

C0550 – Web Applications

UNIT 2 – Visual Studio, Razor Pages and Capturing User Input

CW1 Logbook

- Part of CW1 is a personal logbook of your tutorial work and research
- The first logbook covers this unit

ASP.net Web Apps
Logbook 1 (Unit 2)

Task 1 (2 marks)

Create a simple web forms script which lets a user enter their name into an input field, press a button, and then a message which displays their name should appear. See week 02 slides for the relevant walkthrough. Screenshot your code and resultant functionality working in a browser.

[INSERT SOLUTION AND SCREENSHOTS HERE]

Task 2 (5 marks)

Implement a simple currency convertor script. You may hard-code the exchange rate into the script. See week 02 slides for the relevant notes and suggested interface design.

[INSERT SOLUTION AND SCREENSHOTS HERE]

Question 3 (1 mark): Explain the difference between client-side code and server-side code.

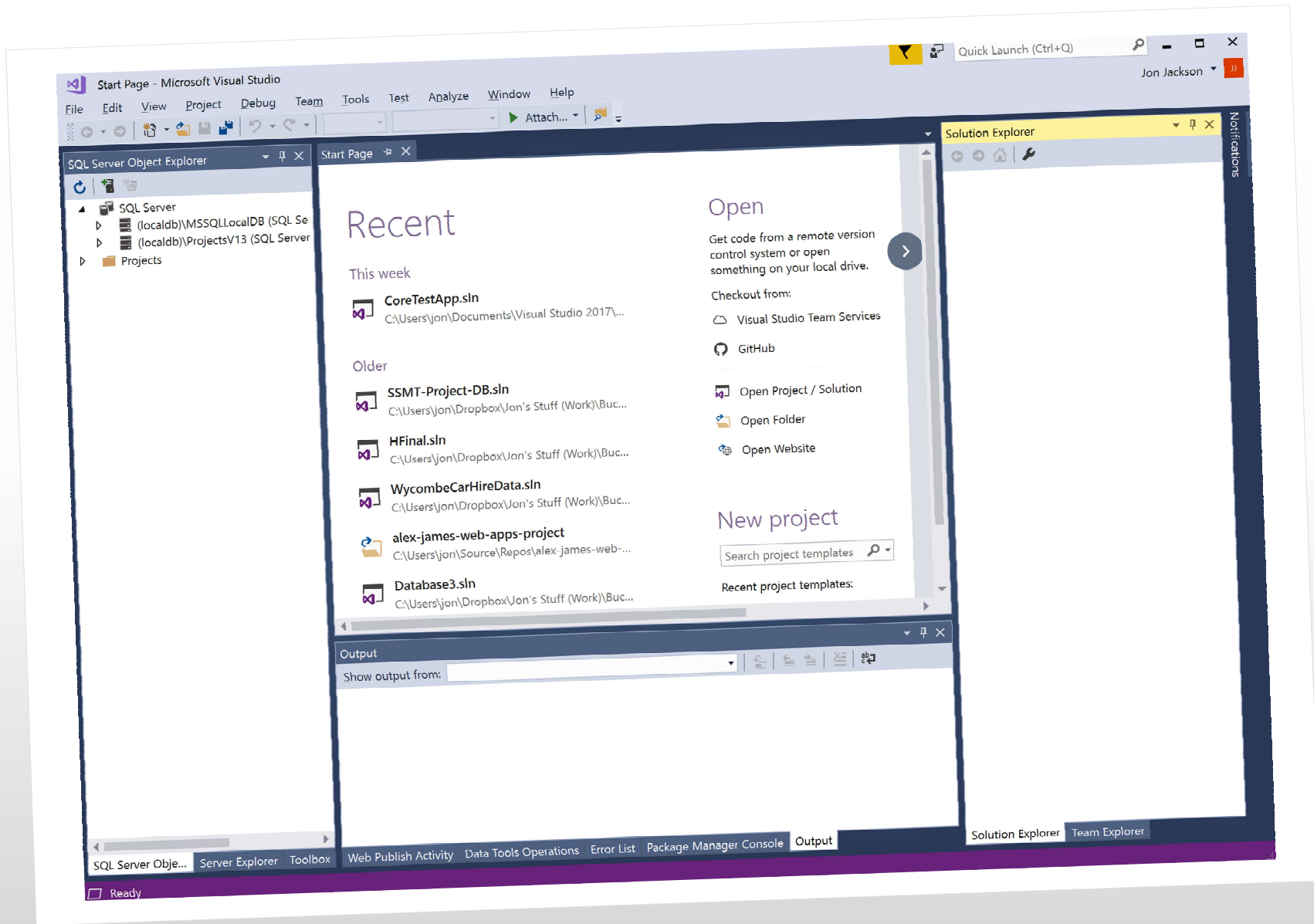
[INSERT ANSWER HERE]

We will be covering...

- Visual Studio basics
- Old-school Web Forms
- Modern Razor Pages

Getting Started: Visual Studio

- Open Visual Studio
- The VS start page allows you to create new or open existing projects
- It shows recently accessed projects.
- As is frequently the case with VS, there are numerous ways to get started



History: Web Forms

How it used to be done...

...a long, long, long time ago...

Old School Web Forms

- Choose File, New, Web Site from menu
- New Web Site dialog box appears
- Ensure C# template chosen from list on left
- Also ensure .NET Framework 4.5 selected from drop-down list at top of page
- Select: ASP.NET Empty Web Site
- Give site a meaningful name
- Save it in an appropriate folder
- Navigate using browse button

Empty Websites v. Web Forms

It is possible to create either 'empty websites' or 'web forms' projects

- *Web Forms* are full-featured websites, with much of the infrastructure already in place.
- *Empty Web Sites* are skeletal with stripped-down web.config files but do not clutter your site with pre-generated files and settings.

