:none;
    margin:0px;
    padding:0px;
}
/* HyperLink cinza, sem sublinhado */
.menuVertical ul>li>a {
    text-decoration:none;
    color:#5f5f5f
}
/* HyperLink muda a cor ao passar o mouse em cima*/
.menuVertical ul>li>a:hover {
    color:#ff6a00;
}
```

- Menu exibido no browser:



8.3. Performance

Os próximos subtópicos estão relacionados à performance.

8.3.1. Desabilite o ViewState (Web Forms)

Se a sua página não realiza PostBack, não há necessidade de armazenar o estado dos controles. Os Web Forms sempre armazenam o estado dos controles por meio do mecanismo **ViewState**.

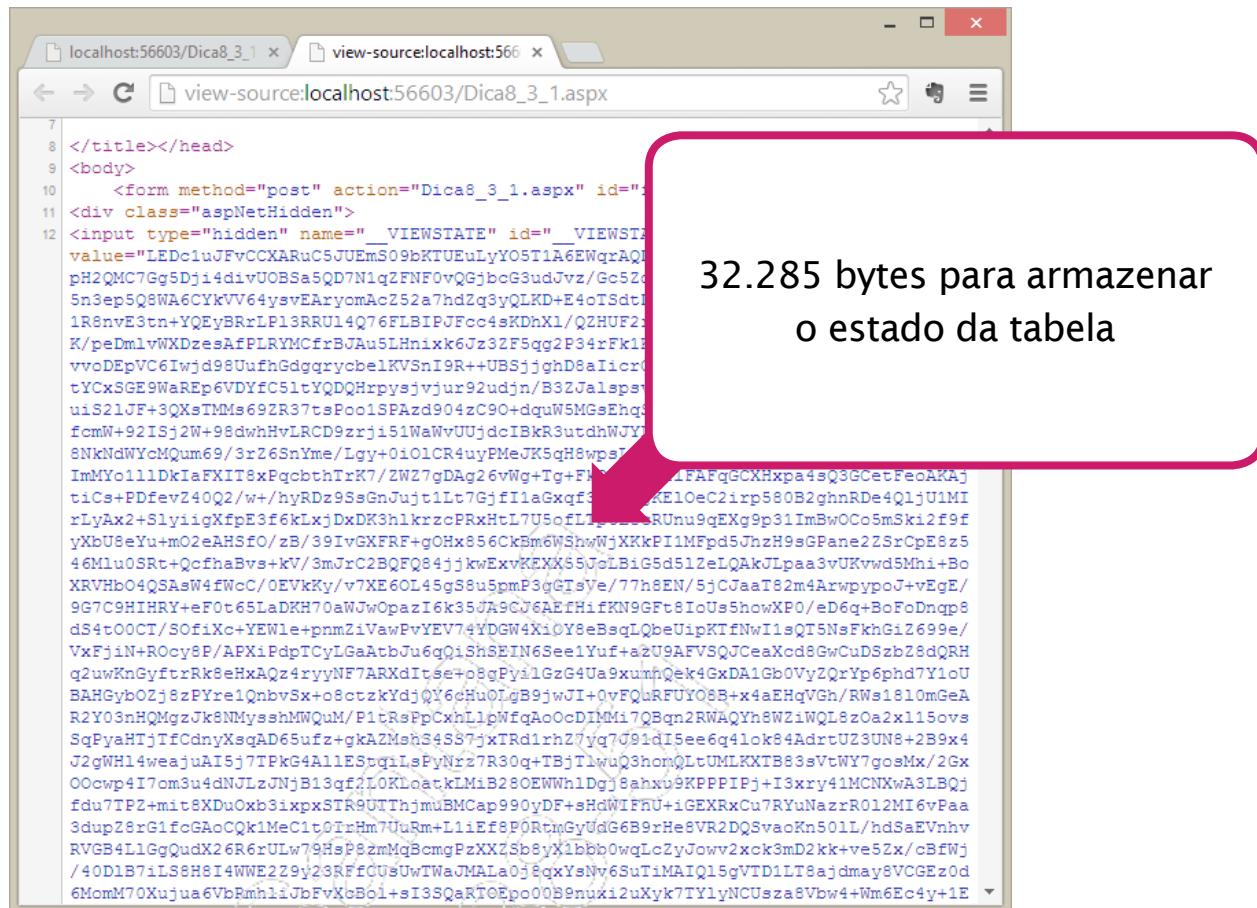
Esse mecanismo armazena, em um controle HTML oculto, as propriedades dos controles para poder reconstruir a página com o mesmo conteúdo. Isso torna o tráfego entre o cliente e o servidor muito maior do que o necessário, nos casos em que não vai haver processamento por meio de PostBack. Veja um exemplo:

- Esta página exibe uma lista. É a tabela **Customers** (clientes) do banco de dados exemplo da Microsoft, **Northwind**:

A screenshot of a Microsoft Edge browser window. The address bar shows 'localhost:56603/Dica8_3_1.aspx'. The page displays a table with data from the 'Customers' table in the Northwind database. The columns are: CustomerID, CompanyName, ContactName, ContactTitle, Address, City, and Region. The data rows are:

CustomerID	CompanyName	ContactName	ContactTitle	Address	City	Region
ALFKI	Alfreds Futterkiste	Maria Anders	Sales Representative	Obere Str. 57	Berlin	
ANATR	Ana Trujillo Emparedados y helados	Ana Trujillo	Owner	Avda. de la Constitución 2222	México D.F.	
ANTON	Antonio Moreno Taqueria	Antonio Moreno	Owner	Mataderos 2312	México D.F.	
AROUT	Around the Horn	Thomas Hardy	Sales Representative	120 Hanover Sq.	London	
BERGS	Berglunds snabbköp	Christina Berglund	Order Administrator	Berguvvägen 8	Luleå	
BLAUS	Blauer See Delikatessen	Hanna Moos	Sales Representative	Forsterstr. 57	Mannheim	

Analisando o código-fonte, é possível ver o controle oculto **ViewState**, que armazena as propriedades dos controles, que, nesse caso, é apenas a **GridView**. O conteúdo desse **ViewState** é de 32.285 caracteres, quase 32 Kb.



Para desabilitar o **ViewState** da página, pode ser usada a diretiva de compilação, ou, dentro do código-fonte, a propriedade **EnableViewState** da classe **Page**, da qual todo Web Form é derivado.

- Desligando o **ViewState** por meio da diretiva da página:

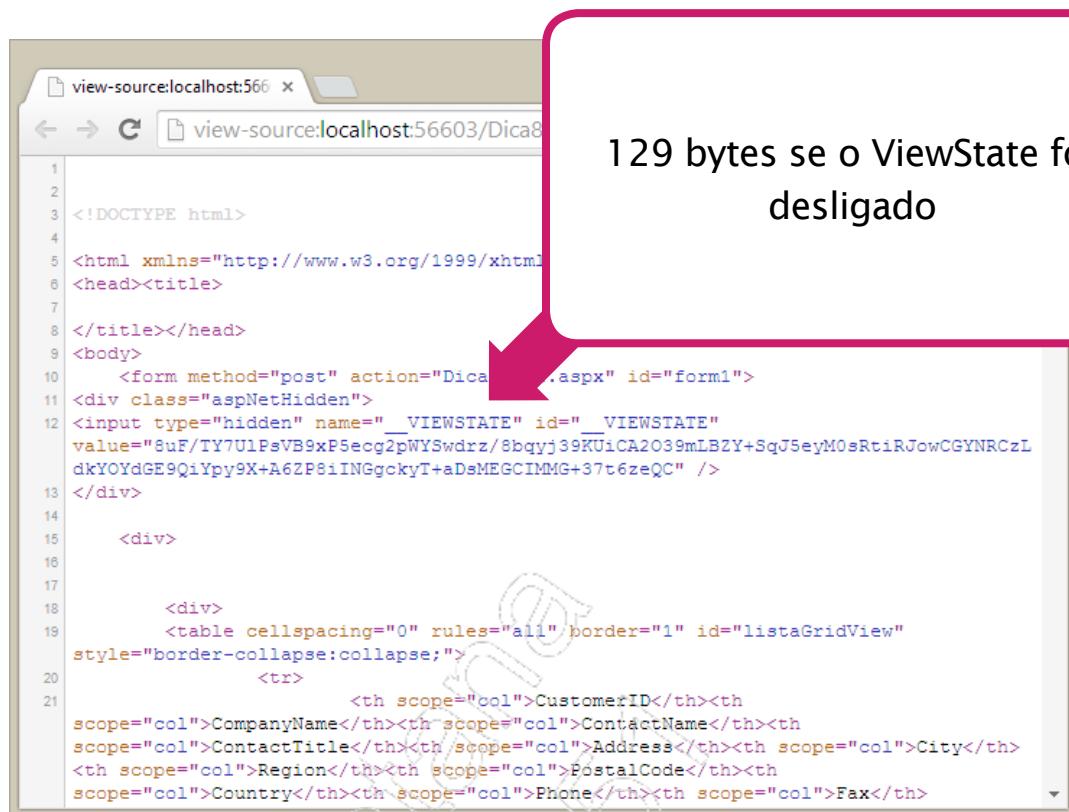
```
<%@ Page EnableViewState="false" Language="C#" ... %>
```

- Desligando o **ViewState** via código:

```
this.EnableViewState = false;
```

Visual Studio 2015 - ASP.NET com C# Fundamentos

- Sem o ViewState, o controle escondido que armazena o estado passou de 32.285 para 129 bytes.



view-source:localhost:566 x view-source:localhost:56603/Dica8.aspx

```
1
2
3 <!DOCTYPE html>
4
5 <html xmlns="http://www.w3.org/1999/xhtml">
6 <head><title>
7
8 </title></head>
9 <body>
10    <form method="post" action="Dica8.aspx" id="form1">
11      <div class="aspNetHidden">
12        <input type="hidden" name="__VIEWSTATE" id="__VIEWSTATE" value="8uF/TY7U1PsvB9xP5ecg2pWYSwdrz/8bqyj39KUiCA2O39mLBZY+SqJ5eyM0sRtiRJowCGYNRCzLdkYOYdGE9QlYpy9X+A6ZP8iINGckyT+aDsMEGCIMMG+37t6zeQC" />
13      </div>
14
15      <div>
16
17          <div>
18              <table cellspacing="0" rules="all" border="1" id="listaGridView" style="border-collapse:collapse;">
19                  <tr>
20                      <th scope="col">CustomerID</th><th scope="col">CompanyName</th><th scope="col">ContactName</th><th scope="col">ContactTitle</th><th scope="col">Address</th><th scope="col">City</th>
21                      <th scope="col">Region</th><th scope="col">PostalCode</th><th scope="col">Country</th><th scope="col">Phone</th><th scope="col">Fax</th>
22                  </tr>
23                  <tr>
24                      <td>1</td>
25                      <td>Adventure Works</td>
26                      <td>Anne Davolio</td>
27                      <td>Sales Representative</td>
28                      <td>123 Main St</td>
29                      <td>Seattle</td>
30                      <td>WA</td>
31                      <td>USA</td>
32                      <td>(503) 555-1234</td>
33                      <td>(503) 555-1235</td>
34                  </tr>
35              </table>
36          </div>
37      </div>
38
39  </form>
```

129 bytes se o ViewState for desligado

Um detalhe curioso é que o controle oculto ainda existe. Afinal de contas, se o mecanismo do ViewState foi desligado na página, por que ainda são gastos 129 bytes para armazenar estado?

O que acontece é que o controle do ViewState será criado sempre que existir qualquer tag com o atributo **runat-server**. Entre outras coisas, essa informação mínima é utilizada para validar a página.

É importante lembrar que o ViewState só poderá ser desligado se a página não faz nenhum PostBack. Se houver qualquer processamento do lado do servidor após a página ser carregada e o ViewState estiver desligado, todos os controles perderão as propriedades quando a página for reprocessada. Os únicos controles que vão manter o estado (e mesmo assim apenas a propriedade principal) são aqueles que enviarem o seu conteúdo por meio do atributo **value**.

O controle **TextBox**, por exemplo, gera uma tag **input**, e isso é enviado por meio do POST do formulário. O valor (**value**) desse input é copiado na propriedade **Text** do TextBox, independente de o ViewState da página estar habilitado ou não.

No exemplo a seguir, a GridView é carregada no evento **Page_Load** na primeira vez que a página é carregada. A propriedade **IsPostBack** informa se a página está sendo chamada pelo método POST do formulário. Na primeira chamada da página, essa propriedade sempre retorna **False**. O método **ObterListaDeClientes** retorna um **DataSet**.

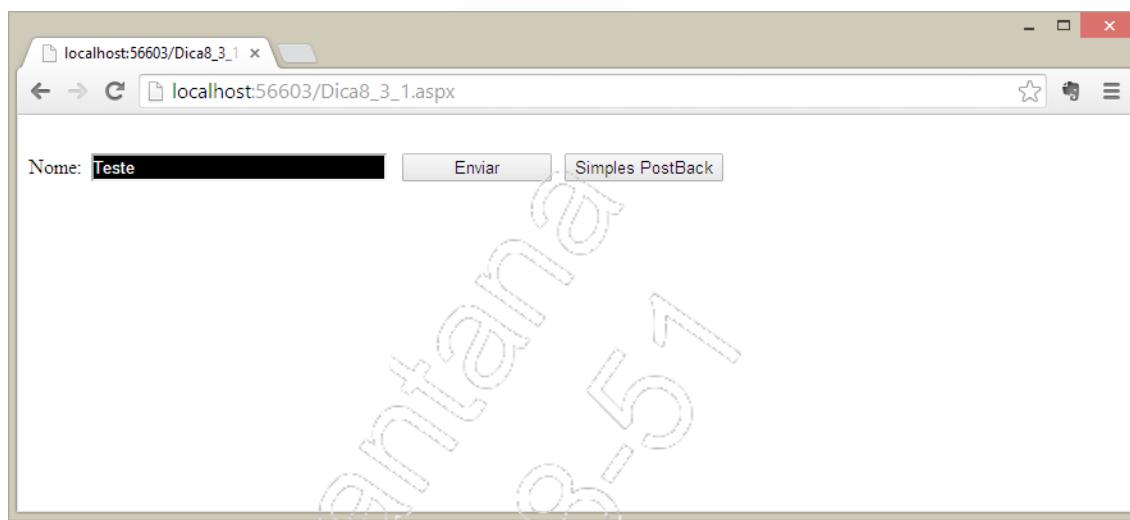
```
protected void Page_Load(object sender, EventArgs e)
{
    if(!Page.IsPostBack)
    {
        listaGridView.DataSource = ObterListaDeClientes();
        listaGridView.DataBind();
    }
}
```

CustomerID	CompanyName	ContactName	ContactTitle	Address	City	Region	PostalCode
ALFKI	Alfreds Futterkiste	Maria Anders	Sales Representative	Obere Str. 57	Berlin		12209
ANATR	Ana Trujillo Emparedados y helados	Ana Trujillo	Owner	Avda. de la Constitución 2222	México D.F.		05021
ANTON	Antonio Moreno Taquería	Antonio Moreno	Owner	Mataderos 2312	México D.F.		05021
AROUT	Around the Horn	Thomas Hardy	Sales Representative	120 Hanover Sq.	London		WA1
BERGS	Berglunds snabbköp	Christina Berglund	Order Administrator	Berguvsvägen 8	Luleå		S-958
BLAUS	Blauer See Delikatessen	Hanna Moos	Sales Representative	Forsterstr. 57	Mannheim		68300
BLONP	Blondesddsl père et fils	Frédérique Citeaux	Marketing Manager	24, place Kléber	Strasbourg		67000
BOLID	Bólido Comidas preparadas	Martin Sommer	Owner	C/ Araquil, 67	Madrid		28021
BONAP	Bon app'	Laurence Lebihan	Owner	12, rue des Bouchers	Marseille		13000
BOTTM	Bottino-Dollar Markets	Elizabeth Lincoln	Accounting Manager	23 Tsawassen Blvd.	Tsawassen	BC	T2F 8

Visual Studio 2015 - ASP.NET com C# Fundamentos

Ao digitar o nome do cliente e clicar em **Enviar**, o evento **Click** do botão é disparado, trocando a cor da fonte e do fundo do TextBox. Repare que os dados da GridView desaparecem, pois o ViewState da página está desligado.

```
protected void botao_Click(object sender, EventArgs e)
{
    nomeTextBox.BackColor = System.Drawing.Color.Black;
    nomeTextBox.ForeColor = System.Drawing.Color.White;
}
```

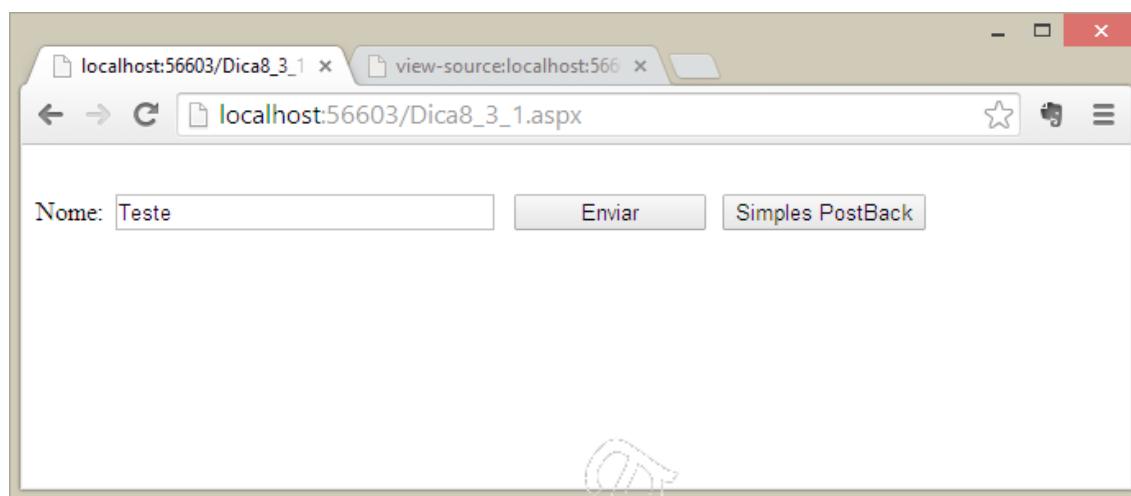


O HTML gerado define a cor do input (**TextBox**) usando CSS e define o atributo **value** com o texto que foi digitado. O HTML a seguir mostra o que foi gerado. Alguns itens do HTML não relevantes para este exemplo foram omitidos.

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

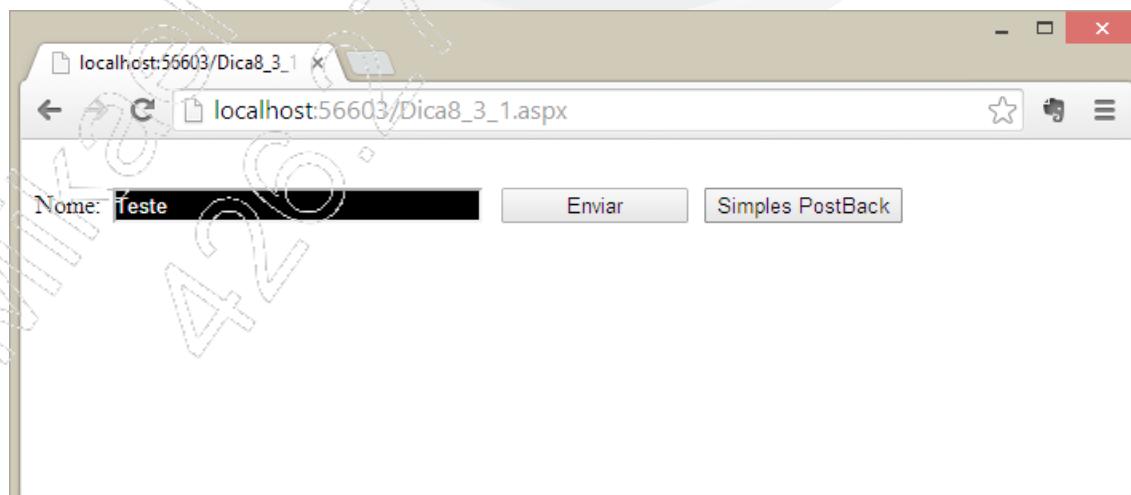
    Nome: 
```

O botão **Simples PostBack** não faz nada além de executar o PostBack do formulário. Repare que o texto do **input (TextBox)** é mantido, mas as cores não. Como o **ViewState** está desligado, apenas a propriedade **value** enviada pelo formulário é preservada.



É possível habilitar e desabilitar o ViewState para apenas alguns controles e não para a página toda. Todo Web Control tem a propriedade **EnabledViewState** porque, assim como a classe **Page**, as classes dos Web Controls são classes derivadas da classe **Control**, onde essa propriedade está definida. Veja o exemplo:

```
<asp:GridView EnableViewState="false" runat="server" />
```



Nesse caso, apenas a GridView perde o conteúdo. Todos os outros controles da página não são afetados por essa configuração.

8.3.2. Use o ViewState (Web Forms)

Este item parece entrar em contradição com o item anterior (desabilite o ViewState), mas o objetivo é deixar claro que é uma boa prática desabilitar esse recurso apenas quando não é necessário manter o estado da página. Por outro lado, o ViewState é um recurso extremamente poderoso para criar páginas interativas, que precisam manter a informação dos controles enquanto o usuário interage com a aplicação.

Quando o usuário necessita inserir um grande número de informações, é melhor dividir o formulário em partes e obter os dados gradativamente, por meio de telas com controles agrupados por assunto. Por exemplo, se for necessário obter detalhadamente as informações de uma pessoa, é interessante utilizar controles que permitam agrupar dados por assunto como informações pessoais, endereço, dados bancários, escolaridade, todos em telas separadas. Os controles mais importantes para essa tarefa são **Panel**, **MultiView** e **Wizard**.

Esses controles têm uma característica em comum: os controles contidos só são renderizados em HTML se estiverem visíveis. Quando não estão visíveis, todas as propriedades são armazenadas no ViewState, estando disponíveis para o processamento no servidor, mas ocupando muito menos espaço do que se estivessem sendo exibidas na página e ocultadas com propriedades de folhas de estilo. Veja um exemplo:

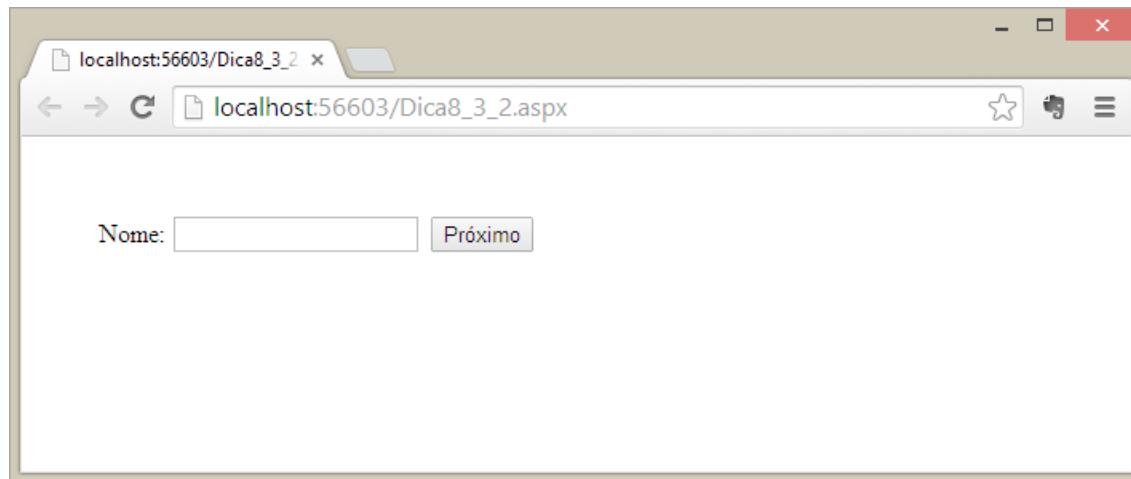
```
<asp:MultiView runat="server" ActiveViewIndex="0">

    <asp:View ID="View1" runat="server">
        <asp:Label runat="server" >Nome:</asp:Label>
        <asp:TextBox runat="server"></asp:TextBox>
        <asp:Button runat="server" Text="Próximo" />
    </asp:View>

    <asp:View ID="View2" runat="server">
        <asp:Calendar runat="server"></asp:Calendar>
    </asp:View>

</asp:MultiView>
```

O primeiro painel cria um Label, um TextBox e um botão.



Como o segundo painel está oculto, o calendário não é gerado. Veja o HTML gerado (algumas tags não relevantes foram omitidas para melhor visualização):

```
<!DOCTYPE html>

<html>
<head></head>
<body>
    <form method="post">
        <input type="hidden" id="__VIEWSTATE" value="/wEPQ4..." />

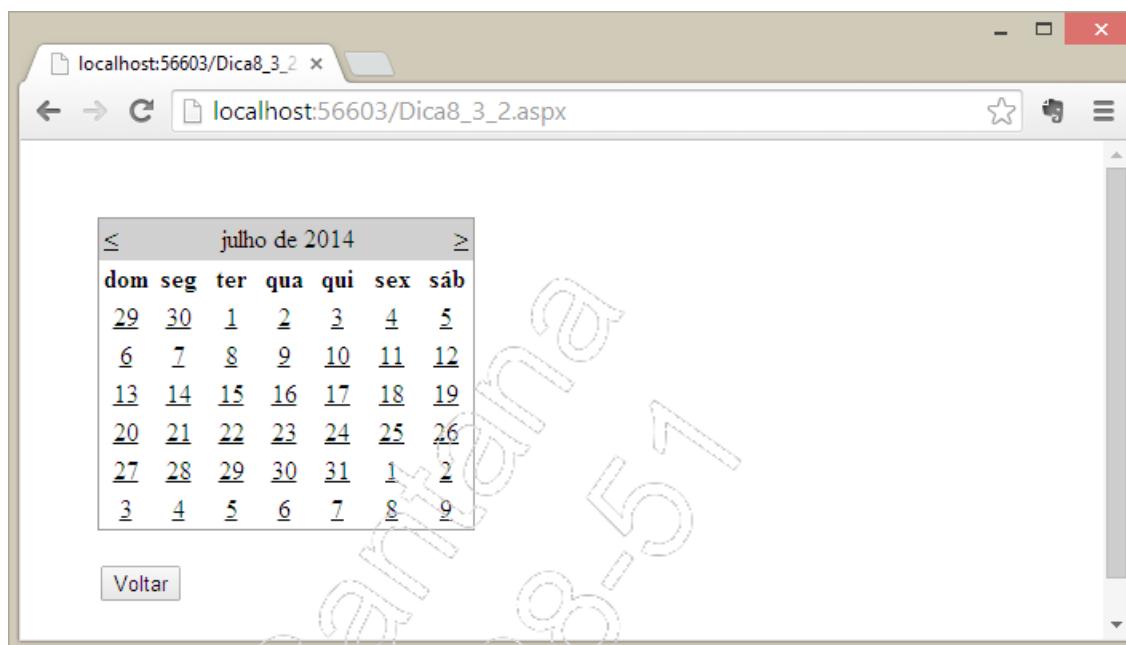
        <span id="nomeLabel">Nome:</span>
        <input name="nomeTextBox" type="text" id="nomeTextBox" />
        <input type="submit" value="Próximo" />

    </form>
</body>
</html>
```

Visual Studio 2015 - ASP.NET com C# Fundamentos

Ao clicar no botão **Próximo**, o código no servidor muda a propriedade **ActiveViewIndex** para 1, fazendo o controle MultiView exibir o segundo painel:

```
protected void proximo_Click(object sender, EventArgs e)
{
    dadosMultiview.ActiveViewIndex = 1;
}
```



Agora, é o primeiro painel que está oculto. O segundo painel é renderizado. Repare que nenhuma informação do primeiro painel foi gerada. Apesar disso, toda informação digitada no TextBox no primeiro painel está disponível. Esses dados estão armazenados no ViewState.

