Profº Elton Rodrigo

Esta apostila foi criada e editada com base em informações contidas nos sites **imasters.com.br**, **caelum.com.br**, **linhadecodigo.com.br**, **infowester.com, oficinadanet.com.br, caelum.com.br, diegomacedo.com.br** e **devmedia.com.br.**

As imagens nela contidas foram capturadas com o recurso *PrintScreen*.

Sumário

1. Introdução

ASP.NET (*Active Server Pages.NET*) compõe a forma de se criar páginas da Internet utilizando-se a plataforma .NET. Esta plataforma provê uma série de serviços, através de classes existentes no Microsoft .NET Framework. Com ele, é possível a construção não apenas de páginas Web, mas também, de aplicativos desktop, aplicativos para dispositivos móveis, objetos de negócios, web services, etc.

Pode-se dizer que a plataforma .NET é similar à plataforma Java, que também oferece diversos serviços. Dentre suas principais diferenças, destacam-se as seguintes características: a plataforma Java utiliza a linguagem de programação Java e seus aplicativos são híbridos (compilados e interpretados, característica que permite que os aplicativos sejam multi-plataforma); enquanto que em .NET, pode-se utilizar Visual Basic.NET, C++, JScript.NET ou C#, e os aplicativos gerados são compilados em uma linguagem chamada MSIL (Microsoft Intermediate Language). No caso das páginas ASP.NET, sua compilação é realizada no momento em que elas são chamadas pela primeira vez através do browser, que apresenta o conteúdo final em HTML (Hypertext Markup Language).

2. O que é C# e .Net

2.1 Um pouco sobre a história do C# e .Net

No final da década de 1990 a **Microsoft** tinha diversas tecnologias e linguagens de programação para resolver muitos problemas diferentes. Toda vez que um programador precisava migrar para uma nova linguagem, era necessário aprender tanto a nova linguagem quanto suas bibliotecas e conceitos. Para solucionar esses problemas, a Microsoft recorreu à linguagem **Java**.

O Java agradou os engenheiros da Microsoft pois com ela podíamos construir programas que eram independentes do ambiente de execução, além de possuir diversas bibliotecas com soluções prontas para diversos problemas. Para lançar produtos baseados no Java, a Microsoft assinou um acordo de licenciamento com a Sun para utilizar o Java em ambiente Windows.

Porém, a linguagem Java possuía um grave problema: ela não se comunicava bem com as bibliotecas de código nativo (código de máquina) que já existiam. Para resolver isso, a Microsoft decidiu criar a sua própria implementação do Java chamado **J++**, que possuía extensões proprietárias que resolviam o problema de comunicação com o código nativo existente. Para o desenvolvimento dessa nova implementação do Java, a Microsoft contratou um engenheiro chamado Anders Hejlsberg, um dos principais nomes por trás do Delphi.

O J++ era uma versão da linguagem Java que só podia ser executada no ambiente Microsoft. Seu código não podia ser executado em mais nenhum ambiente Java, o que violava o licenciamento feito com a Sun e, por isso, a Microsoft foi processada. Uma das mais conhecidas batalhas judiciais da época.

Sem o J++, a Microsoft foi obrigada a repensar sua estratégia sobre como lidar com as diferentes linguagens e tecnologias utilizadas internamente. A empresa começou a trabalhar em uma nova plataforma que seria a base de todas as suas soluções, que posteriormente foi chamada de **.Net**. Esse novo ambiente de desenvolvimento da Microsoft foi desde o início projetado para trabalhar com diversas linguagens de programação, assim diversas linguagens diferentes compartilhariam o mesmo conjunto de bibliotecas. Com isso, para um programador migrar de uma linguagem para outra ele precisaria apenas aprender a linguagem sem se preocupar com as bibliotecas e APIs.

Além de uma plataforma a Microsoft também precisava de uma linguagem de programação. Um novo projeto de linguagem de programação foi iniciado, o projeto **COOL (C-like Object Oriented Language)**. Anders Hejlsberg foi escolhido como engenheiro chefe desse novo projeto. COOL teve seu design baseado em diversas outras linguagens do mercado como Java, C, C++, Smalltalk, Delphi e VB. A ideia era estudar os problemas existentes e incorporar soluções.

Em 2002, o projeto COOL foi lançado como linguagem **C# 1.0**, junto com o ambiente **.Net 1.0**. Atualmente, a linguagem C# está em sua versão 7.2, e o .Net na versão 4.7, tendo evoluído com expressiva velocidade, adotando novidades na sua sintaxe que a diferenciaram bastante do Java e outras concorrentes.

2.2 Máquina virtual

Em uma linguagem de programação como C e Pascal, temos a seguinte situação quando vamos compilar um programa:

O código fonte é compilado para código de máquina específico de uma plataforma e sistema operacional. Muitas vezes o próprio código fonte é desenvolvido visando uma única plataforma!

Esse código executável (binário) resultante será executado pelo sistema operacional e, por esse motivo, ele deve saber conversar com o sistema operacional em questão. Isto é, temos um código executável diferente para cada sistema operacional diferente.

Precisamos reescrever um mesmo pedaço da aplicação para diferentes sistemas operacionais, já que eles não são compatíveis.

O C# utiliza o conceito de máquina virtual. Entre o sistema operacional e a aplicação existe uma camada extra responsável por "traduzir" — mas não apenas isso — o que sua aplicação deseja fazer para as respectivas chamadas do sistema operacional onde ela está rodando no momento.

2.3 O ambiente de desenvolvimento do C#

Nessa apostila escreveremos todo o código utilizando o **Visual Studio**, a versão gratuita da ferramenta de desenvolvimento de aplicações, que é distribuída pela própria Microsoft.

O Visual Studio pode ser encontrado no site:

https://www.visualstudio.com/pt-br/downloads/

A versão que utilizaremos na apostila é a Visual Studio Professional 2013.

Durante a instalação do Visual Studio, o .Net Framework também será automaticamente instalado em sua máquina, então ela estará pronta executar as aplicações escritas em C#.

2.4 Executando aplicações sem o Visual Studio

Como vimos anteriormente, para executarmos uma aplicação C# precisamos da máquina virtual da linguagem além das bibliotecas do .Net Framework. Ao instalarmos o Visual Studio, todo esse ambiente de execução de programas é automaticamente instalado em nossas máquinas, mas e se quisermos executar o programa em um computador que não tenha o Visual Studio instalado, o computador de um cliente, por exemplo?

Nesse caso precisamos instalar apenas o ambiente de execução no computador do cliente. Para isso podemos utilizar um pacote de instalação fornecido pela própria Microsoft, esses são os .Net Framework Redistributable. O pacote de instalação para a última versão do .Net Framework (4.5.1 lançada em 2013) pode ser encontrada no seguinte site:

http://www.microsoft.com/en-us/download/details.aspx?id=40779

2.5 ASP.NET, HTML e C#

As páginas ASP.NET produzem arquivos de extensão ".aspx". Normalmente, elas são divididas em duas seções principais: HTML e, no nosso caso, C# (C Sharp).

Assim, o código C# é colocado dentro do elemento script, antes do código HTML propriamente. É necessário ainda, que seja explicitada a linguagem que está sendo utilizada, o que é feito com uso da diretiva "Page Language", que é adicionada no início do código da página. Um exemplo simples de uma página ASP.NET é o seguinte:

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

<!DOCTYPE html>

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

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

Nota-se, que alguns elementos possuem um atributo *runat*, que possui o valor server. Isto significa que todo o conteúdo deste elemento estará sendo executado no servidor. Isto é necessário pelo fato

de ser o servidor, o lugar onde está instalado o .NET framework (o cliente não necessariamente o terá instalado).

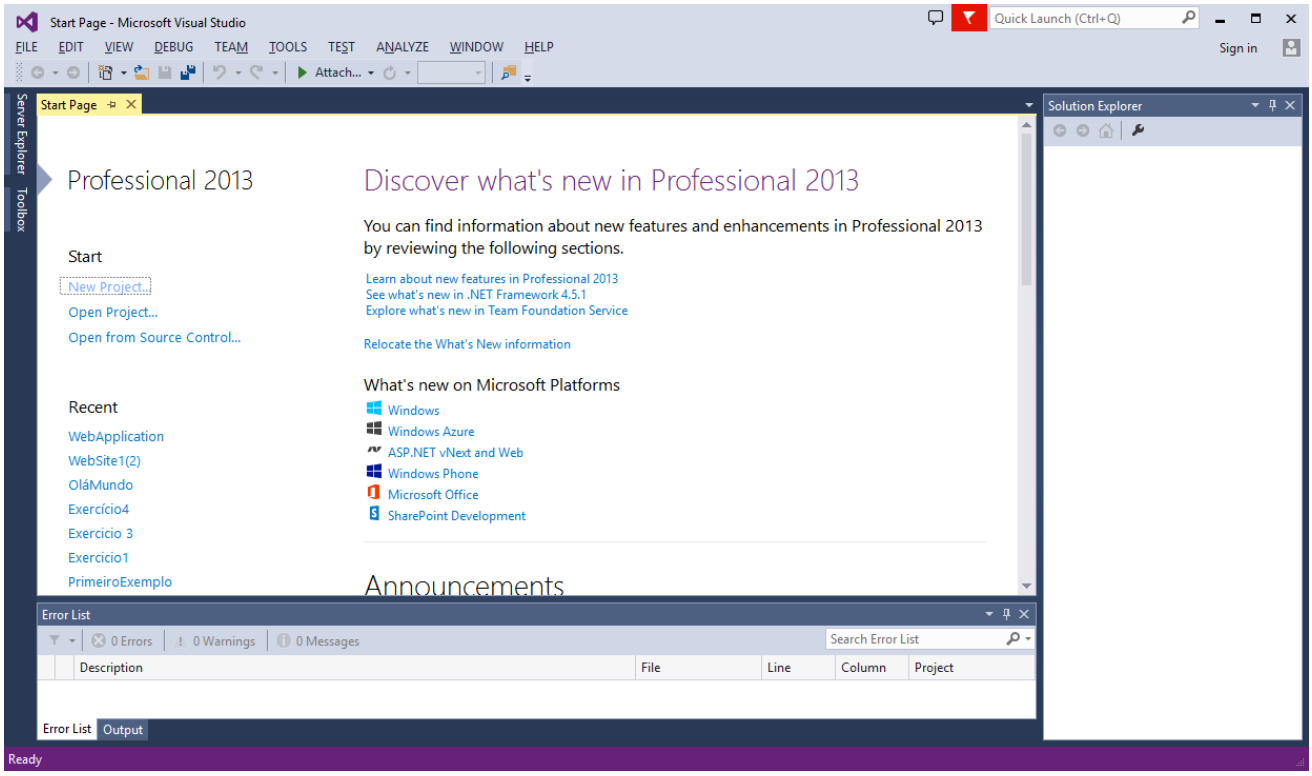
A linguagem C# possui muitas semelhanças com as linguagens C, C++ e Java. Assim, o código é case-sensitive (existe diferença entre letras maiúsculas e minúsculas), e declarações de variáveis, operadores e estruturas de controle são utilizados praticamente da mesma maneira que nestas linguagens. Por exemplo, para se declarar uma variável do tipo inteiro, basta escrever o seguinte:

```
int variavel;
```

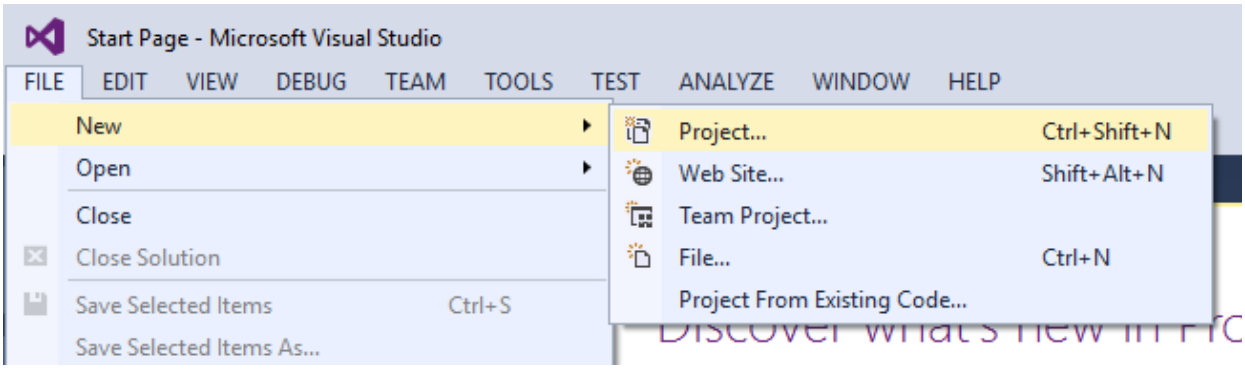
A forma do código ASP.NET se comunicar com o código HTML é através de funções que são invocadas por controles de servidor, por meio do evento *onclick*. Outra forma desta comunicação acontecer ocorre no momento em que a página é carregada, por meio da função *Page_Load*.

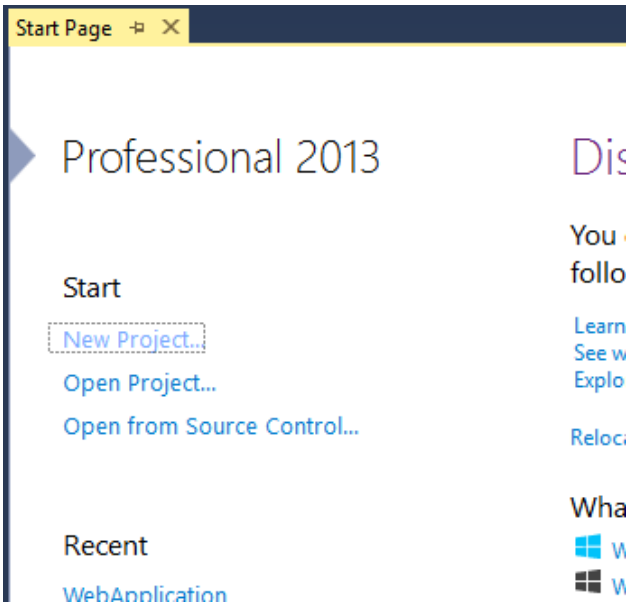
2.6 Criando um Novo Projeto ASP .NET

Abrindo o Visual Studio Express 2013 e você já verá o novo estilo visual adotado pelo Visual Studio.



Abra o menu **File** e clique em **New Project** ou na página inicial (**Start Page**) clique no link - **New Project...**

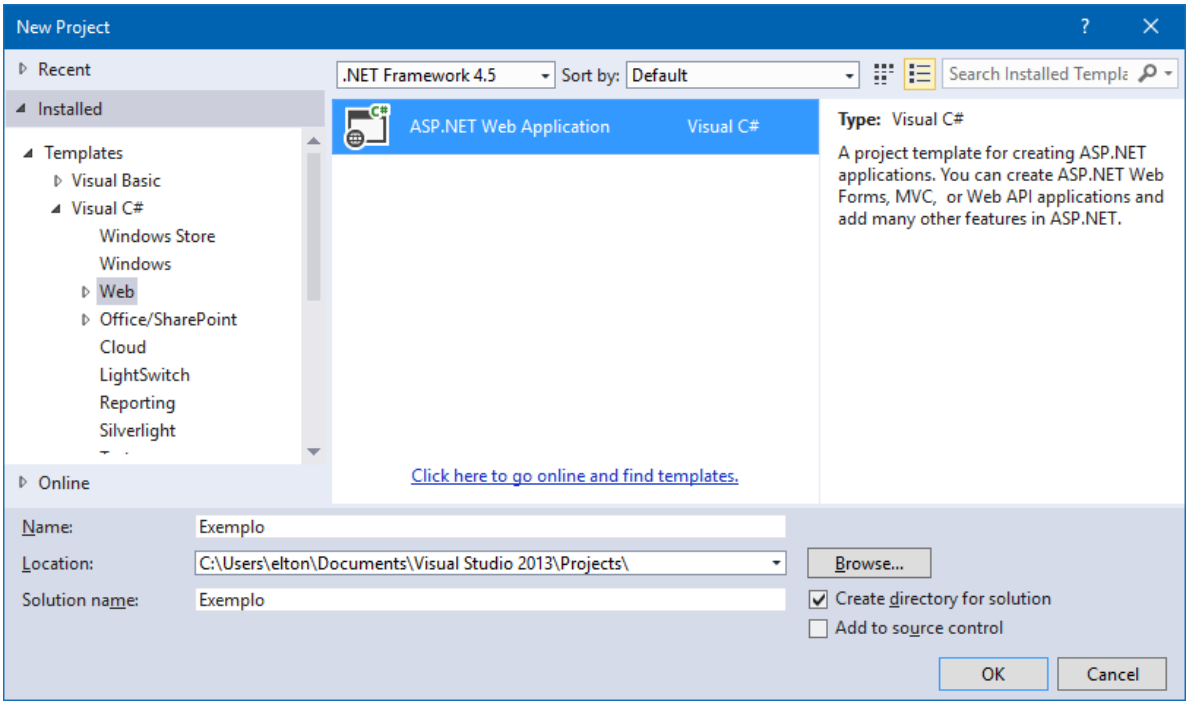




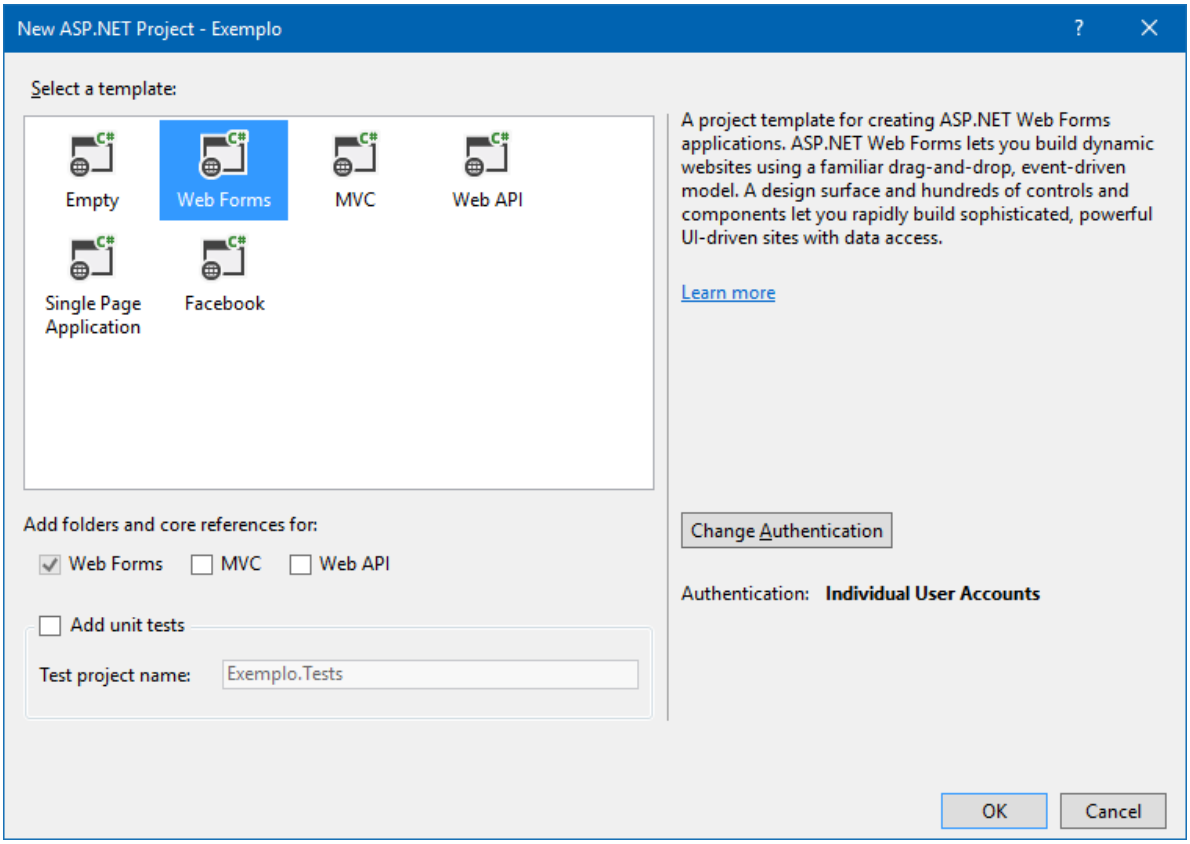
Selecione o **Visual C# -> Web**;

Selecione o projeto **ASP .NET Web Application**;

Informe o nome do projeto e clique no botão **OK**.

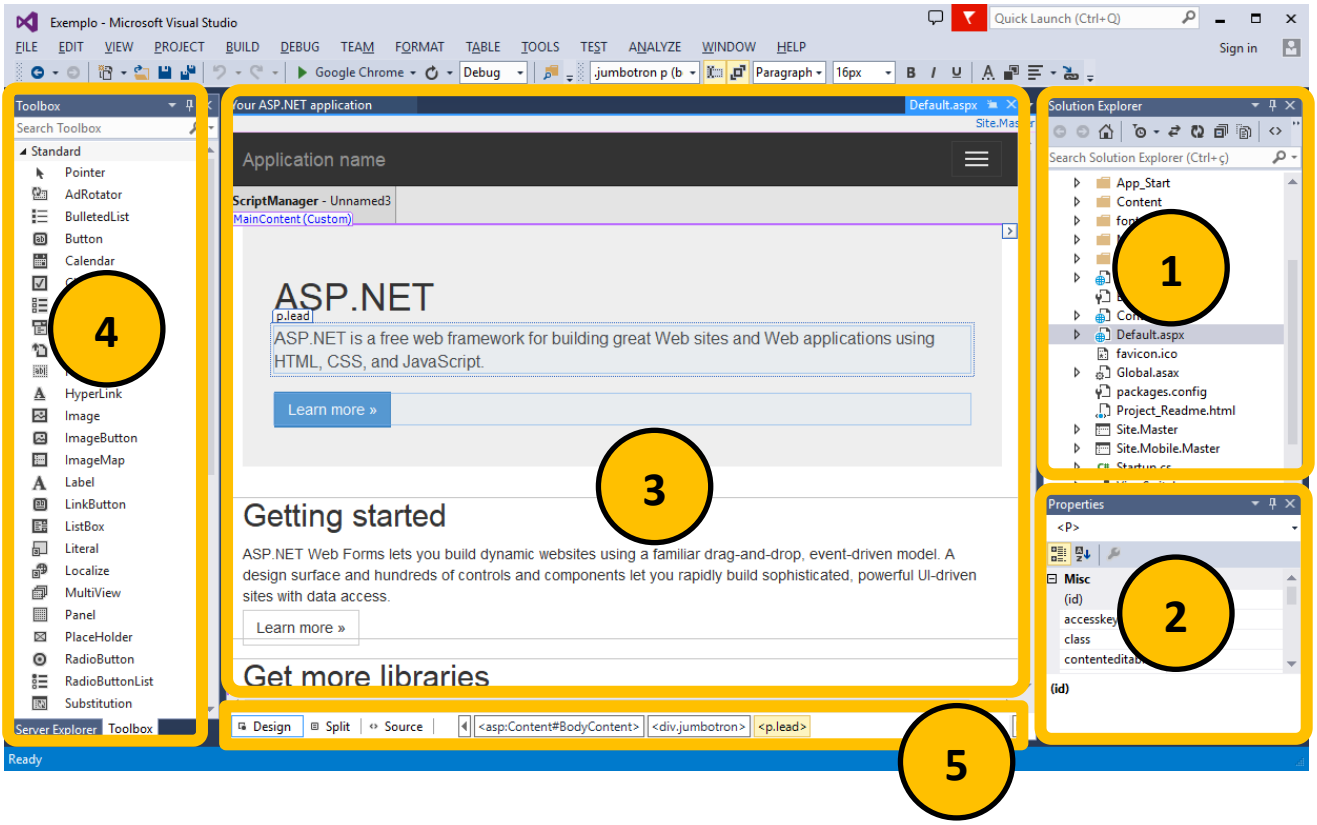


Na próxima tela, selecione a template **Web Forms** e clique em **OK**.



Após alguns segundos será criado um projeto com uma estrutura básica contendo pastas e arquivos e apresentado na janela Solution Explorer;

Abaixo vemos a solução exibida na janela Solution Explorer e a página **Default.aspx** exibida no modo **Design**;

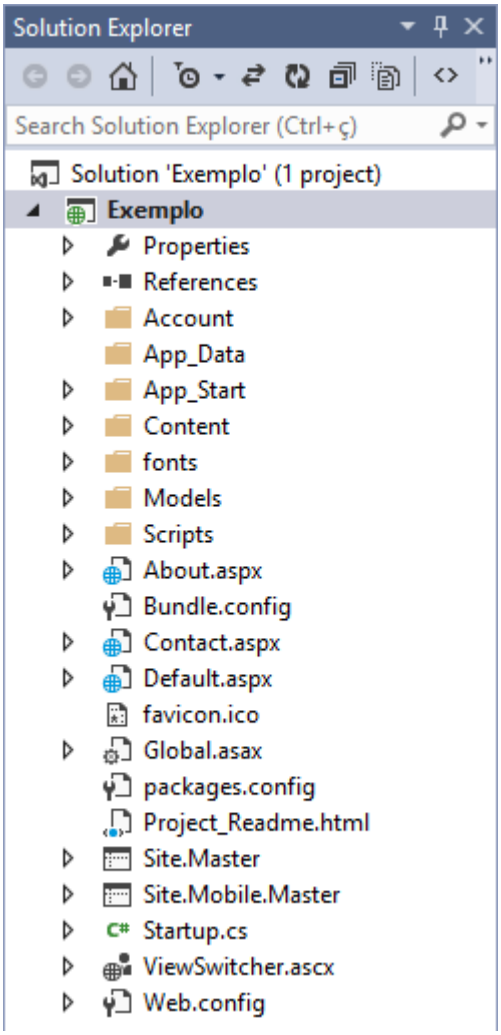


Podemos identificar as seguintes áreas no desenvolvedor:

1. **Janela Solution Explorer** - Exibe a solução, os projetos e seus respectivos arquivos e pastas;
2. **Janela de Propriedades** - Exibe as propriedades do componente selecionado;
3. **Janela de Documentos** - exibe o arquivo selecionado;
4. **ToolBox** - Exibe os controles disponíveis conforme o componente selecionado;
5. **Tab de visões do Documento** - Permite selecionar os modos de exibição do arquivo selecionado;

Você pode alternar para o tipo de exibição **Source** ou no modo **Split** permitir a visualização dos dois tipos: **Design** e o seu respectivo **Source**;

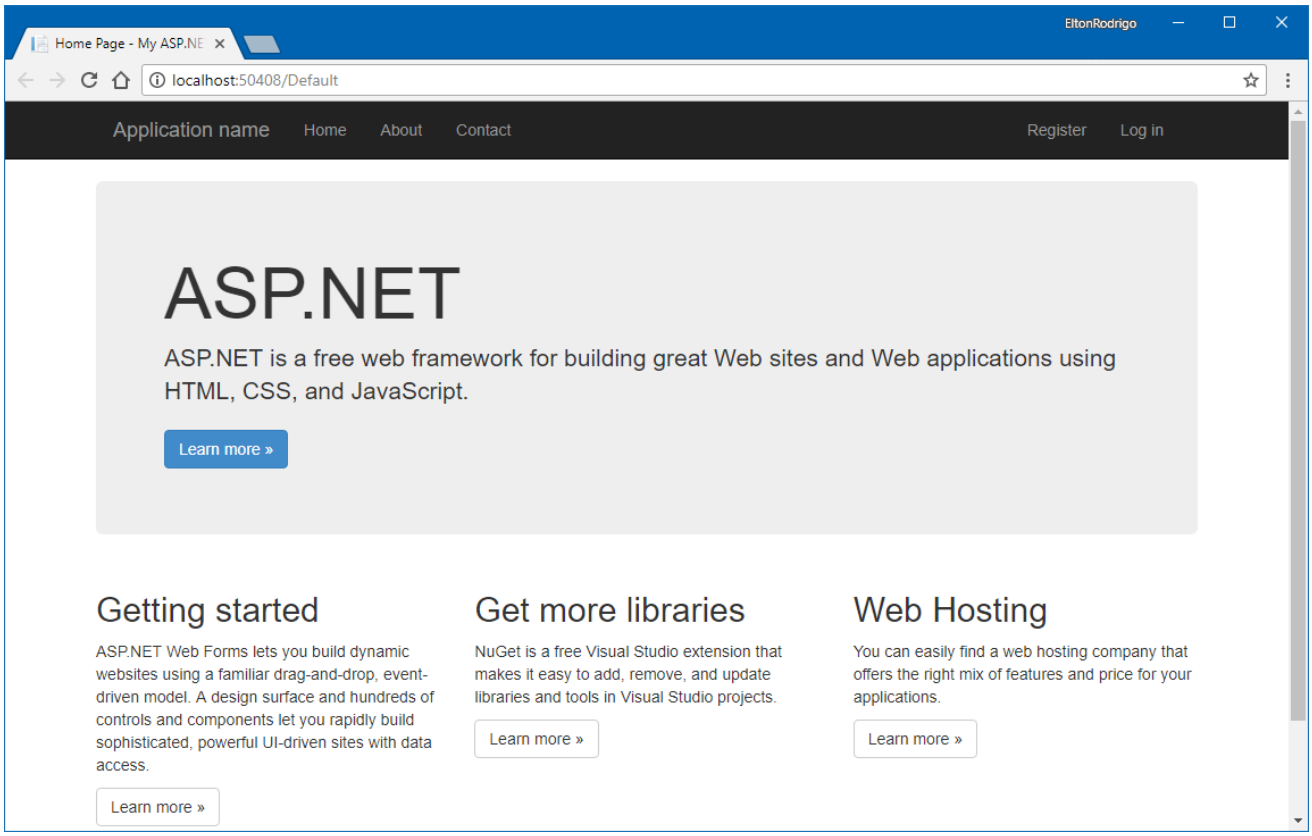
Na janela Solution Explorer podemos gerenciar os arquivos do projeto. Vamos dar uma olhada nas pastas que foram adicionados à sua aplicação no Solution Explorer. O modelo de aplicativo web acrescenta uma estrutura de pasta básica:



O Visual Studio cria algumas pastas e arquivos iniciais para o seu projeto. Dentre os arquivos destacamos os seguintes:

Arquivo	Propósito
Default.aspx	Geralmente é a primeira página a ser exibida quando a aplicação for executada no navegador. (Você pode alterar este comportamento)
Site.Master	Uma página que permite que você crie um layout consistente e use um comportamento padrão para as páginas em seu aplicativo.
Global.asax	Um arquivo opcional que contém o código para responder a nível de aplicação e sessão de a eventos gerados pelo ASP.NET ou por módulos HTTP.
Web.config	O arquivo de configuração da aplicação.

Executando a aplicação no seu navegador web padrão pressionando **F5** será apresentada a seguinte tela:



Há três páginas principais neste aplicativo Web padrão: **default.aspx** (Home), **About.aspx** e **Contact.aspx**.

Cada uma dessas páginas pode ser alcançada a partir da barra de navegação superior. Há também duas páginas adicionais contidos na pasta **Account**, a página **Register.aspx** e página **Login.aspx**. Estas duas páginas permitem que você use os recursos de **membership**(associação) da ASP.NET para criar, armazenar e validar as credenciais do usuário.

O recurso membership ASP.NET armazena as credenciais de seus usuários em um banco de dados criado pela aplicação. Quando os usuários efetuarem o login, o aplicativo vai validar suas credenciais através da leitura desse banco de dados. A pasta Account do projeto contém os arquivos que implementam as várias partes do membership: ***registro, login, alteração de senha e autorização de acesso***.

2.5.1 O que são Web Forms?

O ASP.NET Web Forms são páginas que são baseadas na tecnologia Microsoft ASP.NET, na qual o código que é executado no servidor dinamicamente gera a saída de página Web para o navegador ou dispositivo cliente. Uma página ASP .NET Web Forms automaticamente renderiza o HTML compatível com o navegador para características como estilos, leiaute, etc.

Os Web Forms são compatíveis com qualquer linguagem suportada pelo CLR - Common Language Runtime da plataforma .NET, como o Microsoft Visual Basic e a Microsoft Visual C#. Além disso, os Web Forms são construídos sobre os recursos da plataforma .NET, que oferece benefícios como um ambiente gerenciado, segurança de tipo e herança.

Quando uma página ASP .NET Web Forms roda, a página passa por um ciclo de vida em que se realiza uma série de etapas de processamento. Estas etapas incluem inicialização, instanciamento de controles, restauração e manutenção do estado, a execução de código manipulador de eventos e a renderização.

Quando um servidor Web recebe uma solicitação para uma página, ele encontra a página, processa, envia para o navegador e depois descarta todas as informações da página. Se o usuário solicita a mesma página novamente, o servidor repete toda a sequência, o reprocessando a página a partir do zero.

3. Variáveis e tipos primitivos

Na maioria das aplicações que escrevemos, não estamos interessados em apenas mostrar textos básicos para o usuário. Queremos também armazenar e processar informações.

Em um sistema bancário, por exemplo, estaríamos interessados em armazenar o saldo de uma conta e o nome do correntista. Para armazenar esses dados, precisamos pedir para o C# reservar regiões de memória que serão utilizadas para armazenar informações. Essas regiões de memória são conhecidas como variáveis.

As variáveis guardam informações de um tipo específico. Podemos, por exemplo, guardar um número inteiro representando o número da conta, um texto para representar o nome do correntista ou um número real para representar o saldo atual da conta. Para utilizar uma variável, devemos primeiramente declará-la no texto da aplicação.

Na declaração de uma variável, devemos dizer seu tipo (inteiro, texto ou real, por exemplo) e, além disso, qual é o nome que usaremos para referenciá-la no texto do programa. Para declarar uma variável do tipo inteiro que representa o número de uma conta, utilizamos o seguinte código:

```
int numeroDaConta;
```

Repare no " ; " no final da linha. Como a declaração de uma variável é um comando da linguagem C#, precisamos do ; para terminá-lo.

Além do tipo int (para representar inteiros), temos também os tipos double e float (para números reais), string (para textos), entre outros.

Depois de declarada, uma variável pode ser utilizada para armazenar valores. Por exemplo, se estivéssemos interessados em guardar o valor 1 na variável numeroDaConta que declaramos anteriormente, utilizaríamos o seguinte código:

```
numeroDaConta = 1;
```

Lê-se "numeroDaConta recebe 1". Quando, no momento da declaração da variável, sabemos qual será seu valor, podemos utilizar a seguinte sintaxe para declarar e atribuir o valor para a variável.

```
double saldo = 100.0;
```

3.1 Tipos Primitivos

Vimos que no C# toda variável possui um tipo, utilizamos o **int** quando queremos armazenar valores inteiros e **double** para números reais. Agora vamos descobrir quais são os outros tipos de variáveis do C#.

Tipo	Implementação
byte	Inteiro de 8 bits sem sinal (0 a 255).
sbyte	Inteiro de 8 bits com sinal (-127 a 128).
ushort	Inteiro de 16 bits sem sinal (0 a 65 535).
short	Inteiro de 16 bits com sinal (-32 768 a 32 767).
uint	Inteiro de 32 bits sem sinal (0 a 4 294 967 295).
int	Inteiro de 32 bits com sinal (-2 147 483 648 a 2 147 483 647).
ulong	Inteiro de 64 bits sem sinal (0 a 18 446 744 073 709 551 615).
long	Inteiro de 64 bits com sinal (-9 223 372 036 854 775 808 a 9 223 372 036 854 775 807).
double	Ponto flutuante binário de 8 bytes, 15 dígitos decimais de precisão.
float	Ponto flutuante binário de 4 bytes, 7 dígitos decimais de precisão.
decimal	Ponto flutuante decimal de 128 bits. 28 dígitos decimais de precisão.
bool	Pode ter os valores true e false. Não é compatível com inteiro.
char	Um único caractere Unicode de 16 bits. Não é compatível com inteiro.

Os tipos listados nessa tabela são conhecidos como tipos primitivos ou value types da linguagem C#. Toda vez que atribuímos um valor para uma variável de um tipo primitivo, o C# copia o valor atribuído para dentro da variável.

Agora que conhecemos os tipos primitivos da linguagem C#, vamos ver como é que eles interagem dentro de uma aplicação. Suponha que temos um código que declara uma variável do tipo inteiro e depois tenta copiar seu conteúdo para uma variável long:

```
int valor = 1;
long valorGrande = valor;
```

4. Estruturas de controle

4.1 Tomando decisões no código

Voltando para nosso exemplo de aplicação bancária, queremos permitir um saque somente se o valor a ser retirado for menor ou igual ao saldo da conta, ou seja, se o saldo da conta for maior ou igual ao valor do saque, devemos permitir a operação, do contrário não podemos permitir o saque. Precisamos fazer execução condicional de código.

No C#, podemos executar código condicional utilizando a construção if:

```
if (condicao)
            {
                // Esse código será executado somente se a condição for
verdadeira
            }
```

No nosso exemplo, queremos executar a lógica de saque apenas se o saldo for maior ou igual ao valor do saque:

```
double saldo = 100.0;
            double valorSaque = 10.0;
            if (saldo >= valorSaque)
            {
                // código do saque.
            }
```

O código do saque deve diminuir o saldo da conta e mostrar uma mensagem para o usuário indicando que o saque ocorreu com sucesso:

```
double saldo = 100.0;
            double valorSaque = 10.0;
            if (saldo >= valorSaque)
            {
                saldo = saldo - valorSaque;
                Label1.Text = "Saque realizado com sucesso";
            }
```

Repare que, se a conta não tiver saldo suficiente para o saque, o usuário não é avisado. Então estamos na seguinte situação: "Se a conta tiver saldo suficiente, quero fazer o saque, senão, quero mostrar a mensagem Saldo Insuficiente para o usuário". Para fazer isso, podemos usar o else do C#:

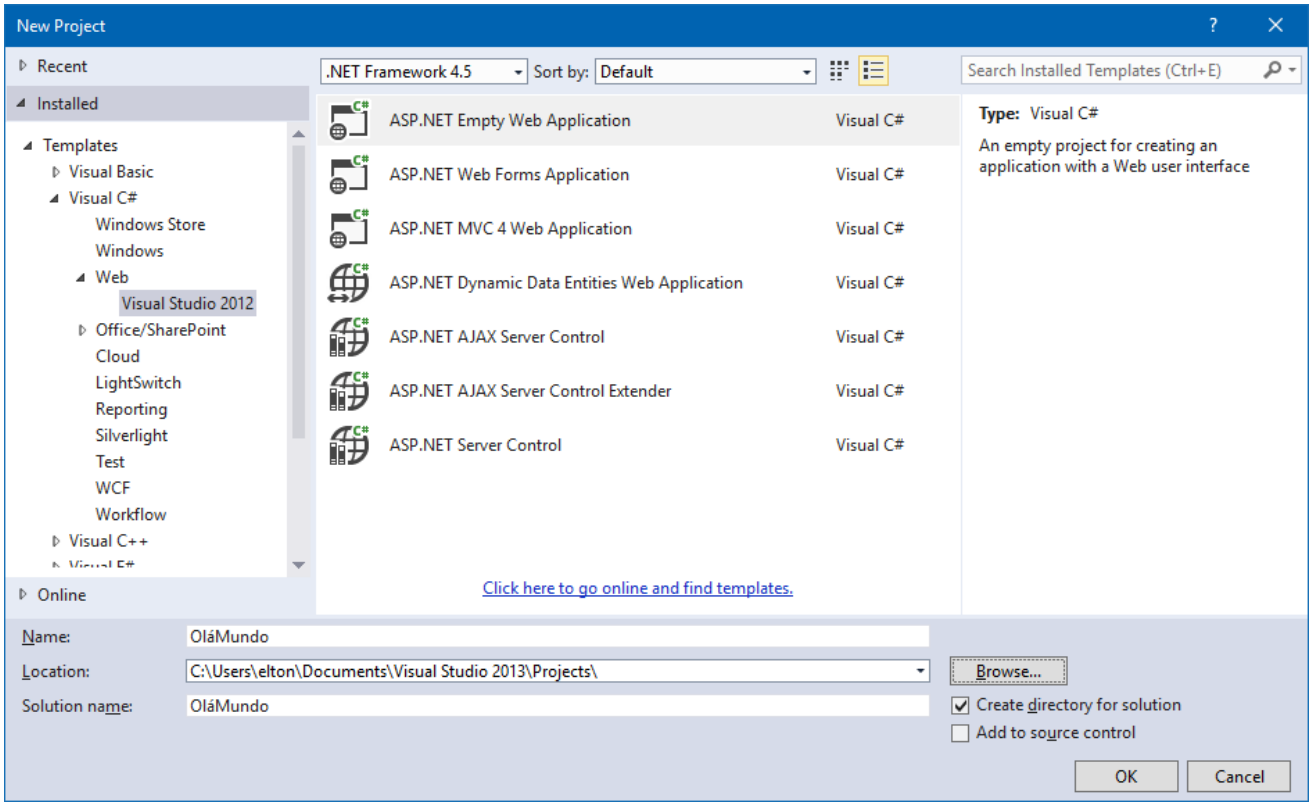
```
double saldo = 100.0;
            double valorSaque = 10.0;
            if (saldo >= valorSaque)
            {
                saldo = saldo - valorSaque;
                Label1.Text = "Saque realizado com sucesso";
            }

            else
            {
                Label1.Text = "Saldo Insuficiente";
            }
```

4.2 Iniciando programa em C# ou ASP.Net

Agora que já entendemos o funcionamento da linguagem C#, vamos começar a desenvolver a primeira aplicação utilizando o Visual Studio. Para criarmos um programa C# utilizando o Visual Studio precisamos inicialmente de um novo projeto.

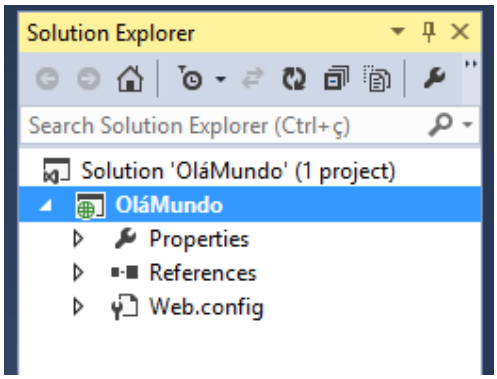
Dentro do Visual Studio 2013, aperte o atalho **Ctrl + Shift + N** para abrir o assistente de criação de novo projeto.



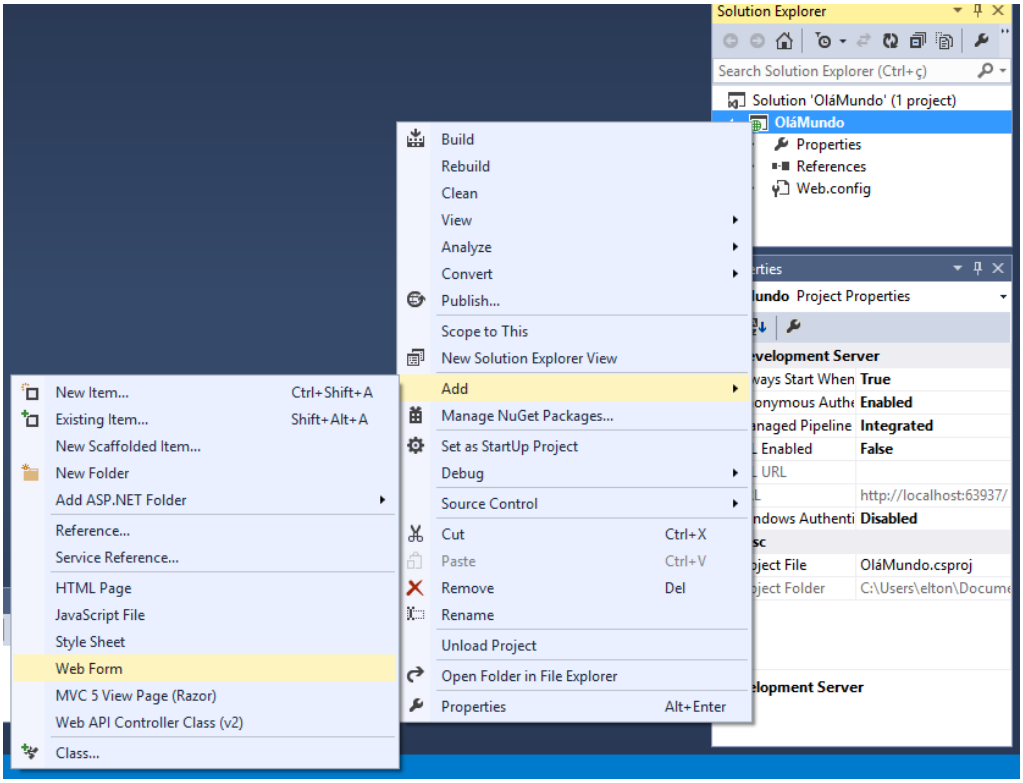
No canto esquerdo da janela do assistente de criação de novo projeto, podemos escolher a linguagem de programação que desejamos utilizar, escolha a opção **Visual C# / Web/ Visual Studio 2012**. Como tipo de projeto escolha a opção **ASP.Net Empty Web Application**, com isso estamos criando uma nova aplicação vazia em ASP.Net utilizando o C#.

No canto inferior da janela, podemos escolher o nome do projeto além da pasta em que ele será armazenado. Utilizaremos "OláMundo" como nome desse novo projeto.

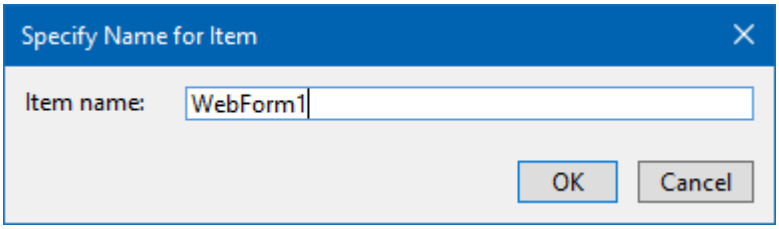
Dentro do painel Solution Explorer será exibido o projeto criado com as bibliotecas necessárias para o desenvolvimento da nossa aplicação.



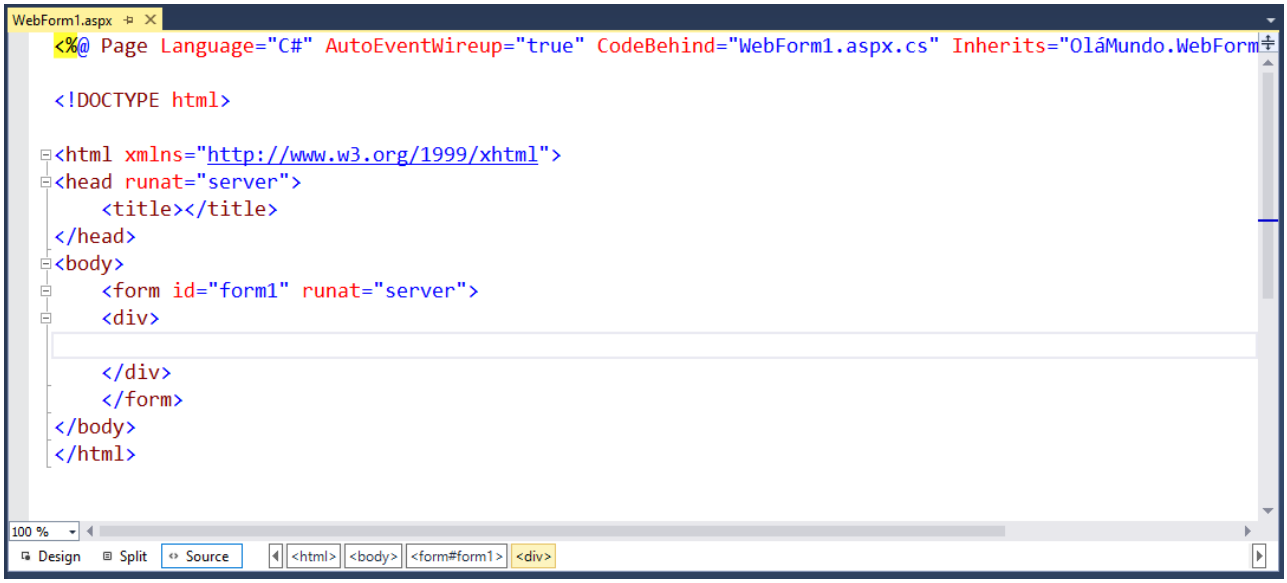
Clique com o botão direito do mouse sobre o projeto "OláMundo", depois clique em **Add / WebForm.**



Será exibido uma caixa pedindo para definir um nome para o nosso primeiro Formulário Web.



Na aba que se abre será criada uma estrutura HTML contendo as tags principais.



Podemos alterar seu modo de exibição através dos botões **Design**, **Split**, e **Source**.