Adding web pages to empty sites

- At top of page click on Website and select add new item
- Web form should be selected. If not, select it.
- Give the new form suitable name
- Ensure that code is placed in separate file
- Leave MasterPages deselected (for now)
- Add new form

Add New Item - Empty

Sort by: Default

Search Installed Templates (Ctrl+E)

- Visual Basic
- Visual C#**
- Online

Icon	Item Name	Type
	Web Form	Visual C#
	Content Page (Razor)	Visual C#
	Empty Page (Razor)	Visual C#
	Helper (Razor)	Visual C#
	Layout Page (Razor)	Visual C#
	Web API Controller Class	Visual C#
	Web Page (Razor)	Visual C#
	Master Page	Visual C#
	Web User Control	Visual C#
	ADO.NET Entity Data Model	Visual C#
	AJAX-enabled WCF Service	Visual C#

Type: Visual C#
A form for Web Applications

Name:

Place code in separate file
 Select master page

Add Cancel

Adding controls to the page

- With Web Forms, Visual Studio allows the developer to see a version of what the end-user will see
- Default view: **scripting** window
- Click on **Design** tab at bottom of screen to see the design view
- **Split** button - allows you to view script and design simultaneously

EDIT VIEW WEBSITE BUILD DEBUG TEAM SQL FORMAT TOOLS TEST ANALYZE WINDOW HELP

Firefox Debug DOCTYPE: XHTML5

TestPage2.aspx* TestPage.aspx* Default.aspx

Toolbox

- Standard
- Pointer
- AdRotator
- BulletedList
- Button
- Calendar
- CheckBox
- CheckBoxList
- DropDownList
- FileUpload
- HiddenField
- HyperLink
- Image
- ImageButton
- ImageMap
- Label
- LinkButton
- ListBox
- Literal
- Localize
- MultiView
- Panel
- Placeholder
- RadioButton
- RadioButtonList
- Substitution
- Table
- TextBox
- View
- Wizard

```
<%@ Page Language="C#" AutoEventWireup="true" CodeFile="TestPage2.aspx.cs" Inherits="TestPage2" %>

<!DOCTYPE html>

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

      <asp:Label ID="lblLabel" runat="server" Text="Hello"></asp:Label>
      <br />
      <asp:TextBox ID="txtTextBox" runat="server"></asp:TextBox>
      <asp:Button ID="btnButton" runat="server" Text="Submit" />

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

100 %

div

Hello

Submit

The Visual Studio toolbox

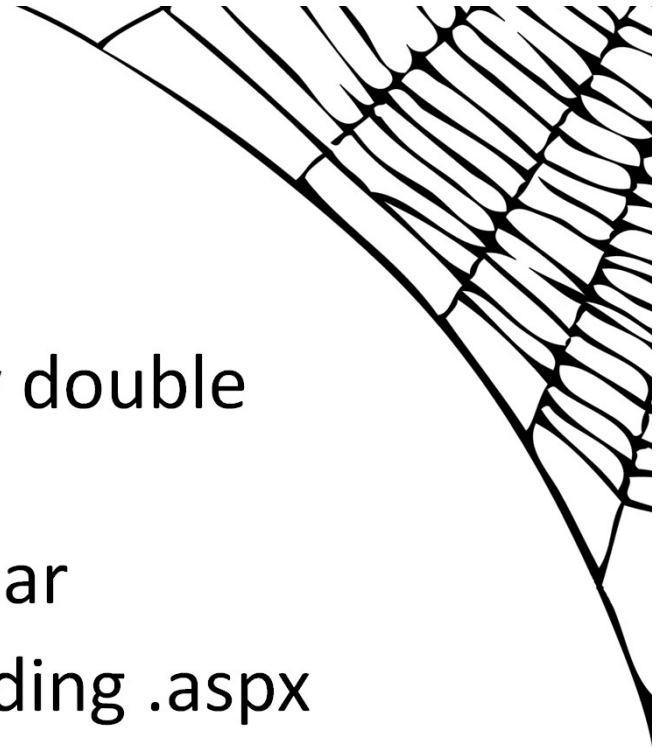
- You can drag controls from the toolbox to build out a form. For example:
 - Place label near top of window
 - Below drag over textBox
 - Place button next to it on same line
- Important to label controls uniquely, correctly and consistently

Sample ASP.NET