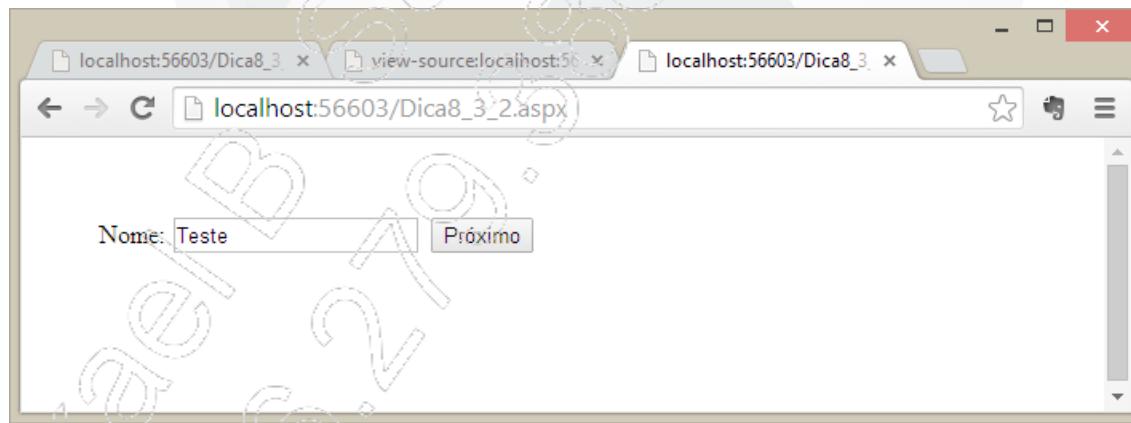
```
<!DOCTYPE html>
<html>
<head></head>
<body>
    <form method="post">

        <input type="hidden" id="__VIEWSTATE"
            value="/wEPDwUJOTYwNTMxMj..." />

        <table cellspacing="0" cellpadding="2" title="Calendário"
            style="border-width:1px; border-style:solid; border-collapse:collapse;"> ... (parte html omitida)...
            <tr><td colspan="7" style="background-color:Silver;"><table
            , '5334')" style="color:Black" title="1 de janeiro">9</a></td></tr>
        </table>

        <input type="submit" value="Voltar" />
    </form>
</body>
</html>
```

Ao clicar no botão **Voltar**, o código no servidor muda a propriedade **ActiveViewIndex** para 0, fazendo o controle MultiView exibir novamente o primeiro painel:



O valor digitado no TextBox, antes de clicar no botão **Próximo** para exibir o calendário, aparece, provando que o ASP.NET está mantendo o estado do controle, mesmo não sendo gerado o HTML.

Nesse caso, o ViewState é extremamente eficiente, gerando apenas o HTML necessário e ainda mantendo a informação disponível de todos os controles da página no ViewState.

No final do processo, todos os dados, visíveis ou não, podem ser obtidos e processados.

8.4. Código

Os próximos tópicos estão relacionados a códigos.

8.4.1. Separar funcionalidades

Um dos princípios de programação que facilitam o desenvolvimento, testes e a implementação de aplicativos é o de separar as funcionalidades. Isso significa que cada parte do seu código, seja ele um método, uma classe ou um componente, deve ter um (e somente um) propósito bem definido.

Esse princípio pode ser seguido em vários níveis. O importante é desenvolver o hábito de criar o código a partir de pequenas funcionalidades e ir conectando-as para atender aos requisitos da solução. Existem vários princípios bem documentados (chamados **Design Patterns**) para cada tipo de situação e ambiente em que o aplicativo vai ser executado.

O objetivo, nesta parte do curso, não é entrar em detalhes de cada um desses Design Patterns, mas sim ter uma visão global de como criar um código mais legível e de mais fácil manutenção usando o princípio da separação das responsabilidades.

É possível criar estruturas no seu código de cinco maneiras: **Métodos, Classes, Componentes, Serviços Windows e Web Services**. Vamos analisar os três primeiros nos próximos subtópicos.

8.4.1.1. Separar por métodos

Uma boa prática para criar User Interfaces (UI) é não colocar dentro dos métodos de eventos funcionalidades que não estejam diretamente relacionadas com a tela. Por exemplo, observe o seguinte método, disparado a partir de um evento **Click** de um botão:

```
void exibirDescontoButton_Click(object sender, EventArgs e)
{
    string estado=estadoTextBox.Text;
    decimal valorCompra=Convert.ToDecimal(valorCompraTextBox.Text);
    decimal desconto, porcentagemDesconto, valorFinal;
    if (string.IsNullOrEmpty(estado))
    {
        mensagemLabel.Text = "O estado deve ser informado";
        return;
    }
    if (valorCompra <= 0 ||
        valorCompra > Convert.ToDecimal(1000000))
    {
        mensagemLabel.Text = "Valor da compra inválido";
        return;
    }
    switch (estado)
    {
        case "SP":
            porcentagemDesconto = 5;
            break;
        case "RJ":
            porcentagemDesconto = 2;
            break;
        default:
            porcentagemDesconto = 0;
            break;
    }
    desconto = valorCompra * (porcentagemDesconto / 100);
    valorFinal = valorCompra - desconto;
    if (desconto == 0) {
        descontoLabel.Text = "Não há desconto.";
    }
    else {
        descontoLabel.Text = desconto.ToString("c");
    }
    valorFinalLabel.Text = valorFinal.ToString("c");
}
```

Qual é o problema desse método? Apesar de estar calculando corretamente o desconto, ele tem muitas responsabilidades. É possível enumerar as seguintes:

1. Obter os valores digitados na interface de usuário;
2. Validar esses valores;
3. Calcular o desconto da compra, segundo as regras de negócio: se o estado onde foi feita a compra for São Paulo, o desconto é de 5%; se o estado for Rio de Janeiro, o desconto é de 2%; e, no restante, não tem desconto;
4. Exibir os valores formatados nos controles. Se não houver desconto, deve aparecer escrito **Não há desconto** e, se houver, deve aparecer o valor do desconto. Havendo ou não desconto, deve ser exibido o **Valor final**.

Definir funcionalidades dentro do código e transferir para métodos é a primeira alternativa para tornar o programa mais claro no seu propósito e mais gerenciável. Esse processo se chama **refatoração**.

O seguinte método pode ser criado para calcular o desconto:

```
private decimal CalcularDesconto(string estado,
decimal valorCompra)
{
    decimal porcentagemDesconto;
    switch (estado)
    {
        case "SP":
            porcentagemDesconto = 5;
            break;
        case "RJ":
            porcentagemDesconto = 2;
            break;
        default:
            porcentagemDesconto = 0;
            break;
    }
    decimal desconto = valorCompra *
(porcentagemDesconto / 100);

    return desconto;
}
```

O evento do botão fica assim:

```
string estado = estadoTextBox.Text;
decimal valorCompra=Convert.ToDecimal(valorCompraTextBox.Text);

decimal desconto, valorFinal;

if (string.IsNullOrEmpty(estado))
{
    mensagemLabel.Text = "O estado deve ser informado";
    return;
}

if (valorCompra <= 0 || valorCompra > Convert.ToDecimal(1000000))
{
    mensagemLabel.Text = "Valor da compra inválido";
    return;
}

desconto = CalcularDesconto(estado, valorCompra);
valorFinal=valorCompra - desconto;

if (desconto == 0)
{
    descontoLabel.Text = "Não há desconto.";
}
else
{
    descontoLabel.Text = desconto.ToString("c");
}
valorFinalLabel.Text = valorFinal.ToString("c");
```

Repare que o método do evento agora não sabe a porcentagem de desconto. Essa responsabilidade foi transferida para outro método. Se a lógica de desconto mudar, o único lugar no código a ser alterado é o método **CalcularDesconto()**. Acabamos de incluir no código o que é chamado de **encapsulamento**.

Outra parte do processo que pode ser isolada é a validação. Podemos criar um método para validar os parâmetros **estado** e **valorCompra** ou incluir a validação dentro do método **Calcular desconto**. Para exemplificar melhor o processo de refatoramento, vamos criar um método para validar.

O método retorna uma lista de string com os erros ou uma lista vazia se não houver erros. Mais uma vez, deixamos a responsabilidade de validar para esse método apenas.

```
Private List<string> ValidarDados(    string estado,
decimal valorCompra)
{
List<string> erros = new List<string>();

if (string.IsNullOrEmpty(estado))
{
erros.Add("O estado deve ser informado");

}

if (valorCompra <= 0 || valorCompra >
Convert.ToDecimal(1000000))
{
    erros.Add("Valor da compra inválido");
}

return erros;
}
```

Outro método será o **ExibirErro**, que recebe uma lista de strings para exibir e cria o código HTML para ser exibido em vermelho. O comportamento dessa parte é muito parecido com o do mecanismo do controle **ValidationSummary**.

```
private void ExibirErro(List<string> erros)
{
var sb = new StringBuilder();
foreach (string erro in erros)
{
    sb.AppendFormat("-{0}<br/>");
}
mensagemLabel.Text = sb.ToString();
mensagemLabel.ForeColor=System.Drawing.Color.Red;

}
```

O método do evento **Click** fica, então, assim:

```
void exibirDescontoButton_Click(object sender,
{
    //Obter Dados
    string estado = estadoTextBox.Text;
    decimal valorCompra=Convert.ToDecimal(valorCompraTextBox.Text);

    //Validar
    var erros = ValidarDados(estado, valorCompra);
    if (erros.Count > 0)
    {
        ExibirErro(erros);
        return;
    }

    //Calcular o Desconto e o Valor Final
    decimal desconto = CalcularDesconto(estado, valorCompra);
    decimal valorFinal = valorCompra - desconto;

    //Exibir
    if (desconto == 0)
    {
        descontoLabel.Text = "Não há desconto.";
    }
    else
    {
        descontoLabel.Text = desconto.ToString("c");
    }
    valorFinalLabel.Text = valorFinal.ToString("c");
}
```

E, finalmente, podemos encapsular a parte que exibe o resultado:

```
void ExibirDesconto(decimal desconto, decimal valorFinal)
{
    if (desconto == 0) {
        descontoLabel.Text = "Não há desconto.";
    }
    else{
        descontoLabel.Text = desconto.ToString("c");
    }
    valorFinalLabel.Text = valorFinal.ToString("c");
}
```

O método do evento **Click**, então, fica da forma exibida a seguir. Repare que é muito mais legível que a primeira versão. Quanto maior a complexidade do programa, maior se mostra essa vantagem.

```
protected void exibirDescontoButton_Click(object sender,
    EventArgs e)
{
    //Obter Dados
    string estado = estadoTextBox.Text;
    decimal valorCompra=Convert.ToDecimal(valorCompraTextBox.Text);

    //Validar
    var erros = ValidarDados(estado, valorCompra);
    if (erros.Count > 0)
    {
        ExibirErro(erros);
        return;
    }

    //Calcular o Desconto e o Valor Final
    decimal desconto = CalcularDesconto(estado, valorCompra);
    decimal valorFinal = valorCompra - desconto;

    //Exibir
    ExibirDesconto(desconto, valorFinal);
}
```

Na versão final, um método virou esses quatro métodos depois do refatoramento:

```
void ExibirDesconto(decimal desconto, decimal valorFinal)
void ExibirErro(List<string> erros)
decimal CalcularDesconto(string estado, decimal valorCompra)
List<string> ValidarDados(string esta, decimal vlCompra)
```

Cada método é responsável por uma única tarefa. Veja a comparação do código original com este. Repare como é muito mais fácil visualizar o que o programa faz e como o código está organizado.

- Refatorado:

```
void exibirDescontoButton_Click(object sender, EventArgs e)
{
    //Obter Dados
    string estado = estadoTextBox.Text;
    decimal valorCompra=Convert.ToDecimal(valorCompraTextBox.Text);

    //Validar
    var erros = ValidarDados(estado, valorCompra);
    if (erros.Count > 0)
    { ExibirErro(erros); return; }

    //Calcular o Desconto e o Valor Final
    decimal desconto = CalcularDesconto(estado, valorCompra);
    decimal valorFinal = valorCompra - desconto;

    //Exibir
    ExibirDesconto(desconto, valorFinal);
}
```

- Original:

```
void exibirDescontoButton_Click(object sender, EventArgs e)
{
    string estado=estadoTextBox.Text;
    decimal valorCompra=Convert.ToDecimal(valorCompraTextBox.Text);
    decimal desconto, porcentagemDesconto, valorFinal;
    if (string.IsNullOrEmpty(estado))
    {
        mensagemLabel.Text = "O estado deve ser informado";
        return;
    }
    if (valorCompra <= 0 ||
        valorCompra > Convert.ToDecimal(1000000))
    {
        mensagemLabel.Text = "Valor da compra inválido";
        return;
    }
    switch (estado)
    {
        case "SP":
            porcentagemDesconto = 5;
            break;
        case "RJ":
            porcentagemDesconto = 2;
            break;
        default:
            porcentagemDesconto = 0;
            break;
    }
    desconto = valorCompra * (porcentagemDesconto / 100);
    valorFinal = valorCompra - desconto;
    if (desconto == 0) { descontoLabel.Text = "Não há desconto."; }
    else {
        descontoLabel.Text = desconto.ToString("c");
    }
    valorFinalLabel.Text = valorFinal.ToString("c");
}
```

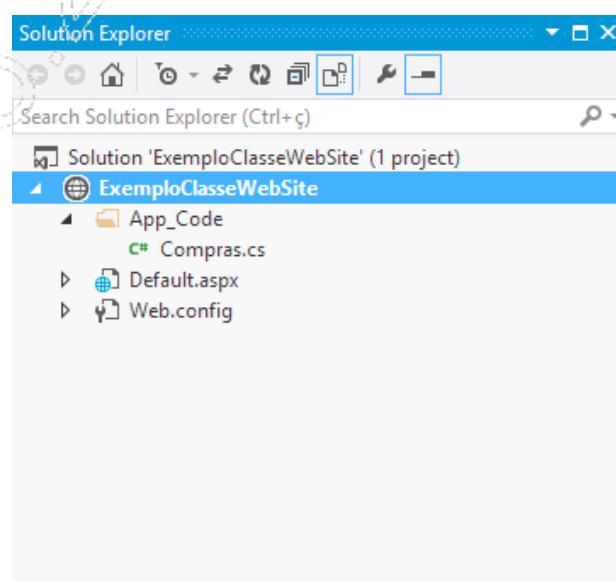
8.4.1.2.Separar por classes

O problema de separar por métodos ocorre quando uma funcionalidade precisa ser usada em outro formulário. Nesse caso, é melhor encapsular os métodos que não fazem parte da interface de usuário em uma classe. Veja o exemplo:

```
//  
//Compras  
//  
public class Compras  
{  
    //  
    // Calcular Desconto  
    //  
    public static decimal CalcularDesconto(string estado,  
                                            decimal valorCompra)  
    {  
        decimal porcentagemDesconto;  
        switch (estado)  
        {  
            case "SP":  
                porcentagemDesconto = 5;  
                break;  
            case "RJ":  
                porcentagemDesconto = 2;  
                break;  
            default:  
                porcentagemDesconto = 0;  
                break;  
        }  
        decimal desconto =  
            valorCompra * (porcentagemDesconto / 100);  
        return desconto;  
    }  
  
    //
```

```
// Validar Dados  
//  
public static List<string> ValidarDados(  
string estado, decimal valorCompra)  
{  
    List<string> erros = new List<string>();  
  
    if (string.IsNullOrEmpty(estado))  
    {  
  
        erros.Add("O estado deve ser informado");  
    }  
  
    if (valorCompra <= 0 || valorCompra >  
        Convert.ToDecimal(1000000))  
    {  
  
        erros.Add("Valor da compra inválido");  
    }  
  
    return erros;  
}  
}
```

Se for usado um projeto Web, a classe pode ser colocada em qualquer lugar. Em um Web site, a classe deve ser colocada na pasta especial **App_Code**.



O código da interface chama os métodos da classe. No exemplo anterior, os métodos na classe **Compras** foram declarados como **static**, portanto não é preciso criar instância dessa classe.

Lembre-se que os métodos declarados com **static** ficam para sempre na memória, até o término da aplicação. Apenas as funcionalidades usadas com frequência devem ser colocadas em métodos estáticos.

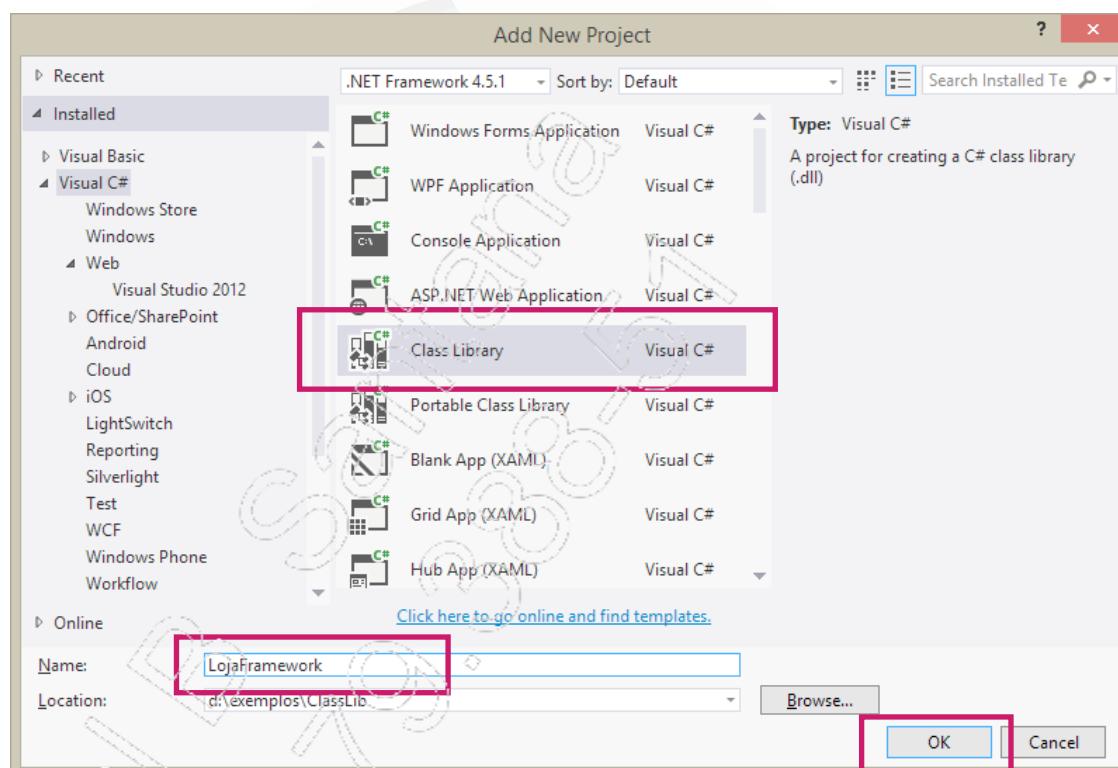
```
protected void exibirDescontoButton_Click(...)  
{  
    //Obter Dados  
    string estado = estadoTextBox.Text;  
    decimal valorCompra=Convert.ToDecimal(valorCompraTextBox.Text);  
  
    //Validar  
    var erros = Compras.ValidarDados(estado, valorCompra);  
  
    if (erros.Count > 0)  
    {  
        ExibirErro(erros);  
        return;  
    }  
  
    //Calcular o Desconto e o Valor Final  
    decimal desconto=Compras.CalcularDesconto(estado, valorCompra);  
  
    decimal valorFinal = valorCompra - desconto;  
  
    //Exibir  
    ExibirDesconto(desconto, valorFinal);  
}
```

Os métodos **exibirErro** e **exibirDesconto** têm como responsabilidade exibir os dados no formulário, portanto não foram movidos para a classe **Compras**.

8.4.1.3.Separar por componente

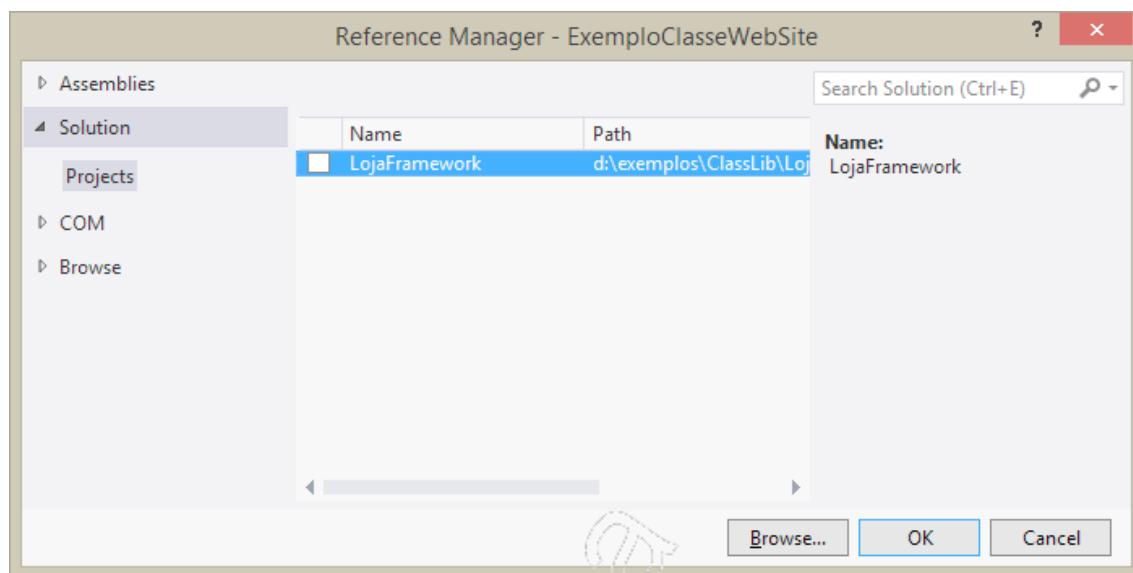
Se for necessário compartilhar a lógica do cálculo do desconto em outros aplicativos, o mais prudente é criar uma biblioteca de classe (**Class Library**). No projeto Web, então, adicionamos uma referência a esse componente para poder utilizá-lo:

1. No menu de contexto da solução, escolha **Add / New Item**;
2. Escolha a opção **Class Library**, nomeie a classe e confirme.



A classe **Compras** pode, então, ficar dentro da biblioteca **LojaFramework**, como no exemplo anterior.

O próximo passo é adicionar uma referência à biblioteca de classe **LojaFramework** no projeto Web atual ou Web site. Usando o menu de contexto do projeto Web, escolha **Add Reference** e escolha o projeto **LojaFramework**.



O código do botão fica desta forma:

```
void exibirDescontoButton_Click(...)
{
    //Obter Dados
    string estado = estadoTextBox.Text;
    decimal valorCompra=Convert.ToDecimal(valorCompraTextBox.Text);

    //Validar
    var erros = LojaFramework.Compras.ValidarDados(estado,
                                                    valorCompra);
    if (erros.Count > 0)
    {
        ExibirErro(erros);
        return;
    }

    //Calcular o Desconto e o Valor Final
    decimal desconto=LojaFramework.Compras.CalcularDesconto(estado,
                                                              valorCompra);
    decimal valorFinal = valorCompra - desconto;

    //Exibir
    ExibirDesconto(desconto, valorFinal);}
}
```

Criar componentes é o melhor caminho para gerar aplicações escaláveis, ou seja, aplicações que podem ser ampliadas e adaptadas com facilidade. Imagine um portal com centenas de usuários on-line no qual uma regra de negócio mudou. Se a lógica da sua aplicação estiver separada por componentes, basta atualizar o componente e copiá-lo para o servidor de produção. Se toda a sua aplicação está dentro do Web site, será necessário recompilar todo o Web site e publicar no servidor, finalizando todas as sessões abertas. Em sites de grande porte, isso é um grande problema e acontecem atualizações frequentemente.

A criação de funcionalidades em diferentes componentes é chamada **arquitetura baseada em camadas** (layered architecture). O modelo mais simples de camadas é o modelo de três camadas: uma para a interface, uma para as regras de negócio e outra para as operações com banco de dados. Essas camadas são conhecidas como **UI** (User Interface), **BLL** (Business Logic Layer) e **DAL** (Data Access Layer).

A camada **UI** pode ser qualquer tipo de interface com o usuário adotando diversas tecnologias: aplicativo Web, Windows Forms, WPF, Silverlight, jQuery, JavaScript, HTML, CSS. Todas essas são tecnologias envolvidas na criação de interfaces.

A camada **BLL** pode ser qualquer tipo de componente: Class Library, Web services, WCF, REST. Todas são tecnologias de componentes que processam informações, mas não destinadas à criação de interfaces com o usuário, apenas a processamento de dados.

A camada **DAL** pode ser qualquer componente ou conjunto de tecnologias para manipular banco de dados: Class Library, ADO, Entity Framework, LINQ, SQL, Hibernate. Na plataforma .NET existe uma variedade muito grande de estratégias para acesso a dados.

8.4.2. Acesso a dados

Os tópicos a seguir estão relacionados ao acesso a dados.

8.4.2.1.String de conexão

O ASP.NET reserva um lugar especial para armazenar string de conexão, que é o arquivo **Web.config**, na seção **ConnectionStrings**. Essa conexão é obtida dentro do código por meio da classe **ConfigurationManager**. A seguir, um exemplo do processo:

- No **Web.config**:

```
<?xml version="1.0"?>
<configuration>
    <connectionStrings>
        <add name="conexao"
            connectionString="data source=... "
            providerName="System.Data.SqlClient"/>
    </connectionStrings>

    <system.web>
        <compilation debug="true" targetFramework="4.5.1"/>
    </system.web>
</configuration>
```

- Dentro do código:

```
using System.Configuration;
...
string conexao = ConfigurationManager
    ..ConnectionStrings["conexao"]
    .ConnectionString;
```

Acontece que armazenar a string de conexão no Web.config nem sempre é a melhor solução, principalmente em aplicações distribuídas. Às vezes, diversos componentes acessam os dados e nem todos estão dentro do ambiente Web. Outro caso comum é a string de conexão mudar de acordo com as regras de negócio.

A solução mais simples para resolver isso é encapsular a string de conexão dentro de um componente ou classe, criando uma propriedade ou método que retorne a string de conexão.

Dentro dessa propriedade ou método, pode-se usar a estratégia de ler do Web.config, como mostrado anteriormente, mas fica mais fácil trocar de estratégia se as necessidades do negócio mudarem. É possível retornar uma string colocada diretamente no código. Pode ser obtida por meio de um serviço, pode ficar em um arquivo de configuração diferente do Web.config, por exemplo.

Se a string de conexão estiver encapsulada dentro dessa propriedade ou método, o modo de obter a string pode mudar, mas nada precisa ser alterado no resto do sistema. O encapsulamento é um recurso muito importante, principalmente em aplicações de grande porte. Veja o exemplo:

```
public static class DbTools
{
    public static string StringDeConexao
    {
        get
        {
            return ConfigurationManager
                ..ConnectionStrings["conexao"]
                .ConnectionString;
        }
    }
}
```

Em qualquer lugar, dentro do código, que tenha acesso a essa classe:

```
var cn = new SqlConnection(DbTools.StringDeConexao);
```

8.4.2.2. Sempre fechar a conexão

O acesso a banco de dados consome muitos recursos do sistema. A melhor estratégia para acesso a dados é abrir a conexão com o servidor, executar os comandos, fechar a conexão e liberar a memória. O principal é garantir que a conexão será finalizada. Portanto, sempre que se conectar no banco, o código deve garantir esse processo. Existem duas maneiras: usando **using** e o bloco **try..catch**.

- **Using**

O bloco de código definido em C# com a palavra **using** garante que a variável que armazena o objeto vai liberar os recursos usados por esse objeto. Isso é feito por meio da interface **IDisposable**, que define um método chamado **Dispose()**. O bloco **using** só pode ser usado em classes que implementam essa interface. Praticamente todas as classes que acessam arquivos (**StreamWriter**, **DbConnection**) implementam essa interface. Veja o exemplo:

```
using (var cn = new SqlConnection(DbTools.StringDeConexao))
{
    cn.Open();
    // operações no banco..
    cn.Close();
}
```

A chamada para o método **Close()**, nesse caso, é opcional. Ao terminar o bloco **using**, será chamado o método **Dispose()**, que fecha a conexão com o banco também. A diferença é que o **Dispose()**, além de fechar a conexão, libera a variável para o **Garbage Collector**. Isso acontece porque o **using** cria uma variável de escopo de bloco, que não existe fora dele. O código, então, mais enxuto, fica assim:

```
using (var cn = new SqlConnection(DbTools.StringDeConexao))
{
    cn.Open();
    //operações no banco..
}
```

- Try..catch

Esta é outra maneira de garantir que a conexão será fechada. É um pouco mais trabalhoso que o bloco **using**, mas fica embutida dentro de controle de exceções.

```
SqlConnection cn = null;
try
{
    cn = new SqlConnection(DbTools.StringDeConexao);
    cn.Open();

    //Operações com o banco...
}

catch (SqlException)
{
    //Tratamento do erro..
}

finally
{
    if (cn != null)
    {
        cn.Close();
    }
}
```

Observe que a string de conexão deve ser declarada fora do bloco **try..catch** para que esteja dentro da visibilidade do bloco **finally**. Deve-se, também, tomar o cuidado de verificar se a variável foi instanciada, pois pode ter havido um erro na criação do objeto, impedindo a instância do objeto de ser carregada na variável.

