# Certificate in C# Programming

## Creating Web Applications in C#

By Randal Root

# Module 03

In this Module, we will see how web controls are used in ASP.NET web pages. We’ll first look at how standard HTML controls can be upgraded to HTML Server controls and then we’ll look at how these differ from the standard ASP.NET Server Controls.

This module corresponds to the 2nd part of the course Textbook,

Beginning ASP.NET 4.5 in C#, by Matthew MacDonald, ISBN-13: 978-1-4302-4251-2.

*“Part 2: Developing ASP.NET Applications*

*The second part of this book delves into the heart of ASP.NET programming and introduces its event-based model. In Chapter 4, you’ll take a look around the Visual Studio design environment and learn a few fundamentals about web forms, events, and HTML5.* ***In Chapters 5 and 6, you learn how to program a web page’s user interface through a layer of objects called server controls.***

*Next you’ll explore two more essentials of ASP.NET programming. Chapter 7 presents techniques for handling errors. Chapter 8 describes strategies for state management. Taken together, the chapters in this part contain all the core concepts you need to design web pages and create a basic ASP.NET website.” (MacDonald, 2012, p.xxxv)*

# The Difference between HTML and ASP.NET Controls

HTML has been around for a very long time, but it was originally intended to display data. A Web Server never processes HTML code. When you want to make your HTML pages more interactive, you can either use JavaScript or add server processing commands. In ASP.NET WebForms, this is accomplished using various "Controls."

“When you create ASP.NET Web pages, you can use these types of controls:

* **HTML server control[s]** HTML elements exposed to the server so you can program them. HTML server controls expose an object model that maps very closely to the HTML elements that they render.
* **Web server controls:** Controls with more built-in features than HTML server controls. Web server controls include not only form controls such as buttons and text boxes, but also special-purpose controls such as a calendar, menus, and a tree view control. Web server controls are more abstract than HTML server controls in that their object model does not necessarily reflect HTML syntax.
* **Validation controls:** Controls that incorporate logic to enable you to what users enter for input controls such as the TextBox control. Validation controls enable you to check for a required field, to test against a specific value or pattern of characters, to verify that a value lies within a range, and so on. For more information, see ASP.NET Validation Controls.
* **User controls:** Controls that you create as ASP.NET Web pages. You can embed ASP.NET user controls in other ASP.NET Web pages, which is an easy way to create toolbars and other reusable elements. For more information, see ASP.NET User Controls.” (MSDN, <http://msdn.microsoft.com/en-us/library/vstudio/zsyt68f1(v=vs.100).aspx>)

## Standard HTML Server Controls

HTML controls have been used for decades. While HTML code cannot process on a Web Server, the contents of the HTML input tags can be sent to the Web Server to be processed by another programming Language like C# or PHP.

#### Demo01HTMLControls.aspx

<%@ Page Language="C#" %>

<!DOCTYPE html>

<html xmlns="http://www.w3.org/1999/xhtml">

<head runat="server">

<title>Demo1</title>

<script type="text/javascript">

// With **Standard HTML controls you write JavaScript** to add functionality

function StdHTMLCtrl() {

document.getElementById("Text1").value += " from Std HTML Control";

alert("Pause before sending to server")

document.forms[0].submit() //Send to Server

}

</script>

<script runat="server">

// **Std HTML controls don’t expose server-side events**, but you can still get Form data

protected void Page\_Load(object sender, EventArgs e)

{

//Note: The Server does not have direct access to the HTML control’s object model

//but, you **use the Request object** to access its data

Response.Write(Request.Form["Text1"]);

}

</script>

</head>

<body>

<%--NOTE: the **runat attribute** is **required** to allow ASP.NET Processing --%>

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

<div>

<p>

This is a standard HTML textbox and button.

You write your own JavaScript to interact with it. <br />

<!-- NOTE: You **must include a NAME attribute** for Processing Std HTML on the Server! -->

