Module 8: **Applying Styles to ASP.NET MVC 5 Web Applications**

# ****Contents:****

|  |  |
| --- | --- |
|  | [Module Overview](https://www.skillpipe.com/api/2.1/content/urn:uuid:4ed7876c-8864-4cf5-91ee-491726dfe322@2020-12-11T17:05:09Z/OPS/html/20486C08.html#P1) |
| **Lesson 1:** | [Using Layouts](https://www.skillpipe.com/api/2.1/content/urn:uuid:4ed7876c-8864-4cf5-91ee-491726dfe322@2020-12-11T17:05:09Z/OPS/html/20486C08.html#P2) |
| **Lesson 2:** | [Applying CSS Styles to an MVC Application](https://www.skillpipe.com/api/2.1/content/urn:uuid:4ed7876c-8864-4cf5-91ee-491726dfe322@2020-12-11T17:05:09Z/OPS/html/20486C08.html#P3) |
| **Lesson 3:** | [Creating an Adaptive User Interface](https://www.skillpipe.com/api/2.1/content/urn:uuid:4ed7876c-8864-4cf5-91ee-491726dfe322@2020-12-11T17:05:09Z/OPS/html/20486C08.html#P4) |
| **Lab:** | [Applying Styles to MVC 5 Web Applications](https://www.skillpipe.com/api/2.1/content/urn:uuid:4ed7876c-8864-4cf5-91ee-491726dfe322@2020-12-11T17:05:09Z/OPS/html/20486C08.html#P5) |
|  | [Module Review and Takeaways](https://www.skillpipe.com/api/2.1/content/urn:uuid:4ed7876c-8864-4cf5-91ee-491726dfe322@2020-12-11T17:05:09Z/OPS/html/20486C08.html#P6) |

# ****Module Overview****

While building web applications, you should apply a consistent look and feel to the application. You should include consistent header and footer sections in all the views. ASP.NET MVC 5 includes features such as cascading style sheets (CSS) styles and layouts that enhance the appearance and usability of your web application. ASP.NET MVC 5 also includes features such as mobile-specific views that allow you to build applications for different browsers or mobile devices.

### ****Objectives****

After completing this module, you will be able to:

|  |  |
| --- | --- |
| • | Apply a consistent layout to ASP.NET MVC 5 applications by using CSS and layout. |
| • | Develop device-specific views to support various browsers. |
| • | Build adaptive user interface for mobile devices. |

# Lesson 1: ****Using Layouts****

You need to build multiple views to support the operations of the application, such as creating an order and querying order history. However, several maintenance issues arise while changing the common part of the application layout, because of which you need to update each view. To resolve these maintenance issues, you can build a common module or a shared view. A shared view that helps to store the application logic is called a layout. ASP.NET MVC 5 includes features that help simplify the process of creating and using layouts. You can further simplify the application management process by using the \_ViewStart file, to apply the layout to each view, instead of individually editing each view.

## ****Lesson Objectives****

After completing this lesson, you will be able to:

|  |  |
| --- | --- |
| • | Describe layouts. |
| • | Describe how to use layouts. |
| • | Describe the \_ViewStart file. |

## ****What Are Layouts?****

The ASP.NET MVC 5 Razor engine includes a feature called layouts. Layouts are also called template views. Layouts enable you to define a common style template, and then apply it to all the views in a web application. The functionality of layouts is similar to that of the master page in a traditional ASP.NET web application. You can use layouts to define the content layout or logic that is shared across views.

You can define multiple layouts in an ASP.NET MVC 5 application, and each layout can have multiple sections. You can define these sections anywhere in the layout file, even in the **<head>** section of the HTML. Sections enable you to output dynamic content to multiple, non-contiguous, regions of the final response.

**Question**: What are some common scenarios when you would use layouts?

## ****Creating a Layout****

While creating layouts, you need to store the layout files in the \Views\Shared folder of the project. The \Views\Shared folder is the default location, where you can store common view files or templates.