8.4.3. DataReader, DataSet e Generics

Esta é uma decisão de estratégia muito importante porque pode impactar na performance e na integração dos dados do aplicativo.

Existem três tipos de dados frequentemente utilizados nos métodos que retornam informações do banco de dados. São eles:

- **DataReader**

Características:

- Conectado com o banco;
- A conexão deve ficar aberta enquanto os dados são lidos;
- Somente leitura;
- Cursor apenas avança.

Prós:

- Extremamente rápido;
- Ocupa pouca memória (um único registro na memória por vez).

Contras:

- Não é fortemente tipado;
- Não pode ser usado em Web Controls para paginação automática;
- Não é navegável;
- Não pode ser usado para alterar dados.

Melhor uso:

- Relatórios;
- Grandes quantidades de dados.

- **DataSet**

Características:

- Desconectado com o banco;
- Armazenado na memória;
- Navegável.

Prós:

- Total suporte dos componentes de tela;
- Flexível, pois não tem estrutura fixa;
- Exige pouca programação.

Contras:

- Não é fortemente tipado (existe também o DataSet tipado, mas está sendo substituído por novas tecnologias);
- Pode ocupar muita memória.

Melhor uso:

- Relatórios sem estrutura fixa;
- Pouca quantidade de dados;
- Necessidade de manipular dados em blocos;
- Exportação de dados para Excel ou CSV.

- **Classes (Generics)**

Características:

- Desconectado com o banco;
- Armazenado na memória.

Prós:

- Extremamente rápido;
- Fortemente tipado;
- Total suporte de componentes (se forem implementadas as interfaces corretas);
- Flexível;
- Diversos frameworks facilitam a programação.

Contras:

- Exige mais programação (se não usar frameworks);
- Pode ocupar muita memória.

Melhor uso:

- Definição do modelo de dados da aplicação;
- Transferência de dados entre aplicativos;
- Possibilidade de passar informações entre camadas.

A decisão de qual modelo usar depende de diversos fatores. De um modo geral, é possível iniciar a decisão tendo alguns conceitos em mente:

- **DataReader** é sempre uma escolha a ser considerada quando performance é uma exigência importante ou quando a quantidade de dados a ser manipulada é muito grande;
- **DataSet** é uma boa estratégia quando os dados retornados não são fixos. Por exemplo, relatórios em que o cliente escolhe quais campos quer ver;
- **Classes** é a estratégia correta quando o objetivo é modelar os dados da aplicação, criar telas de manipulação de dados e quando se deseja ter controle total da lógica da aplicação para não ficar explicitamente ligado em um banco de dados.

Pontos principais

Atente para os tópicos a seguir. Eles devem ser estudados com muita atenção, pois representam os pontos mais importantes do capítulo.

- Use o Web Control **Label** apenas se deseja formatar o conteúdo. Para exibir textos simples, use o controle **Literal**;
- Sempre associe o Label que representa uma legenda com o TextBox que representa o campo. Para isso, use a propriedade **AssociatedControlId**;
- Use **HtmlEncoding** se for exibir dados que tenham caracteres especiais, como > ou <;
- Use folhas de estilo para formatar formulários e não tabelas. Tente sempre fazer a tela se adaptar ao tamanho disponível no dispositivo;
- Faça menus com as tags **UL**, **LI** e formato de folhas de estilo;
- Se a interface do usuário não faz uso do PostBack, desligar o ViewState faz a página carregar mais rápido;
- Os Web Controls **MultiView** e **Wizard** fazem um bom uso do **ViewState**. Use-os sempre que o formulário for composto de muitos campos;
- Ao codificar, separe as funcionalidades em métodos, classes ou componentes. Cada parte do seu código deve ter sempre uma única finalidade;
- Crie uma classe e armazene a string de conexão com uma propriedade estática dessa classe;
- Sempre que abrir uma conexão com o banco, feche-a usando a estrutura **using** ou o bloco **try..catch**;
- Use **DataReader** quando precisar ler dados diretamente do banco. Funciona com qualquer banco e com qualquer quantidade de dados;
- Use **DataSet** quando precisar paginar informações do banco usando os recursos dos controles Web e principalmente quando os dados que serão exibidos são desconhecidos;
- Use classes personalizadas para definir o modelo de dados do seu sistema, para ter melhor performance e para ter controle total dos dados que trafegam no aplicativo.

8

Boas práticas em ASP.NET

Teste seus conhecimentos

Mikael
Santana
426.293.8057



IMPACTA
EDITORA

1. Qual é o atributo do Web Control Label que o associa a outro controle, como TextBox ou DropDownList?

- a) Text
- b) ID
- c) FieldID
- d) AssociatedControlID
- e) ContentPlaceHolder

2. Das alternativas a seguir, qual apresenta a maneira mais flexível para criar menus?

- a) Dentro de uma tabela (table, tr, td).
- b) Cada item do menu seguido de um
.
- c) Dentro da tag <HEAD>.
- d) Usando listas (ul, li).
- e) Usando metadados.

3. Qual é o nome do controle oculto que armazena as informações de estado da página nos Web Forms?

- a) __VIEWSTATE
- b) Session
- c) HiddenControl
- d) Post
- e) Application

4. Como se chama o processo de melhorar a estrutura interna do código para torná-lo mais claro no seu propósito e de mais fácil manutenção?

- a) Compilação
- b) Reestruturação
- c) Codificação
- d) Recolocação
- e) Refatoração

5. Em qual caso é aconselhável desligar o ViewState?

- a) Quando se usa MVC.
- b) Quando a página não faz PostBack.
- c) Quando a página é do tipo Web Page.
- d) Quando a página tem poucos controles.
- e) Quando a página tem muitos controles.

8

Boas práticas em ASP.NET

Mãos à obra!

Mikael
426.273.8857



IMPACTA
EDITORA

Laboratório 1

Objetivos:

- Criar um programa que gere gráficos;
- Usar o componente **Chart**;
- Preencher listas com enumeradores;
- Propriedade **AutoPostBack**;
- Propriedade **EnableViewState**.



1. Crie um novo projeto Web vazio chamado **Cap08Lab01**;
2. Adicione um Web Form chamado **Default.aspx**;
3. Adicione um tema chamado **padrao**;
4. Adicione uma folha de estilos chamada **estilos.css**;

5. Configure no Web.config o tema **padrao** como o tema das páginas;

```
<?xml version="1.0"?>

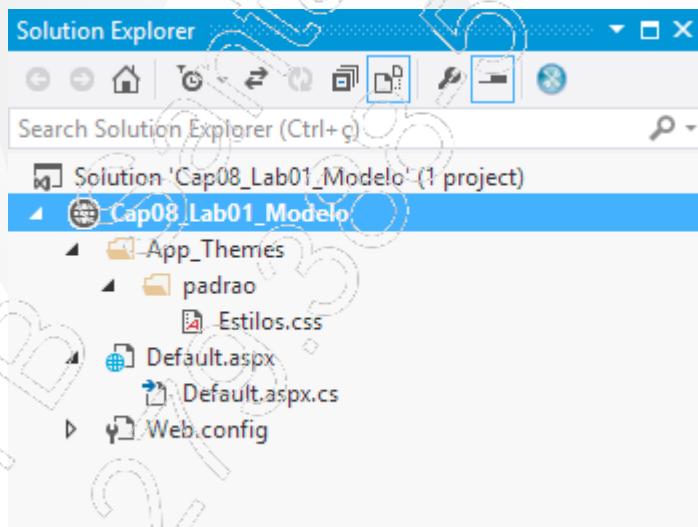
<configuration>

    <system.web>
        <pages theme="padrao"></pages>
        <compilation debug="false" targetFramework="4.5.1" />
        <httpRuntime targetFramework="4.5.1" />

    </system.web>

</configuration>
```

6. O projeto deve estar com estes arquivos:



7. Na folha de estilos, configure os estilos das páginas:

```
* {
    margin:0px;
}

body {
    font-family:'Century Gothic';
    margin:0px;
}
```

8. Na página **default.aspx**, insira a divisão do cabeçalho:

```
<!DOCTYPE html>

<html xmlns="http://www.w3.org/1999/xhtml">
  <head runat="server">
    <title></title>
  </head>
  <body>
    <form id="form1" runat="server">

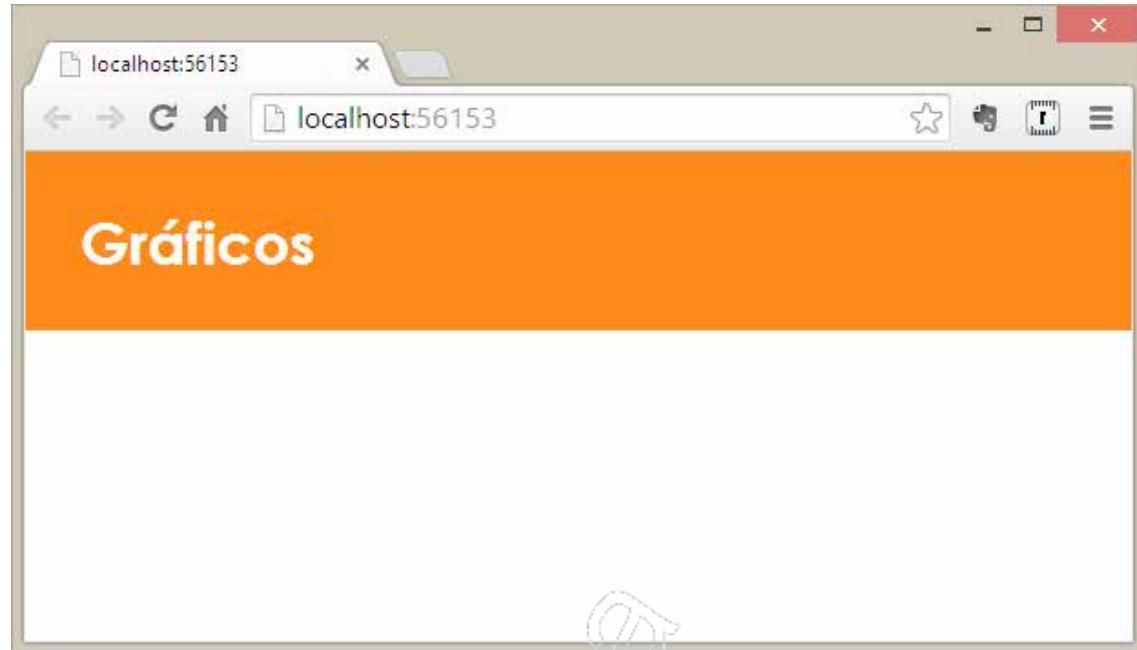
      <div id="topo">
        <h1>Gráficos</h1>
      </div>

    </form>
  </body>
</html>
```

9. Na folha de estilos, insira o código CSS para essa divisão:

```
* {
  margin:0px;
}
body {
  font-family:'Century Gothic';
  margin:0px;
}
#topo {
  padding:30px;
  background-color:#ff6a00;
  color:#fff;
  margin-bottom:40px;
}
```

10. Visualize o resultado:



11. Insira as divisões para o menu e a mensagem e o label para as mensagens:

```
<div id="topo">
    <h1>Gráficos</h1>
</div>

<div id="dadosGrafico">
    <div class="formGrafico">
        </div>
    <asp:Label CssClass="mensagem"
        ID="mensagemLabel"
        runat="server">
        </asp:Label>
    </div>
```

Visual Studio 2015 - ASP.NET com C# Fundamentos

12. Adicione os estilos para a área de menu e a mensagem de erro:

```
#dadosGrafico {  
  
    max-width:300px;  
    float:left;  
    margin-top:30px;  
    margin-left:30px;  
}  
  
.formGrafico {  
  
    border-radius:5px;  
    background-color:#ffd800;  
    padding:20px;  
}  
  
.mensagem {  
  
    display:block;  
    float:left;  
    padding:5px;  
    margin-top:20px;  
}
```

13. Visualize a página:



14. Insira os itens de tela para configurar o tipo do gráfico e manipular os valores dentro da div com o atributo class **formGrafico**:

```
<asp:Label CssClass="legenda"  
    runat="server"  
    Text="Tipo de gráfico:>  
</asp:Label>  
  
<div class="campo">  
  
    <asp:DropDownList CssClass="campoInLine"  
        ID="tipoDropDownList"  
        runat="server"  
        AutoPostBack="True">  
    </asp:DropDownList>  
  
    <asp:CheckBox runat="server"  
        ID="tdCheckBox"  
        AutoPostBack="true"  
        Text="3D"  
        Checked="true" />  
</div>  
  
<asp:Label CssClass="legenda"  
    ID="descricaoLabel"  
    runat="server"  
    Text="Legenda:>  
</asp:Label>  
  
<asp:TextBox CssClass="campo"  
    ID="legendaTextBox"  
    runat="server">  
</asp:TextBox>
```

Visual Studio 2015 - ASP.NET com C# Fundamentos

```
<asp:Label CssClass="legenda"
           ID="valorLabel"
           runat="server"
           Text="Valor:">
</asp:Label>

<asp:TextBox   CssClass="campo"
               ID="valorTextBox"
               runat="server">
</asp:TextBox>

<div class="botoes">

    <asp:Button CssClass="botao"
                ID="incluirButton" runat="server"
                Text="Adicionar" />

    <asp:Button   CssClass="botao"
                  ID="excluirButton"
                  runat="server"
                  Text="Excluir" />

    <asp:Button   CssClass="botao"
                  ID="limparTudoButton"
                  runat="server"
                  Text="Limpar" />
</div>
```

15. Defina os estilos do formulário:

```
.campo {
    display:block;
    margin-bottom:10px;
    min-width:100px;
}
.campoInLine {
    display:inline-block;
    min-width:150px;
    margin-right:10px;
}

.legenda {
    display:block;
}
```

```
.botao {  
    display:inline-block;  
    width:80px;  
    border-radius:3px;  
    border:1px solid #ff6a00;  
    background-color:#ff6a00;  
    padding:5px;  
    color:#fff;  
}  
  
.botoes {  
    margin-top:40px;  
    display:block;  
}
```

16. Visualize o resultado:



Visual Studio 2015 - ASP.NET com C# Fundamentos

17. Na página default.aspx, insira a tag para conter o gráfico:

```
<div id="dadosGrafico">
    <div class="formGrafico">
        ...
    </div>
```

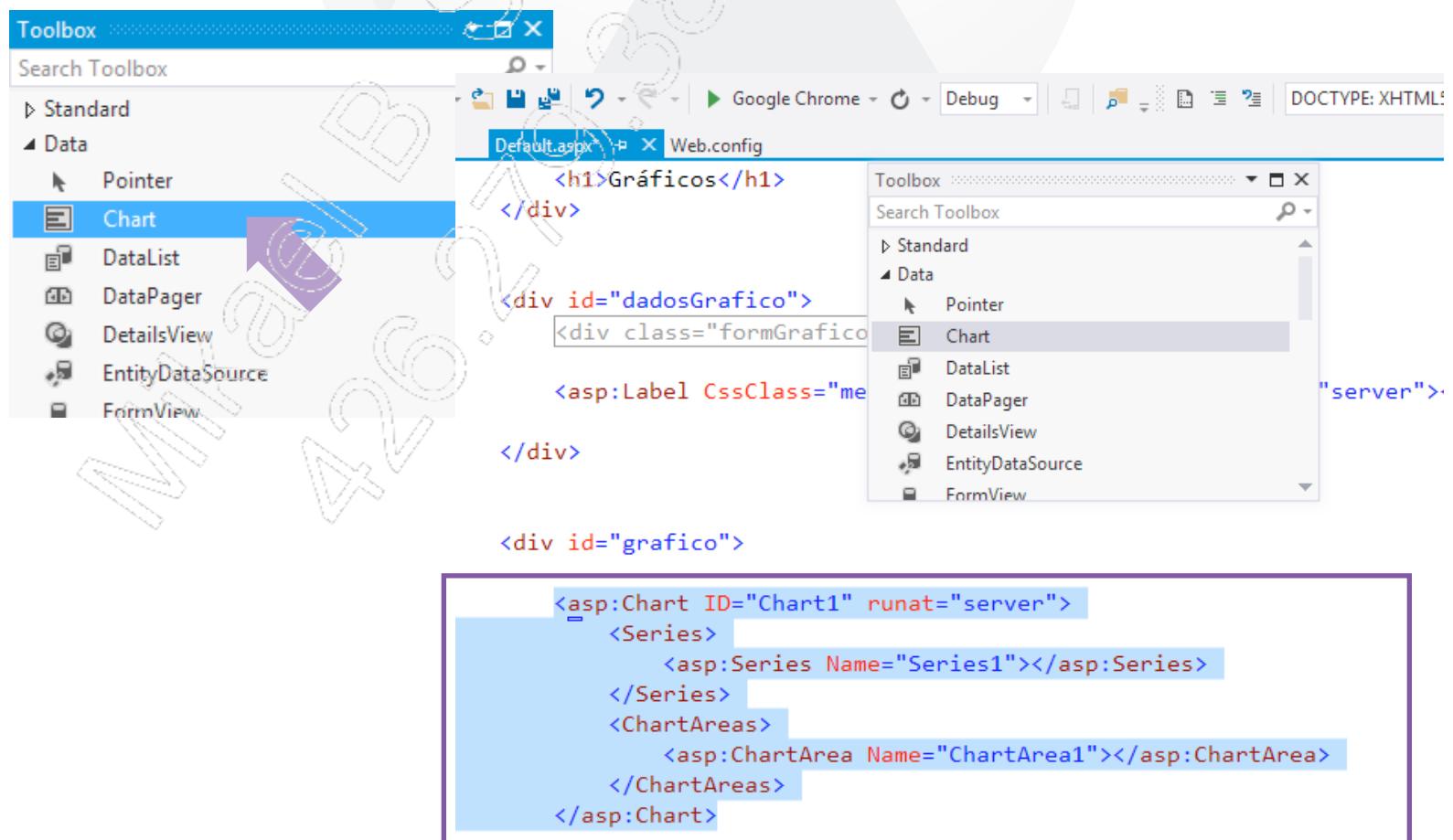
```
<div id="grafico">

</div>
```

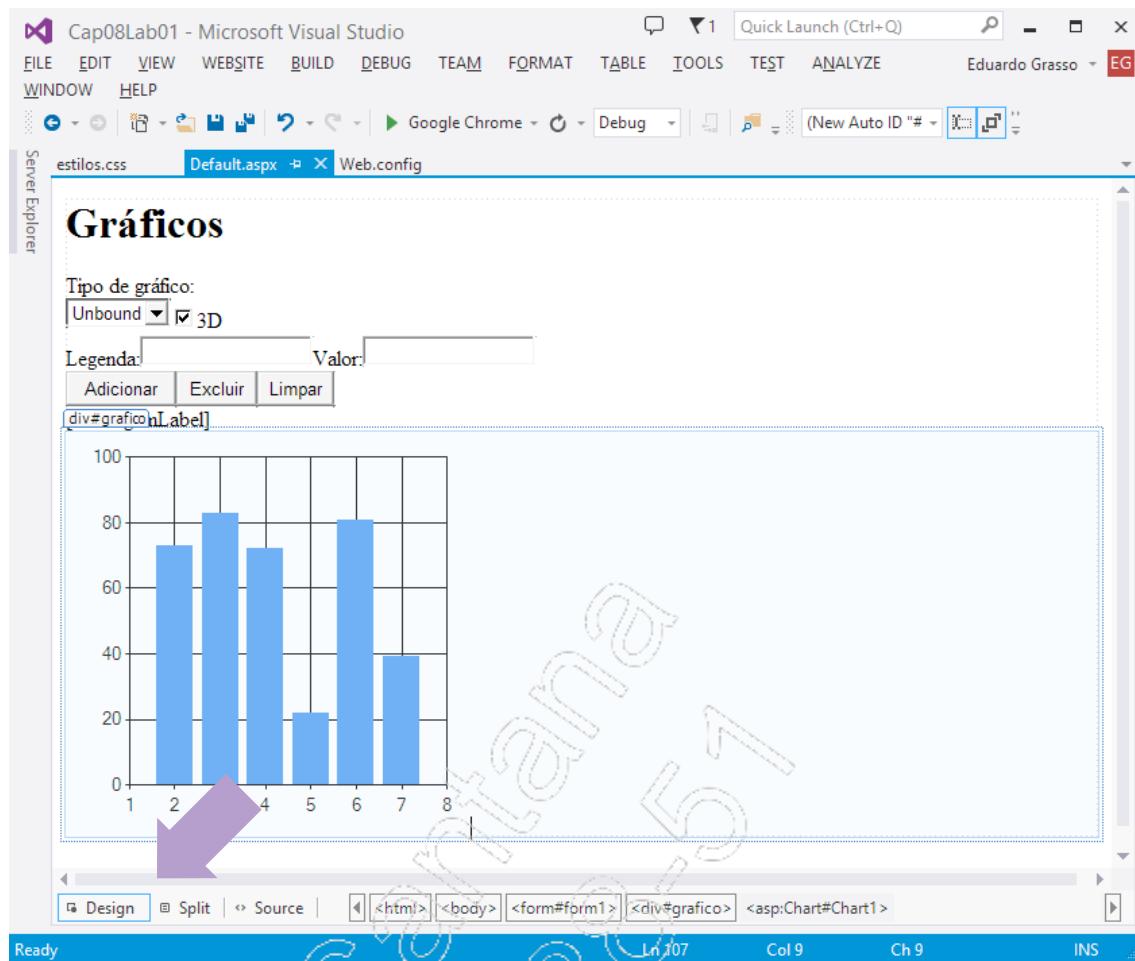
18. Na folha de estilo, adicione o estilo para a moldura do gráfico:

```
#grafico {
    padding:30px;
    float:left;
}
```

19. Dentro da div com id **gráfico**, arraste o controle **Chart**, que está dentro da guia **Data**:



20. Mude para o modo **Design**. Este passo é necessário para carregar as bibliotecas;



21. Altere alguns parâmetros do gráfico, acrescentando as propriedades **EnableViewState** e **Width**:

```
<asp:Chart ID="Chart1" runat="server"
    EnableViewState="true" Width="800px" >

    <Series>
        <asp:Series Name="Series1"></asp:Series>
    </Series>
    <ChartAreas>
        <asp:ChartArea Name="ChartArea1">
        </asp:ChartArea>
    </ChartAreas>
</asp:Chart>
```

22. Insira o rodapé após o div **gráfico** e os estilos:

```
<div id="rodape">
    <p>
        Desenvolvido para estudar
        ASP.NET (c) 2014 Impacta
    </p>
</div>
```

```
#rodape {
    position:fixed;
    bottom:10px;
    left:10px;
}
```

23. Visualize a página:



24. No code-behind, importe o seguinte namespace: **System.Web.UI.DataVisualization.Charting** e crie o método **AtualizarGrafico**:

```
using System.Web.UI.DataVisualization.Charting;  
  
...  
  
//  
// Atualizar Gráfico  
//  
private void AtualizarGrafico()  
{  
    Chart1.ChartAreas[0].Area3DStyle.Enable3D =  
        tdCheckBox.Checked;  
  
    Chart1.Series[0].ChartType =  
        (SeriesChartType)Enum.Parse(typeof(SeriesChartType),  
            tipoDropDownList.SelectedValue);  
  
    Chart1.Visible = Chart1.Series[0].Points.Count > 0;  
}
```

25. No evento **Page_Load**, altere o método para criar um gráfico de exemplo:

```
protected void Page_Load(object sender, EventArgs e)  
{  
    if (!Page.IsPostBack)  
    {  
  
        tipoDropDownList.DataSource =  
            Enum.GetNames(typeof(SeriesChartType));  
  
        tipoDropDownList.DataBind();  
  
        tipoDropDownList.SelectedValue = "Column";  
  
        Chart1.Series[0].Points.Add(10).AxisLabel = "Jan";  
        Chart1.Series[0].Points.Add(15).AxisLabel = "Fev";  
        Chart1.Series[0].Points.Add(5).AxisLabel = "Mar";  
        Chart1.ChartAreas[0].Area3DStyle.WallWidth = 10;  
  
        Form.DefaultButton = "incluirButton";  
  
        AtualizarGrafico();  
    }  
}
```

26. Teste o programa:

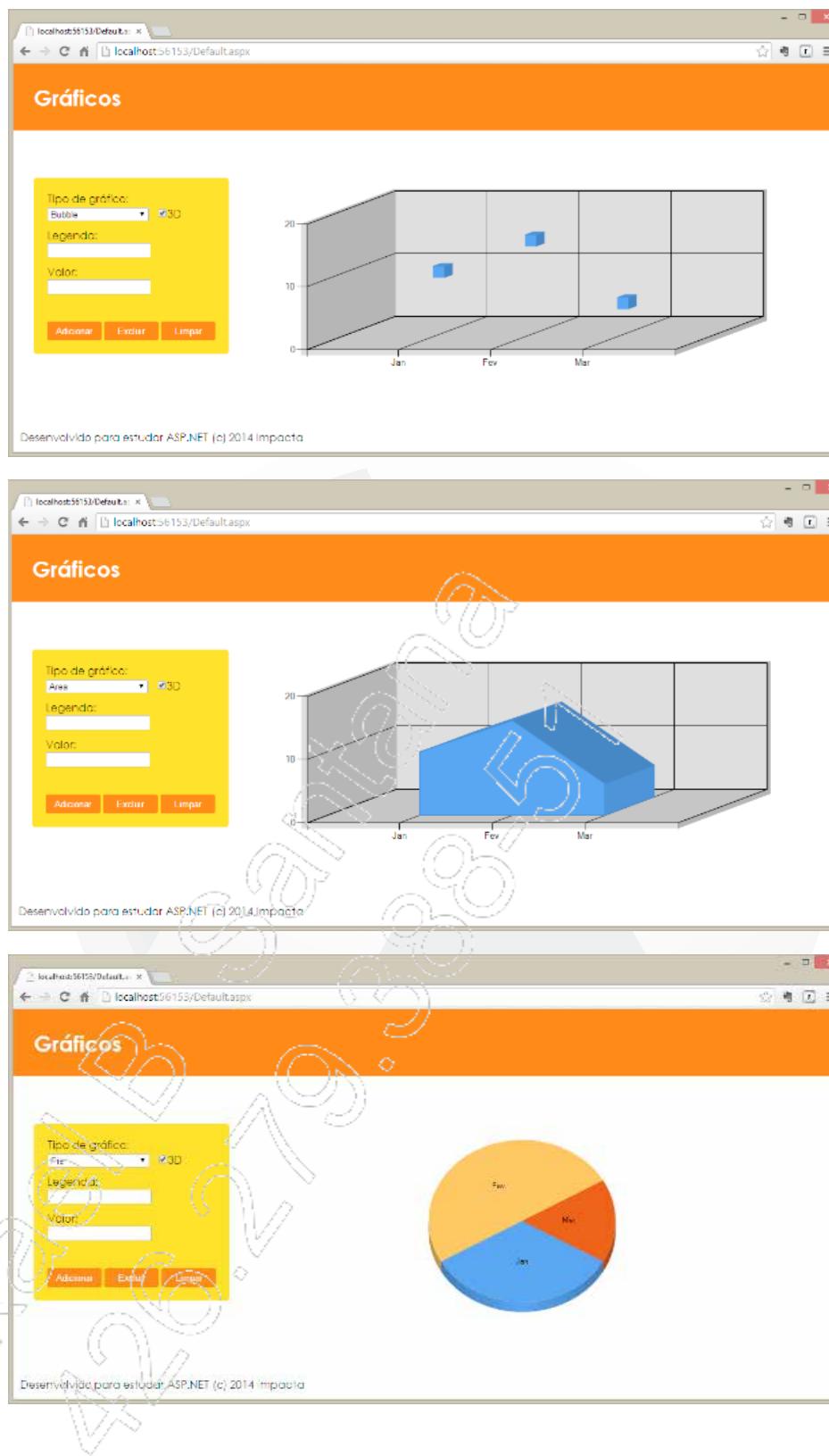


27. Defina o evento da DropDownList para alterar o tipo de gráfico:

```
protected void DropDownList1_SelectedIndexChanged(
    object sender, EventArgs e)
{
    AtualizarGrafico();
}

<asp:DropDownList
    CssClass="campoInLine"
    runat="server"
    OnSelectedIndexChanged=
        "tipoDropDownList_SelectedIndexChanged">
</asp:DropDownList>
```