<input id="Text1" name="Text1" type="text" />

<input id="Button1" type="button" value="button" onclick="StdHTMLCtrl()" />

</p>

</div>

</form>

</body>

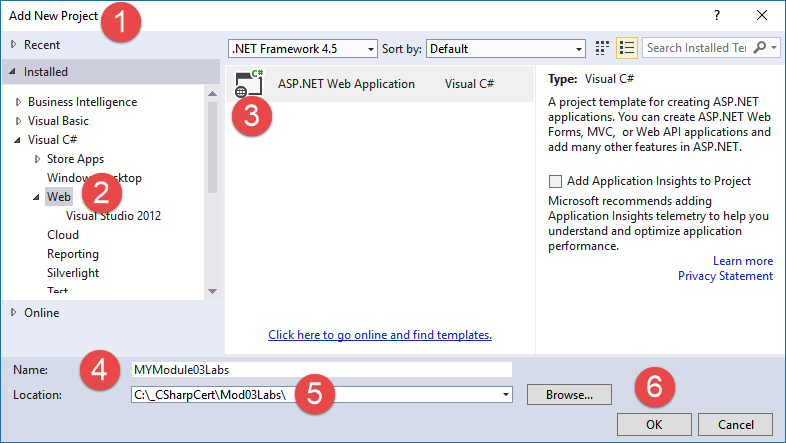
</html>

**Note:** ASP.NET's **MVC applications only use HTML controls**, not HTML Server Controls.

### LAB 01: Creating an ASP.NET using HTML Controls

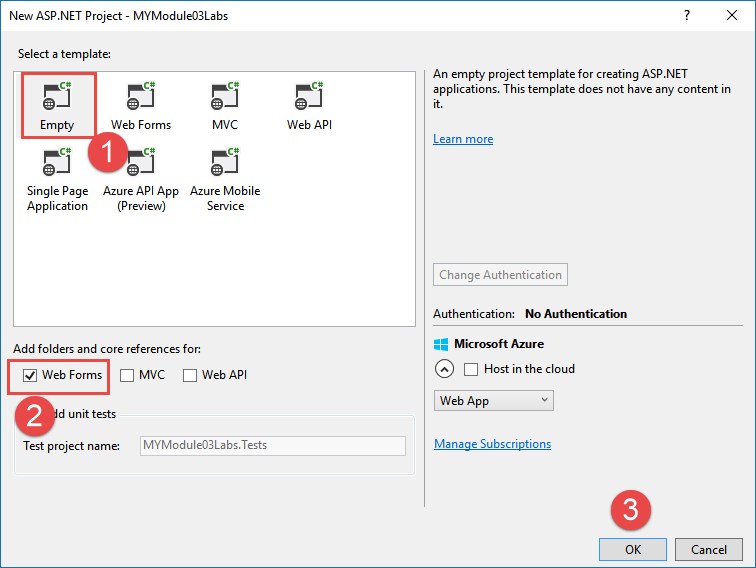
In this lab, you will:

1) Create a new Visual Studio solution with an ASP.NET **Empty** Web Application project called **MY**Module03Labs.



#### Creating the Lab03 project

2) Select an Empty Project template with Web Forms references.

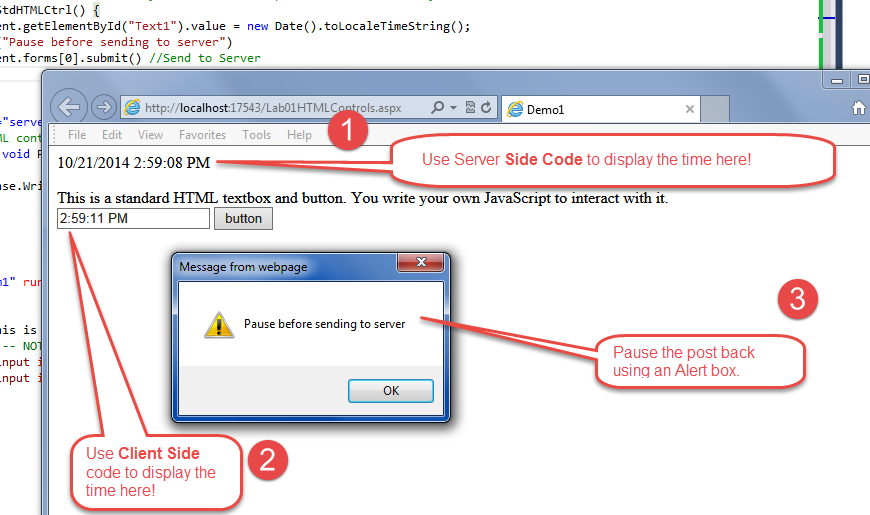


#### ***Setting the P***roject Template

3)Add an ASP.NET web page in it called Lab01.aspx.

4) Add code to the page to return the results shown.

Tip: Start with the code from Demo01HTMLControls.aspx and modify it!



#### ***The Lab 1 Results***

**This lab should take about 5 to 10 minutes**

## The Page.ISPostBack Property

When an **HTML page** is **first requested** of the Web Server, the page is sent **without server-side processing**. However, if an **ASPX page is requested**, the Web Server will try to **process** any Server-side code on the **first and all additional requests**.

When you do not want Server-side code to run when the page is first requested you can use the IsPostBack property to test if the page is being rendered for the first time or is responding to a “post back” from the client.

#### Demo02PostBack.aspx

<%@ Page Language="C#" %>

<!DOCTYPE html>

<html xmlns="http://www.w3.org/1999/xhtml">

<head runat="server">

<title>Demo2</title>

<script type="text/javascript">

// With Standard HTML controls you write JavaScript to add functionality

function StdHTMLCtrl() {

var strData = document.getElementById("Text1").value += " from Std HTML Control";

alert("Pause before sending this (" + strData + " ) to the server")

document.forms[0].submit() //Send to Server

}

</script>

<script runat="server">

protected void Page\_Load(object sender, EventArgs e)

{ //When PostBack == true, it means that it is being SENT from a client

if (IsPostBack == true)

{ Response.Write(Request.Form["Text1"] + "with additional Server Results"); }

else if (IsPostBack == false)

{ Response.Write("The browser requested the page for the first time"); }

}

</script>

</head>

<body>

<%--NOTE: the runat attribute is required to allow ASP.NET Processing --%>

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

<div>

<p>

This is a standard HTML textbox and button.

You write your own JavaScript to interact with it.<br />

<!-- NOTE: You must include a NAME attribute for Processing Std HTML on the Server! -->

<input id="Text1" name="Text1" type="text" />

<input id="Button1" type="button" value="button" onclick="StdHTMLCtrl()" />

</p>

</div>

</form> </form>

</body>

</html>

**Note:** ASP.NET's **MVC applications do not use PostBacks**.

## HTML Server Controls

The web server can **only access the data** of a standard HTML control object and **not its object model**. If you want to access to these object’s properties and methods, on the Web Server, you **can upgrade those controls to *HTML Server Controls***. This upgrade is **mostly** used when a **company** **has an existing web page** that they do not want to completely re-write!