**A Layout View**

<!DOCTYPE html>

<html>

<head>

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

<title>@ViewBag.Title</title>

</head>

<body>

<div>

@RenderBody()

</div>

</body>

</html

In the preceding example, the **@RenderBody()** method indicates to the rendering engine where the content of the view goes.

**ViewBag** is an object that the layout and view shares. You can use the **ViewBag** object to pass information between a view and a layout. To pass information, you need to add a property to the **ViewBag** object, in the ViewPage of the ViewController or View file, and use the same property in the layout file. Properties help you control the content in the layout, to dynamically render webpages from the code in the view file. For example, consider that the template uses the **ViewBag.Title** property to render the **<title>** content in the view. This property helps define the **Title** property of the **ViewBag** object in the view and retrieve the property in the layout. This retrieval is possible because the code in the view file runs before the layout runs.

**Using Sections in Layouts**

<!DOCTYPE html>

<html>

<head>

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

<title>@ViewBag.Title</title>

</head>

<body>

<div id="menu">

@RenderSection("MenuBar",required:false)

</div>

<div>

@RenderBody()

</div>

</body>

</html>

The **MenuBar** parameter in the **RenderSection()** helper method specifies the name of the section that you want to render in the layout. The **required** parameter is optional; it allows you to determine if the section you render is required. Consider that a section is required, and you do not implement the section in the layout file. In this case, ASP.NET MVC 5 displays the **Section not defined** exception at runtime. Implementing sections in a layout file makes it easier to track content errors. If a section is not required, you can choose to not include it in the layout.

**Question**: Why do you have multiple sections in a layout?

## ****Linking Views and Layouts****

After defining the layout, you should link the layout to the view files. You should first remove the content that is not required anymore in the view. Then, you need to create the link between the view and the layout, so that the content removed from the view is not reflected in the layout.

**Linking to a View**

@{

ViewBag.Title = "Details";

Layout = "~/Views/Shared/SiteLayout.cshtml";

}

<h2>Details</h2>

You can use the **ViewBag.Title** property to pass page title information from the view to the layout. You can define other properties along with the **ViewBag** object, such as **<meta>** elements in the **<head>** section, and enable them to pass information to the layout.

**Using the @section Directive**

@{

ViewBag.Title = "Details";

Layout = "~/Views/Shared/SiteLayout.cshtml";

}

<h2>Details</h2>

@section MenuBar {

<p> this is menu</p>

}

In the preceding example, you set the layout file to display sections at the top of each view file. Usually, you have the same layout across the entire web application or section. You can define the layout for an application or section, by using the **\_ViewStart** file. During runtime, the code in the **\_ViewStart** file runs before all the other views in the web application. Therefore, you can place all common application logics in the **\_ViewStart** file.

**The \_ViewStart File**

@{

Layout = "~/Views/Shared/SiteLayout.cshtml";

}

**Question**: When should you use the **\_Viewstart** file?

# Lesson 2: ****Applying CSS Styles to an MVC Application****

Cascading Style Sheets (CSS) is an industry standard for applying styles to HTML pages. Different methods of applying CSS to a webpage are available. These methods include external CSS file, inline CSS, and CSS code block in HTML. Developers usually use an external CSS file, because this file is shared across multiple pages and it helps apply a consistent style across the application. You need to know how to import styles into a web application, to ensure consistency in the appearance of the application.

## ****Lesson Objectives****

After completing this lesson, you will be able to:

|  |  |
| --- | --- |
| • | Import styles into an ASP.NET MVC 5 web application. |
| • | Apply a consistent look and feel to an MVC 5 web application. |

## ****Importing Styles into an MVC Web Application****

After creating CSS styles, you should import these styles into the web application. After importing the CSS file into the web application, you need to modify the layout of the web application, so that you can use the CSS styles that you imported. You can modify the layout of a web application, by using the **<link>** element.

