### Exercise 4: Creating a View

So far you have been returning strings from controller actions. Although that is a useful way of understanding how controllers work, it is not how your real Web applications are built. Views are components that provide a better approach for generating HTML back to the browser with the use of template files.

In this exercise you will learn how to add a layout master page to setup a template for common HTML content, a StyleSheet to enhance the look and feel of the site and finally a View template to enable HomeController to return HTML.

#### **Task 1 - Modifying the file \_layout.cshtml**

The file **~/Views/Shared/\_layout.cshtml** allows you to setup a template for common HTML to use across the entire website. In this task you will add a layout master page with a common header with links to the Home page and Store area.

1. If not already open, start **VS Express for Web**.
2. In the **File** menu, choose **Open Project**. In the Open Project dialog, browse to **Source\Ex04-CreatingAView\Begin**, select **Begin.sln** and click **Open**. Alternatively, you may continue with the solution that you obtained after completing the previous exercise.
   1. If you opened the provided **Begin** solution, you will need to download some missing NuGet packages before continue. To do this, click the **Project** menu and select **Manage NuGet Packages**.
   2. In the **Manage NuGet Packages** dialog, click **Restore** in order to download missing packages.
   3. Finally, build the solution by clicking **Build** | **Build Solution**.

**Note:** One of the advantages of using NuGet is that you don't have to ship all the libraries in your project, reducing the project size. With NuGet Power Tools, by specifying the package versions in the Packages.config file, you will be able to download all the required libraries the first time you run the project. This is why you will have to run these steps after you open an existing solution from this lab.

1. The file **\_layout.cshtml** contains the HTML container layout for all pages on the site. It includes the **<html>** element for the HTML response, as well as the **<head>** and **<body>** elements. **@RenderBody()** within the HTML body identify regions that view templates will be able to fill in with dynamic content.

HTML(C#)

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8" />

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

<title>@ViewBag.Title</title>

@Styles.Render("~/Content/css")

@Scripts.Render("~/bundles/modernizr")

</head>

<body>

@RenderBody()

@Scripts.Render("~/bundles/jquery")

@RenderSection("scripts", required: false)

</body>

</html>

1. Add a common header with links to the Home page and Store area on all pages in the site. In order to do that, add the following code below <body> statement.

HTML(C#)

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8" />

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

<title>@ViewBag.Title</title>

@Styles.Render("~/Content/css")

@Scripts.Render("~/bundles/modernizr")

</head>

<body>

**<header>**

**<div class="content-wrapper">**

**<div class="float-left">**

**<p class="site-title">@Html.ActionLink("ASP.NET MVC Music Store", "Index", "Home")</p>**

**</div>**

**<div class="float-right">**

**<nav>**

**<ul id="menu">**

**<li>@Html.ActionLink("Home", "Index", "Home")</li>**

**<li>@Html.ActionLink("Store", "Index", "Store")</li>**

**</ul>**

**</nav>**

**</div>**

**</div>**

**</header>**

@RenderBody()

@Scripts.Render("~/bundles/jquery")

@RenderSection("scripts", required: false)

</body>

</html>

1. Include a div to render the body section of each page. Replace **@RenderBody()** with the following higlighted code:

HTML(C#)

...

<body>

<header>

<div class="content-wrapper">

<div class="float-left">

<p class="site-title">@Html.ActionLink("ASP.NET MVC Music Store", "Index", "Home")</p>

</div>

<div class="float-right">

<nav>

<ul id="menu">

<li>@Html.ActionLink("Home", "Index", "Home")</li>

<li>@Html.ActionLink("Store", "Index", "Store")</li>

</ul>

</nav>

</div>

</div>

</header>

**<div id="body">**

**@RenderSection("featured", required: false)**

**<section class="content-wrapper main-content clear-fix">**

**@RenderBody()**

**</section>**

**</div>**

...