```
1 <%@ Page Language = "C#" AutoEventWireup = "true"
2 CodeFile = "Default.aspx.cs" Inherits = "_Default" %>
3 <!DOCTYPE html!>
4 <html>
5 <head runat = "server">
6 <title > Simple Page</title>
7 </head>
8 <body>
9 <form ID = "form1" runat = "server">
10 <div>
11 <asp:Label ID = "Label1" runat = "server" Text = "Type
something:" />
13 <br />
14 <asp:TextBox ID = "TextBox1" runat = "server" />
15 <asp:Button ID = "Button1" runat = "server" Text = "Button"
/>
16 </div>
17 </form> </body> </html>
```

Writing behind-code

- You can find the behind-code by double clicking on the design window
- A behind-code window will appear
- It can also be accessed by expanding .aspx file you're currently working in
- Note extension .aspx.cs to indicate that you are producing C#
- ASP.NET has auto-generated a class Page_Load event handler



dio

BUILD DEBUG TEAM SQL TOOLS TEST ANALYZE WINDOW HELP

Firefox Debug

Default2.aspx.cs Default2.aspx TestPage.aspx Default.aspx Web.config

Default2 Page_Load(object sender, EventArgs e)

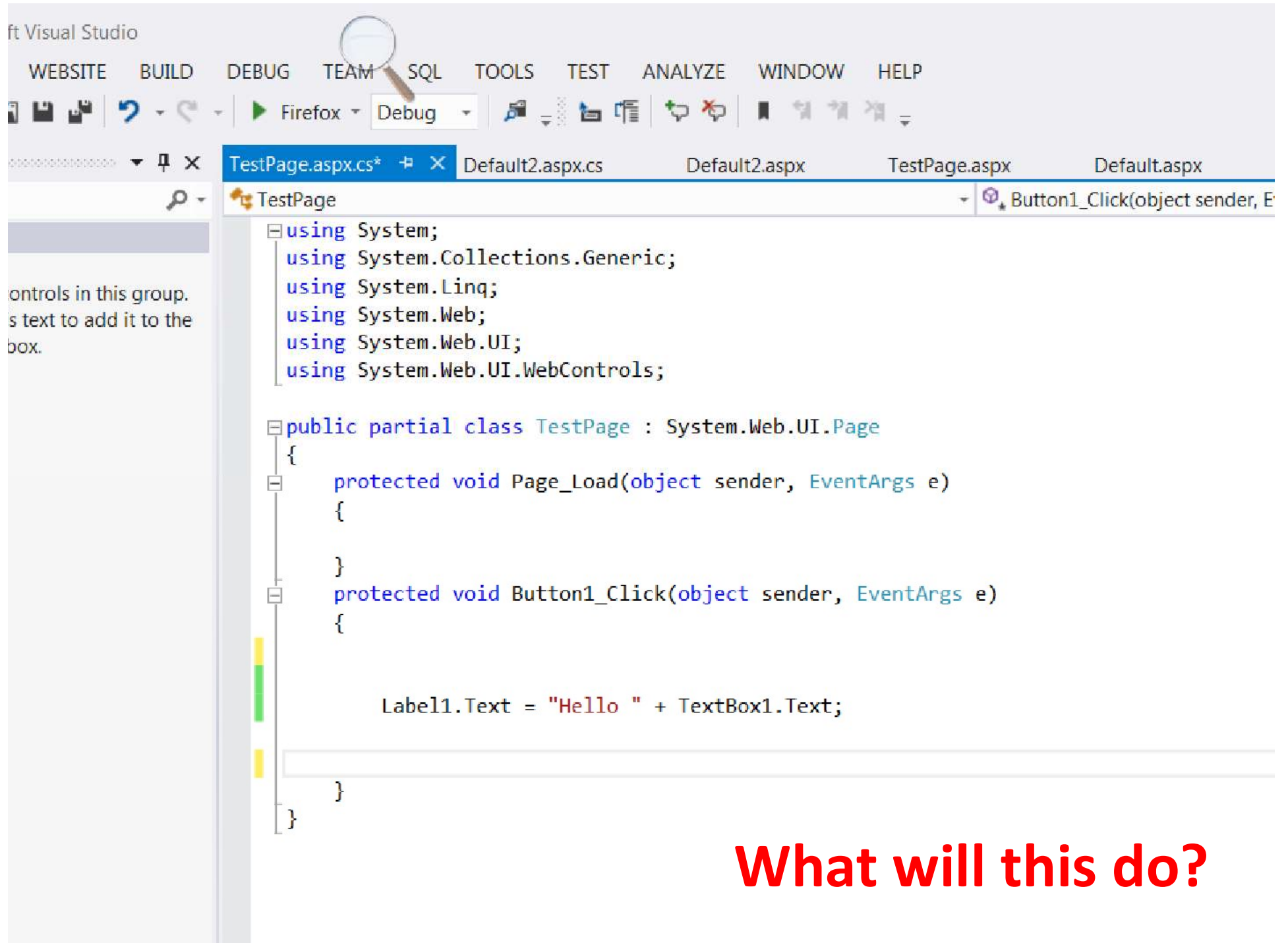
```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;

public partial class Default2 : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
    {
    }
}
```

is group.
it to the

Adding code to Event handlers

- ASP.NET Web Forms is event-driven so code is placed in event handlers
- The event we're interested in is button click
- Return to design window and double-click button
- You will see relevant event handler
- Add some code ...



Now for the More modern
stuff...

Razor Pages

- A Razor Page is very similar to the “view” component of ASP.NET MVC
- The Razor Pages framework provides the developer with full control over rendered HTML.
- The framework is built on top of ASP.NET Core MVC, and is enabled by default when MVC is enabled in a .NET Core application.
- You do not need to have any knowledge or understanding of MVC to work with Razor Pages.

References:

- <https://stackify.com/asp-net-razor-pages-vs-mvc/>
- <https://docs.microsoft.com/en-us/aspnet/core/razor-pages/>
- <https://www.learnrazorpages.com/>

Razor Pages

- Here is a basic example of a Razor Page using inline code within a @functions block.
- It is recommended to put the PageModel code in a separate file (a bit like code behind files with ASP.NET Web Forms)