#### Demo03HTMLServerControls.aspx:

<%@ Page Language="C#" %>

<!DOCTYPE html>

<html xmlns="http://www.w3.org/1999/xhtml">

<head runat="server">

<title>Demo3</title>

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

// With Standard HTML controls you write JavaScript to add functionality

function StdHTMLCtrl() {

var strData = document.getElementById("Text1").value += " from Std HTML Control";

alert("Pause before sending this (" + strData + " ) to the server")

document.forms[0].submit() //Send to Server

}

**function HTMLSvrCtrl() {**

**document.getElementById("Text2").value += " Client Side Result";**

**alert("Pause before sending to server")**

**document.forms[0].submit() //Send to Server**

**}**

</script>

**<script runat="server">**

protected void Page\_Load(object sender, EventArgs e)

{ //When PostBack == true, it means that data is being SUBMITED from a client

if (IsPostBack == true)

{ Response.Write(Request.Form["Text1"] + " Server-side Results "); }

else if (IsPostBack == false)

{ Response.Write("The browser requested the page for the first time"); }

}

//HTML Server controls have access to Server-side Event handling!

**protected void Button2\_Click(object sender, EventArgs e)**

**{**

**//Note how simple it is to work with the Web Server Control!**

**Response.Write(Text2.Value + "<br />");**

**Response.Write("<b><i>Note:</i></b>Response.Write works, but is not recommended!");**

**}**

**</script>**

<style type="text/css">

.auto-style1 {

color: #FF0000;

}

</style>

</head>

<body>

<p>

NOTE: You must include an<span class="auto-style1"> <strong>"onsubmit" attribute it the Form tag</strong></span><br />

if you want to include client side scripting with HTML Server controls!<br />

This is due to the Auto generated JavaScript block added automatically by Microsoft.<br />

<span class="auto-style1"><strong>Use View Source to see the Client Side code </strong></span>added and how<br />

it checks to see if the attribute has been added before it allows Client Side event handling!

<br />

</p>

<**form id="form1" runat="server" onsubmit="return HTMLSvrCtrl()">**

<!-- Required for Client Side Event Handling!-->

<div>

<p>

These are a standard HTML textbox and button.<br />

You write your own JavaScript to interact with them.<br />

<input id="Text1" name="Text1" type="text" />

Note how the textbox data is reset each time!<br />

<!-- NOTE: You must include a NAME attribute for Processing Std HTML on the Server ! -->

<input id="Button1" type="button" value="button" onclick="StdHTMLCtrl()" />

</p>

**<p>**

**These are the same standard HTML controls turned into HTML Server Controls<br />**

**<%-- Note the runat and the OnServerClick attributes!--%>**

**<input id="Text2" type="text" runat="server" />**

**Note how the textbox data is kept automatically!<br />**

**<input id="Button2" type="button" value="button" runat="server"**

**onserverclick="Button2\_Click" /><br />**

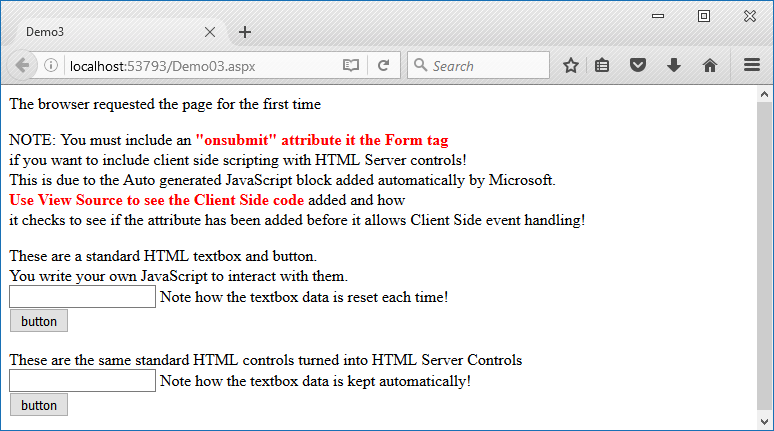
**</p>**

</div>

</form>

</body>

</html>



#### Results of Demo03

## Web Server Controls

**Web Server Controls** are the **typical** way to work with **ASP.NET Web Forms**. Instead of upgrading your HTML controls to HTML Server Controls, you can **start new web pages using Web Server Controls**.

#### Demo04ASPWebServerControls.aspx:

<%@ Page Language="C#" %>

<!DOCTYPE html>

<html xmlns="http://www.w3.org/1999/xhtml">

<head runat="server">

<title>Demo4</title>

<script type="text/javascript">

function ASPWebSvrCtrl() {

alert(document.getElementById("TextBox1").value += " Client Side Results");

//You DON'T Need this NOW since ASP.NET has created its own JavaScript submit code!

**//document.forms[0].submit**()

/\*\*\* NOTE: If you do use this it will overwrite your Svr results \*\*\*/

}

</script>

<script runat="server">

protected void Button1\_Click(object sender, EventArgs e)

{

TextBox1.Text += " with Server-side Results";

}

</script>

</head>

<body>

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

<div>

<**asp**:TextBox ID="TextBox1" runat="server" TextMode="MultiLine" Height="100px" Width="200px"></asp:TextBox>

<**asp**:Button ID="Button1" runat="server" Text="Button"

**OnClick**="Button1\_Click"

**OnClientClick**="ASPWebSvrCtrl()"

**UseSubmitBehavior**="True" />

<%-- The UseSubmitBehavior determines if ASP.NET will add a JavaScript submit method!--%>

</div>

</form>

</body>

</html>

### LAB 02: ASP.NET Web Server Controls

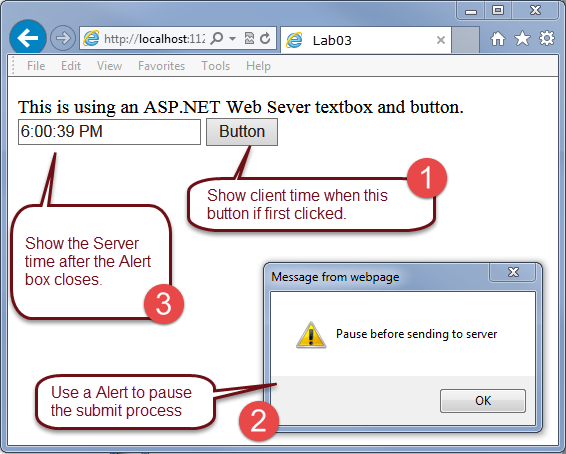
In this lab, you will:

1) Add a new WebForm called Lab02.aspx to your Module03Labs project.