</body>

</html>

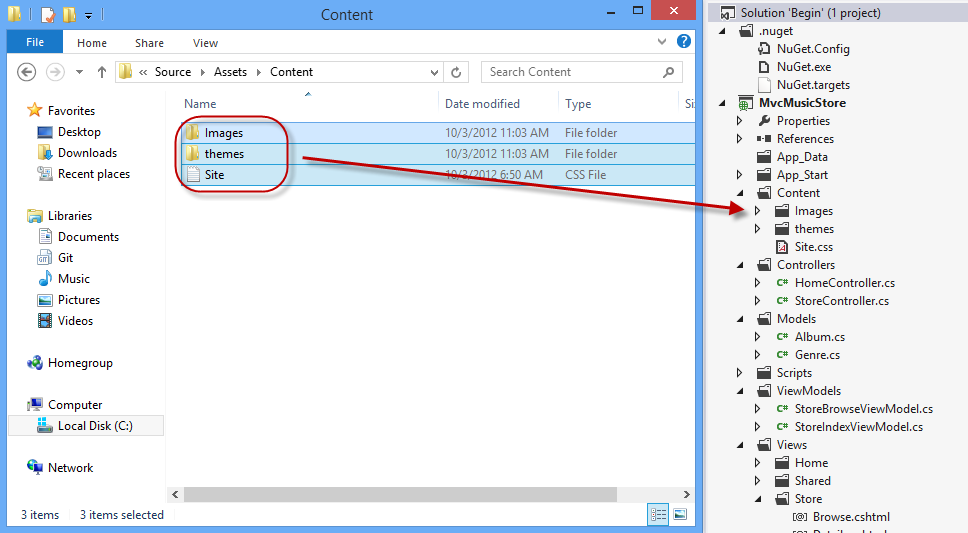
**Note:** Did you know? Visual Studio 2012 has snippets that make it easy to add commonly used code in HTML, code files and more! Try it out by typing **<div>** and pressing **TAB** twice to insert a complete **div** tag.

#### **Task 2 - Adding CSS Stylesheet**

The empty project template includes a very streamlined CSS file which just includes styles used to display basic forms and validation messages. You will use additional CSS and images (potentially provided by a designer) in order to enhance the look and feel of the site.

In this task, you will add a CSS stylesheet to define the styles of the site.

1. The CSS file and images are included in the **Source\Assets\Content** folder of this Lab. In order to add them to the application, drag their content from a **Windows Explorer** window into the **Solution Explorer** in Visual Studio Express for Web, as shown below:



Dragging style contents

1. A warning dialog will appear, asking for confirmation to replace **Site.css** file and some existing images. Check **Apply to all items** and click**Yes**.

#### **Task 3 - Adding a View Template**

In this task, you will add a View template to generate the HTML response that will use the layout master page and CSS added in this exercise.

1. To use a View template when browsing the site's home page, you will first need to indicate that instead of returning a string, the**HomeController Index** method will return a **View**. Open **HomeController** class and change its **Index** method to return an **ActionResult**, and have it return **View()**.

(Code Snippet - ASP.NET MVC 4 Fundamentals - Ex4 HomeController Index)

C#

public class HomeController : Controller

{

//

// GET: /Home/

**public ActionResult Index()**

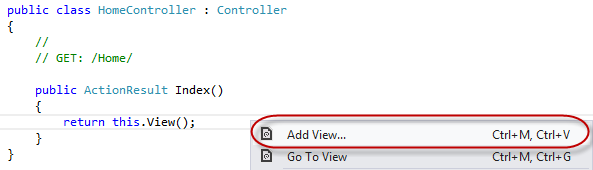
**{**

**return this.View();**

**}**

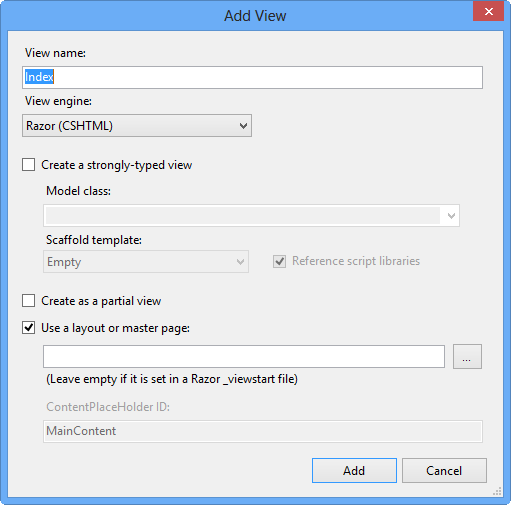
}

1. Now, you need to add an appropriate View template. To do this, **right-click** inside the **Index** action method and select **Add View**. This will bring up the **Add View** dialog.