```
@page
@model IndexModel
@using Microsoft.AspNetCore.Mvc.RazorPages

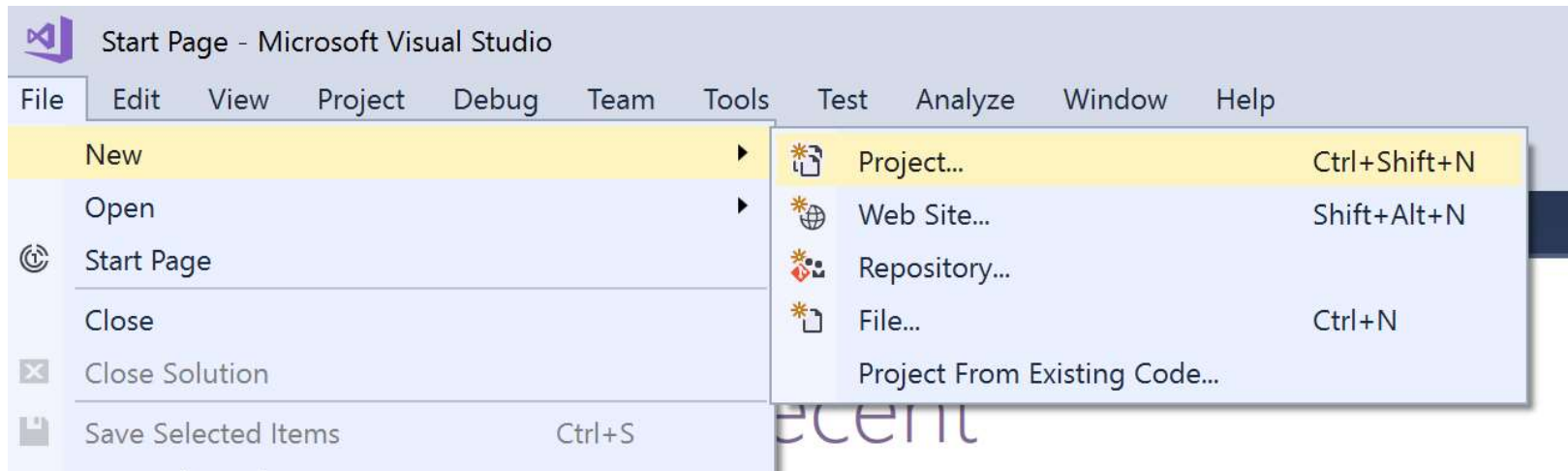
@functions {
    public class IndexModel : PageModel
    {
        public string Message { get; private set; } = "In page model: ";

        public void OnGet()
        {
            Message += $" Server seconds { DateTime.Now.Second.ToString() }";
        }
    }
}