28. Teste os diferentes os tipos de gráficos:



29. Crie o código do botão Limpar:

```
protected void limparTudoButton_Click(
    object sender, EventArgs e)
{
    mensagemLabel.Text = string.Empty;
    Chart1.Series[0].Points.Clear();
    AtualizarGrafico();
}
```

30. Crie o código do botão Incluir:

```
protected void incluirButton_Click(
    object sender, EventArgs e)
{
    mensagemLabel.Text = string.Empty;
    double valor = 0;
    bool ok = double.TryParse(
        valorTextBox.Text, out valor);
    if (!ok)
    {
        mensagemLabel.Text = "Digite um número válido";
        return;
    }

    Convert.ToDouble(valorTextBox.Text);
    string legenda = legendaTextBox.Text;

    var d = new DataPoint();
    d.IsValueShownAsLabel = true;
    d.AxisLabel = legenda;
    d.LabelToolTip = legenda;
    d.YValues = new double[] { valor };
    Chart1.Series[0].Points.Add(d);

    AtualizarGrafico();
    legendaTextBox.Focus();
}
```

31. Crie o código do botão Excluir:

```
protected void excluirButton_Click(
    object sender, EventArgs e)
{
    double valor = 0;
    bool ok = double.TryParse(
        valorTextBox.Text, out valor);
    if (!ok)
    {
        mensagemLabel.Text = "Digite um número válido";
        return;
    }

    mensagemLabel.Text = string.Empty;

    string legenda = legendaTextBox.Text;

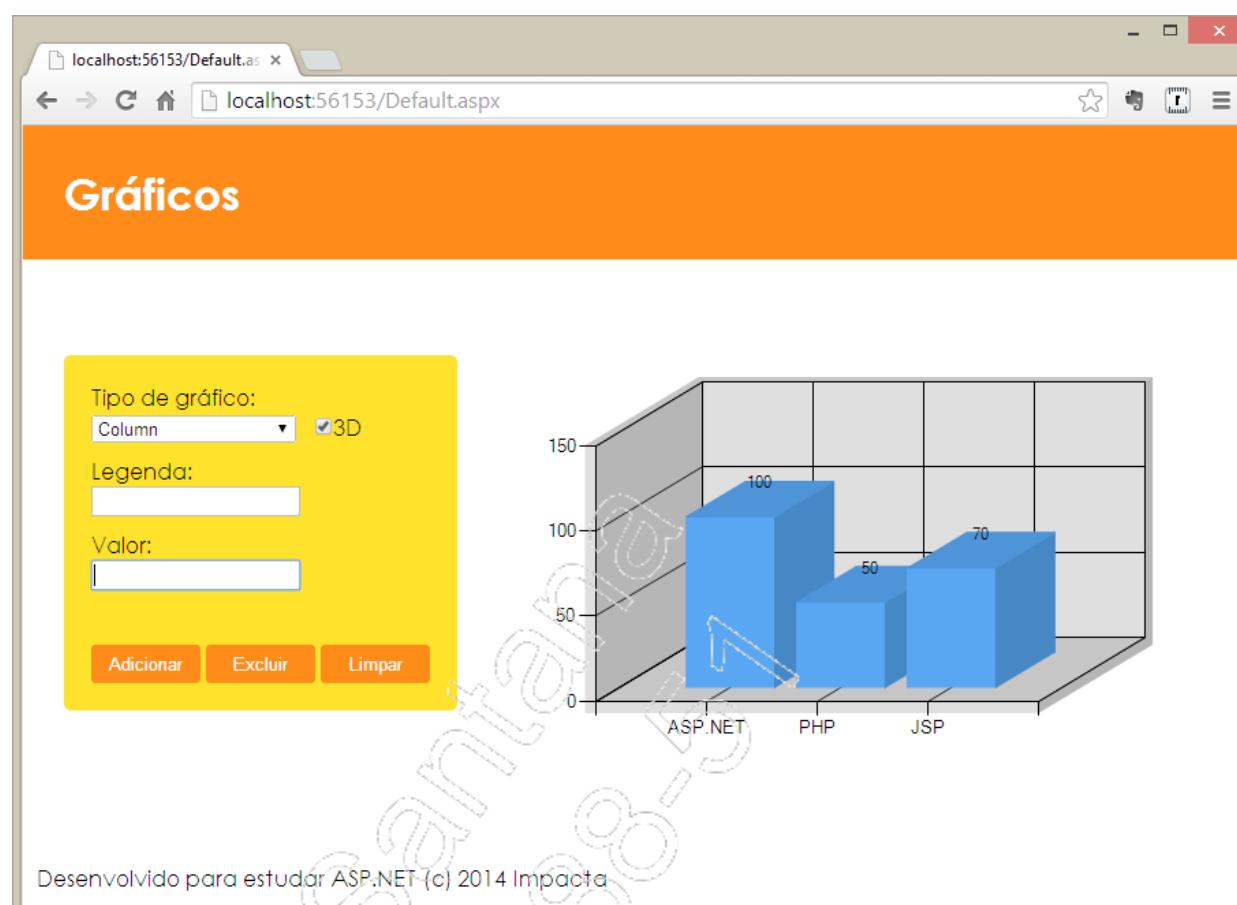
    DataPoint dataPointExcluir = Chart1.Series[0].Points
        .AsEnumerable<DataPoint>()
        .Where(m => m.AxisLabel == legenda)
        .FirstOrDefault();

    if (dataPointExcluir != null)
    {
        Chart1.Series[0].Points.Remove(dataPointExcluir);
    }

    AtualizarGrafico();
}
```

Visual Studio 2015 - ASP.NET com C# Fundamentos

32. Teste o programa usando o botão **Limpar**, digite as legendas e valores de outros itens, clicando em **Adicionar**. Para excluir, digite no campo **Legenda** o item a ser excluído e clique em **Excluir**.



9

AJAX

- ✓ Como funciona o AJAX;
- ✓ AJAX com JavaScript;
- ✓ AJAX com jQuery;
- ✓ Uso de AJAX com Web Forms;
- ✓ Exemplo de uso;
- ✓ AJAX com MVC;
- ✓ AJAX Control Toolkit.



IMPACTA
EDITORA

9.1. Introdução

AJAX é um modelo de programação Web em que a aplicação cliente, quando precisa atualizar a página, requisita ao servidor apenas a informação necessária, e não a página toda, como no modelo tradicional.

Esse processo reduz bastante a quantidade de informação que é transferida pela rede, reduzindo o tempo de resposta do servidor e, com isso, melhorando muito a experiência do usuário, que interage pela página como se estivesse usando um aplicativo desktop.

A sigla **AJAX** significa **Asynchronous JavaScript and XML** (JavaScript e XML não sincronizado). JavaScript é linguagem de programação cujos programas são executados no navegador; XML é o formato da mensagem transmitida entre o cliente e o servidor; e não sincronizado é o modo como as solicitações são feitas, sem esperar uma resposta e capturando o resultado posteriormente num processo chamado **CallBack**.

Em 2005, muitos programadores e Web designers utilizavam diferentes técnicas para tornar melhor a experiência do usuário quando navegava pelas páginas de um site ou aplicativo Web. Nessa época, Jesse James Garrett, um programador de São Francisco, escreveu um artigo chamado **AJAX: A New Approach to Web Applications** (AJAX: uma nova abordagem para aplicações Web), em que criava esse termo.

Jesse usava a palavra AJAX para resumir o conjunto de tecnologias utilizadas para criar aplicações Web para seus clientes. O artigo ainda pode ser visto pelo site **WayBackMachine**, que armazena páginas da Web para registro histórico, pelo seguinte endereço: <https://web.archive.org/web/20080702075113/http://www.adaptivepath.com/ideas/essays/archives/000385.php>.

The screenshot shows a Wayback Machine interface with a green header bar containing links for 'blog', 'reports', 'latest essay', 'newsletter', 'essay archives' (which is highlighted in blue), and 'reading list'. The main content area features a portrait of Jesse James Garrett, the author of the essay. The title 'Ajax: A New Approach to Web Applications' is displayed above the text. The text discusses the evolution of web interaction design, comparing it to desktop software and highlighting the asynchronous nature of Ajax. To the right, a sidebar titled 'Recent Essays' lists several other essays by different authors, such as 'Project Management for Creative Teams: Art and Science' and 'Kate Discusses the Role of Design in Business with Nathan Shedroff'. The bottom of the page includes a 'Printer-friendly version' link and a 'Newsletter' sign-up section.

No artigo, Jesse explica que o termo AJAX descreve parte da tecnologia. Para criar uma experiência real, são utilizadas, entre outras, as seguintes tecnologias: JavaScript, XML, HTML, CSS, DOM, JSON, XMLHttpRequest e XSLT, isso sem contar tecnologias do lado do servidor, como PHP, ASP.NET, SQL e JSP. AJAX é apenas uma palavra fácil de lembrar.

9.2. Como funciona o AJAX

Uma boa maneira de entender o funcionamento AJAX é a comparação do modelo tradicional com o assíncrono.

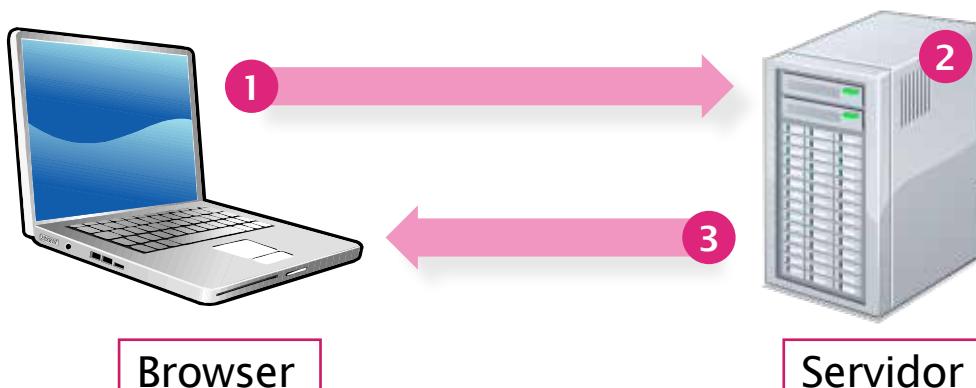
- **Web do modo tradicional**

1. O browser faz uma requisição;
2. O servidor recebe e processa a solicitação e envia uma resposta, que é a página HTML;
3. O browser recebe, processa e exibe a página;
4. Se a página precisar ser alterada, começa novamente, desde o passo 1.



- **Web usando AJAX**

1. O browser faz uma requisição (request);
2. O servidor recebe e processa a solicitação e envia uma resposta (response), que é a página HTML inteira;
3. O browser recebe a página e a exibe;



4. Se a página precisar ser alterada, o browser executa um programa em JavaScript;
5. O programa em JavaScript chama um Web Service e não espera uma resposta do servidor. O processamento é liberado imediatamente e uma função fica aguardando o retorno;
6. O navegador continua executando suas funções;
7. A mensagem de retorno (CallBack) é disparada quando a resposta chega do servidor. O programa em JavaScript atualiza a parte da página que precisa ser modificada.



As tecnologias envolvidas para criar uma página usando AJAX são as seguintes:

- Veja tecnologias do lado do cliente:
 - **HTML**: É a página que vai ser atualizada;
 - **JavaScript**: É a linguagem que realiza o processo de chamar o servidor e atualizar a página;
 - **DOM (Document-Object-Model)**: É o modelo de objeto do browser, ou seja, como os elementos HTML são representados na memória do navegador. O JavaScript usa o DOM para localizar e manipular o elemento na página;

- **CSS:** Embora não esteja diretamente relacionado com o processo de transmissão de dados, as folhas de estilo são frequentemente utilizadas para manipular elementos da tela, escondendo ou exibindo itens, alterando a aparência ou sendo utilizado para ajustar o layout da página para diversos dispositivos;
 - **XML:** É um dos formatos de mensagem que o JavaScript utiliza para se comunicar com o servidor;
 - **JSON:** É outro formato utilizado para mensagens.
-
- Agora, algumas tecnologias do lado do servidor:
 - **Web Services:** São páginas que retornam XML/JSON, e não HTML, e usam protocolos de comunicação como GET e POST;
 - **WCF (Windows Communications Foundations):** É a evolução dos Web Services. O WCF é um conjunto de protocolos, ferramentas e padrões que permitem a comunicação entre aplicativos com muito mais recursos do que os Web Services;
 - **PHP:** Outra plataforma bastante utilizada, principalmente por aplicativos open source;
 - **JSP:** A implementação Java dos Web Services.

As tecnologias citadas nessa lista são algumas dentre centenas de combinações possíveis de plataforma, ambientes, linguagens e protocolos. Na verdade, qualquer servidor que retorne alguma informação pode ser utilizado no fluxo de procedimentos do modelo AJAX.

É possível usar apenas o conjunto JavaScript/HTML/Web Service para criar as páginas no padrão AJAX. Porém, programar diretamente em JavaScript pode ser uma árdua tarefa, mesmo para um programador experiente. Isso porque, dentre outros motivos, o JavaScript não é compilado, o que torna difícil a detecção de erros.

Não é à toa que as bibliotecas (frameworks) JavaScript se tornaram tão utilizadas. Uma das bibliotecas mais famosas é o **jQuery**. Com o jQuery, tarefas que requerem dezenas de linhas de código podem ser feitas com apenas uma ou duas expressões. Veja um exemplo:

As duas listas a seguir fazem a mesma coisa: localizam um elemento do documento, mudam a cor desse elemento para azul e inserem um conteúdo dentro com o texto **Bem-vindo**. Repare que, em jQuery, é necessário muito menos código. Além disso, o jQuery torna os processos de manipulação da página compatíveis com todos os browsers. Isso é necessário porque o DOM pode variar de um navegador para outro, fazendo que um programa que funcione perfeitamente bem quando roda no Internet Explorer possa falhar completamente quando é executado em outro navegador, como Chrome, por exemplo.

- Em JavaScript:

```
var titulo=document.getElementById('tituloDiv');
titulo.style.color = 'blue';
titulo.innerHTML='Bem Vindo';
```

- Em jQuery:

```
$('#titulo').css('color', 'blue').text('Bem Vindo');
```

9.3. AJAX com JavaScript

Para criar uma chamada do tipo AJAX para um servidor, é utilizado o objeto **XMLHttpRequest**. Esse objeto faz uma chamada para um servidor e recebe a resposta de maneira assíncrona, ou seja, o processamento não para a fim de esperar o retorno. Em vez disso, uma função é designada para ser executada quando a resposta chegar. Esse processo é chamado **CallBack**.

Visual Studio 2015 - ASP.NET com C# Fundamentos

As principais propriedades do objeto **XMLHttpRequest** são as seguintes:

Propriedade	Descrição
onreadystatechange	Propriedade que define a função que será chamada quando a resposta do servidor chegar.
Open(comando, url)	Método que abre a comunicação com o servidor por meio do parâmetro URL, usando o comando GET ou POST.
send	Método que inicia a chamada.
readyState	Propriedade que retorna o status da chamada. Esse status vai de 0 a 4, sendo: 0 – Chamada não iniciada; 1 – Conexão estabelecida; 2 – Iniciando o recebimento; 3 – Recebendo a resposta; 4 – Resposta recebida.
Status	A condição da resposta recebida, sendo: 200 – Recebida com sucesso; 404 – URL não encontrada; 407 – Acesso negado; 500 – Erro no servidor.
responseText	O conteúdo enviado pelo servidor no formato texto.
responseXml	O conteúdo enviado pelo servidor no formato XML.

Veja este exemplo:

Dentro da tag BODY de uma página HTML, é definido um botão, que, quando clicado, vai preencher uma div com informações no servidor. O exemplo a seguir cria um botão e define que uma função chamada **teste()** vai ser executada pelo navegador quando o usuário clicar. Não há um envio de formulário ao servidor porque o tipo do input é button, e não submit.

```
<input type="button" value="Clique aqui" onclick="teste()" />

<div id="mensagem"></div>
```

Dentro da tag HEAD, a função declarada na tag **input**, deve ser escrita dentro de um elemento script:

```
<head runat="server">  
  <script type="text/javascript">  
  
    function teste()  
{  
  
}  
  
  </script>  
</head>
```

O primeiro passo desse script é criar o objeto **XMLHttpRequest** e definir a função que vai ser chamada quando houver uma resposta do servidor:

```
function teste()  
{  
  
  var obj = new XMLHttpRequest();  
  obj.onreadystatechange = servidorCallBack;
```

```
  function servidorCallBack()  
  {  
  
  }  
  
}
```

O passo seguinte é definir a conexão com o servidor por meio do método **open** e iniciar o processo com o método **send**.

Visual Studio 2015 - ASP.NET com C# Fundamentos

O primeiro parâmetro do método **open** é o comando (GET/POST) e o segundo é a URL que retorna os dados. A URL pode ser um arquivo, um Web Service ou qualquer recurso que retorne informações em um formato conhecido.

```
function teste()
{
    var obj = new XMLHttpRequest();
    obj.onreadystatechange = servidorCallBack;

    obj.open('GET', 'dados.xml');
    obj.send();
```

```
function servidorCallBack()
{
}
```

Conteúdo do arquivo **dados.xml**:

```
<?xml version="1.0" encoding="utf-8" ?>
<nome>Bem vindo ao Ajax</nome>
```

Quando a resposta vier, a função **CallBack** se encarrega de processá-la. A propriedade **readystate** retorna em que ponto está o processo, a transmissão do servidor. Quando o valor dessa propriedade chegar em 4 é porque a resposta terminou de ser transmitida e pode ser lida.

Após esse processo, é necessário verificar o conteúdo da propriedade **status**, que retorna um número indicando o resultado do processamento.

Se tudo ocorreu bem, a propriedade **status** deverá retornar **200**. Outros números identificam a origem do problema. O status **404**, por exemplo, é Página ou recurso não encontrado.

```
function teste()
{
    var obj = new XMLHttpRequest();

    obj.onreadystatechange = servidorCallBack;

    obj.open('GET', 'dados.xml');

    obj.send();

    function servidorCallBack()
    {
        if (obj.readyState== 4) {
            if (obj.status == 200) {

            }
        }
    }
}
```

Se a resposta chegou com sucesso, os dados transmitidos podem ser obtidos por meio das propriedades **responseXML** ou **responseText** do objeto **XMLHttpRequest**.

Visual Studio 2015 - ASP.NET com C# Fundamentos

Os dados podem vir nos formatos XML, Text, JSON, CSV ou outro formato personalizado. Se o formato for XML, os elementos e os atributos podem ser obtidos por meio dos mesmos recursos usados para ler os elementos da página, pois o DOM (Document Object Model) que o browser usa é um documento XML.

```
function servidorCallBack()
{
    if (obj.readyState== 4) {
        if (obj.status == 200) {

            //Obtém o documento XML
            var doc = obj.responseXML;

            //Obtém um array de elementos chamado ‘nome’
            var elementos =
                doc.getElementsByTagName('nome');

            //Obtém o elemento
            var elementoNome = elementos[0];

            //Obtém o texto desse elemento
            var nome = elementoNome.textContent;

            //Obtém o elemento no documento
            //que receberá a mensagem
            var elementoMensagem =
                document.getElementById('mensagem');

            //Define que o texto desse elemento
            //é o texto recebido do servidor
            elementoMensagem.innerHTML = nome;
        }
    }
}
```

Se o status não for 200 (OK), uma descrição do status pode ser obtida por meio da propriedade **statusText**.

```
function servidorCallBack()
{
    if (obj.readyState== 4) {
        if (obj.status == 200) {

            //Obtém o documento XML
            var doc = obj.responseXML;

            //Obtém um array de elementos chamado 'nome'
            var elementos =
                doc.getElementsByTagName('nome');

            //Obtém o elemento
            var elementoNome = elementos[0];

            //Obtém o texto desse elemento
            var nome = elementoNome.textContent;

            //Obtém o elemento no documento
            //que receberá a mensagem
            var elementoMensagem =
                document.getElementById('mensagem');

            //Define que o texto desse elemento
            //é o texto recebido do servidor
            elementoMensagem.innerHTML = nome;
        }
        else {
            alert('erro:' +obj.status+'\r'+obj.statusText);
        }
    }
}
```

9.4. AJAX com jQuery

A biblioteca jQuery simplifica bastante o processo de escrever códigos em JavaScript. Suas funções internas encapsulam a lógica necessária para localizar objetos no documento, para manipular suas propriedades e métodos e realizar chamadas AJAX no servidor.

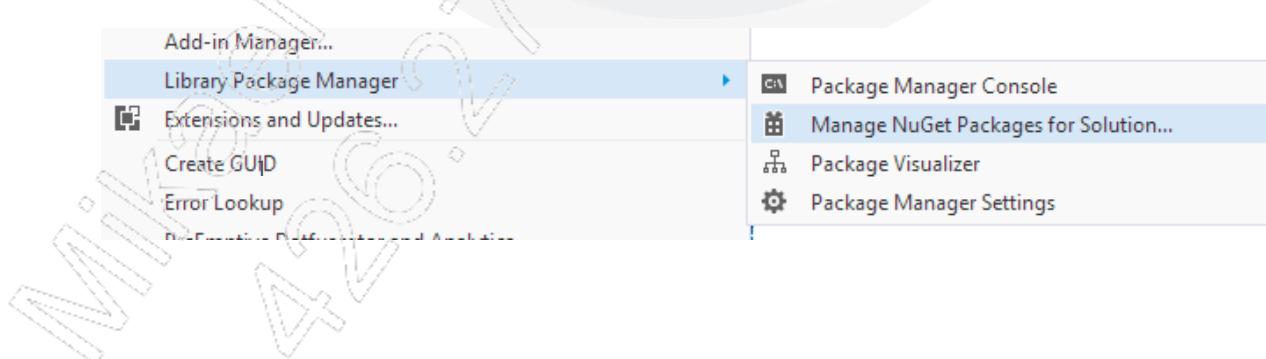
O primeiro passo é incluir a biblioteca jQuery no programa. Para isso, basta inserir o arquivo em JavaScript no projeto e criar uma tag **script** apontando para esse arquivo.

9.4.1. NuGet

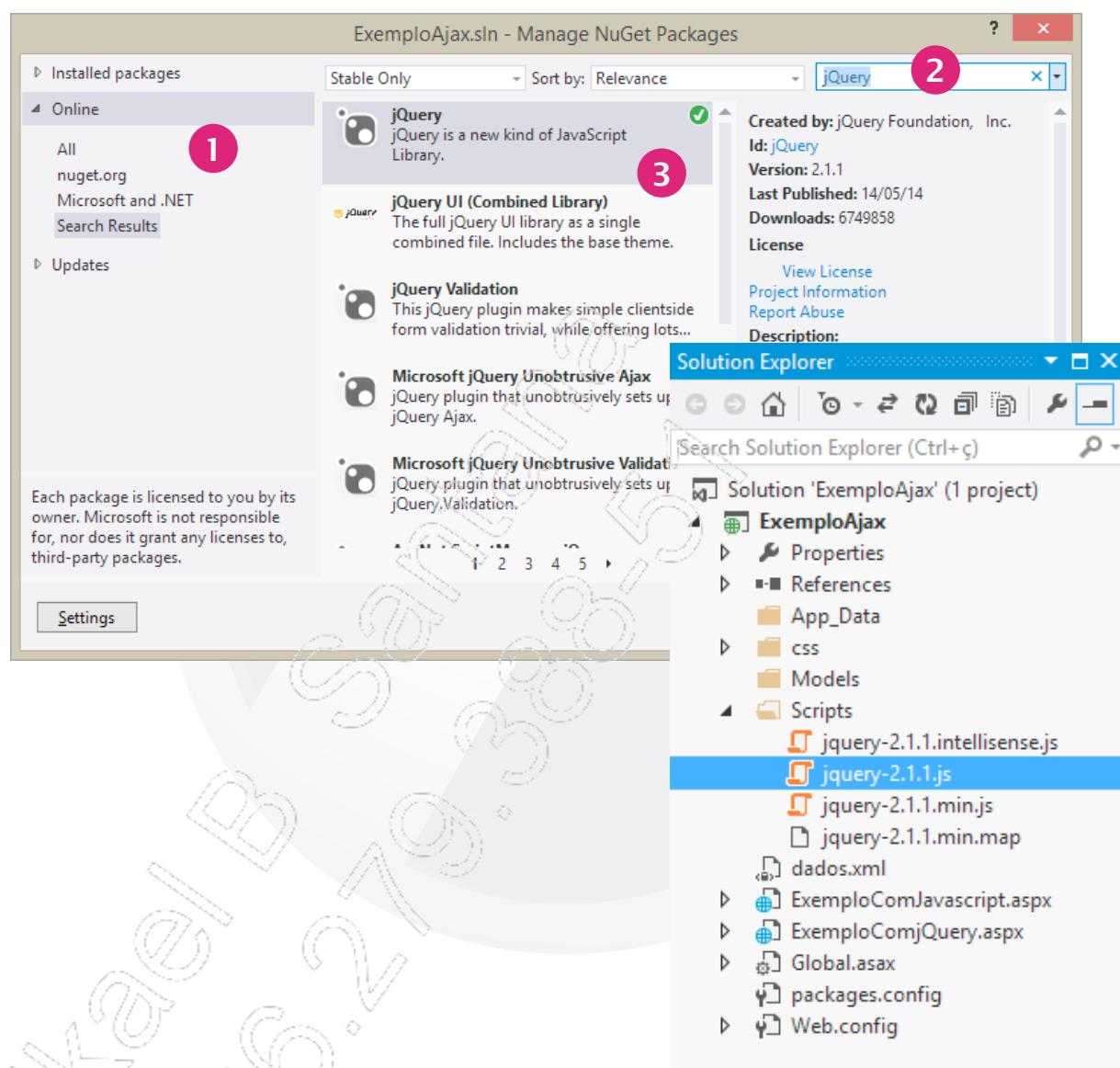
Esse processo pode ser feito manualmente, obtendo o arquivo do site e incluindo no projeto. Mas é muito mais prático usar um recurso do Visual Studio chamado **NuGet**, que é um framework open source para gerenciar plug-ins e componentes. O NuGet cuida automaticamente do processo de obter a última versão do componente, criar o ambiente adequado no projeto e preparar os arquivos necessários para distribuição.

- **Instalando o jQuery pelo NuGet:**

1. No menu **Tools**, escolha **Library Package Manager** e **Manage NuGet Packages for Solution**;



2. Escolha **Online** (1), procure **jQuery** na caixa de procura (2) e clique em **jQuery** (3) para instalar. No **Solution Explorer**, alguns arquivos serão incluídos. O principal é **jquery-(versao).js**, que é o código em JavaScript desse componente. Existe uma outra versão reduzida (sem espaços e com nome de funções modificadas), usada para publicar em ambiente de produção, chamada **jquery-(versao).min.js**.