**Linking to a Style Sheet**

<!DOCTYPE html>

<html>

<head>

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

<title>@ViewBag.Title</title>

<link href="~/Views/Shared/StyleSheet1.css" rel="stylesheet" type="text/css" />

</head>

<body>

<div id="menu">

@RenderSection("MenuBar",required:false)

</div>

<div>

@RenderBody()

</div>

</body>

</html>

CSS selectors help browsers to determine how the CSS styles should be applied. You can use various selectors, such as class and id selectors, to apply styles to HTML elements.

**CSS class Selector**

You can define a CSS class selector to specify a style for a group of elements. To apply the class selector to an HTML element, you need to add the **class** attribute to the HTML element. You can use the **.<class>** syntax to add the style in the CSS file.

**Using a Class**

@{

ViewBag.Title = "Details";

Layout = "~/Views/Shared/SiteLayout.cshtml";

}

<h2>Details</h2>

@section MenuBar {

<p class=”menu”> this is menu</p>

}

**Applying a Style to a Class**

.menu

{

font-weight:bold;

}

**CSS id Selector**

You can use the CSS id selector to specify a style for any unique element in your HTML code. To apply the id selector to an HTML element, you need to add the **id** attribute and a unique name to the HTML element. You can use the **#<id>** syntax to add the style in the CSS file.

**Using The id Selector**

@{

ViewBag.Title = "Details";

Layout = "~/Views/Shared/SiteLayout.cshtml";

}

<h2>Details</h2>

@section MenuBar {

<p id="leftmenu"> this is menu</p>

}

**Creating an ID Style**

#leftmenu

{

font-size:16px;

}

**Question**: What are some common scenarios when you would use the class selector? What are some common scenarios when you would use the id selector?

## ****Demonstration: How to Apply a Consistent Look and Feel****

In this demonstration, you will see how to:

|  |  |
| --- | --- |
| • | Create a layout. |
| • | Apply the layout to an MVC view. |

### ****Demonstration Steps****

|  |  |
| --- | --- |
| • | You will find the steps in the “Lesson 2: Applying CSS Styles to an MVC Application“ section on the following page: <https://github.com/MicrosoftLearning/20486-DevelopingASPNETMVCWebApplications/blob/master/Instructions/20486C/20486C_MOD08_DEMO.md>. |

# Lesson 3: ****Creating an Adaptive User Interface****

ASP.NET MVC 5 applications facilitate adaptive user interface to render content on different devices. Adaptive user interface is a type of user interface that renders content based on the capability of the target web browser or device. You need to ensure that your application supports mobile devices, so that it reaches all types of audience. You also need to know how to use media queries and mobile-specific views to ensure that your application is effective on mobile devices.

## ****Lesson Objectives****

After completing this lesson, you will be able to:

|  |  |
| --- | --- |
| • | Describe the HTML5 **viewport** attribute. |
| • | Explain how CSS media queries apply specific CSS based on the capabilities of the browser. |
| • | Explain how you can use MVC 5 templates to render views based on mobile device screen size. |

## ****The HTML5 Viewport Attribute****

Adaptive rendering allows you to customize your web application to display differently, based on the capabilities of the web browser or device.

Mobile browsers such as Internet Explorer use the **viewport** attribute to render webpages in a virtual window. This virtual window is usually wider than the application screen. The **viewport** attribute helps eliminate the need to reduce the size of layout of each page. Reducing the size of the layout can break or distort the display of non-mobile-optimized web applications. Creating the application interface by using the **viewport** attribute enables users to zoom into the different areas of a webpage.

**Additional Reading:**For more information about the **viewport** attribute, visit: <http://go.microsoft.com/fwlink/?LinkID=288967&clcid=0x409>

The **viewport** tag is a meta tag that helps to control the width and height of webpages, while it renders to web browsers.

**Using the Viewport Tag**

<meta name="viewport" content="width=device-width, initial-scale=1, maximum-scale=1">

The **width** and **height** properties help to specify the width and height of the virtual viewport window. You can specify the width in pixels. You can use the keyword **device-width** to enable the content to fit the native screen size of the browser.

The **initial-scale** property controls the initial scale or zoom level of the webpage. The **maximum-scale**, **minimum-scale**, and **user-scalable** properties control the other scalability features of the webpage.

**Question**: How can you control the size of the virtual viewport window?

## ****CSS Media Queries****

You may sometimes need to apply different CSS styles in your application, to support different browsers. HTML5 includes CSS media queries, which are special selectors that begin with **@media**. Media queries allow conditional application of CSS styles, based on the device conditions or browser capabilities. You can apply media queries in CSS and HTML.

**Using a Media Query**

@media only screen and (max-width: 500px) {

header{

float: none;

}

}

**Using a Media Query in the Link Element**

<link rel="stylesheet" type="text/css" href="smallscreen.css" media="only screen and (max-width: 500px)" />

You can use CSS media queries to apply CSS styles when the screen size is less than 500 pixels. However, you can use CSS media queries only for the screen layout, but not the print layout.

The following table describes properties that you can include in a media query.

| **Property** | **Description** |
| --- | --- |
| **width** | The width of the targeted display area, which includes the browser window in desktop and mobile devices. In desktop computers, when you resize the browser window, the width of the browser changes. However, on most mobile browsers, you cannot resize the browser window. This implies that the width of the browser remains constant. |
| **height** | The height of the targeted display area, which includes the browser window in desktop and mobile devices. |
| **device-width** | The width of the entire screen of a device. For a desktop with a screen resolution of 1,024x768, the device-width is usually 1,024 pixels. |
| **device-height** | The height of the entire screen of a device. For a desktop with a screen resolution of 1,024x768, the device-height is usually 768 pixels. |
| **orientation** | The orientation of the device. If the device-width is larger than **the device-height**, the orientation value is set to **landscape**; otherwise, it is set to **portrait**. |
| **aspect-ratio** | The ratio of the **width** and **height** properties. |
| **device-aspect-ratio** | The ratio of the **device-width** and **device-height** properties. The following example illustrates the device-aspect-ratio for a device with a screen resolution of 1,280x720.  @media screen and (device-aspect-ratio: 16/9) { }  @media screen and (device-aspect-ratio: 1280/720) { }  @media screen and (device-aspect-ratio: 2560/1440) { } |
| **color** | The number of bits per color component of the device. If the device is not a color device, the value is zero. |
| **color-index** | The number of entries in the color lookup table, of the output device. |
| **monochrome** | The number of bits per pixel in a monochrome frame buffer. For non-monochrome devices, this value is zero. |
| **resolution** | The resolution of the output device, or the density of the pixels. The common units for this property include dpi and dpcm. |
| **scan** | The scanning process of TV output devices. |
| **grid** | The property that detects whether the output is in the grid or bitmap format. Grid-based devices return a value of one; all other devices return a value of zero. |

**Question**: Why would you choose to use CSS media queries, instead of using C# code, to define styles for specific browsers?

## ****MVC 5 Templates and Mobile-Specific Views****

ASP.NET MVC 5 includes two features, mobile display mode and custom display mode, which help you to create webpages for mobile devices and different browsers.

ASP.NET MVC 5 enables you to override views for mobile devices by using a different set of view files, rather than using a configuration. When ASP.NET MVC 5 receives a request from a mobile browser, it analyses the request for views with the naming convention **[view].mobile.cshtml**. If ASP.NET MVC 5 detects a view with the mentioned naming convention, ASP.NET MVC 5 will serve the request by using the mobile version of the view; otherwise, it returns the request to the standard view.

Consider that your web application includes a layout that is specific to a browser. In this case, you can create browser-specific views for that browser by checking the **UserAgent** string of that browser. The **UserAgent** string helps identify a browser.

**Question**: Why would you choose device-specific display modes over CSS media queries?

# ****Lab: Applying Styles to MVC 5 Web Applications****

### ****Scenario****

You have created a good amount of the photo-handling functionality for the Photo Sharing web application. However, stakeholders are concerned about the basic black-and-white appearance of the application. In addition, they want the titles and menus to appear on every page.