<h2>In page sample</h2>
<p>
    @Model.Message
</p>
```

Razor Pages - Walkthrough

Razor Pages Walkthrough



New Project ? X

Recent .NET Framework 4.6.1 Sort by: Default Search (Ctrl+E)

- Installed
 - Visual C#
 - Windows Classic Desktop
 - Web
 - .NET Core**
 - .NET Standard
 - Cloud
 - Test
 - WCF
 - Visual Basic
 - SQL Server
 - Azure Data Lake
 - Stream Analytics
 - Other Project Types
- Online

Not finding what you are looking for?
[Open Visual Studio Installer](#)

Icon	Project Name	Type
	Console App (.NET Core)	Visual C#
	Class Library (.NET Core)	Visual C#
	Unit Test Project (.NET Core)	Visual C#
	xUnit Test Project (.NET Core)	Visual C#
	ASP.NET Core Web Application	Visual C#

Type: Visual C#
 Project templates for creating ASP.NET Core applications for Windows, Linux and macOS using .NET Core or .NET Framework.

Name:

Location:

Solution:

Solution name:

Create directory for solution
 Create new Git repository

.NET Core ASP.NET Core 2.0 [Learn more](#)

Empty Web API Web Application Web Application (Model-View-Controller) Angular

React.js React.js and Redux

A project template for creating an ASP.NET Core application with example ASP.NET Core Razor Pages content.

[Learn more](#)

Change Authentication

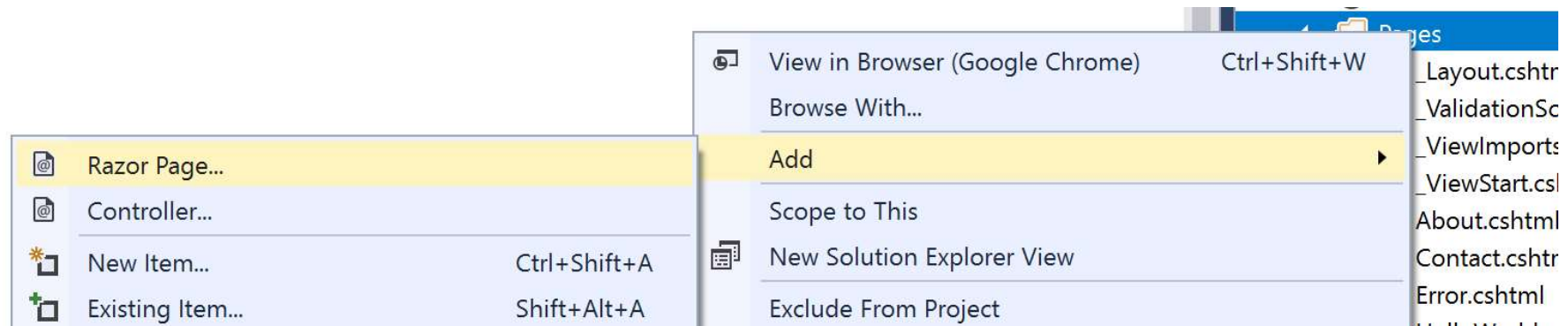
Authentication **No Authentication**

Enable Docker Support
OS: Windows

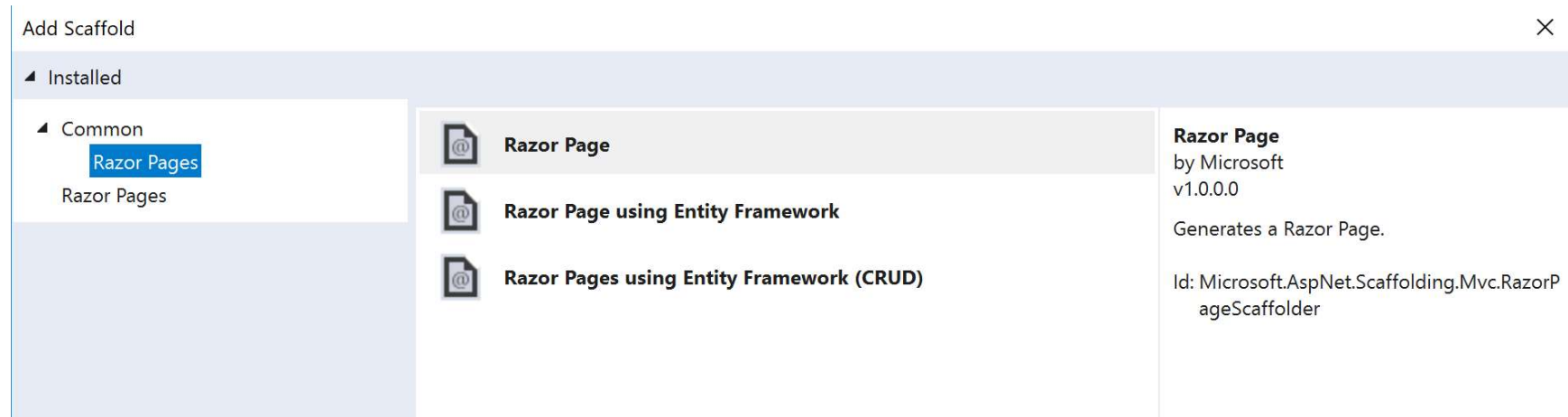
Requires [Docker for Windows](#)
Docker support can also be enabled later [Learn more](#)

OK Cancel

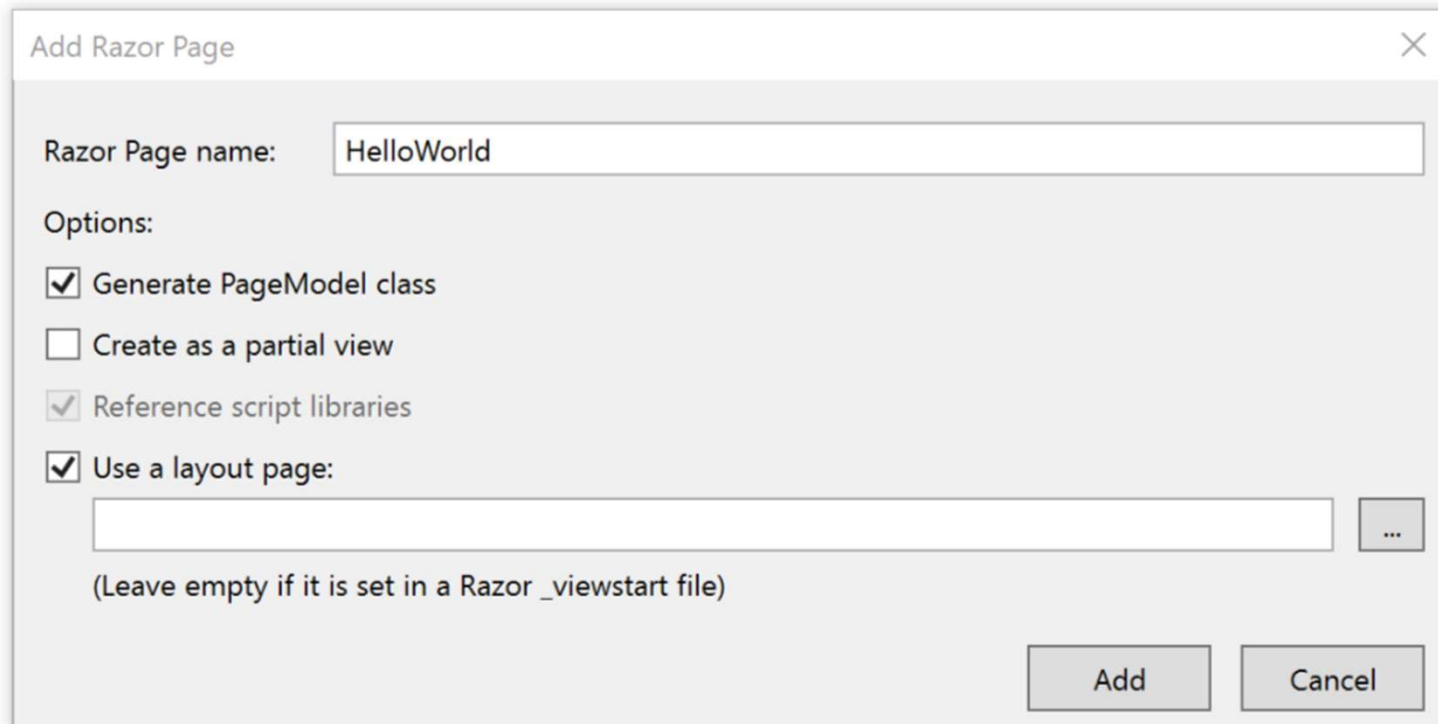
Add a new Razor Page to the project - 1



Add a new Razor Page to the project - 2



Add a new Razor Page to the project - 3



Add Razor Page

Razor Page name: HelloWorld

Options:

- Generate PageModel class
- Create as a partial view
- Reference script libraries
- Use a layout page:

...

(Leave empty if it is set in a Razor _viewstart file)

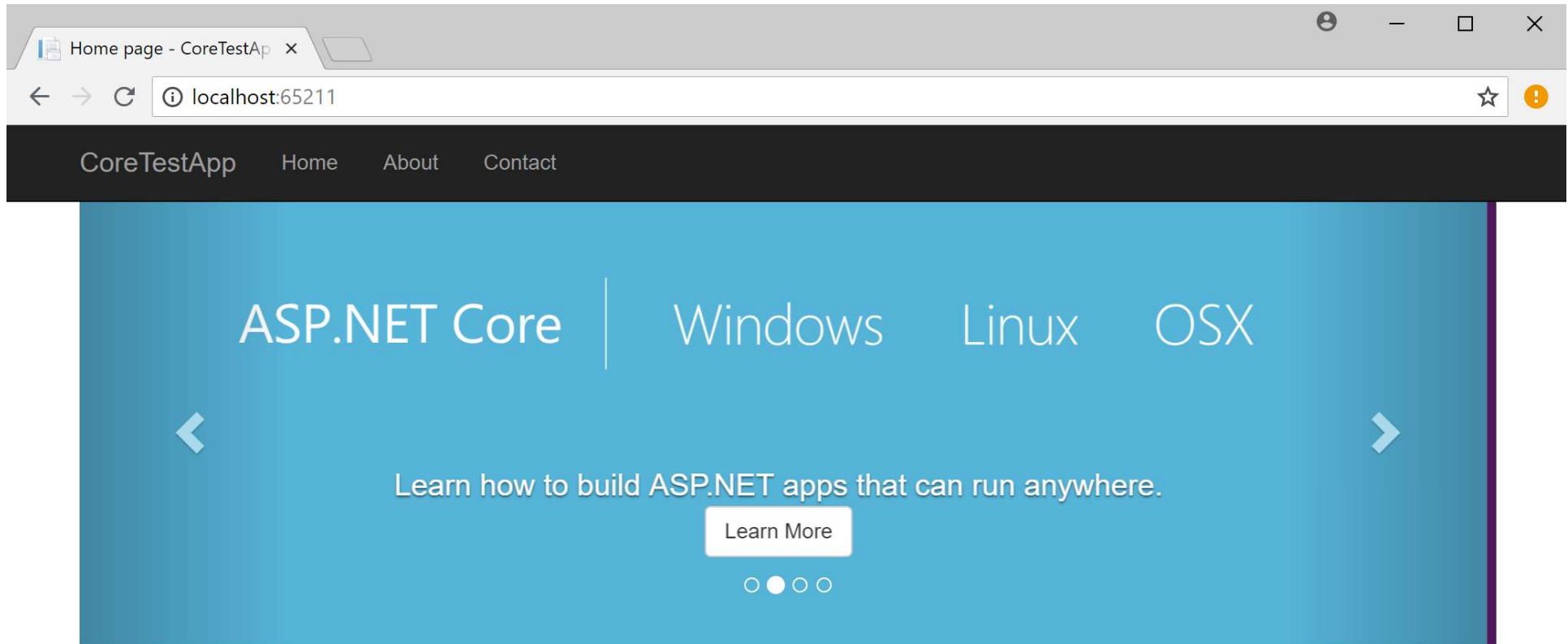
Add Cancel

What's created...

The screenshot displays the Visual Studio IDE with the following components:

- Code Editor:** Shows the content of `HelloWorld.cshtml`. The code includes a page directive, a model reference, a view data assignment, a paragraph tag, and a section for rendering a partial view.
- Solution Explorer:** Shows the project structure for 'CoreTestApp'. The 'Pages' folder is expanded, showing the following files:
 - `_Layout.cshtml`
 - `_ValidationScriptsPartial.cshtml`
 - `_ViewImports.cshtml`
 - `_ViewStart.cshtml`
 - `About.cshtml`
 - `Contact.cshtml`
 - `Error.cshtml`
 - `HelloWorld.cshtml` (selected)
 - `HelloWorld.cshtml.cs`
 - `HelloWorld_Page`
 - `Index.cshtml`

Run it...



Application uses

- Sample pages using ASP.NET Core Razor Pages
- [Bower](#) for managing client-side libraries
- Theming using [Bootstrap](#)

How to

- [Working with Razor Pages.](#)
- [Manage User Secrets using Secret Manager.](#)
- [Use logging to log a message.](#)
- [Add packages using NuGet.](#)
- [Add client packages using](#)

Overview

- [Conceptual overview of what is ASP.NET Core](#)
- [Fundamentals of ASP.NET Core such as Startup and middleware.](#)
- [Working with Data](#)
- [Security](#)
- [Client side development](#)

Run & Deploy

- [Run your app](#)
- [Run tools such as EF migrations and more](#)
- [Publish to Microsoft Azure App Service](#)

Folders Explained

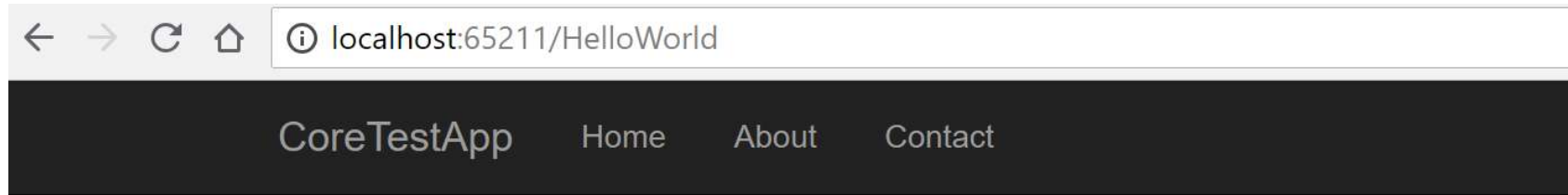
File or folder	Purpose
<i>wwwroot</i>	Contains static assets. See Static files .
<i>Pages</i>	Folder for Razor Pages .
<i>appsettings.json</i>	Configuration
<i>Program.cs</i>	Configures the host of the ASP.NET Core app.
<i>Startup.cs</i>	Configures services and the request pipeline. See Startup .

See <https://docs.microsoft.com/en-us/aspnet/core/tutorials/razor-pages/razor-pages-start?view=aspnetcore-2.1>

Make a change...

```
HelloWorld.cshtml  X HelloWorld.cshtml.cs
1  @page
2  @model CoreTestApp.Pages.HelloWorldModel
3  @{
4      ViewData["Title"] = "HelloWorld";
5  }
6
7  <h2>HelloWorld</h2>
8
9  <h3>The time on the server is @DateTime.Now</h3>
10
11
12  @section Scripts {
13      @{await Html.RenderPartialAsync("_ValidationScriptsPartial");}
14  }
15
```


Run it...



HelloWorld

The time on the server is 06/09/2018 13:31:41

© 2017 - CoreTestApp

URL Routing

The associations of URL paths to pages are determined by the page's location in the file system. The following table shows a Razor Page path and the matching URL:

File name and path	matching URL
<i>/Pages/Index.cshtml</i>	<input type="text" value="/"/> or <input type="text" value="/Index"/>
<i>/Pages/Contact.cshtml</i>	<input type="text" value="/Contact"/>
<i>/Pages/Store/Contact.cshtml</i>	<input type="text" value="/Store/Contact"/>
<i>/Pages/Store/Index.cshtml</i>	<input type="text" value="/Store"/> or <input type="text" value="/Store/Index"/>

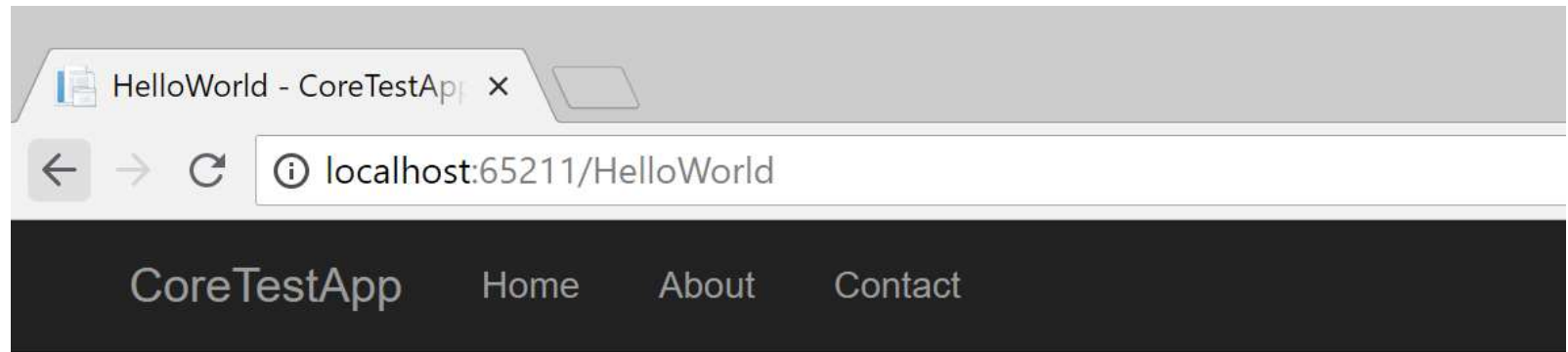
Hello World