9.4.2. Usando jQuery

Uma vez instalado o jQuery, é necessário obter uma referência no código HTML. Isso pode ser conseguido arrastando o arquivo **jQuery-(versao).js** para a sessão HEAD da página HTML:

```
<head runat="server">
    <title></title>

    <script src="Scripts/jquery-2.1.1.js"></script>

</head>
```

Não há diferença no código HTML usado no exemplo em JavaScript para chamar a função que será criada:

```
<!DOCTYPE html>

<html xmlns="http://www.w3.org/1999/xhtml">
    <head runat="server">
        <title></title>
        <script src="Scripts/jquery-2.1.1.js"></script>

        </script>
    </head>

    <body>
        <form id="form1" runat="server">
            <h1>Ajax com jQuery</h1>

            <input type="button"
                   value="Clique aqui"
                   onclick="teste()" />

            <div id="mensagem"></div>

        </form>
    </body>
</html>
```

O código necessário para realizar o processo de carregar conteúdo no servidor, no entanto, é muito mais simples do que o equivalente em JavaScript:

```
<!DOCTYPE html>

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title></title>
    <script src="Scripts/jquery-2.1.1.js"></script>

    <script type="text/javascript">

        function teste() {

            $('#mensagem').load('dados.xml');
        }

    </script>
</head>
<body>
    <form id="form1" runat="server">
        <h1>Ajax com jQuery</h1>
        <input type="button"
value="Clique aqui"
onclick="teste()" />
        <div id="mensagem"></div>
    </body>
</html>
```

De modo resumido, o que acontece no jQuery é:

- A parte `$('#mensagem')` seleciona o elemento com id #mensagem;
- A parte `.load('dados.xml')` carrega o conteúdo do arquivo (fazendo uma chamada AJAX) dentro do elemento selecionado.

O jQuery cuida dos detalhes como obter um array de elementos, usar a propriedade adequada para inserir texto, extrair o conteúdo do XML e tornar tudo isso compatível com todos os browsers. Usar o jQuery torna o processo de escrever código em JavaScript muito mais rápido e eficiente.

9.5. Uso de AJAX com Web Forms

O ASP.NET disponibiliza Web Controls e um framework em JavaScript que facilitam a programação no modelo AJAX. Usando os Web Controls, fica totalmente transparente para o programador todo o processo das chamadas usando JavaScript, XMLHttpRequest, XML, DOM e HTML.

9.5.1. Suporte no lado servidor

Os controles do lado servidor renderizam a saída do tipo AJAX para o navegador e interagem com scripts de código cliente. Estes controles de servidor ASP.NET orientados para AJAX são os mais frequentemente utilizados:

- Controle **ScriptManager**

O **ScriptManager** deve estar em todas as páginas que desejam trabalhar com a funcionalidade AJAX. Ele tem como funções o registro do script da biblioteca do AJAX na página e a renderização parcial da página, além de suportar localização e scripts comuns de usuário e colaborar com chamadas de volta para o servidor.

- Controle **ScriptManagerProxy**

Para gerenciar como os servidores renderizam os scripts em uma página Web, normalmente é utilizado um controle **ScriptManager**. Porém, quando, numa página, já existe um controle **ScriptManager** (por exemplo, em páginas mestre, que tipicamente armazenam esse controle) e é necessário gerenciar referências de script e serviço em componentes aninhados como páginas de conteúdo e controles de usuário, utilizamos o controle **ScriptManagerProxy**.

As páginas de conteúdo armazenam o controle **ScriptManagerProxy** e trabalham com o verdadeiro controle **ScriptManager** por meio do proxy para acessar o controle **ScriptManager** contido na página mestre caso uma segunda instância desse controle seja encontrada em uma mesma página. Essa exceção é também aplicada em casos de controles de usuário.

- Controle **UpdatePanel**

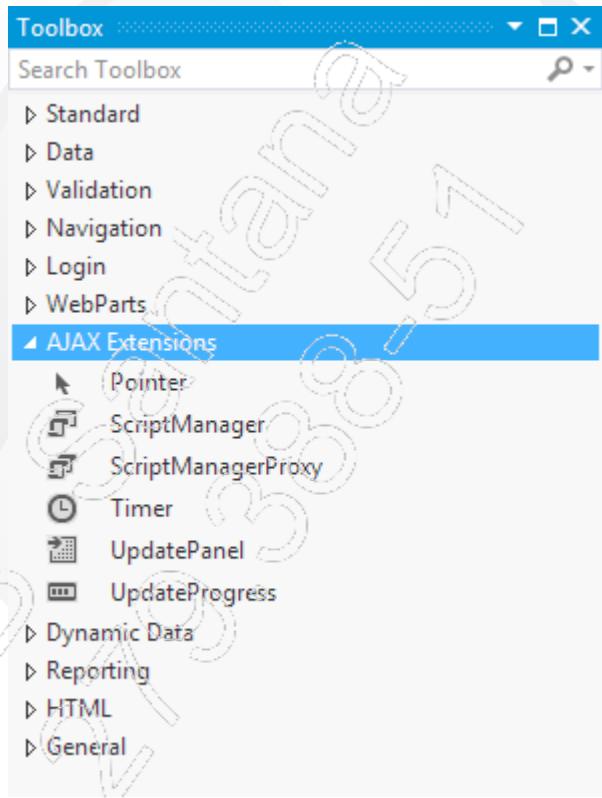
Este controle une eventos e controles de lado servidor específicos para renderizar atualizações parciais de páginas.

- Controle **UpdateProgress**

Quando atualizações parciais de página são feitas pelos controles **UpdatePanel**, o controle **UpdateProgress** permite exibir um conteúdo HTML enquanto a resposta do servidor não chega ao navegador.

- Controle **Timer**

O controle **Timer** tem a função de emitir PostBacks em intervalos definidos, sejam eles PostBacks típicos, ou seja, postando a página completa, ou PostBacks periódicos de partes das páginas, quando é usado em união com o controle **UpdatePanel**.



9.6. Exemplo de uso

O processo de uso do AJAX.NET é composto de três passos básicos:

1. Incluir na página um controle **ScriptManager**;
2. Incluir na página um **UpdatePanel**;
3. Incluir Web Controls tradicionais dentro do **UpdatePanel**.

O primeiro passo é inserir o **ScriptManager**, responsável por criar o código JavaScript que vai realizar a chamada AJAX para cada UpdatePanel que for inserido. É necessário que esse controle fique antes de qualquer outro controle que use a renderização parcial. Normalmente é colocado logo abaixo do **form** da página. Esse controle não é renderizado, portanto não aparece nada na visualização da página.

```
<%@ Page Language="C#" ....%>
<!DOCTYPE html>
<html>

<head>...</head>

<body>
    <form id="form1" runat="server">

        <asp:ScriptManager ID="ScriptManager1" runat="server">
            </asp:ScriptManager>

        <h1>Exemplo AJAX.NET</h1>
    </form>
</body>
</html>
```

O UpdatePanel é utilizado para criar a renderização parcial. Deve ser colocado após o ScriptManager. No exemplo a seguir, são inseridos dois Labels, um dentro e um fora do UpdatePanel. Isso vai servir para mostrar como o UpdatePanel é isolado do processo PostBack da página. Os controles dentro de um UpdatePanel devem ficar dentro da tag **ContentTemplate**:

```
<%@ Page Language="C#" ....%>
<!DOCTYPE html>
<html>
<head>...</head>
<body>
    <form id="form1" runat="server">
        <asp:ScriptManager ID="ScriptManager1" runat="server">
        </asp:ScriptManager>

        <h1>Exemplo AJAX.NET</h1>

        <section>
            <p>
                <asp:Label runat="server" ID="loadLabel"></asp:Label>
            </p>

            <asp:UpdatePanel ID="UpdatePanel1" runat="server">
                <ContentTemplate>
                    <asp:Button runat="server"
                        ID="confirmarButton"
                        Text="Clique aqui" onclick="confirmarButton_Click" />
                    <asp:Label runat="server" id="mensagemLabel"></asp:Label>
                </ContentTemplate>
            </asp:UpdatePanel>

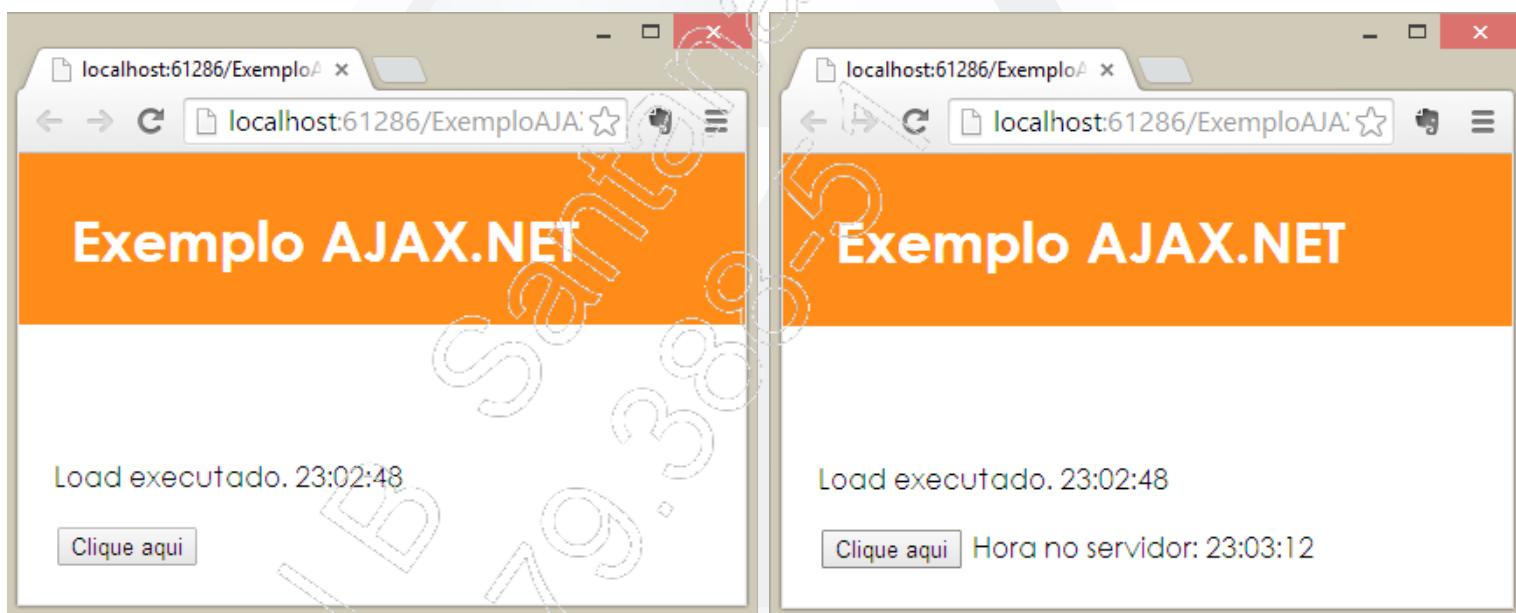
            </section>
        </form>
    </body>
</html>
```

Visual Studio 2015 - ASP.NET com C# Fundamentos

O código da página atualiza os dois Labels. Um no evento **Page_Load** e o outro no evento **Click** do botão:

```
protected void Page_Load(object sender, EventArgs e)
{
    loadLabel.Text = "Load executado. " + DateTime.Now.ToString();
}

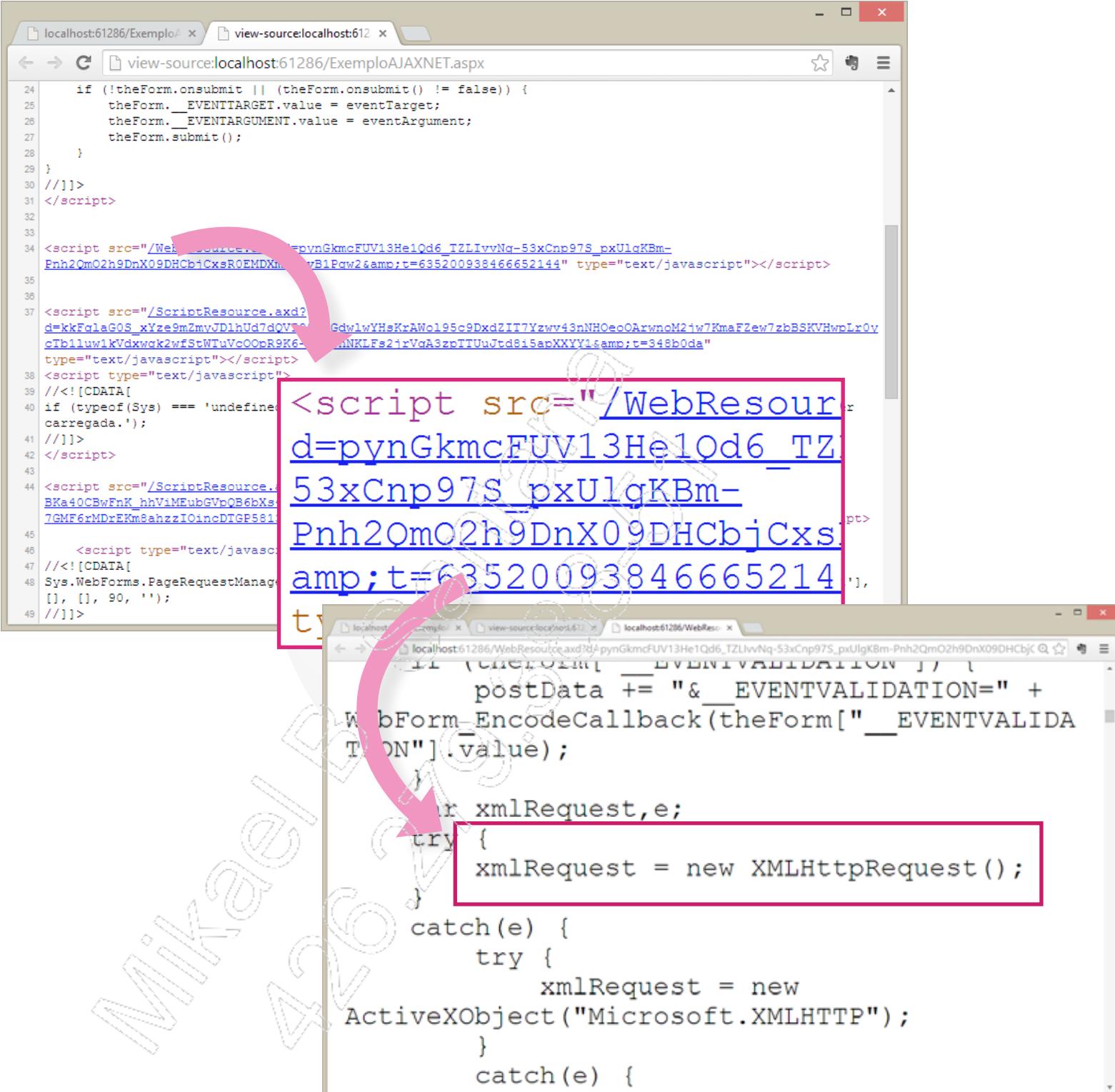
protected void confirmarButton_Click(object sender, EventArgs e)
{
    mensagemLabel.Text = "Hora no servidor: " + DateTime.Now.
    ToString();
}
```



Na execução da página, é possível perceber que o evento **Page_Load** ocorre apenas uma vez. Quando é clicado o botão, a hora do primeiro Label não muda.

No AJAX.NET, todo o processo de JavaScript e objetos XML ficam transparentes para o programador, que continua programando da maneira tradicional, como se estivesse usando controles que fazem PostBack na página.

Na verdade, o ASP.NET utiliza todos os elementos tradicionais: JavaScript, XMLHttpRequest, DOM etc. Isso fica evidente analisando o código-fonte da página gerada:



The screenshot shows two browser windows. The top window displays the source code of a page named 'ExemploAJAXNET.aspx'. The bottom window displays the source code of a file named 'WebResource.axd'. Both files contain JavaScript code. Red boxes highlight specific portions of the code in both windows.

```

// ...
if (!theForm.onsubmit || (theForm.onsubmit() != false)) {
    theForm.__EVENTTARGET.value = eventTarget;
    theForm.__EVENTARGUMENT.value = eventArgument;
    theForm.submit();
}
//]]>
</script>

<script src="/WebResource.axd?d=pynGkmcFUV13He1Qd6_TZLlvvNq-53xCnp97S_pxUlgKBm-Pnh2QmO2h9DnX09DHcbjCxsROEMDXh&t=635200938466652144" type="text/javascript"></script>

<script src="/ScriptResource.axd?d=kkFdlaGOS_xYze9m2myJDLhUd7dQVtGdwlyVHeKrAwI95c9DxdZIT7Yzvv43nNHOeoOArwnoM2iw7KmaF2ew7zbBSKVHwpLr0y&t=348b0da" type="text/javascript"></script>
<script type="text/javascript">
//<![CDATA[
if (typeof(Sys) === 'undefined') Sys = {};
Sys.WebForms.PageRequestManager.getInstance().add_endRequest(function() {
    // ...
});
//]]&gt;
&lt;/script&gt;

&lt;script src="/ScriptResource.axd?d=BKa40CBwFnK_hhViMEubGVpQB6bXa7GMF6rMDrEKm8ahzzIOincDTGF581&amp;t=635200938466652144" type="text/javascript"&gt;
//<![CDATA[
Sys.WebForms.PageRequestManager.getInstance().add_endRequest(function() {
    // ...
});
//]]&gt;
</pre>


```

postData += "&__EVENTVALIDATION=" +
WebForm_EncodeCallback(theForm["__EVENTVALIDATION"].value);
}

var xmlRequest,e;
try {
 xmlRequest = new XMLHttpRequest();
}
catch(e) {
 try {
 xmlRequest = new
ActiveXObject("Microsoft.XMLHTTP");
 }
 catch(e) {

```

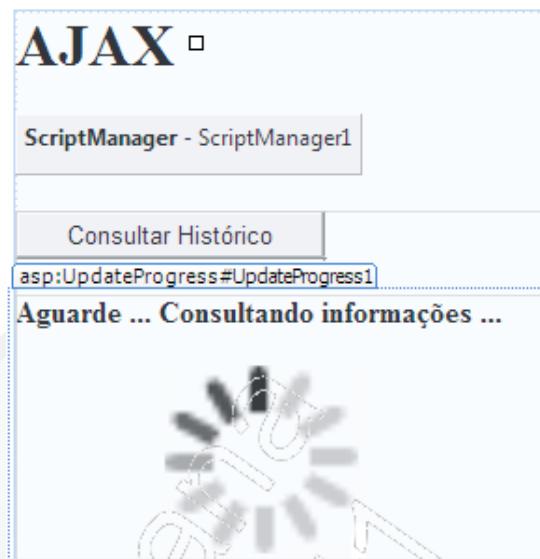

```

O controle **UpdateProgress** tem grande utilidade quando devemos disponibilizar ao usuário o tempo de duração de uma operação, ação que é realizada por meio do componente **BackgroundWorker** e do controle **Progress** no Windows Forms. Porém, no caso do **UpdateProgress**, uma estratégia diferente é utilizada.

Visual Studio 2015 - ASP.NET com C# Fundamentos

Com AJAX no ASP.NET, os controles **UpdateProgress** são mostrados durante PostBacks assíncronos. Portanto, quando um PostBack assíncrono é ativado pelo controle **UpdatePanel**, todos os controles **UpdateProgress** na página tornam-se visíveis.

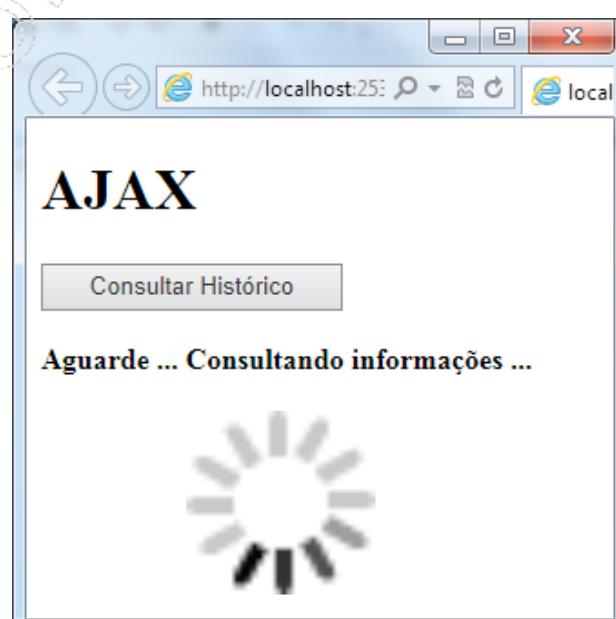
Vejamos um exemplo de inserção de um controle **UpdateProgress** na página:



A seguir, temos o código que permite programar a exibição do controle **UpdateProgress**:

```
protected void consultarButton_Click(object sender, EventArgs e)
{
    //Simulando uma consulta que demora 5 segundos para responder
    System.Threading.Thread.Sleep(5000);
}
```

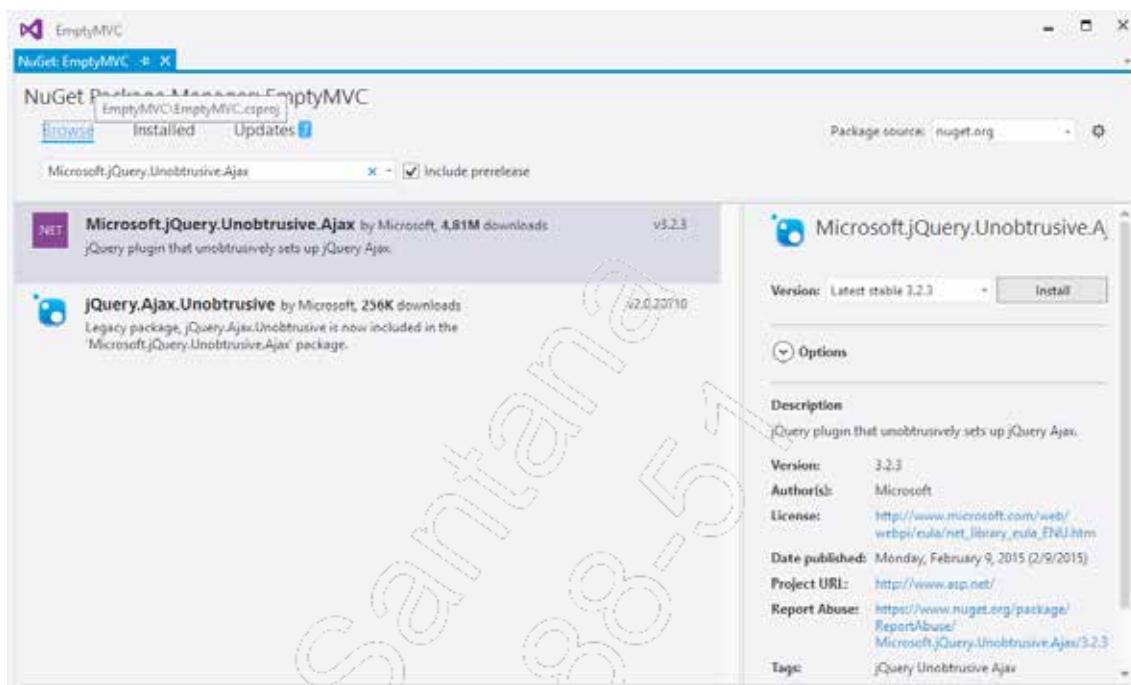
O resultado do **UpdateProgress** no navegador é exibido a seguir:



9.7. AJAX com MVC

O MVC contém uma biblioteca para realizar chamadas AJAX diretamente, sem utilizar manualmente jQuery, JavaScript ou outro componente.

Em primeiro lugar, é necessário instalar o componente, via NuGet, chamado **Microsoft.jQuery.Unobtrusive.Ajax**.



Em seguida, uma referência à biblioteca deverá ser adicionada depois do jQuery, na página principal de layout:

```
<script src="~/Scripts/jquery-1.10.2.min.js"></script>
<script src="~/Scripts/bootstrap.min.js"></script>
<script src="~/Scripts/jquery.unobtrusive-ajax.min.js"></script>
```

Visual Studio 2015 - ASP.NET com C# Fundamentos

Esta é toda a configuração necessária. Para utilizar o AJAX, é necessário usar o método de extensão apropriado:

```
@Ajax.ActionLink("Testar Ajax", "TesteAjax", new AjaxOptions()
{
    HttpMethod = "GET",
    AllowCache = false,
    InsertionMode = InsertionMode.Replace,
    UpdateTargetId = "mensagemDiv"
})

<div id="mensagemDiv"></div>
```

O método **Ajax.ActionLink** cria uma âncora com metadados que são utilizados pela biblioteca jQuery. São três parâmetros passados:

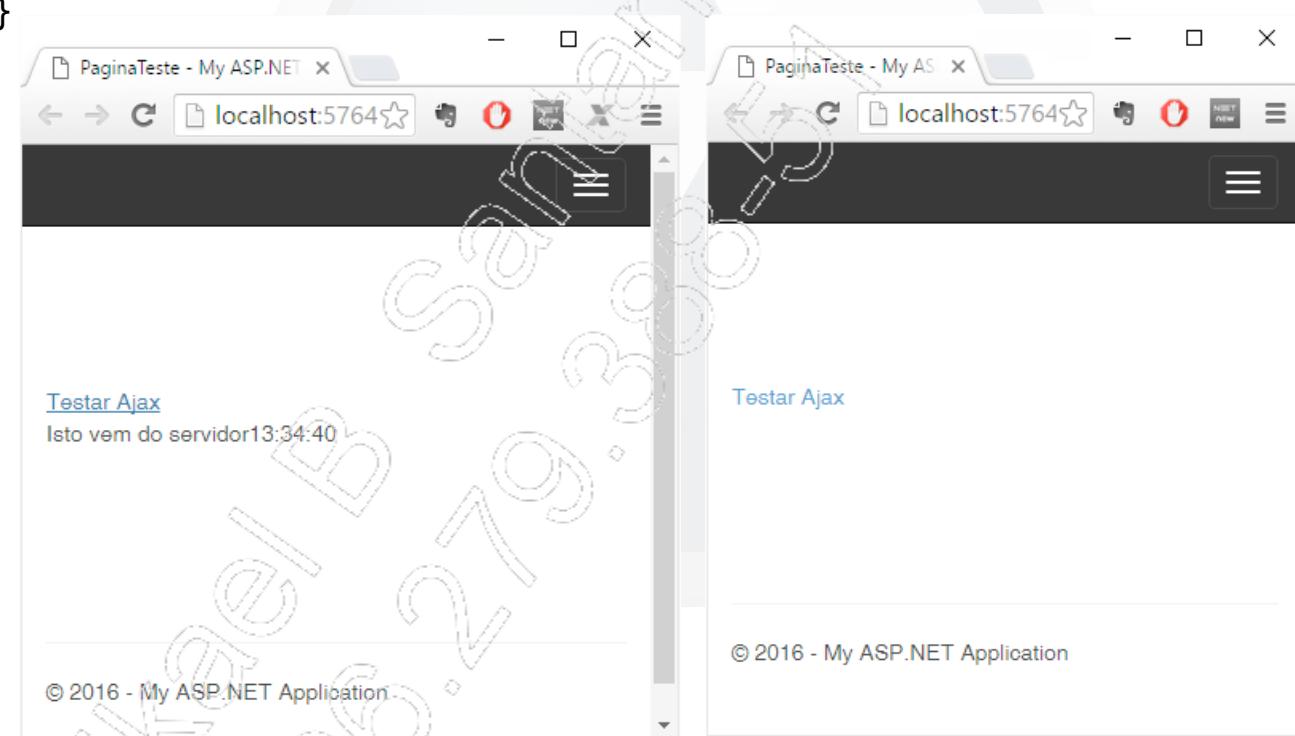
```
@Ajax.ActionLink(Texto(1),
                  Nome do Método(2),
                  Instância da Classe AjaxOptions(3))
```

- **1 – Texto:** É o texto que será exibido no link;
- **2 – Nome do método:** É o método que será executado;
- **3 – AjaxOptions:** Instância da classe **AjaxOptions** com parâmetros de chamada. O parâmetro **UpdateTargetId** representa o id do elemento que será atualizado. O parâmetro **InsertionMode** define que o elemento destino var ser substituído pelo conteúdo do servidor ou se vai manter o conteúdo atual. O parâmetro **AllowCache** define que não é para o navegador armazenar o resultado na memória. E, finalmente, o parâmetro **HttpMethod** define se será utilizado GET, POST, PUT, DEL ou outro verbo de comando HTTP.