2) Add a Textbox and a Button Web Server Controls to the form.

3) Add code to get the same results as seen in Lab 1 and 2.

1. Use Client code to show the time when the button is first clicked.
2. Have an Alert pause the Client’s submission.
3. Display the time using Server code.



#### Lab 03 Results

**This lab should take about 5 to 10 minutes**

### Validation Controls

Validations controls are a **subset of Web Server Controls that include pre-made validation logic**. These controls streamline web page development and offer many advantages.

#### Demo05ValidationControls.aspx:

<%@ Page Language="C#" %>

<!DOCTYPE html>

<html xmlns="http://www.w3.org/1999/xhtml">

<head runat="server">

<title>Demo5</title>

<%-- Note that no Client or Server code is needed for this to work! --%>

</head>

<body>

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

<div>

<asp:TextBox ID="TextBox1" runat="server"></asp:TextBox>&nbsp;

**<asp:RequiredFieldValidator ID="RequiredFieldValidator1" runat="server"**

**ErrorMessage="(This field is required!)"**

**ControlToValidate="TextBox1"></asp:RequiredFieldValidator>**

<br />

<asp:Button ID="Button1" runat="server" Text="Button" />

</div>

<asp:ValidationSummary ID="ValidationSummary1" runat="server" />

</form>

</body>

</html>

##### **Important:** Since VS 2012, you need to add code to the Web.config file before the validation controls will work. Not doing so causes the error shown here!

#### A Common Error when using Validation Controls

To fix the error, add code to the Web.config file as shown here:

#### Web.config

<configuration>

**<appSettings>**

**<add key="ValidationSettings:UnobtrusiveValidationMode" value="None" />**

**</appSettings>**

<system.web>

<compilation debug="true" targetFramework="4.5" />

<httpRuntime targetFramework="4.5" />

</system.web>

</configuration>

Here are some links to more information:

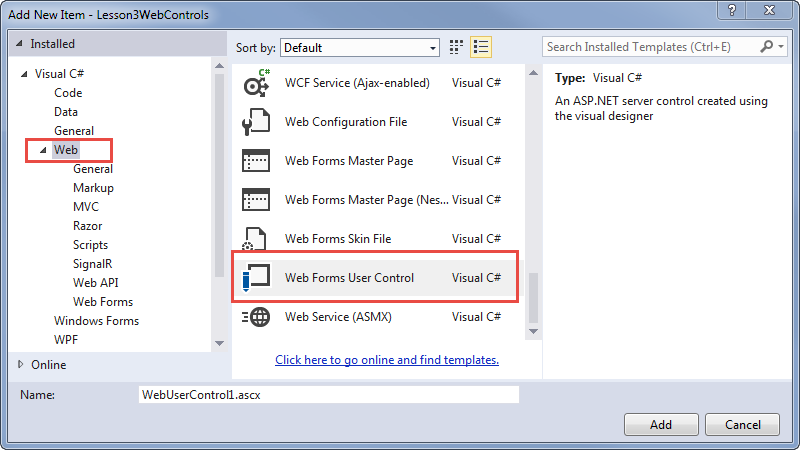
<http://www.codeproject.com/Articles/465613/WebForms-UnobtrusiveValidationMode-requires-a>

<http://en.wikipedia.org/wiki/Unobtrusive_JavaScript>

If you would like more information, check out this video by Chris Pels of Microsoft, on how to use the validation controls in ASP.NET. <http://www.asp.net/web-forms/videos/how-do-i/how-do-i-use-validation-controls-in-aspnet>

## Web User Controls

**Web User Controls allow you to group existing Web Server Controls**. These groups can then be tested as a unit and then added to other web pages as needed.



#### Adding a Web User Control to the Project

You must first create a Web User Control before **you add it to one or more pages**. The code for a Web User Control can include most any content. However, remember that **all content will be placed in the ASP.NET form** of the host page, so plan accordingly!

#### Demo06WebUserControl.ascx

<%@ Control Language="C#" %>

Client Script,<br />

<script type="text/javascript">

alert("demo");

</script>

Server Code,<br />

<script runat="server">

protected void Button1\_Click(object sender, EventArgs e)

{

TextBox1.Text = "Text from the Web User Control";

}

</script>

CSS,<br />

<style>

.redColor { color:red; }

</style>

Standard HTML, and<br />

Web Server Controls are all placed in the BODY of the host page.<br />

<asp:TextBox ID="TextBox1" runat="server" Width="362px"></asp:TextBox>

<br />

<asp:Button ID="Button1" runat="server" Text="Button" class="redColor" OnClick="Button1\_Click" />

&nbsp;View Source to see this placement!

Once you create the Web User Control, you add it to a host page by including a User Control tag. This Tag must be in a Form tag with the runat= “server” attribute.