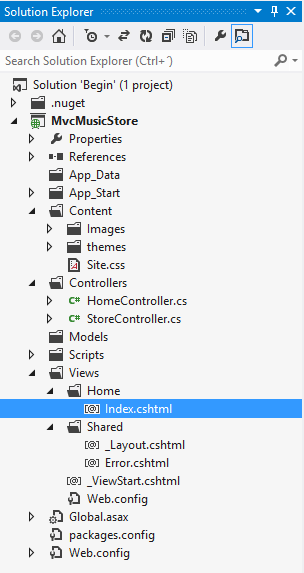
Adding a View from within the Index method

1. The **Add View** Dialog will appear to generate a View template file. By default, this dialog pre-populates the name of the View template so that it matches the action method that will use it. Because you used the **Add View** context menu within the **Index** action method within the HomeController, the **Add View** dialog has Index as the default view name. Click **Add**.



Add View Dialog

1. Visual Studio generates an **Index.cshtml** view template inside the **Views\Home** folder and then opens it.



Home Index view created

**Note:** name and location of the **Index.cshtml** file is relevant and follows the default ASP.NET MVC naming conventions.

The folder \Views\**Home** matches the controller name (**Home**Controller). The View template name (**Index**), matches the controller action method which will be displaying the View.

This way, ASP.NET MVC avoids having to explicitly specify the name or location of a View template when using this naming convention to return a View.

1. The generated View template is based on the **\_layout.cshtml** template earlier defined. Update the ViewBag.Title property to **Home**, and change the main content to **This is the Home Page**, as shown in the code below:

HTML

**@{**

**ViewBag.Title = "Home";**

**}**

**<h2>This is the Home Page</h2>**

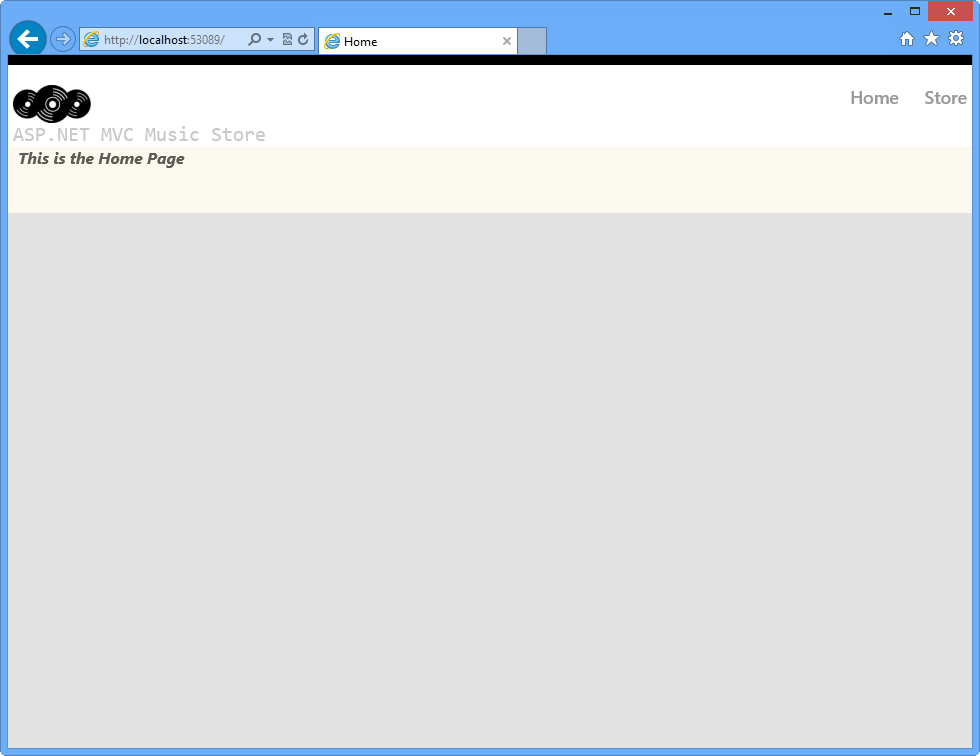
1. Select **MvcMusicStore** project in the Solution Explorer and Press **F5** to run the Application.

#### **Task 4: Verification**

In order to verify that you have correctly performed all the steps in the previous exercise, proceed as follows:

With the application opened in a browser, you should note that:

1. The HomeController's Index action method found and displayed the **\Views\Home\Index.cshtml** View template, even though the code called **return View()**, because the View template followed the standard naming convention.
2. The Home Page displays the welcome message defined within the **\Views\Home\Index.cshtml** view template.
3. The Home Page is using the **\_layout.cshtml** template, and so the welcome message is contained within the standard site HTML layout.



Home Index View using the defined LayoutPage and style