O método que será chamado pelo comando AJAX pode retornar uma string ou uma **PartialView**. Neste exemplo, o retorno é apenas uma string:

```
public class HomeController : Controller
{
    public ActionResult Index()
    {
        return View();
    }

    public string TesteAjax()
    {
        return "Isto vem do servidor" +
            DateTime.Now.ToString("dd/MM/yyyy HH:mm:ss");
    }
}
```



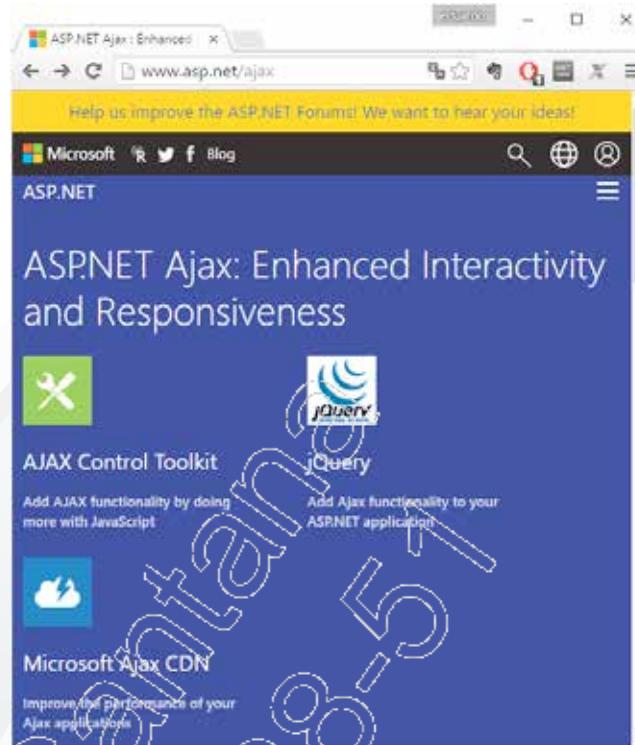
9.8. AJAX Control Toolkit

O **Control Toolkit** do AJAX no ASP.NET é um conjunto de componentes que engloba inúmeras capacidades fornecidas pelo AJAX, entre elas a possibilidade de criar controles e extensores personalizados. O download do Control Toolkit do AJAX no ASP.NET deve ser feito pelo próprio Web site do ASP.NET AJAX, pois ele não está incluído automaticamente no Visual Studio.

Visual Studio 2015 - ASP.NET com C# Fundamentos

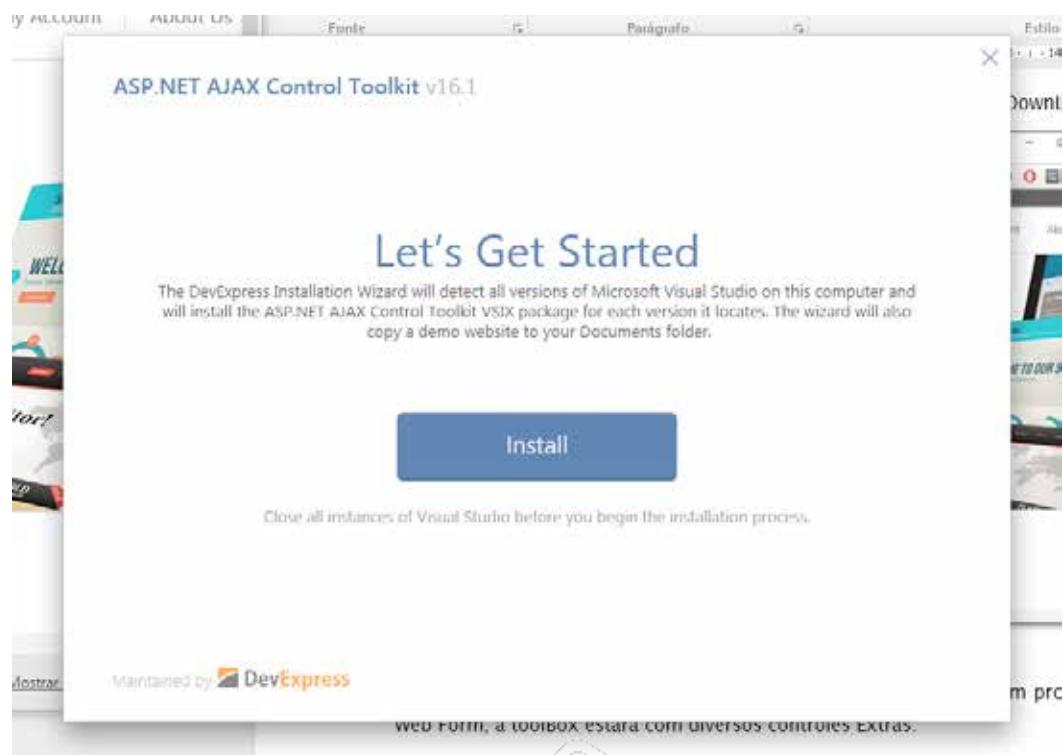
Esse projeto é open source, o que significa que é possível não apenas obter os componentes para usar em qualquer projeto mas também obter o código-fonte para modificar, estudar como foi desenvolvido ou utilizar apenas partes dos componentes.

Para obter o AJAX Control Toolkit, entre na página <http://www.asp.net/ajax>.

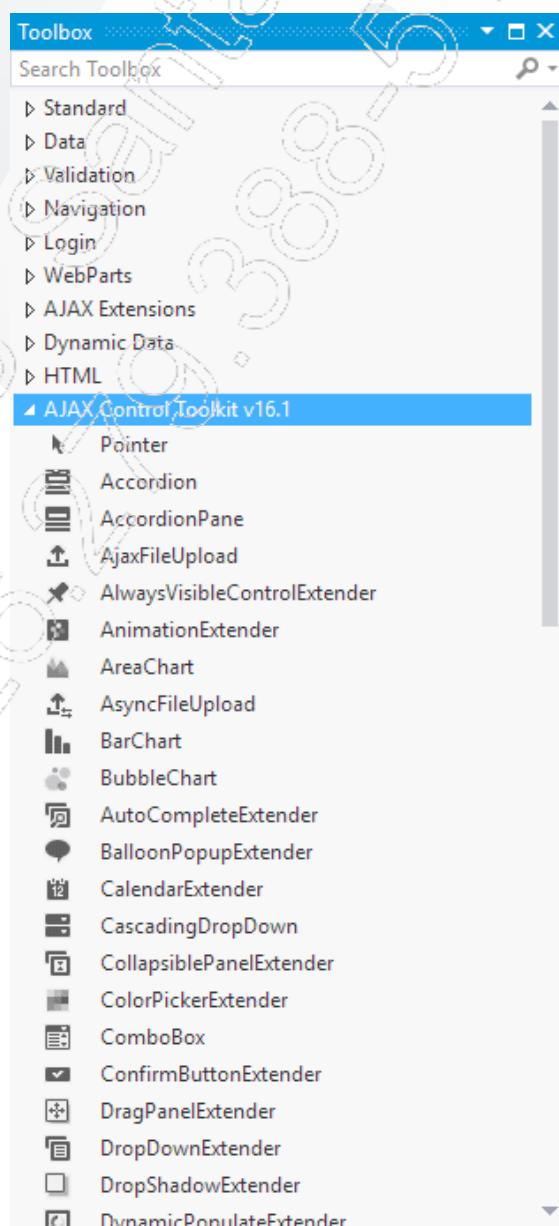


Ao clicar em **AJAX Control Toolkit**, você será desviado para a página de download:





Após executar e instalar o programa, ao reabrir o Visual Studio em um projeto iWeb Form, a toolbox estará com diversos controles extras:



9.8.1. Usando o AJAX Control Toolkit

Os Web Controls disponíveis estão em duas categorias: **Web Controls** e **Extenders** (extensores). Os **Web Controls** são componentes de arrastar-e-soltar com toda a funcionalidade completa. Os extensores usam um controle existente, adicionando funcionalidades a este.

Os principais extensores estão listados a seguir:

Extensor	Descrição
Accordion	Este extensor é composto de um grupo de controles AccordionPane . Ele demonstra um grupo de painéis, um por vez, e funciona do mesmo modo que diversos CollapsiblePanels quando são expandidos um de cada vez.
AlwaysVisibleControl	Utilizarnos este extensor para fixar em uma página um controle que desejamos tornar sempre visível, fixo em um ponto definido da página, enquanto o fundo da página e outros elementos continuam se movendo quando arrastamos a barra de rolagem.
Animation	Quando desejamos utilizar conteúdo de animação, utilizamos uma interface limpa fornecida por este extensor.
AsyncFileUpload	Faz upload de um arquivo de modo assíncrono, no segundo plano.
AutoComplete	Comunica-se com um servidor Web com informações para completar automaticamente entradas de texto com sugestões prováveis.
Calendar	Fornece funcionalidade de escolha de data ao controle TextBox do lado cliente de um modo personalizável.
CascadingDropDown	Funciona com o controle DropDownList , preenchendo de modo automático um conjunto seguinte de DropDownList, que é disponibilizado de acordo com a resposta do anterior.
CollapsiblePanel	Para inserirmos um painel articulado em uma página Web, usamos o extensor CollapsiblePanel , que funciona com controles Panel .

Extensor	Descrição
ConfirmButton	Funcionando com o controle Button (e seus derivados), o ConfirmButton tem como utilidade disponibilizar mensagens quando for necessária uma confirmação do usuário.
DragPanel	Quando aplicado a controles Panel , permite ao usuário arrastar um painel pela tela.
DropDown	Aplica um menu drop-down estilo SharePoint.
DropShadow	O DropShadow aplica uma sombra em um painel, quando utilizado em controles Panel .
DynamicPopulate	Utiliza uma string HTML retornada por um serviço Web ou chamada de método da página que substitui o conteúdo de um controle.
FilteredTextBox	Garante a inserção em uma caixa de texto apenas de caracteres definidos como válidos.
HoverMenu	Em controles Web que possuem uma janela pop-up com informações adicionais, este extensor transforma a janela pop-up em um menu flutuante, que aparece quando colocamos o ponteiro do mouse sobre o controle.
ListSearch	Tendo como alvo uma ListBox ou uma DropDownList , este extensor nos permite realizar buscas conforme o usuário vai digitando o nome do item desejado, tecla por tecla.
MaskedEdit	Aplica uma máscara em um controle TextBox , fazendo com que a caixa de texto limite os tipos e a disposição de caracteres que podem ser inseridos.
ModalPopup	Cria uma janela pop-up que impede a continuidade das atividades normais até que ela receba a devida atenção do usuário, do mesmo jeito que em caixas de diálogo modais no Windows.
MutuallyExclusiveCheckBox	Este extensor agrupa controles CheckBox por meio de uma chave. Com isso, como vários controles utilizam a mesma chave, somente será exibida uma caixa de verificação selecionada por vez.

Visual Studio 2015 - ASP.NET com C# Fundamentos

Extensor	Descrição
NoBot	O controle NoBot tem a função de detectar e prevenir automaticamente spams ou bots, sem necessidade da ação do usuário. Mesmo que a interação humana torne o processo muito mais efetivo, este controle é indicado para páginas Web em que não é necessária plena efetividade e é totalmente invisível.
NumericUpDown	Direcionado ao controle TextBox, botões de up e down são criados para nos permitir aumentar ou diminuir valores da TextBox.
PagingBulletedList	Este extensor é direcionado ao controle BulletedList e permite paginação sortida no lado cliente.
PasswordStrength	Trabalha com o controle TextBox para que, quando o usuário final inserir uma senha, seja exibida sua força, ou seja, seu grau de complexidade.
PopupControl	O PopupControl pode funcionar direcionado a todos os controles, abrindo uma janela pop-up com conteúdo definido.
Rating	Este extensor permite ao usuário avaliar um conteúdo por meio de um sistema de imagens, em que, normalmente, usa-se de uma a cinco estrelas.
ReorderList	Este controle gera uma lista com dados vinculados e marcadores. É possível interagirmos com os itens dessa lista para reordená-los conforme desejado.
ResizableControl	O ResizableControl pode funcionar direcionado a todos os controles, permitindo ao usuário redimensionar o controle ao qual este extensor está associado.

Extensor	Descrição
RoundedCorners	Transforma qualquer elemento de uma página Web com cantos em 90° em que for aplicado em objetos com cantos arredondados.
Seadragon	Este controle fornece um conjunto de botões com as funções de mais ou menos zoom, reset zoom e ativação de tela cheia, para visualização de imagens. Também podemos arrastar imagens com o mouse e aplicar mais zoom com o duplo-clique.
Slider	Cria um controle de estilo deslizante, ligado a um controle TextBox, para aumentar ou diminuir um valor numérico.
SlideShow	Este controle gerencia botões com a função de navegar por imagens, avançando ou retrocedendo, e ativar ou parar o slide show automático.
Tabs	O extensor Tab gera guias em que o conteúdo pode ser distribuído.
TextBoxWatermark	Exibe um texto como marca d'água em um controle TextBox , normalmente com alguma instrução sobre a caixa de texto. Quando a caixa é selecionada, o texto em marca d'água desaparece e o texto inserido pelo usuário tem o estilo normal.
ToggleButton	Ligado ao controle CheckBox , este extensor permite a utilização de imagens personalizadas para indicar o estado do CheckBox.
UpdatePanelAnimation	Gera uma animação associada ao UpdatePanel.
ValidatorCallout	Este extensor gera pequenas janelas pop-up que aparecem próximas aos elementos de UI que contêm dados incorretos para chamar a atenção do usuário.

9.8.1.1.Exemplo de Extender: ConfirmButton

Observe os passos adiante:

1. Adicione um ScriptManager na página (na guia **AJAX Extensions**):

```
<asp:ScriptManager ID="ScriptManager1" runat="server"></asp:ScriptManager>
```

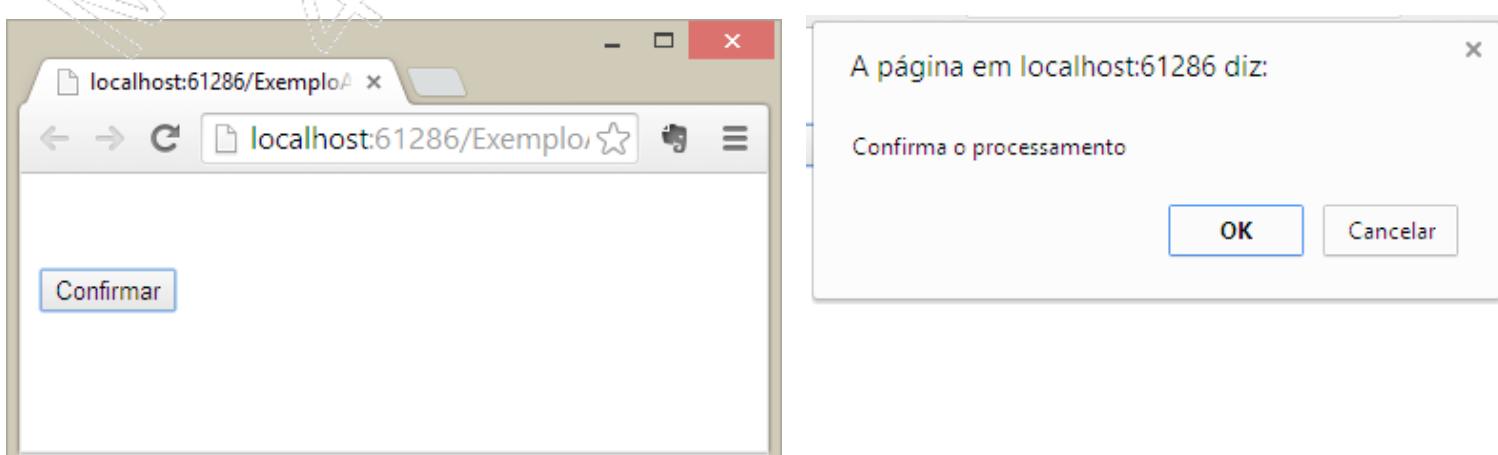
2. Adicione **Button**, **Label** e **ConfirmButtonExtender**. A principal propriedade de um extender é **TargetControlID**, que define qual controle vai ser estendido. O **ConfirmButtonExtender** usa a propriedade **ConfirmText** para incluir uma janela de diálogo em JavaScript que permite ao usuário confirmar ou cancelar uma ação de um botão:

```
<ajaxToolkit:ToolkitScriptManager ID="ToolkitScriptManager1"
runat="server">
</ajaxToolkit:ToolkitScriptManager>

<asp:Button ID="OkButton"
runat="server"
Text="Confirmar"
OnClick="OkButton_Click" />

<asp:Label ID="mensagemLabel"
runat="server"
Text="Label">
</asp:Label>

<ajaxToolkit:ConfirmButtonExtender ID="OkButton_ConfirmButtonExtender"
runat="server"
ConfirmText="Confirma o processamento" Enabled="True"
TargetControlID="OkButton">
</ajaxToolkit:ConfirmButtonExtender>
```



9.8.1.2.Exemplo de WebControl: TabContainer

O Web Control **TabContainer** cria painéis que são ativados com botões estilizados. A aparência dos painéis lembra as interfaces Windows Forms que usam guias.

1. Adicione um **ScriptManager** na página;

2. Adicione um **TabContainer** na página.

Os painéis do **TabContainer** são formados pela tag **TabPanel**, e o conteúdo é colocado dentro desses painéis por meio da tag **ContentTemplate**:

```
<ajaxToolkit:TabContainer ID="TabContainer1"
    runat="server"
    Width="400px"
    ActiveTabIndex="0">

    <ajaxToolkit:TabPanel runat="server"
        HeaderText="Página 1"
        ID="TabPanel1">

        <ContentTemplate>
            <p>Este é o primeiro painel</p>
        </ContentTemplate>

    </ajaxToolkit:TabPanel>

    <ajaxToolkit:TabPanel runat="server"
        HeaderText="Página 2"
        ID="TabPanel2">

        <ContentTemplate>
            <p>Este é o segundo painel</p>
            <p>
                <asp:Calendar runat="server">
                </asp:Calendar>
            </p>

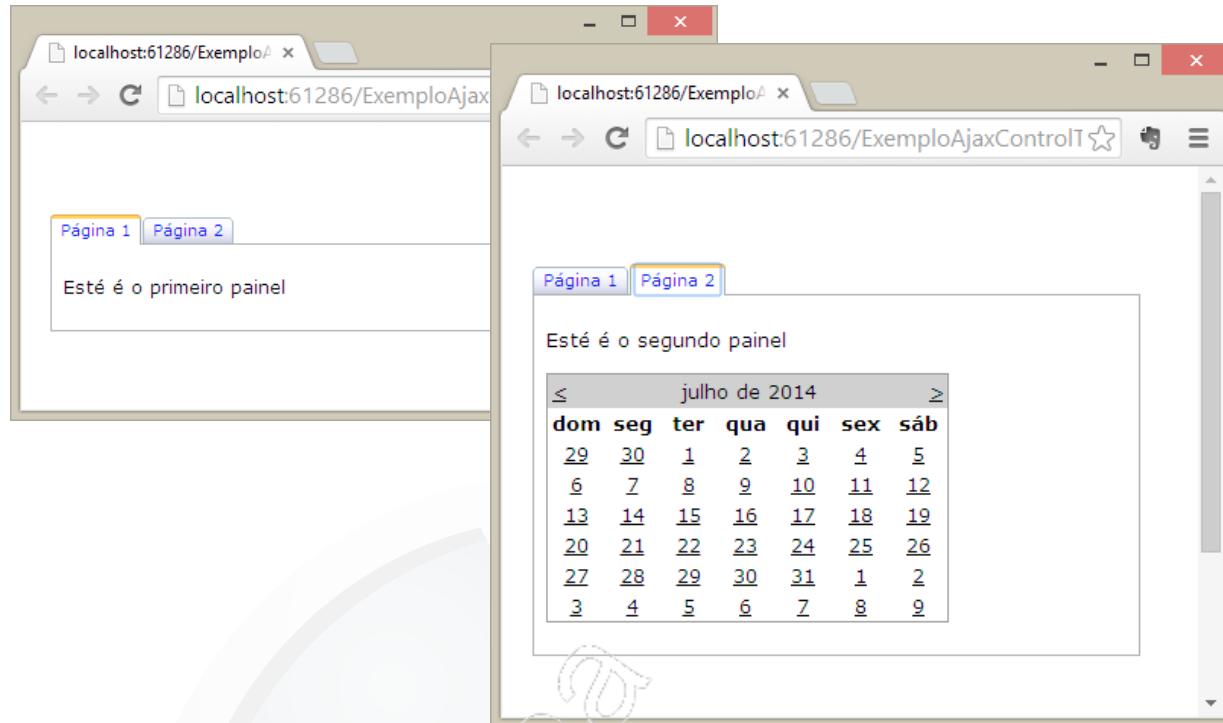
        </ContentTemplate>

    </ajaxToolkit:TabPanel>

</ajaxToolkit:TabContainer>
```

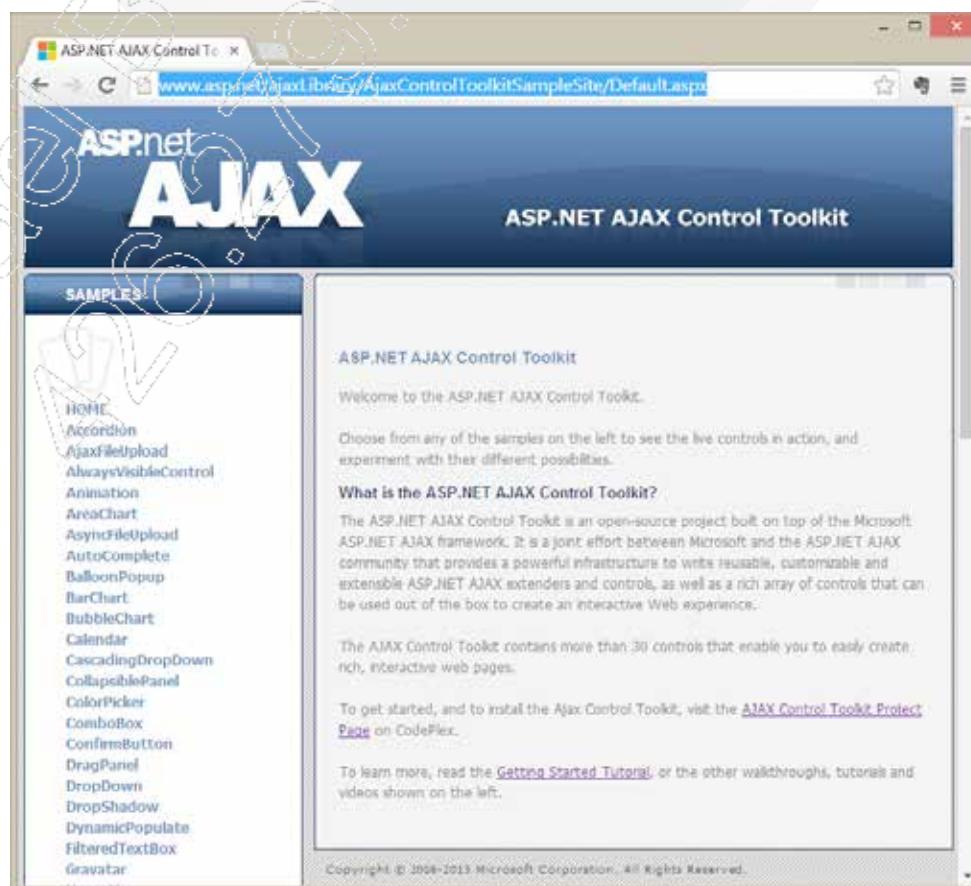
Visual Studio 2015 - ASP.NET com C# Fundamentos

Veja o resultado:

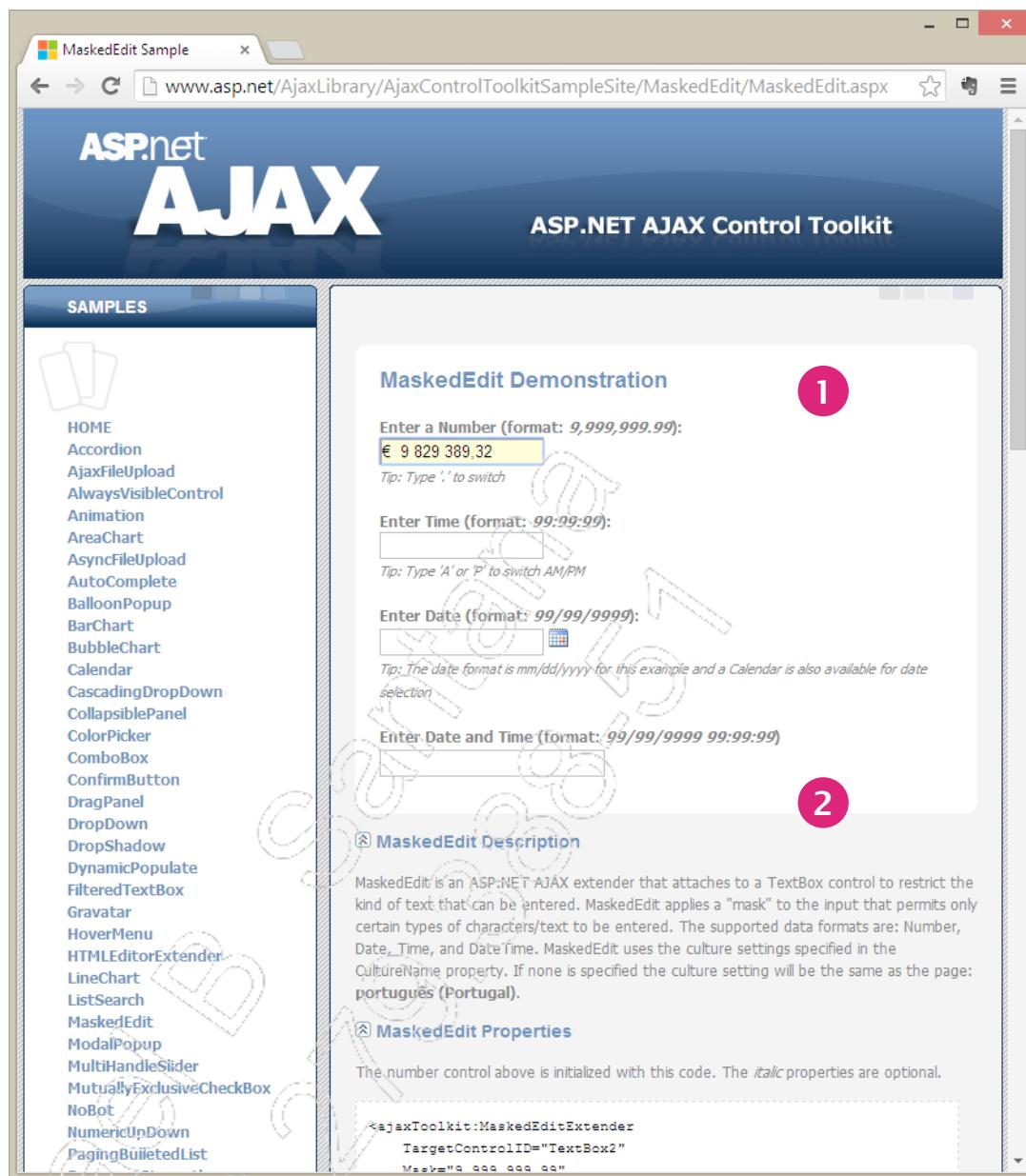


9.8.2. Exemplos

Ao instalar o AJAX Control Toolkit, uma aplicação Web de exemplo é instalada na pasta **Documentos**, dentro de **ASP.NET AJAX Control Toolkit / Sample Site**. Este é um projeto com exemplos da maioria dos controles com explicações detalhadas das propriedades.