To resolve these issues, your manager asked you to implement the following user interface features:

|  |  |
| --- | --- |
| • | A layout for all webpages. The layout should include common elements, such as the main menu and breadcrumb controls, which should appear on every page of the application. |
| • | A style sheet and images for all webpages. The web design team has provided an HTML mockup application to show how the final product should look. This mockup includes a style sheet and image files. You need to import these files and apply them to every page of the application. |
| • | A mobile-specific view. The web application should be accessible from mobile devices such as mobile phones and tablets. In particular, you need to ensure that devices with narrow screens can access photos easily. |

### ****Objectives****

After completing this lab, you will be able to:

|  |  |
| --- | --- |
| • | Apply a consistent look and feel to the web application. |
| • | Use layouts to ensure that common interface features, such as the headers, are consistent across the entire web application. |
| • | Ensure that the web application renders smoothly on screens of different sizes and aspect ratios. |

##### ****Lab Setup****

Estimated Time: 60 minutes

You will find the high-level steps on the following page: <https://github.com/MicrosoftLearning/20486-DevelopingASPNETMVCWebApplications/blob/master/Instructions/20486C/20486C_MOD08_LAB_MANUAL.md>.

You will find the detailed steps on the following page: <https://github.com/MicrosoftLearning/20486-DevelopingASPNETMVCWebApplications/blob/master/Instructions/20486C/20486C_MOD08_LAK.md>.

### ****Exercise 1: Creating and Applying Layouts****

##### ****Scenario****

In this exercise, you will:

|  |  |
| --- | --- |
| • | Browse through the Photo Sharing web application without a layout applied. |
| • | Create a new layout and link the application to the view by using a \_ViewStart.cshtml file. |
| • | Modify the home index and photo display views to use the new layout. |
| • | Browse through the resulting web application. |

### ****Exercise 2: Applying Styles to an MVC Web Application****

##### ****Scenario****

In this exercise, you will:

|  |  |
| --- | --- |
| • | Examine a mockup web application that shows the look-and-feel the web designers have created for the Photo Sharing application. |
| • | Import a style sheet, with the associated graphic files from the mockup application, to your web application, and then update the HTML element classes to apply those styles to the elements in views. |
| • | Examine the changes to the user interface after the styles have been applied. |

### ****Exercise 3: Optional—Adapting Webpages for Mobile Browsers****

##### ****Scenario****

In this exercise, you will:

|  |  |
| --- | --- |
| • | Create a new layout for mobile devices. |
| • | Add a media query to the web application style sheet to ensure that the photo index is displayed on small screens. |
| • | Test the settings applied to the application by using a small browser and changing the user agent string. |

Complete this exercise if time permits.

### ****Review Question(s)****

**Check Your Knowledge**

**Discovery**

**When you first browsed the web application in Exercise 1, why was the menu and the breadcrumb trail visible on the home page, but not on the All Photos page or any other page?**

Show solution Reset

**Check Your Knowledge**

**Discovery**

**When you first viewed the site as a mobile browser in Exercise 3, what are the problems you came across with the display of the site heading and menu?**

Show solution Reset

# ****Module Review and Takeaways****

In this module, you learned how to apply a consistent look and feel to a web application, and share other common components, such as headers and footers, between all views. You also learned how to use the CSS and display modes to adapt the web application for smaller screens and mobile devices. You also familiarized yourself with HTML5 elements that allow you to develop web applications that work on various browsers and devices.

### ****Real-world Issues and Scenarios****

When you develop web applications, you need to create applications that work on different devices and browsers, such as iPhone, iPad, Google Chrome, Microsoft Edge and Internet Explorer. In such cases, you can use the HTML5 elements and features in MVC 5, such as mobile-specific views and media queries to create applications that work well in various browsers and devices.

### ****Review Question(s)****