#### Demo06WebFormPageWithUserControl.aspx

<html xmlns="http://www.w3.org/1999/xhtml">

<head runat="server">

<title></title>

</head>

<body>

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

<div>

**<uc1:Demo06WebUserControl ID="Demo06WebUserControl1" runat="server" />**

</div>

</form>

</body>

</html>

Web user controls are a **handy way to group common components** that will be used in multiple web forms. Chris Pels also posted this video on how to create Web User controls in ASP.NET if you would like to see more.

<http://www.asp.net/web-forms/videos/how-do-i/how-do-i-create-a-custom-user-control-in-aspnet>

**Note:** The Web User Control concept **became the "Partial View" concept in MVC.**

### LAB 03: ASP.NET Web User Controls

In this lab, you create a Web User control that hosts a simple login page that uses ASP.NET Validation Controls:

1) Add a new ASP.NET page called Lab03.

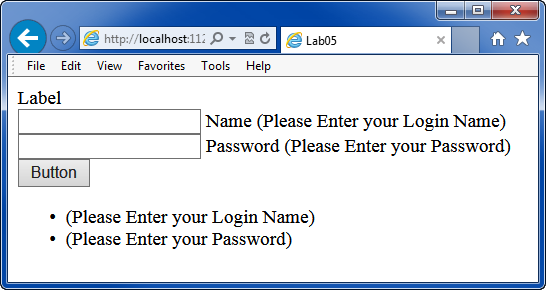
2) Browse the page and verify that it still works.

3) Add a new Web User Control called Lab03WebUserControlLogin.ascx.

4) Add controls and C# code to the Web User control that will display the Name and Password when the button is clicked!

5) Add Validation controls that require the user to enter text in each textbox.

6) Add the Web User Control to the Lab03 page.



#### Lab 03 Results

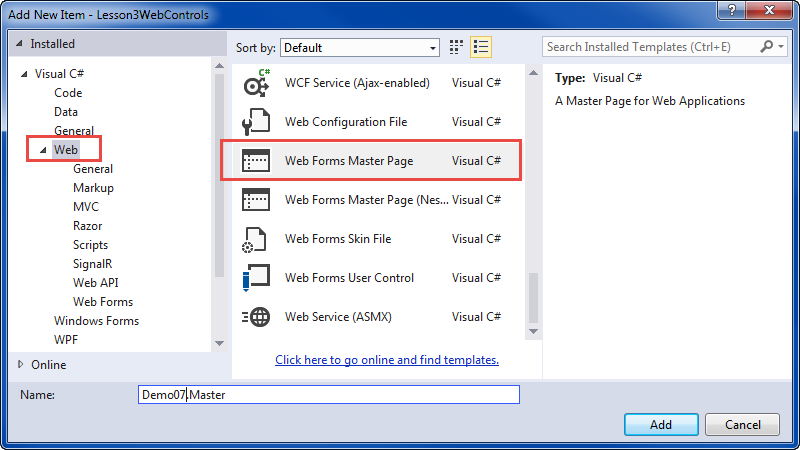
**This lab should take about 5 to 10 minutes**

## Master – Content Pages

Something similar to Web User Controls is the use of **Master and Content pages**. The Master pages host one or more Content pages which **are rendered as a single page** for the Browser. However, this method allows for much more control as to where content is placed.

### Master Pages

You need to **first create a Master** page in your project to use this option. You do so in the same way as adding a Web Form.



#### Adding a Master Page

**Note:** The Master and Content pages **became the "Layout" and "View" pages in MVC.**

You **then add code** to the Master page. The code can include **Client Script, Server Code, CSS, Standard HTML, and Web Server Controls**. However, you cannot browse the Master page directly. Instead must browse it using a Content page.

#### Demo07.Master

<%@ Master Language="C#"%>

<!DOCTYPE html>

<html xmlns="http://www.w3.org/1999/xhtml">

<head runat="server">

<title>Demo7Master</title>

**<%--Client Script --%>**

<script type="text/javascript">

alert("Master Page JavaScript");

</script>

**<%--Server Code--%>**

<script runat="server">

protected void Button1\_Click(object sender, EventArgs e)

{

TextBox1.Text = "Text from the Master's Server Control";

}

</script>

**<%--CSS--%>**

<style>

.redColor {

border: medium dashed #FF0000;

}

</style>

**<asp:ContentPlaceHolder ID="*ContentPlaceHead*" runat="server">**

**</asp:ContentPlaceHolder>**

</head>

<body>

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

Client Script, Server Code, CSS, Standard HTML, and Web Server Controls

<br />

can all placed in both the Master page and Content pages.<br />

<asp:TextBox ID="TextBox1" runat="server" Width="362px"></asp:TextBox>

<br />

<asp:Button ID="Button1" runat="server" Text="Button" OnClick="Button1\_Click" />

&nbsp;View Source to see this placement!

<hr class="redColor" />

<div id="Div1">

**<asp:ContentPlaceHolder ID="*ContentPlaceHolderFormDiv1*" runat="server">**

**</asp:ContentPlaceHolder>**

</div>

<div id="Div2">

**<asp:ContentPlaceHolder ID="*ContentPlaceHolderFormDiv2*" runat="server">**

**</asp:ContentPlaceHolder>**

</div>

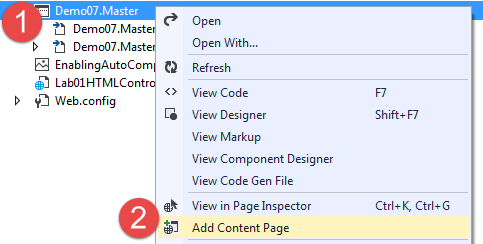
</form>

</body>

</html>

### Content Pages

The easiest way to **add a Content page** is through the context menu as shown here.



#### Adding a Content Page

The newly created **content page references the Master page** in its directive at the top of the page. **Note** the reference to the **Content Place Holders** from the Master page. **Test the Master page by browsing the Content page as-is**!

#### WebForm1.asxp

<%@ Page Title="" Language="C#" **MasterPageFile="~/Demo07.Master"** %>

<asp:Content ID="Content1" **ContentPlaceHolderID="ContentPlaceHead"** runat="server">

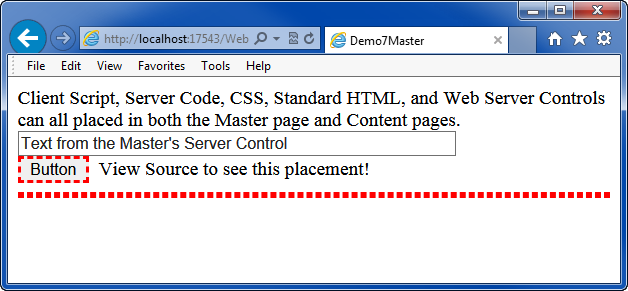
</asp:Content>

<asp:Content ID="Content2" **ContentPlaceHolderID="ContentPlaceHolderFormDiv1"** runat="server">

</asp:Content>

<asp:Content ID="Content3" **ContentPlaceHolderID="ContentPlaceHolderFormDiv2"** runat="server">

</asp:Content>



#### WebForm1.aspx Results

### Adding More Content

After you test the page as-is, you add additional content to the Content Page. Once again it can include almost anything:

#### Demo7Content.aspx

<%@ Page Title="Demo07" Language="C#" MasterPageFile="~/Demo07.Master" %>

<asp:Content ID="Content1" ContentPlaceHolderID="ContentPlaceHead" runat="server">

**<%--Client Script, --%>**

<script type="text/javascript">

alert("Client Page JavaScript");

</script>

**<%--Server Code,--%>**

<script runat="server">

protected void Button2\_Click(object sender, EventArgs e)

{

TextBox2.Text = "Text from the Content Page's Server Control";

}

</script>

**<%--CSS--%>**

<style>

.blueColor {

border: medium dashed #0000FF;

}

</style>

</asp:Content>

<asp:Content ID="Content2" ContentPlaceHolderID="ContentPlaceHolderFormDiv1" runat="server">

Client Script, Server Code, CSS, Standard HTML, and Web Server Controls

<br />

can all placed in both the Master page and Content pages.<br />

<asp:TextBox ID="TextBox2" runat="server" Width="362px"></asp:TextBox>

<asp:Button ID="Button2" runat="server" Text="Button" class="blueColor" OnClick="Button2\_Click" />

<br />

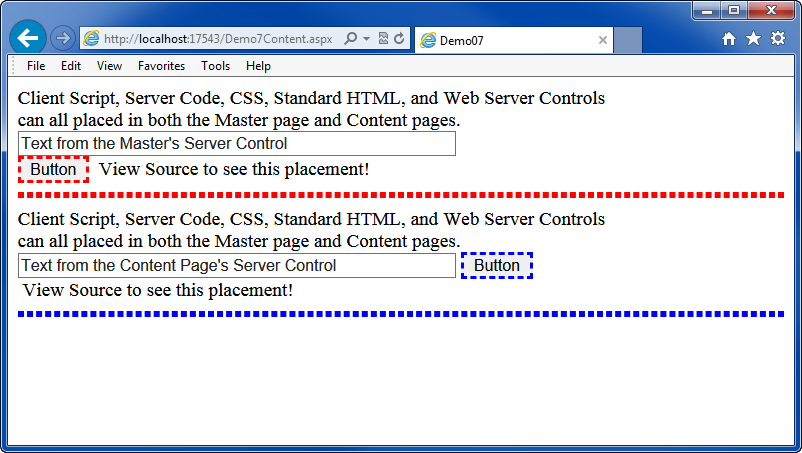
</asp:Content>

<asp:Content ID="Content3" ContentPlaceHolderID="ContentPlaceHolderFormDiv2" runat="server">

&nbsp;View Source to see this placement!

<hr class="blueColor" />

</asp:Content>



#### Demo7Content.aspx Results

### LAB 04: ASP.NET Master and Content Pages

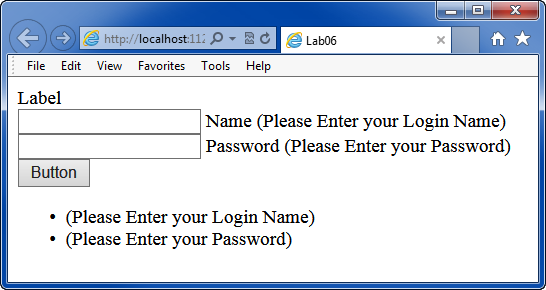
In this lab, you create a Master and a Content page. Add code to the Content page to support a simple login page that uses ASP.NET Validation Controls:

1) Add a new ASP.NET page called Lab04.Master.

2) Create a Content Page for the Master and browse this page to verify that it works.

3) Add controls and C# code to the content that will display the Name and Password when the button is clicked!

5) Add Validation controls that require the user to enter text in each textbox.



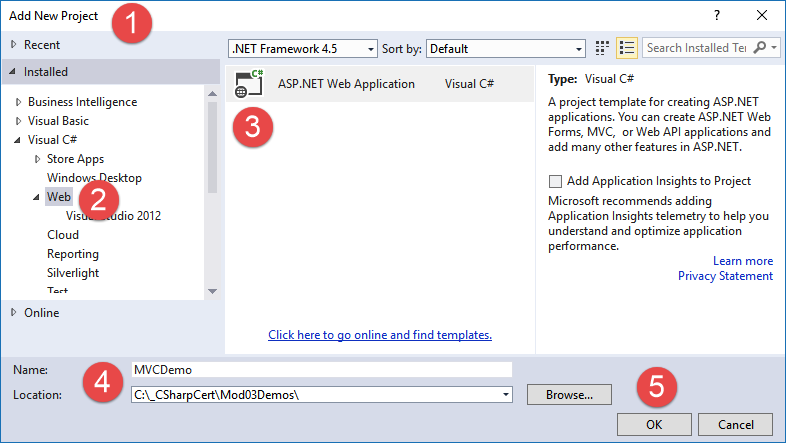
#### Lab 06 Results

**This lab should take about 5 to 10 minutes**

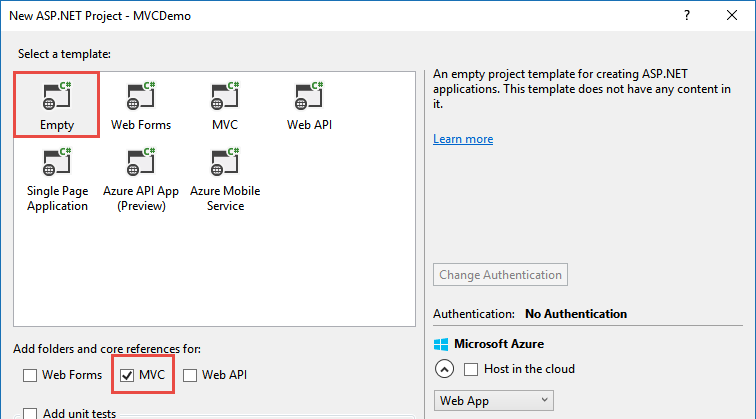
# MVC (Not covered in the book)

Microsoft created a new model for ASP.NET websites using the Model-View-Controller pattern, which has become popular in web development. This **pattern formally separates the Data, UI, and Processing layers of an application**. This separation helps make your web applications more module and consistent between projects.

To **create an MVC web project**, you start as if creating a Web Form project. However, instead of the Web Forms option, you **select the MVC checkbox**!

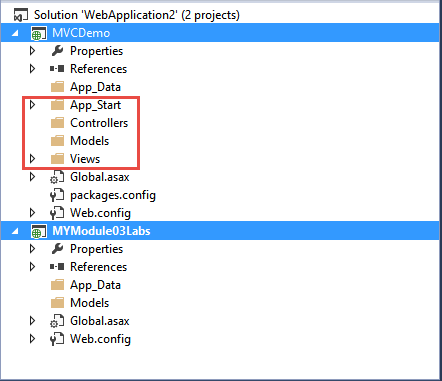


#### Starting an MVC project

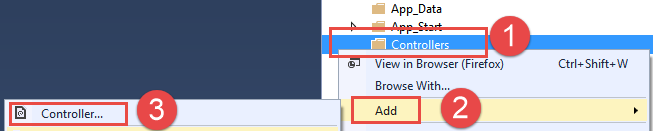


#### Checking the MVC checkbox

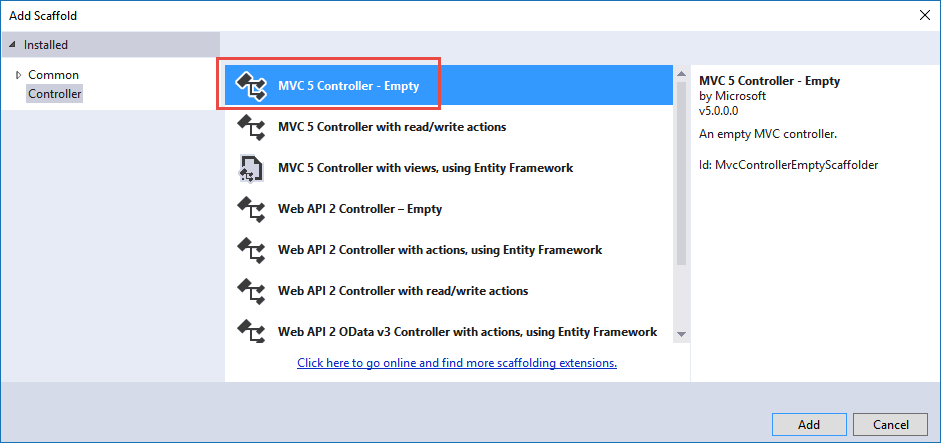
Note the **differences in the structure** of an MCV and a Web Forms project. We now have a **formal separation** between **processing (Controllers), data (Models), and presentation (Views)** code.



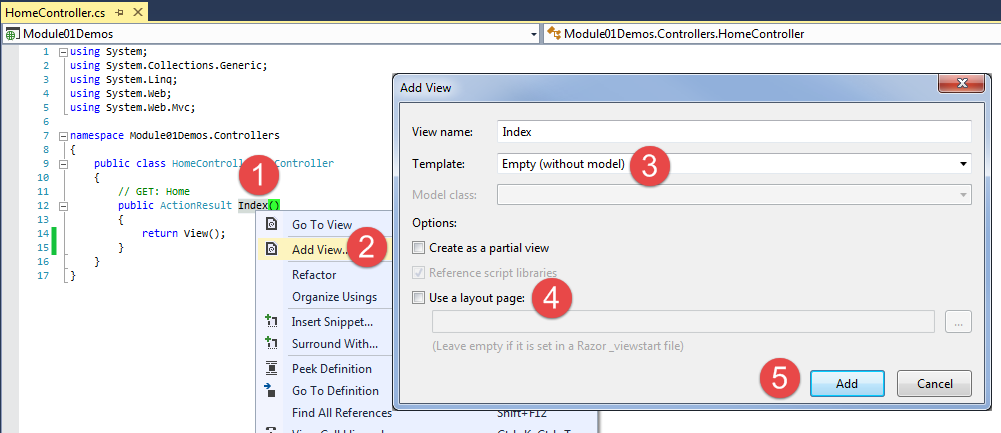
To create an MVC page, you **need at least a processing Controler and presentation View**. It's easiest to start with the Controler and then add a View afterward.



Microsoft includes several pre-configured templates to choose from, but at first, I **recommend you use the Empty Template** option.



Once you have a controller, you can Add View for the controller, by **right-clicking on the Controler's Index method and choosing "Add View"** from the context menu. This selection launches a dialog that allows you to configure and create the view.



Finally, you **add code** to the View, using a mixture of **HTML, CSS, JavaScript, and ASP.NET Razor code**.

#### Code for the Index.cshtml page

@{

Layout = null;

}

<!DOCTYPE html>

<html>

<head>

<meta name="viewport" content="width=device-width" />

<title>Index</title>

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

**// With Standard HTML controls you write JavaScript to add functionality**

**function StdHTMLCtrl() {**

**document.getElementById("Text1ID").value += " from Std HTML Control";**

**alert("Pause before sending to server")**

**document.forms[0].submit() //Send to Server**

**}**

**</script>**

@\*Server Script Blocks **will not work** with MVC!

<script runat="server">

protected void Page\_Load(object sender, EventArgs e)

{

Response.Write(Request.Form["Text1"]);

}

</script> \*@

</head>

<body>

<div>

**@{ //This is razor code (c#)**

**if (IsPost)**

**{**

**string TransferedData = Request["Text1Name"];**

**<p>**

**You entered: @TransferedData <br />**

**</p>**

**}**

**else**

**{**

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

**HTML Textbox: <input type="text" id="Text1ID" name="Text1Name"**

**value="" /><br />**

**<input type="button" value="Submit" onclick="StdHTMLCtrl()" />**

**</form>**

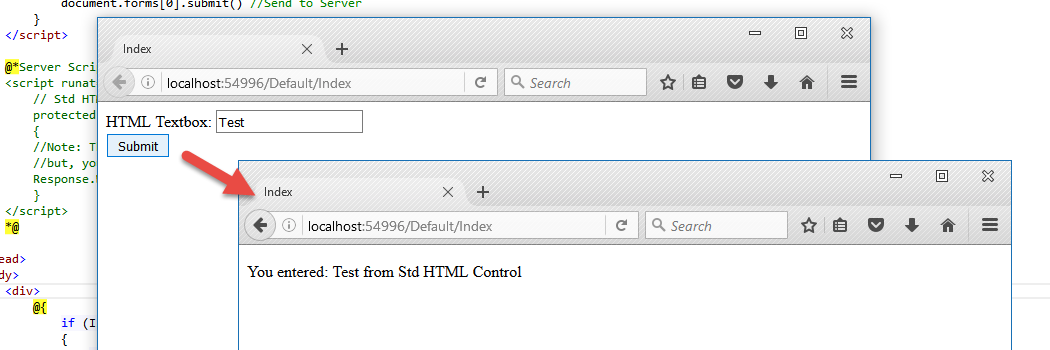
**}**

**}**

</div>

</body>

</html>



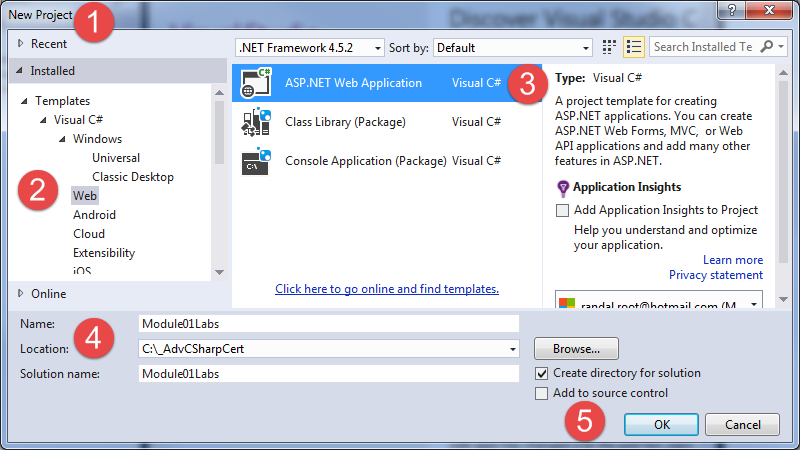
The outcome of a simple MVC page.

***Note:*** *We cover MVC in more detail later in the course.*

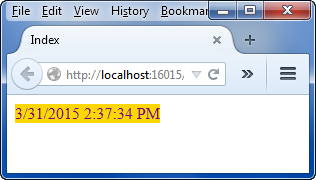
### LAB 05: ASP.NET MVC Applications

In this lab, you create an MVC page in a new project.

1. **Add** a new ASP.NET Web Application to your current Solution called **Module03Lab05**.



1. **Select** the **Empty Template** option and **add the MVC** references. Do not check the host in cloud option.
2. **Create** a new Controller called “***Lab6Controller,****”* and an "Index" View the displays the current time. Your results should look like this:



Tip: Use a Razor script block and C# code to get the current time and display the results, as shown in the last demo.

**This lab should take about 5 to 10 minutes**