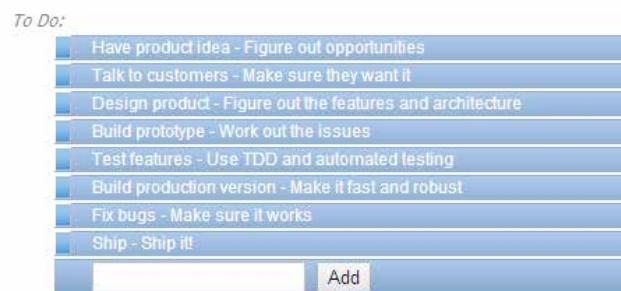
Além da demonstração, toda tela mostra as principais propriedades dos controles. Por exemplo, a tela do controle **MaskedEdit** mostra como o controle aparece (1) na tela do usuário e, na parte inferior (2), quais as principais propriedades:



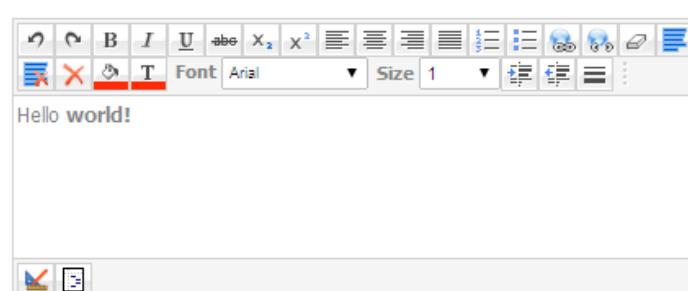
Visual Studio 2015 - ASP.NET com C# Fundamentos

Vale a pena explorar os exemplos do AJAX Control Toolkit. São dezenas de controles com as mais diversas funcionalidades, e essa biblioteca é atualizada constantemente. A todo momento, novos controles passam a fazer parte desse grupo de componentes.

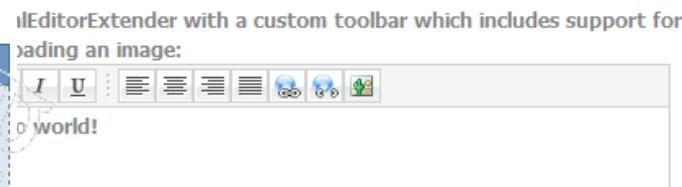
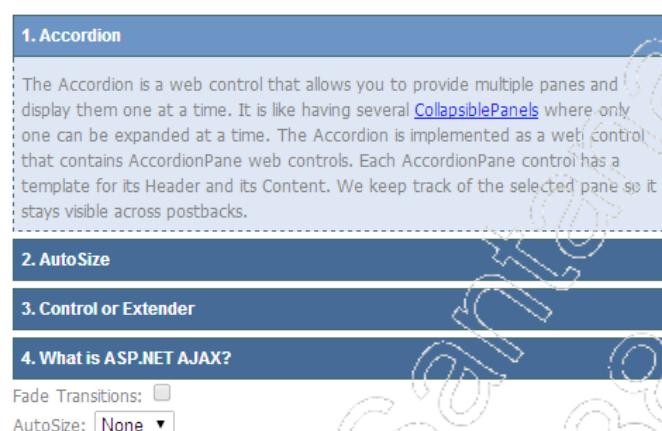
ReorderList Demonstration



HTMLEditorExtender Demonstration



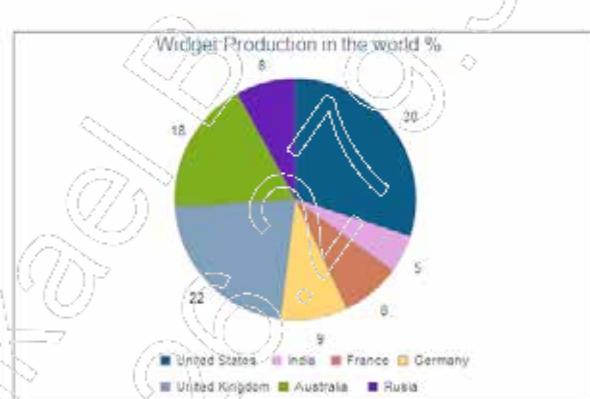
Accordion Demonstration



Seadragon control with default properties



PieChart Demonstration



BalloonPopupExtender Demonstration

Click inside either of the two TextBox controls below or click the link to see a demonstration of the BalloonPopupExtender control.



AreaChart Demonstration



Pontos principais

Atente para os tópicos a seguir. Eles devem ser estudados com muita atenção, pois representam os pontos mais importantes do capítulo.

- O AJAX no ASP.NET é uma aplicação rica de Internet (Rich Internet Applications, ou RIA), cuja prioridade é fornecer uma experiência de usuário cada vez melhor, gerando complementos totalmente novos para o protocolo Web padrão;
- A dependência de uma participação mais ativa do navegador durante o processo é uma das principais diferenças que o AJAX traz para a programação na Web. A utilização do AJAX possui diversas vantagens, como processamento no próprio navegador, elementos de aplicações desktop, atualizações parciais de páginas, compatibilidade com diversos navegadores, novas capacidades e melhorias na autenticação e personalização, entre outras;
- O suporte para AJAX no ASP.NET pode ser feito de duas maneiras: no lado servidor, em que o servidor renderiza saídas para a visualização correta das páginas, e no lado cliente, constituído pelas bibliotecas JavaScript do AJAX;
- O **Control Toolkit** é um conjunto de componentes que engloba inúmeras capacidades fornecidas pelo AJAX, nos permitindo criar controles e extensores personalizados por meio de um poderoso kit de desenvolvimento de software. Seu download deve ser feito pelo próprio Web site do ASP.NET AJAX;
- O controle **Timer** é um temporizador de utilização simples, que permite que eventos sejam realizados com certa regularidade, gerando PostBacks automaticamente para o servidor, de acordo com o intervalo de tempo definido;
- O controle **UpdateProgress** disponibiliza ao usuário o tempo de duração de uma operação, e são mostrados durante PostBacks assíncronos, quando se tornam visíveis na página;
- Os controles extensores têm a função de aumentar os recursos disponibilizados pelo ASP.NET. Devem ser aplicados em controles já existentes para estender sua funcionalidade;
- O MVC utiliza a biblioteca **Microsoft.jQuery.Unobtrusive.Ajax**, que cria chamadas via JavaScript diretamente para os métodos dos controles, sem a necessidade de manipular diretamente o JavaScript ou jQuery.

9

AJAX

Teste seus conhecimentos

Mikael B Santana
426.279.3857



IMPACTA
EDITORA

1. Qual tecnologia não está diretamente relacionada com o modelo AJAX?

- a) JavaScript
- b) SQL
- c) XML
- d) HTML
- e) DOM

2. Qual propriedade é usada pelo objeto XMLHttpRequest para definir a função CallBack que receberá a resposta do servidor?

- a) readyState
- b) status
- c) responseText
- d) onreadystatechange
- e) responseXml

3. Qual Web Control é obrigatório em toda página que usa AJAX.NET?

- a) jQuery
- b) UpdatePanel
- c) Timer
- d) ScriptManager
- e) ScriptManagerProxy

4. Quais os tipos de componentes disponíveis no AJAX Control Toolkit?

- a) Cliente e servidor.
- b) Web Controls e Extenders.
- c) GET e POST.
- d) HTML e JavaScript.
- e) Web Services e Web Forms.

5. Qual biblioteca facilita a criação de programas em JavaScript?

- a) Class Library
- b) System.Web
- c) jQuery
- d) CSS
- e) AJAX

9

AJAX

Mãos à obra!

Mikael B. Santana
426.279.3857



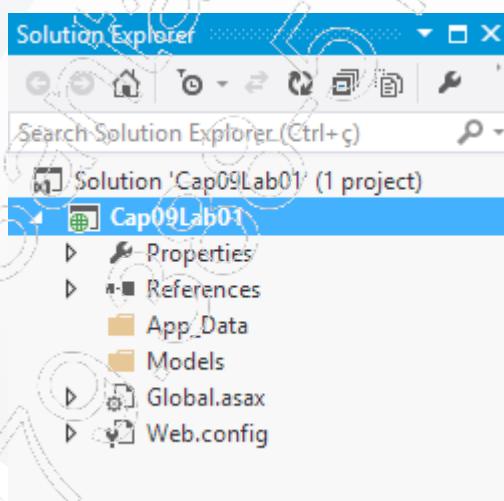
IMPACTA
EDITORA

Laboratório 1

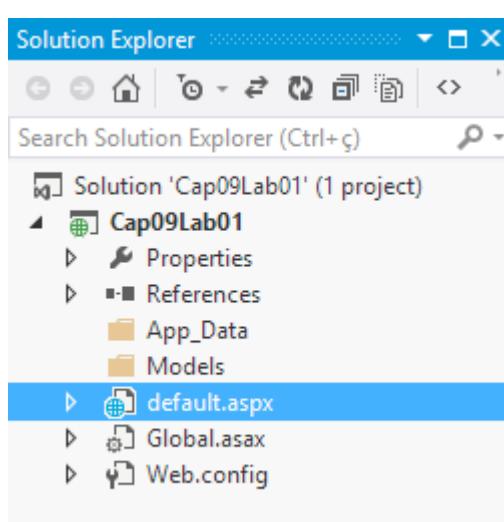
Objetivos:

- Criar um programa chat on-line;
- Usar o Web Control **Timer**;
- Usar o objeto **Application**;
- Usar o evento **Application_Start**;
- Separar as funcionalidades;
- Usar o ViewState.

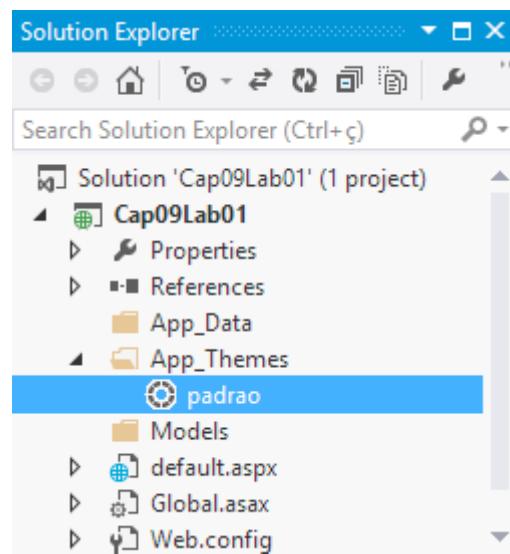
1. Crie um novo projeto Web vazio chamado **Cap09Lab01**;



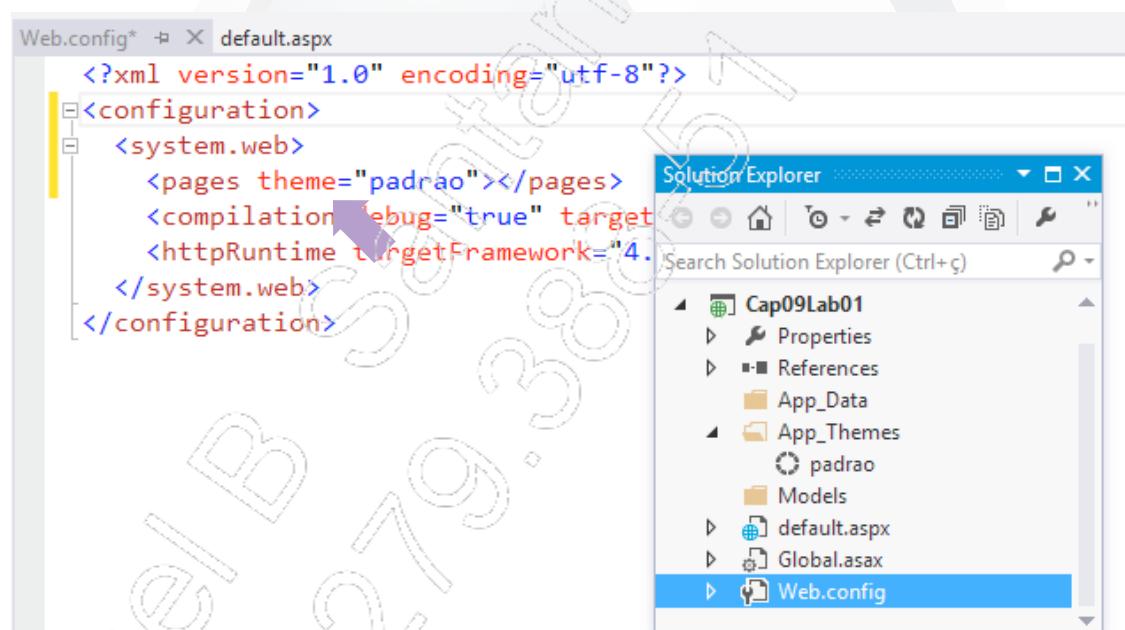
2. Adicione um Web Form chamado **default.aspx**;



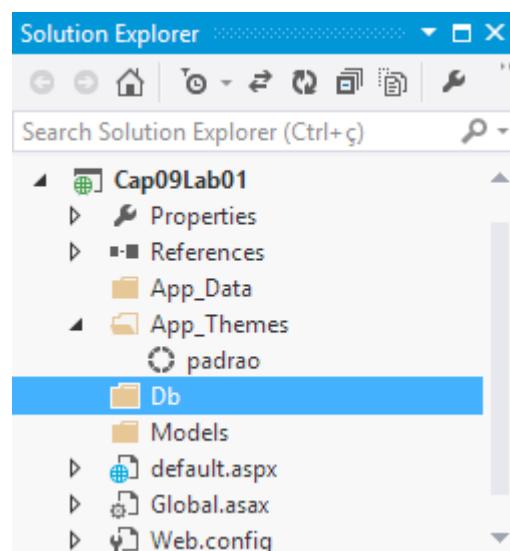
3. Adicione um tema chamado **padrao**;



4. Defina o tema **padrao** como o tema das páginas do Web site;

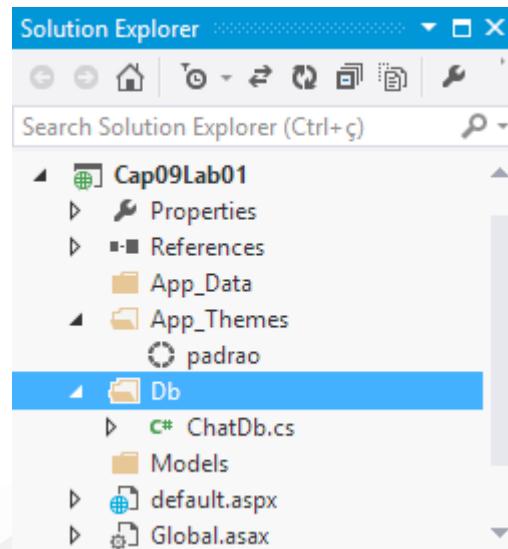


5. Crie uma pasta chamada **Db**;

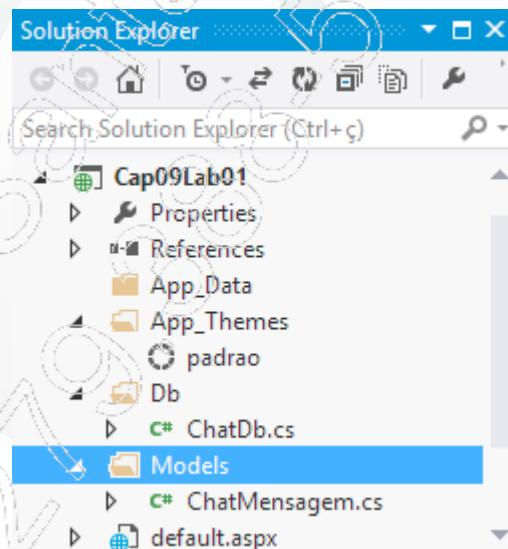


Visual Studio 2015 - ASP.NET com C# Fundamentos

6. Dentro da pasta **Db**, crie uma classe chamada **ChatDb**:



7. Dentro da pasta **Models**, crie uma classe chamada **ChatMensagem**:



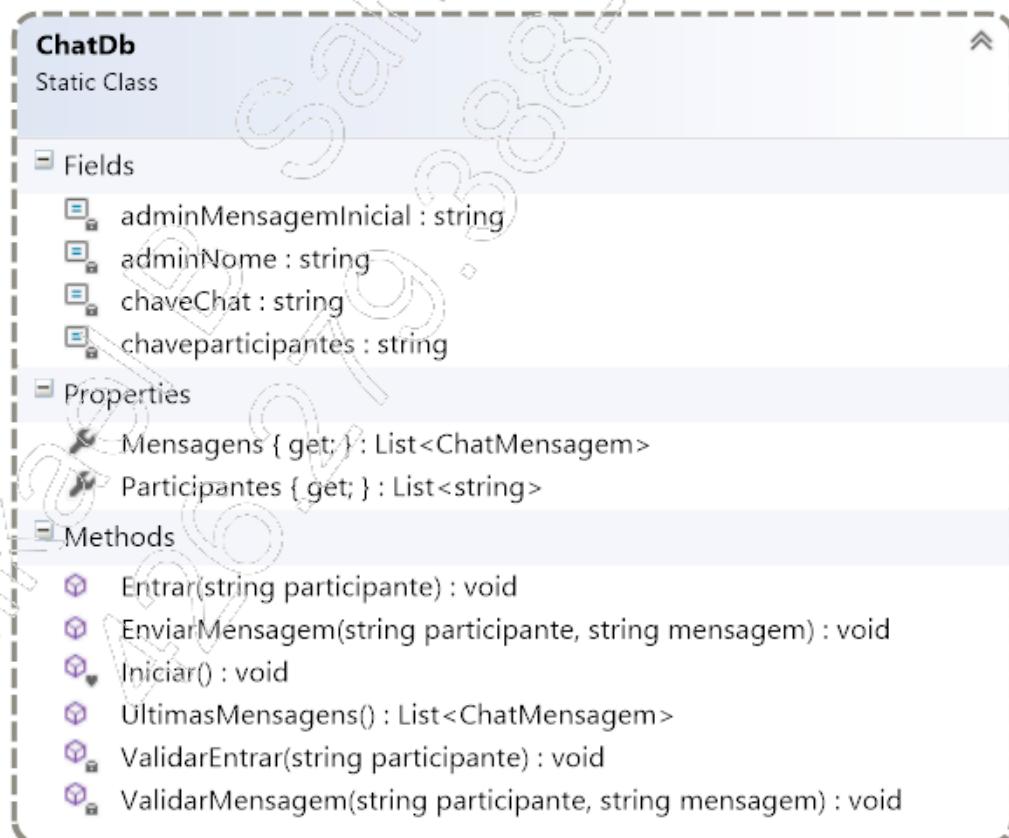
8. A classe **ChatMensagem** contém propriedades que armazenam os dados de uma mensagem (o nome do participante, a data e a hora):



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

namespace Cap09Lab01.Models
{
    public class ChatMensagem
    {
        public string Participante { get; set; }
        public DateTime DataHora { get; set; }
        public string Mensagem { get; set; }
    }
}
```

9. A classe **ChatDb** exibe os métodos de manipulação das mensagens e o login do participante:



Visual Studio 2015 - ASP.NET com C# Fundamentos

10. A classe **Db** expõe o método **Iniciar**, que cria a lista de mensagens e a lista de participantes. Esse método deve ser chamado quando a aplicação é ativada pela primeira vez:

```
using Cap09Lab01.Models;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;

public static class ChatDb
{
    //Chaves para gravar em Application
    private const string chaveChat = "chat";
    private const string chaveparticipantes =
        "participantes";
    private const string adminNome = "admin";
    private const string adminMensagemInicial =
        "Chat Iniciado com sucesso";

    //
    // Iniciar: Cria os itens da aplicação
    //
    internal static void Iniciar()
    {
        // Lista de Mensagens
        var lista = new List<ChatMensagem>();
        HttpContext
            .Current
            .Application
            .Add(chaveChat, lista);

        // Lista de Participantes On-Line
        var participantes = new List<string>();
        HttpContext
            .Current
            .Application
            .Add(chaveparticipantes, participantes);

        participantes.Add(adminNome);
    }
}
```

11. No arquivo **Global.asax**, dois objetos da aplicação serão usados para armazenar a lista de usuários on-line e as mensagens. O método **Application_Start** é ativado quando a aplicação vai ao ar pela primeira vez (ou quando é recompilada);

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

namespace Cap09Lab01
{
    public class Global : System.Web.HttpApplication
    {
        protected void Application_Start(
            object sender, EventArgs e)
        {
            Db.ChatDb.Iniciar();
        }
    }
}
```

Visual Studio 2015 - ASP.NET com C# Fundamentos

12. Na página **default.aspx**, crie o **título**, um **label** e um **multiview** para conter o formulário de login e a área de exibição do chat:

```
<%@ Page Language="C#" AutoEventWireup="true" . . . . .>

<!DOCTYPE html>

<html>
  <head runat="server">
    <title></title>
  </head>

  <body>
    <form id="form1" runat="server">

      <h1>ASP.NET - Chat</h1>

      <asp:Label ID="erroLabel"
        Visible="false"
        CssClass="erro"
        runat="server"></asp:Label>

      <asp:MultiView ID="MultiView1"
        ActiveViewIndex="0"
        runat="server">

        <asp:View ID="View1" runat="server">
        </asp:View>

        <asp:View ID="View2" runat="server">
        </asp:View>

      </asp:MultiView>

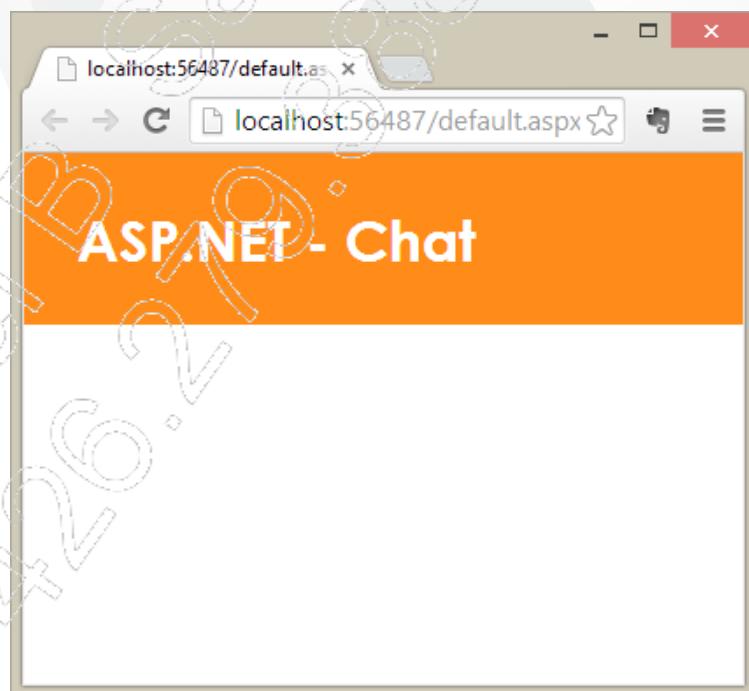
    </form>
  </body>
</html>
```

13. Adicione uma folha de estilo no tema padrão, insira as primeiras definições de formatação:

```
body {  
    margin:0px;  
    font-family:'Century Gothic'  
  
}
```

```
h1 {  
  
    margin:0px;  
    background-color:#ff6a00;  
    color:#ffffff;  
    padding:30px;  
    margin-bottom:40px;  
}
```

14. Visualize o resultado:



Visual Studio 2015 - ASP.NET com C# Fundamentos

15. Na página **default.aspx**, no primeiro **view**, insira o formulário de login:

```
<form id="form1" runat="server">
    <h1>ASP.NET ...
    <asp:Label ...
    <asp:MultiView ...

        <asp:View ID="View1" runat="server">

            <section>

                <asp:Label CssClass="legenda"
                    runat="server"
                    Text="Insira seu nome">
                </asp:Label>

                <asp:TextBox CssClass="campo"
                    runat="server"
                    ID="nomeTextBox">
                </asp:TextBox>

                <asp:Button CssClass="botao"
                    runat="server"
                    Text="Entrar no Chat"
                    ID="entrarButton" />

                <asp:Label runat="server"
                    ID="mensagemLabel">
                </asp:Label>

            </section>

        </asp:View>

        <asp:View ID="View2...">
        </asp:MultiView>
    </form>
    ...

```

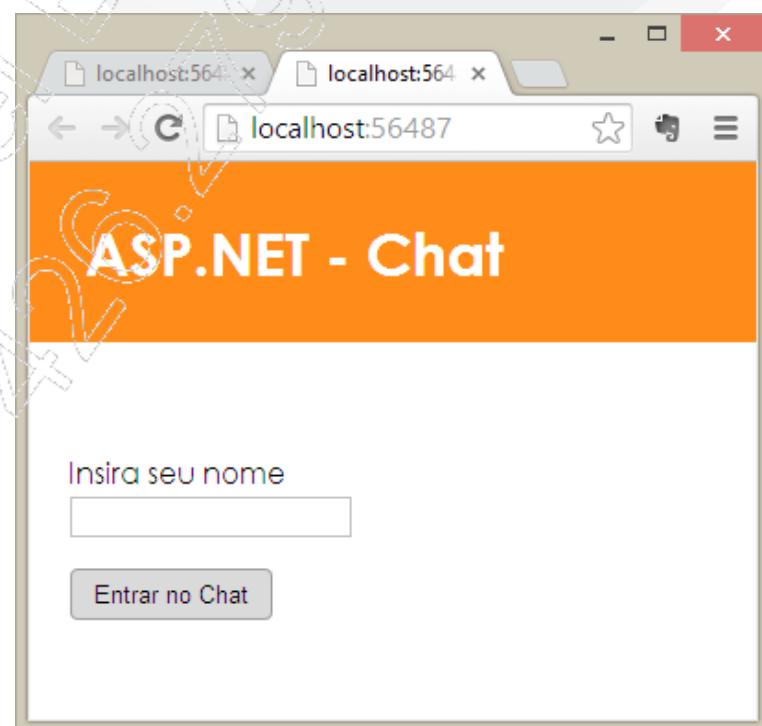
16. Na folha de estilos, insira as definições de formatação do formulário e visualize:

```
section {  
    padding:20px;  
}
```

```
.legenda {  
    display:block;  
}  
}
```

```
.campo {  
    display:block;  
}  
}
```

```
.botao {  
    margin-top:15px;  
    padding:5px 12px 5px 12px;  
    border-radius:4px;  
    border:1px solid #808080;  
    background-color:#ccc;  
}  
}
```



Visual Studio 2015 - ASP.NET com C# Fundamentos

17. Na view2 (página default.aspx), insira o HTML da área de chat:

```
<form id="form1" runat="server">
<asp:View ID="View2" runat="server">

    <section id="chat">
        <asp:Label runat="server"
            CssClass="legenda"
            ID="digiteMensagemLabel">
            Digite sua mensagem
        </asp:Label>

        <asp:TextBox runat="server"
            CssClass="campo"
            TextMode="MultiLine"
            Rows="3"
            ID="mensagemTextBox">
        </asp:TextBox>

        <asp:Button runat="server"
            CssClass="botao"
            Text="Enviar Mensagem"
            ID="enviarMensagemButton"/>

        <asp:Button runat="server"
            CssClass="botao"
            Text="Sair"
            ID="sairButton" />

        <div class="chat-box">
            <asp:Repeater
                runat="server"
                ID="chatRepeater"
                ItemType="Cap09Lab01.Models.ChatMensagem">
                <ItemTemplate>

                    </ItemTemplate>
            </asp:Repeater>

        </div>
    </section>

</asp:View>
```

18. Na tag **ItemTemplate** do **repeater**, escreva o código para vincular o chat. A propriedade **ItemType** define o tipo que será vinculado, permitindo usar o **IntelliSense** do Visual Studio para definir as propriedades do objeto vinculado:

```
<asp:Repeater  
    runat="server"  
    ID="chatRepeater"  
    ItemType="Cap09Lab01.Models.ChatMensagem">
```

```
<ItemTemplate>
```

```
<div class='chat-mensagem-box'>  
  
    <div class='chat-mensagem-participante'>  
        <%# Item.DataHora.ToString("T") %>  
        <%# Item.Participante %>  
    </div>  
  
    <div class="chat-mensagem-texto">  
        <%# Item.Mensagem %>  
    </div>  
</div>
```

```
</ItemTemplate>  
</asp:Repeater>
```

19. Na folha de estilos, insira as definições da área de mensagens:

```
.chat-mensagem-box {  
  
    border-top:1px solid #ccc;  
    clear:both;  
  
    font-size:80%;  
}
```