```
HelloWorld.cshtml  X HelloWorld.cshtml.cs
1  @page
2  @model CoreTestApp.Pages.HelloWorldModel
3  @{
4      ViewData["Title"] = "HelloWorld";
5  }
6
7  <h2>HelloWorld</h2>
8
9  <h3>The time on the server is @DateTime.Now</h3>
10
11  <p>
12      Enter your name.
13  </p>
14  <div asp-validation-summary="All"></div>
15  <form method="POST">
16      <div>Name: <input asp-for="name" /></div>
17      <input type="submit" />
18  </form>
19
20  <div id="result">@this.Model.name</div>
21
22
23  @section Scripts {
24      @{await Html.RenderPartialAsync("_ValidationScriptsPartial");}
25  }
```

Hello World

```
HelloWorld.cshtml | HelloWorld.cshtml.cs | CoreTestApp | CoreTestApp.Pages.HelloWorldModel
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Threading.Tasks;
5  using Microsoft.AspNetCore.Mvc;
6  using Microsoft.AspNetCore.Mvc.RazorPages;
7  using Microsoft.Extensions.Primitives;
8
9  namespace CoreTestApp.Pages
10 {
11     public class HelloWorldModel : PageModel
12     {
13         public StringValues name;
14
15         public void OnGet()
16         {
17         }
18
19         public void OnPost()
20         {
21             this.name = Request.Form["name"];
22         }
23     }
24 }
25
```

Run it...



HelloWorld

The time on the server is 06/09/2018 17:31:38

Type name
and hit
submit



Enter your name.

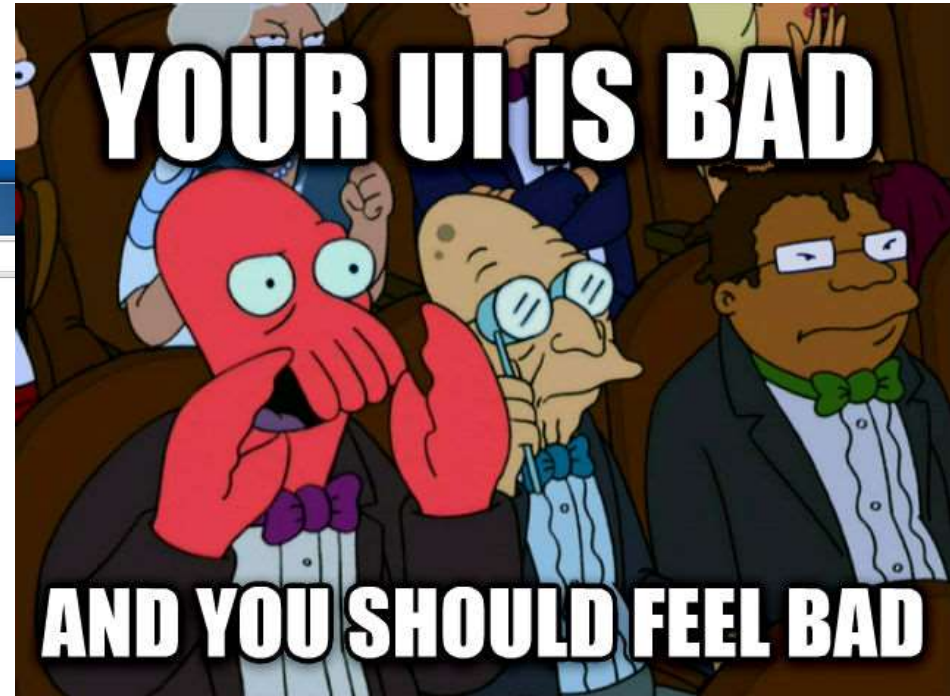
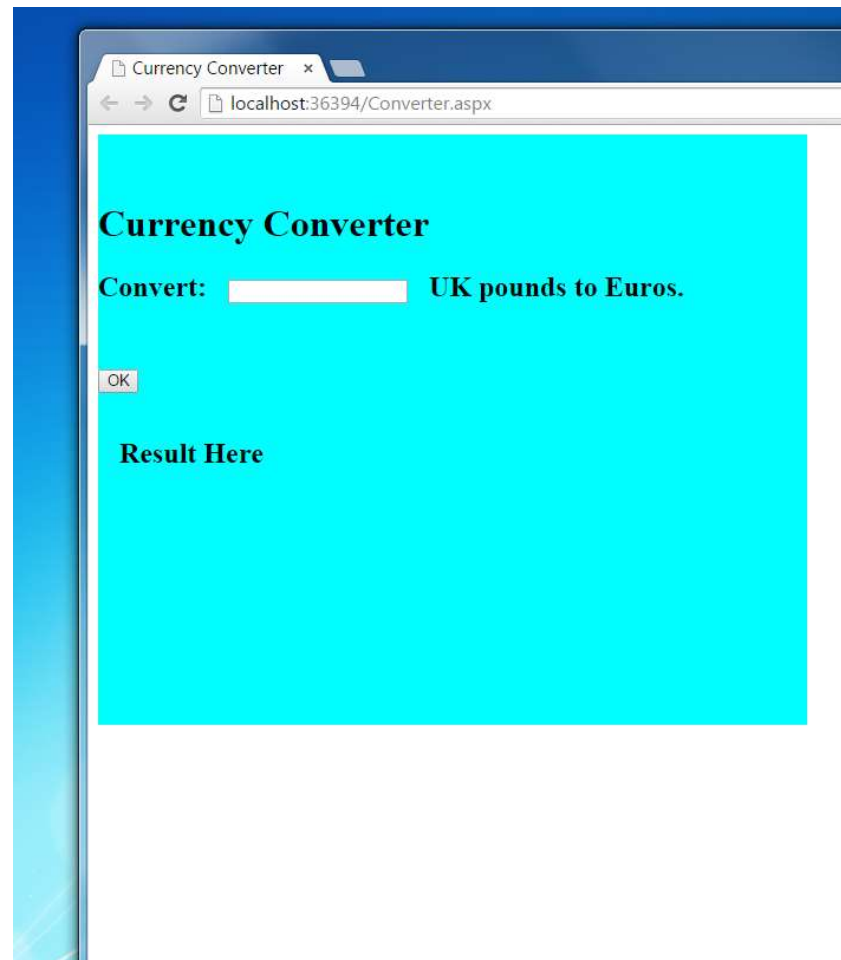
Name:

Jon

Exercise

- Build a simple currency converter
- The user inputs the number of pounds (sterling) they have
- The webpage outputs the equivalent in euros
- Hard-code the exchange rate – don't worry about looking up live exchange rates
- Interface should (not) look something like the next slide

Currency converter



Design

- When building web-based systems vitally important to develop an intuitive user interface.
- Why??

SUMMARY

- Familiarised ourselves with Visual Studio
- Created a simple website project
- Captured user input and processed it in the “code behind” file of a web page by using an event handler
- Explored the various ways to build a simple currency converter