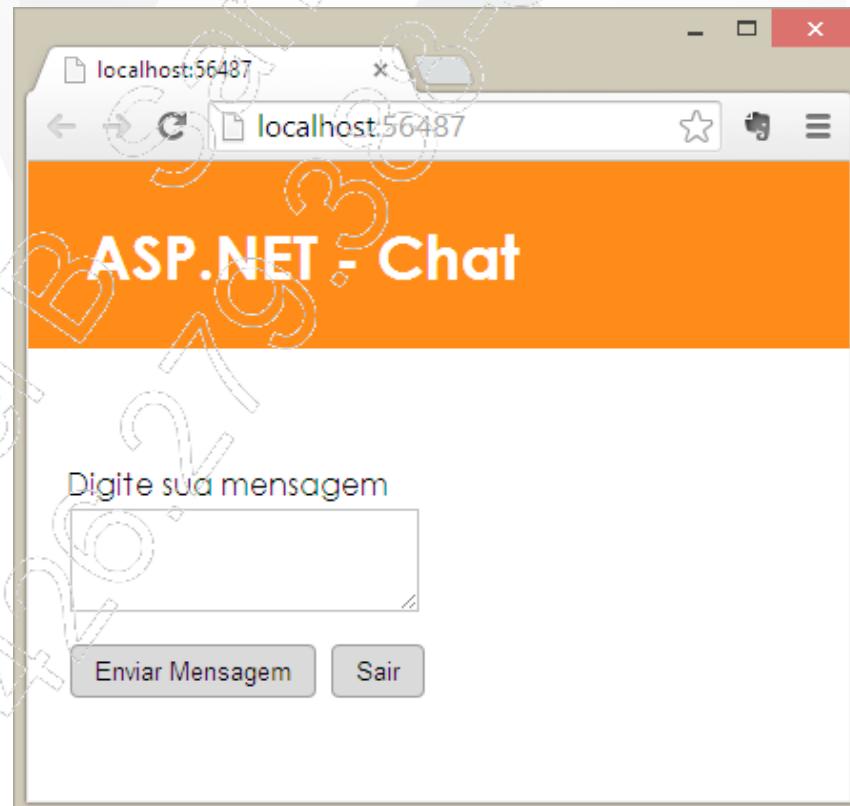
```
.chat-mensagem-participante {  
    font-weight:bold;  
    float:left;  
    width:120px;  
    padding:5px  
}
```

```
.chat-mensagem-texto{  
    float:left;  
    padding:5px  
}  
}  
}
```

```
.chat-box {  
    margin-top:20px;  
}
```

20. Altere a propriedade **ActiveViewIndex** do MultiView para visualizar a área de chat:

```
<asp:MultiView ID="MultiView1"  
    ActiveViewIndex="1"  
    runat="server">
```



21. As propriedades **Mensagens** e **Participantes** retornam a lista de mensagens e a lista de participantes do chat, gravadas no objeto **Application**:

```
public static class ChatDb
{
    //Chaves para gravar em Application
    private const string chaveChat = "chat...";

    // Iniciar: Cria os itens da aplicação
    internal static void Iniciar()...

    //
    // Mensagens
    //
    public static List<ChatMensagem> Mensagens
    {
        get
        {
            return
                (List<ChatMensagem>)HttpContext
                    .Current
                    .Application[chaveChat];
        }
    }

    //
    // Participantes
    //
    public static List<string> Participantes
    {
        get
        {
            return (List<string>)HttpContext
                .Current
                .Application[chaveparticipantes];
        }
    }
}

// Fim da Classe ChatDb
```

Visual Studio 2015 - ASP.NET com C# Fundamentos

22. O método **ValidarMensagem** faz a validação dos dados de uma mensagem; antes de realizar a gravação da mensagem, realizam o login do usuário:

```
public static class ChatDb
{
    //Chaves para gravar em Application
    private const string ...;

    // Iniciar: Cria os itens da aplicação
    internal static void Iniciar() ...

    // Mensagens
    public static List<ChatMensagem> Mensagens....;

    // Participantes
    public static List<string> Participantes....;
```

```
//
// Validar Mensagem
//
private static void ValidarMensagem(
    string participante, string mensagem)
{
    if (string.IsNullOrEmpty(participante) ||
        participante.Trim().Length == 0)
    {
        throw new Exception(
            "Participante não informado");
    }

    if (string.IsNullOrEmpty(mensagem) ||
        mensagem.Trim().Length == 0)
    {
        throw new Exception("Mensagem não informada");
    }
}

} // Fim da Classe ChatDb
```

23. O método **EnviarMensagem** grava a mensagem do objeto **Application**:

```
public static class ChatDb
{
    //Chaves para gravar em Application
    private const string ... 

    // Iniciar: Cria os itens da aplicação
    internal static void Iniciar() ...

    // Mensagens
    public static List<ChatMensagem> Mensagens.... 

    // Participantes
    public static List<string> Participantes.... 

    // Validar Mensagem
    private static void ValidarMensagem.... 

    // 
    // Enviar Mensagem
    //
    public static void EnviarMensagem(
        string participante, string mensagem)
    {
        ValidarMensagem(participante, mensagem);

        var msg = new ChatMensagem();
        msg.DataHora = DateTime.Now;
        msg.Participante = participante;
        msg.Mensagem = mensagem;

        Mensagens.Add(msg);
    }

} // Fim da Classe
```

Visual Studio 2015 - ASP.NET com C# Fundamentos

24. O método **ValidarEntrar** valida a entrada do usuário no chat:

```
public static class ChatDb
{
    //Chaves para gravar em Application
    private const string ...

    // Iniciar: Cria os itens da aplicação
    internal static void Iniciar() ...

    // Mensagens
    public static List<ChatMensagem> Mensagens.....

    // Participantes
    public static List<string> Participantes.....

    // Validar Mensagem
    private static void ValidarMensagem.....

    // Enviar Mensagem
    public static void EnviarMensagem.....
```

```
//
// Validar Entrar
//
private static void ValidarEntrar(string participante)
{
    if (string.IsNullOrEmpty(participante))
    {
        throw new Exception(
            "Informe o nome do participante");
    }
    foreach (string nome in participantes)
    {
        if (participante == nome)
        {
            throw new Exception(
                "Este nome já está sendo usado.");
        }
    }
}
```

```
// Fim da Classe
```

25. O método Entrar grava o usuário na lista de usuários:

```
public static class ChatDb
{
    //Chaves para gravar em Application
    private const string ...

    // Iniciar: Cria os itens da aplicação
    internal static void Iniciar() ...

    // Mensagens
    public static List<ChatMensagem> Mensagens.....

    // Participantes
    public static List<string> Participantes.....

    // Validar Mensagem
    private static void ValidarMensagem.....

    // Enviar Mensagem
    public static void EnviarMensagem.....

    // Validar Entrar
    private static void ValidarEntrar.....

    //
    // Entrar
    //
    public static void Entrar(string participante)
    {
        ValidarEntrar(participante);
        Participantes.Add(participante);

        string msg = string.Format(
            "{0} entrou no chat.",
            participante);

        EnviarMensagem(adminNome, msg);
    }
}

// Fim da Classe
```

26. Finalmente, o método **UltimasMensagens** apresenta as últimas mensagens enviadas:

```
public static class ChatDb
{
    //Chaves para gravar em Application
    private const string ...

    // Iniciar: Cria os itens da aplicação
    internal static void Iniciar() ...

    // Mensagens
    public static List<ChatMensagem> Mensagens.....

    // Participantes
    public static List<string> Participantes.....

    // Validar Mensagem
    private static void ValidarMensagem.....

    // Enviar Mensagem
    public static void EnviarMensagem.....

    // Validar Entrar
    private static void ValidarEntrar.....

    // Entrar
    public static void Entrar...

    //
    // Últimas Mensagens
    //
    public static List<ChatMensagem> UltimasMensagens()
    {
        var lista = ( from m in Mensagens
                     orderby m.DataHora descending
                     select m)
                    .Take(100);

        return lista.ToList();
    }
}

// Fim da Classe
```

27. Esta é a listagem completa da classe ChatDb:

```
using Cap09Lab01.Models;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;

namespace Cap09Lab01.Db
{
    public static class ChatDb
    {

        //Chaves para gravar em Application
        private const string chaveChat = "chat";
        private const string chaveparticipantes =
            "participantes";
        private const string adminNome = "admin";
        private const string adminMensagemInicial =
            "Chat Iniciado com sucesso";

        //
        // Mensagens
        //
        public static List<ChatMensagem> Mensagens
        {
            get
            {
                return (List<ChatMensagem>)HttpContext
                    .Current
                    .Application[chaveChat];
            }
        }

        //
        // Participantes
        //
        public static List<string> Participantes
        {
            get
            {
                return (List<string>)HttpContext
                    .Current
                    .Application[chaveparticipantes];
            }
        }
    }
}
```

Visual Studio 2015 - ASP.NET com C# Fundamentos

```
//  
// Entrar  
//  
public static void Entrar(string participante)  
{  
    ValidarEntrar(participante);  
    Participantes.Add(participante);  
  
    string msg = string.Format(  
        "{0} entrou no chat.", participante);  
    EnviarMensagem(adminNome, msg);  
}
```

```
//  
// Validar Entrar  
//  
private static void ValidarEntrar(string participante)  
{  
    if (string.IsNullOrEmpty(participante))  
    {  
        throw new Exception(  
            "Informe o nome do participante");  
    }  
  
    foreach (string nome in Participantes)  
    {  
        if (participante == nome)  
        {  
            throw new Exception(  
                "Este nome já está sendo usado.");  
        }  
    }  
}
```

```
//  
// Enviar Mensagem  
//  
public static void EnviarMensagem(  
    string participante, string mensagem)  
{  
    ValidarMensagem(participante, mensagem);  
  
    var msg = new ChatMensagem();  
    msg.DataHora = DateTime.Now;  
    msg.Participante = participante;  
    msg.Mensagem = mensagem;  
  
    Mensagens.Add(msg);  
}
```

```
//  
// Validar Mensagem  
//  
private static void ValidarMensagem(  
    string participante, string mensagem)  
{  
    if (string.IsNullOrEmpty(participante) ||  
        participante.Trim().Length == 0)  
    {  
        throw new Exception(  
            "Participante não informado");  
    }  
  
    if (string.IsNullOrEmpty(mensagem) ||  
        mensagem.Trim().Length == 0)  
    {  
        throw new Exception("Mensagem não informada");  
    }  
}
```

Visual Studio 2015 - ASP.NET com C# Fundamentos

```
//  
// Últimas Mensagens  
//  
public static List<ChatMensagem> UltimasMensagens()  
{  
    var lista = (from m in Mensagens  
                orderby m.DataHora descending  
                select m).Take(100);  
  
    return lista.ToList();  
}
```

```
//  
// Iniciar  
//  
internal static void Iniciar()  
{  
    var lista = new List<ChatMensagem>();  
    HttpContext  
        .Current  
        .Application.Add(chaveChat, lista);  
  
    var participantes = new List<string>();  
    HttpContext  
        .Current  
        .Application  
        .Add(chaveparticipantes, participantes);  
  
    participantes.Add(adminNome);  
}
```

```
}// Fim da Classe  
}// Fim do Namespace
```

28. No arquivo **default.aspx**, insira os métodos para exibir dados na interface e teste o programa:

```
using Cap09Lab01.Db;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;

namespace Cap09Lab01
{
    public partial class Default : System.Web.UI.Page
    {
```

```
        //
        // Exibir
        //
        private void Exibir(string msg)
        {
            erroLabel.Text = msg;
            erroLabel.Visible = true;
        }
```

```
        //
        // Ocultar
        //
        private void Ocultar()
        {
            erroLabel.Visible = false;
        }
```

```
        //
        // Exibir Login
        //
        private void ExibirLogin()
        {
            MultiView1.ActiveViewIndex = 0;
        }
```

```
//  
// Load  
  
protected void Page_Load(  
    object sender, EventArgs e)  
{  
    if (!Page.IsPostBack)  
    {  
        ExibirLogin();  
    }  
}  
  
}// Fim da Classe  
}// Fim do namespace
```

29. Escreva o código do evento exibirChat:

```
public partial class Default : System.Web.UI.Page  
{  
  
    // ErroExibir  
    private void ErroExibir(...)  
  
    // ErroOcultar  
    private void ErroOcultar(...)  
  
    // Exibir Login  
    private void ExibirLogin(...)  
  
    // Load  
    protected void Page_Load(...)
```

```
//  
// ExibirChat  
//  
private void ExibirChat()  
{  
    try  
    {  
        ErroOcultar();  
  
        chatRepeater.DataSource =  
            ChatDb.UltimasMensagens();  
        chatRepeater.DataBind();  
        MultiView1.ActiveViewIndex = 1;  
    }  
    catch (Exception ex)  
    {  
        ErroExibir(ex.Message);  
    }  
}  
  
}// Fim da Classe
```

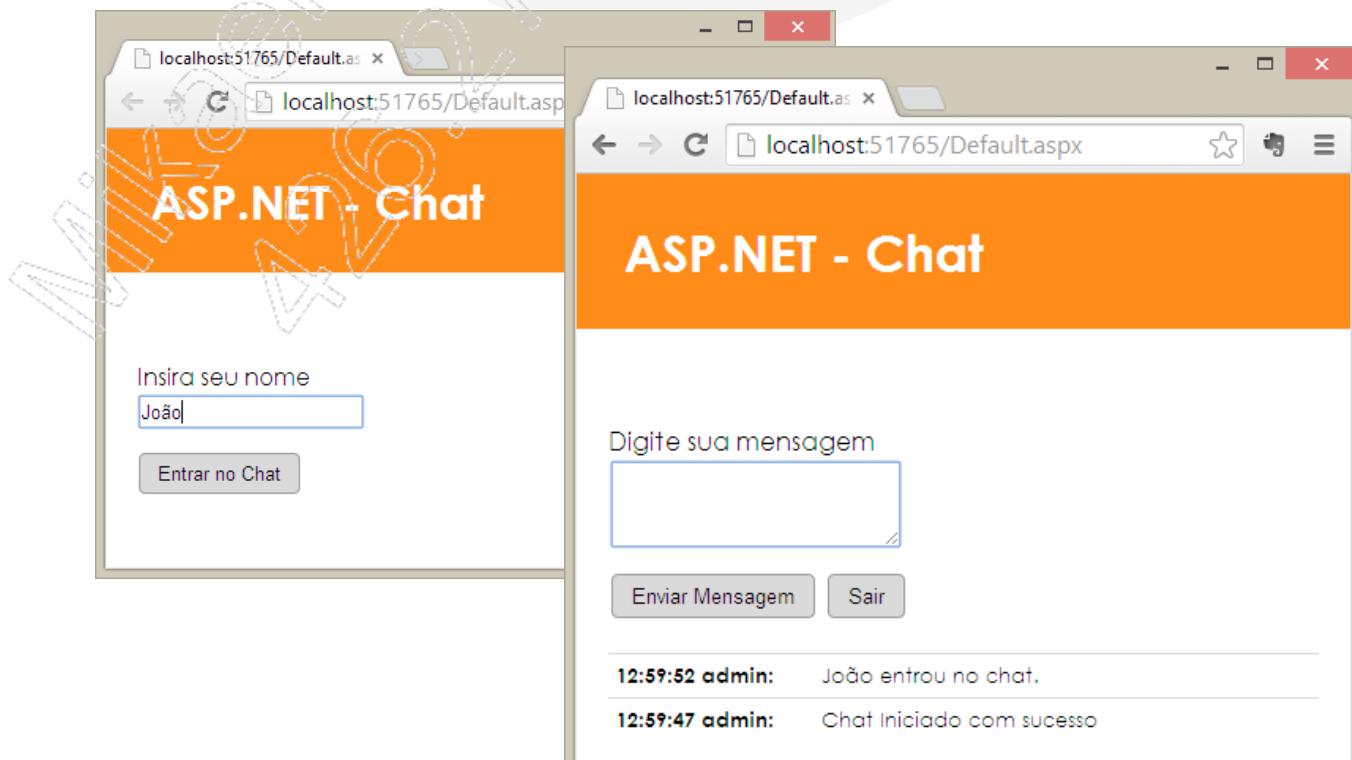
30. Escreva o código do evento entrarButton:

```
public partial class Default : System.Web.UI.Page  
{  
  
    // Exibir  
    private void Exibir...  
  
    // Ocultar  
    private void Ocultar()...  
  
    // Exibir Login  
    private void ExibirLogin()...  
  
    // Load  
    protected void Page_Load(...  
  
    // ExibirChat  
    private void ExibirChat()...
```

Visual Studio 2015 - ASP.NET com C# Fundamentos

```
//  
// Entrar  
//  
protected void entrarButton_Click(  
    object sender, EventArgs e)  
{  
    try  
{  
        ErroOcultar();  
  
        string nome = nomeTextBox.Text;  
        ChatDb.Entrar(nome);  
        ViewState["nome"] = nome;  
        ExibirChat();  
        mensagemTextBox.Focus();  
        mensagemTextBox.Text = string.Empty;  
  
    }  
    catch (Exception ex)  
{  
        ErroExibir(ex.Message);  
    }  
}  
  
}// Fim da Classe
```

31. Teste o programa:



32. Escreva o código do evento enviarMensagem:

```
public partial class Default : System.Web.UI.Page
{
    // ErroExibir
    private void ErroExibir...

    // ErroOcultar
    private void ErroOcultar()...

    // Exibir Login
    private void ExibirLogin()...

    // Load
    protected void Page_Load(...)

    // ExibirChat
    private void ExibirChat()...

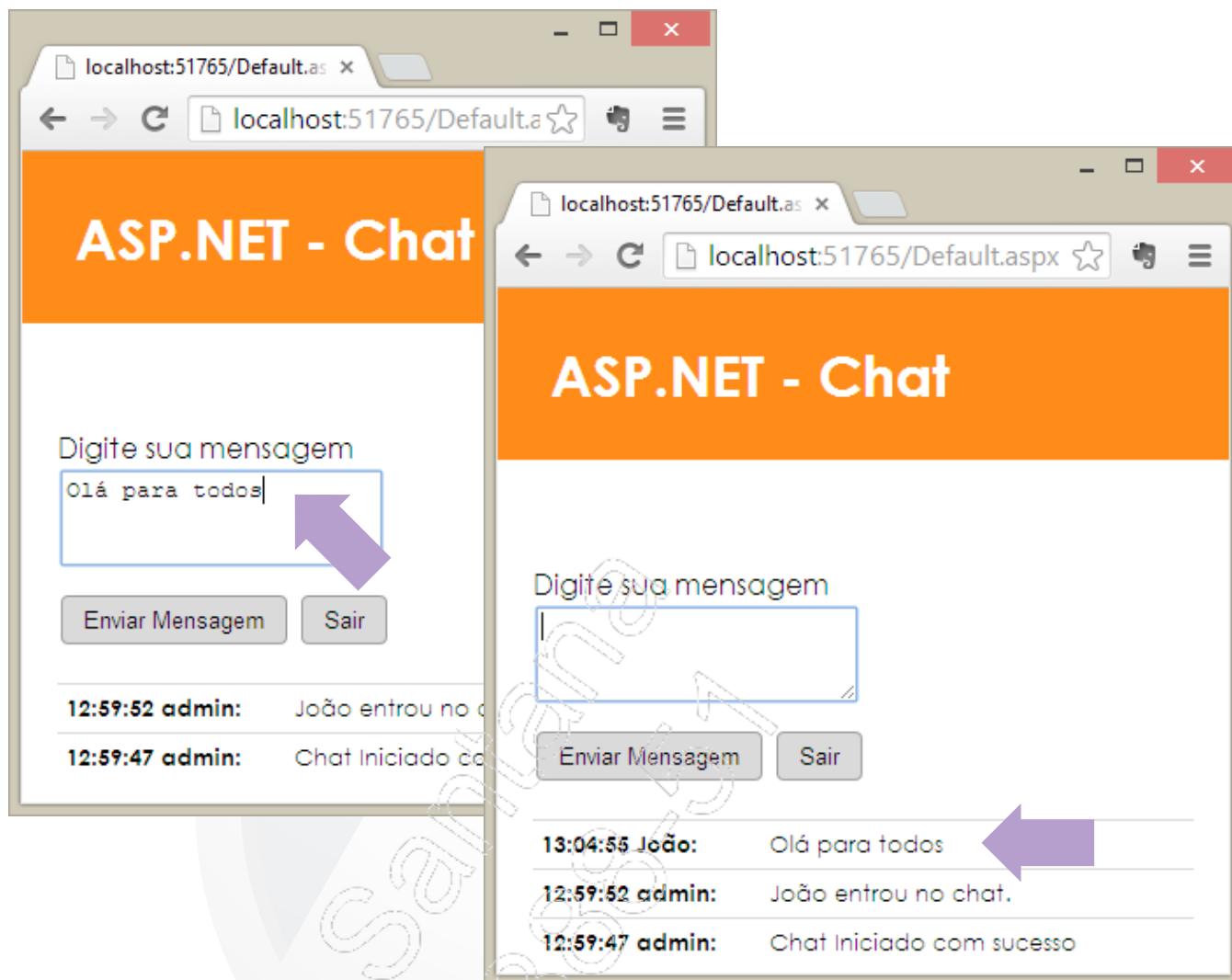
    // Entrar
    protected void entrarButton_Click.....

    //
    // Enviar mensagem
    //
    protected void enviarMensagemButton_Click(
        object sender, EventArgs e)
    {
        string nome = (string)ViewState["nome"];
        string mensagem = mensagemTextBox.Text;

        try
        {
            ErroOcultar();
            ChatDb.EnviarMensagem(nome, mensagem);
            ExibirChat();
            mensagemTextBox.Focus();
            mensagemTextBox.Text = string.Empty;
        }
        catch (Exception ex)
        {
            ErroExibir(ex.Message);
        }
    }
}

// Fim da Classe
```

33. Teste o envio de mensagem:



34. Escreva o código do botão Sair:

```
public partial class Default : System.Web.UI.Page
{
    // Exibir Erro
    private void ExibirErro...
    // Ocultar Erro
    private void OcultarErro()...

    // Exibir Login
    private void ExibirLogin()...

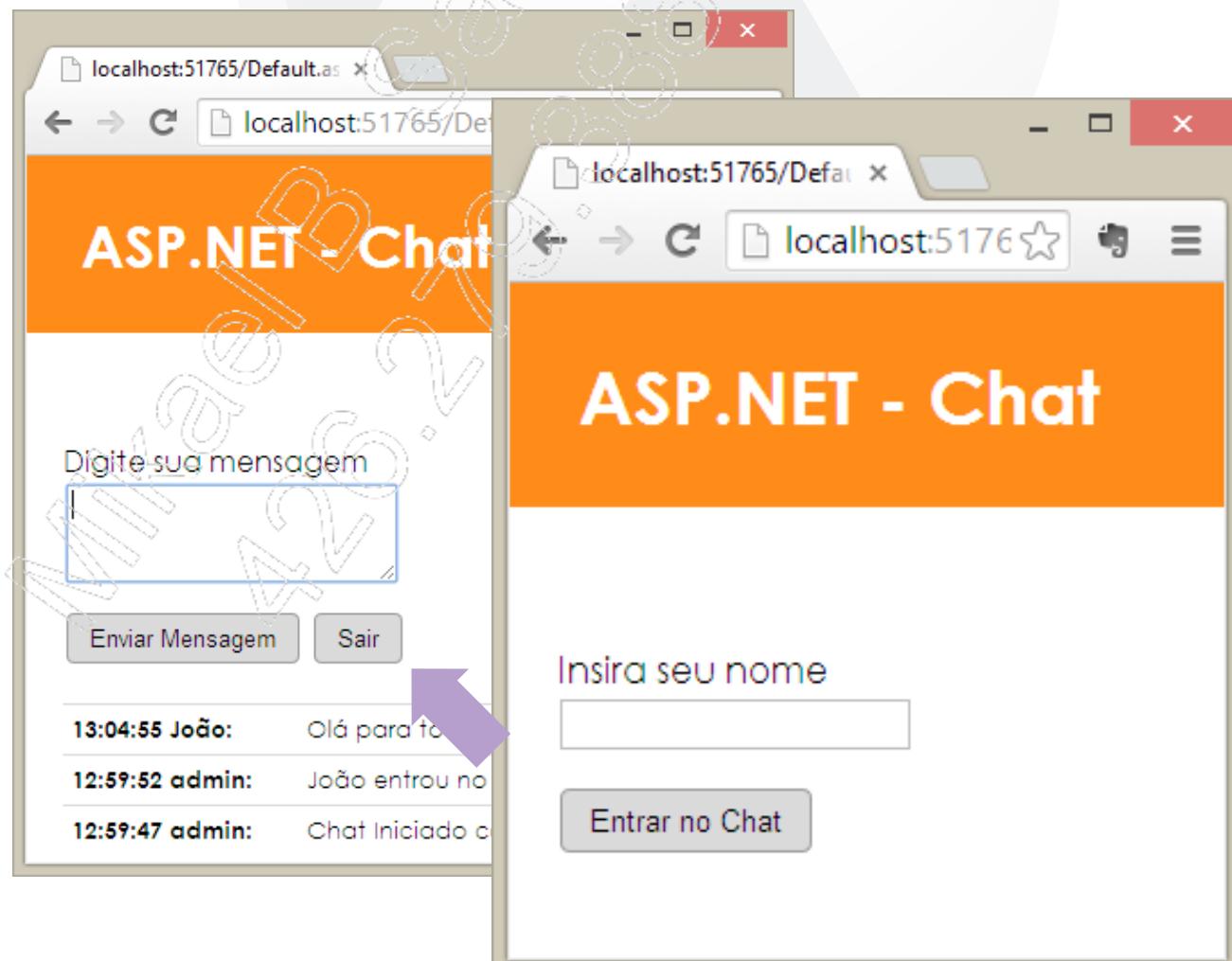
    // Load
    protected void Page_Load(...)

    // Exibir Chat
    private void ExibirChat()...
```

```
// Entrar  
protected void entrarButton_Click....  
  
// Enviar mensagem  
protected void enviarMensagemButton_Click...
```

```
// Sair  
protected void sairButton_Click(  
    object sender, EventArgs e)  
{  
    ErroOcultar();  
    ViewState.Remove("nome");  
    nomeTextBox.Text = string.Empty;  
    ExibirLogin();  
}  
}
```

35. Teste o evento Sair:



Visual Studio 2015 - ASP.NET com C# Fundamentos

36. Para que o chat seja atualizado automaticamente, insira um controle **Timer** e **UpdatePanel** do ASP.NET AJAX. Atenção: A div **chat-box** vai ficar dentro do **content-template** no UpdatePanel;

```
<form id="form1" runat="server">

    <asp:ScriptManager ID="ScriptManager1" runat="server">
        </asp:ScriptManager>

    <h1>ASP.NET - Chat</h1>
    <asp:MultiView ...>
        <asp:View ID="View1" runat="server">
            ...
            </asp:View>

        <asp:View ID="View2" runat="server">
            <section id="chat">
                <asp:Label .... mensagem</asp:Label>
                <asp:TextBox ...>
                <asp:Button ... Text="Enviar Mensagem" />
                <asp:Button Text="Sair" ...>
            </section>
        </asp:View>
    </asp:MultiView>

    <asp:UpdatePanel ID="UpdatePanel1" runat="server">
        <ContentTemplate>

            <asp:Timer ID="Timer1" runat="server" Interval="4" OnTick="Timer1_Tick">
            </asp:Timer>

            <div class="chat-box">
                ...
            </div>

        </ContentTemplate>
    </asp:UpdatePanel>

</form>
```

37. Escreva o código do evento **Tick** controle **Timer**:

```
public partial class Default : System.Web.UI.Page
{
    // Exibir
    private void Exibir...

    // Ocultar
    private void Ocultar()...

    // Exibir Login
    private void ExibirLogin()...

    // Load
    protected void Page_Load(...)

    // Exibir Chat
    private void ExibirChat()...

    // Entrar
    protected void entrarButton_Click(...)

    // Enviar mensagem
    protected void enviarMensagemButton_Click(...)

    // Sair
    protected void sairButton_Click(...)

    // Timer1
    protected void Timer1_Tick(
        object sender, EventArgs e)
    {
        ExibirChat();
    }
}
```

Visual Studio 2015 - ASP.NET com C# Fundamentos

38. Teste o código abrindo dois navegadores e verificando se os dados são atualizados automaticamente:

