Session: 7

Consistent Styles and Layouts

Objectives

- Define and describe how to use layout to maintain consistent look and feel
- Explain the process of creating a custom layout
- Explain how to implement styles in an application
- Explain and describe how to create adaptive styles

Consistent Look and Feel Using Layouts

- ASP.NET MVC Framework allows you to use layouts that simplify the process of maintaining consistent look and feel in an application.
- You can use layout that allows you to specify a style template to be used across the views in an application.
- In this layout, you can define the structure and common styles for the views.

_Layout.cshtml File

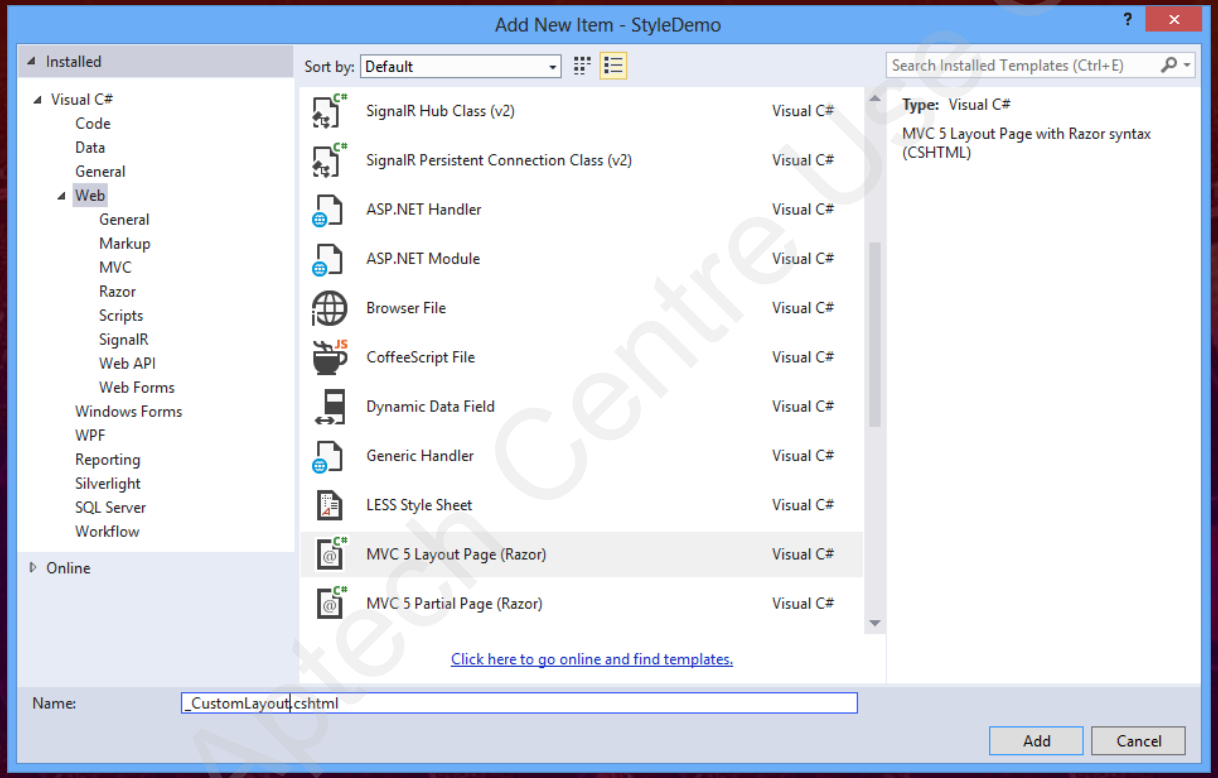
- When you create an ASP.NET MVC Web application in Visual Studio 2013, a _Layout.cshtml file is automatically added in the View/Shared folder.
- This file represents the layout that you can apply to the views of the application.
- This _Layout.cshtml file contains the basic structure of a view.
- While developing an application, you can also modify the _Layout.cshtml based on your requirements.

Custom Layout 1-6

- While developing an ASP.NET MVC Web application, sometime you might require creating your custom layout.
- Creating a layout in Visual Studio 2013 is similar to creating a view.
- By default, the name of the layout file is preceded by the underscore (_) character.
- The extension of a layout file is .cshtml.
- To create a custom layout using Visual Studio 2013, you need to perform the following steps:
  - Right-click the Shared folder under the Views folder in the Solution Explorer window.
  - Select Add → New Item from the context menu that appears. The Add New Item dialog box is displayed.
  - Select MVC 5 Layout Page (Razor) template.
  - Enter _CustomLayout in the Name text field.

Custom Layout 2-6

- Following figure shows the Add New Item dialog box:



- Click Add. This will add the newly created layout in the Views/Shared folder.

Custom Layout 3-6

- You can display unique content associated with a view using the following methods:
  - @RenderBody(): Allows you to define a placeholder in the layout to insert the main content of the view.
  - @RenderSection(): Allows you to display a section specified in the view.
- You can use the @RenderBody() method to define a placeholder in the layout that needs to be inserted in the main content of the view.
- While using the @RenderBody() method, you should remember that, you can use it only once in a layout.

Custom Layout 4-6

- Following code snippet shows a layout named _CustomLayout.cshtml file that uses the @RenderBody() method:

Snippet

```
<!DOCTYPE html>
<html lang="en">
<head>
<title> ASP.NET MVC Application</title>
</head>
<body>
<p>
    @RenderBody()
</p>
</body>
</html>
```

- In this code, the @RenderBody() method inside the <p> tag inserts the content of the view.

Custom Layout 5-6

- You can use the @RenderSection() method to display a section specified in the view.
- A layout allows you to add one or more sections in it.
- Following code snippet shows the _CustomLayout.cshtml layout that uses the @RenderSection() method:

Snippet

```
<!DOCTYPE html>
<html>
<head>
<title>@ViewBag.Title</title>
</head>
<body>
<div id="main-content">@RenderBody()</div>
<div>@RenderSection("TestSection")</div>
</body>
</html>
```

- In this code, the @RenderSection() method is used to add a section.

Custom Layout 6-6

- Sometimes, you may not want to use a specified section in some of the views in your application.
- In such situation, you can use an overloaded method of the @RenderSection() method in your layout.
- This overloaded method uses the required attribute with the false value.
- Following code snippet shows how to use an overloaded method of the @RenderSection() method:

Snippet

```
<div>@RenderSection("TestSection", required: false)</div>
```

- In this code, the @RenderSection() method indicates that a view that uses this layout can provide content for the TestSection section. As this is not mandatory, a view using this layout may not contain any section named TestSection.

Specifying a Layout for a View 1-2

- While developing an ASP.NET MVC Web application, you can also specify a layout for either a particular view or for all the views.
- You can specify a layout for a view by using the Layout property.
- Following code snippet shows specifying a layout for a view:

Snippet

```
@{
Layout = "~/Views/Shared/CustomLayout.cshtml";
}
```

- In this code, the CustomLayout.cshtml file is specified as the layout for a view available in the application.
- You can also specify a particular layout for all the views available in your application by using the Layout property in the _ViewStart.cshtml file.
- This file is available under the Views folder.

Specifying a Layout for a View 2-2

- When you create an ASP.NET MVC Web application in Visual Studio 2013, the _ViewStart.cshtml file is created by default under the Views folder.
- This file specifies the default layout to be used for the views available.
- Following code snippet shows the default layout in the _ViewStart.cshtml file:

Snippet

```
@{
Layout = "~/Views/Shared/_Layout.cshtml";
}
```

- In this code, the default layout is specified as _Layout.cshtml.
- When you run the application, the code available in the _ViewStart.cshtml file is executed first.
- Then, the code of the available views in the same directory or within a sub-directory is executed.

Nested Layout

- While developing Web applications, sometimes you might need to maintain some standard features.
- In such scenarios, you can use nested layouts.
- A nested layout:
  - Refers to a layout that is derived from a parent layout and is responsible for defining the structure of a page.
  - Uses the @RenderBody() method only once.
  - It can have multiple sections that can be rendered by using the @RenderSection() method.

Implementing Styles

- Besides using the layouts, in an ASP.NET Web application, you can also use styles that allow you to apply consistent formatting across all the views available.
- Styles are used to define a set of formatting options, which can be reused to format different HTML helpers on a single view or on multiple views.

CSS Files 1-8

- A .css file in an ASP.NET Web application allows you to define a style that needs to be applied in the application.
- Whenever you create an ASP.NET MVC Web application in Visual Studio 2013, a .css file named Site.css is automatically created under the Content folder.
- Following code snippet shows using style definitions in the Site.css file:

Snippet

```
H2 {
font-weight: bold;
font-size: medium;
color: red;
font-family: Times New Roman; }
```

- This code contains a style definition that will be applied to all the <h2> tags in the views that references the test.css file.
- Here, <h2> tag acts as a selector that specifies the elements on which the specified style needs to be applied.

CSS Files 2-8

- You can also define same style for more than one element by combining selectors as a single group.
- Following code snippet shows combining two elements in the Site.css file:

Snippet

```
H1,H2 {COLOR: BLUE;}
```

- This code will display all the text that are specified inside the <h1> and <h2> elements in blue color.
- Besides applying styles using elements, you can also use the following types of CSS selectors:
  - ID selector: Is used to identify an element that you need to style differently from the rest of the page. Each element on a view can have a unique Id, which contains a hash symbol (#) followed by the element Id.

CSS Files 3-8

- Following code snippet specifies an id for the <p> tag:

Snippet

```
<p id="uname">Mobile Phone Users</p>
```

- Now, you can define style separately for the element, whose Id is uname.
- Following code snippet shows defining style specifications to be applied to the element, whose Id is uname:

Snippet

```
#uname
{
color:green;
font-size:20pt;
font-weight:bold;
}
```

- This code selects the element with the id named, uname and applies the specified styles on it.

CSS Files 4-8

- Class selector:
  - Is used to specify styles for a group of elements.
  - Allows applying a style on several elements in a view.

- Following code snippet shows how to use class selector to define styles:

Snippet

```
<p class="red"> First paragraph </p>
<p class="red"> Second paragraph </p>
<p class="green"> Third paragraph </p>
<p class="green"> Fourth paragraph </p>
<p class="blue"> Fifth paragraph </p>
```

- This code uses the class selector to define styles for multiple elements.

CSS Files 5-8

- Following code snippet shows applying different style to the five paragraphs:

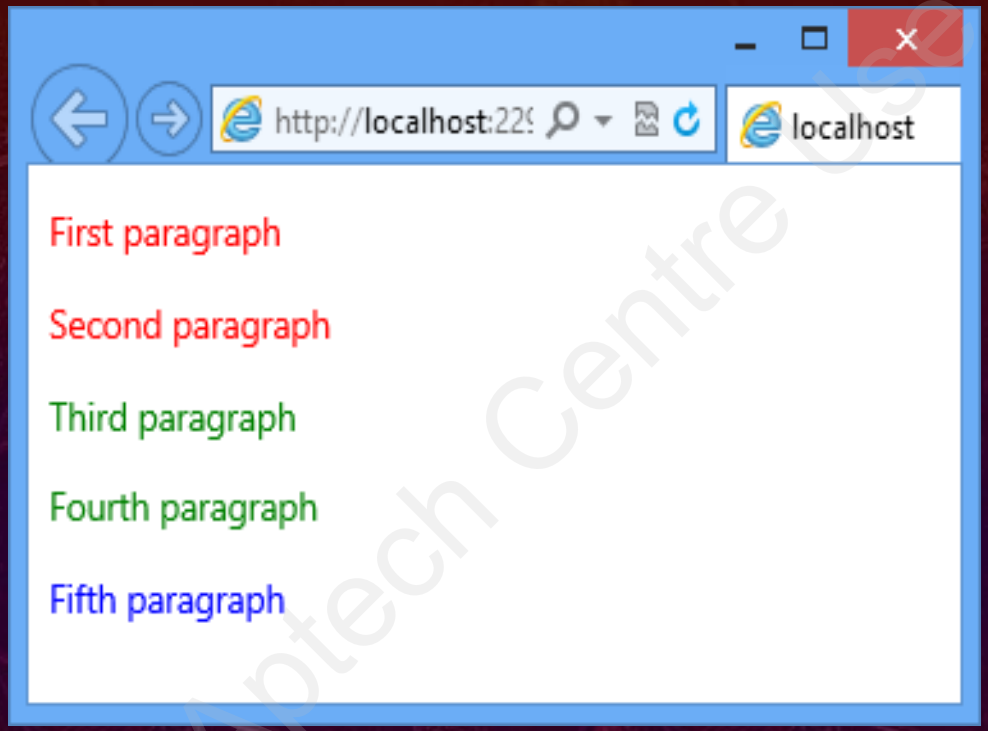
Snippet

```
.red
{
color:red;
}
.green
{
color:green;
}
.blue
{
color:blue
}
```

- This code will display the first two paragraphs in red color, the next two paragraphs in green color, and the last paragraph is in blue color.

CSS Files 6-8

- Following figure shows the output of applying different style to five paragraphs:



CSS Files 7-8

- Universal selector is used to specify styles for all available elements in a particular area of a page or the whole page.
- To define universal selector, you should use the * symbol.
- Following code snippet shows the tag that contains some elements:

Snippet

```
<div class="Customers">
<p id="category1">Mobile users</p>
<p id="category2">Laptop users</p>
<p id="category3">Desktop users</p>
</div>
```

- This code creates a tag that contains three <p> tags with data inside it.
- Now, to apply styles to all the elements in the preceding tag, you can use the universal selector.

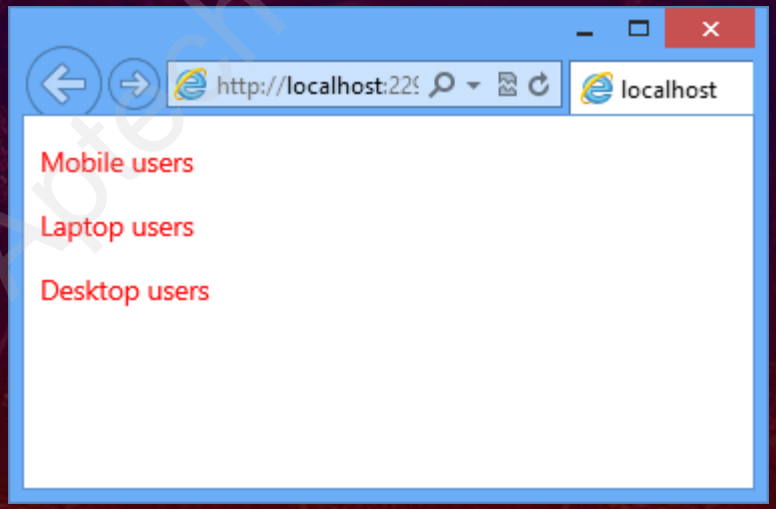
CSS Files 8-8

- Following code snippet shows applying styles using the universal selectors:

Snippet

```
div.Customers *
{
color:red;
}
```

- This code defines styles using the universal selector in the Site.css file.
- Following figure shows the output of applying styles using the universal selector:



Importing Styles in Views

- Once you have defined styles in the Site.css file, you need to link it with HTML documents on which the styles are to be implemented.
- For this, you can use the <link> tag that enables linking the views along with style sheet together.
- Following code snippet shows how to use the <link> tag:

Snippet

```
<link href="~/content/Site.css" rel="stylesheet" type="text/css" />
```

- In this code:
  - The Site.css file is referred to by using the href attribute of the <link> tag.
  - The value of the rel attribute specifies that the document will use a style sheet.
  - Finally, the value of the type attribute specifies that the MIME type of the linked document is text/css.

Adaptive User Interface 1-11

- At recent time it is possible to access Web application from various devices, such as tabs, laptops, or mobile phones.
- Due to the different screen sizes and browser capability, the page size and visual appearance of a Web application may change accordingly.
- Adaptive rendering enables automatic scaling down of a page and redrawing the page according to a device screen.
- You can create an adaptive UI in order to implement adaptive rendering using one of the following mechanisms:
  - Viewport meta tag
  - CSS media queries

Adaptive User Interface 2-11

- The browsers of a mobile device can render pages of a Web application and scale them in a manner so that they can properly appear in the screen area of the device.
- These pages are rendered in a virtual window known as a viewport.
- Viewport is typically wider than the actual width of a mobile device.
- You can control the size and scaling of pages of a Web application using the viewport <meta> tag in the head section of the view, as shown in the following code snippet:

Snippet

```
<meta name="viewport" content="width=250">
```

- In this code, the width property is set to 250 pixels. As a result, the pages of the Web application will be appear on a canvas of width 250 pixels. It will also be scaled properly inside the screen area.

Adaptive User Interface 3-11

- You can also use the device-width keyword to set the width of a viewport to the native screen size of the browser.
- This allows you to create the content, which can be adjusted according to the device specifications.
- Following code snippet shows how to use the device-width keyword:

Snippet

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

- In this code, the width property is set to the device-width value. This specifies the width of the device screen.

Adaptive User Interface 4-11

- You can use CSS media queries to enable your application to support different browsers and devices.
- To use CSS media queries you need to use the @media keyword.
- Using these queries you can apply different conditional CSS styles to support different device conditions or browser capabilities.
- Following code snippet shows the content of an Index.cshtml view:

Snippet

```
<html>
<head>
<title>Sample Article</title>
<link href="~/content/Site.css" rel="stylesheet" type="text/css" />
</head>
<body>
<div id="container">
<div id="article1">
<h1>Cricket</h1> Cricket is a bat-and-ball game played between two
teams of 11 players each on a field at the center of which is a
rectangular 22-yard long pitch. Each team takes its turn to bat,
attempting to score runs, while the other team fields. Each turn is
known as an innings. There are several for mats of the cricket game.
</div>
```

Adaptive User Interface 5-11

- Following code snippet shows the content of an Index.cshtml view:

Snippet

```
<div></div>
<div></div>
<div id="article2">
<h1>Football</h1>Football refers to a number of sports that involve, to varying degrees, kicking a ball with the foot to score a goal. The most popular of these sports worldwide is association football, more commonly known as just "football" or "soccer". The variations of football are known as football codes.
</div>
</div>
</body>
```

- This code creates the Index.cshtml view that contains sample articles on two topics.
- Once you have created the Index.cshtml view, then, you can apply styles in the Site.css file to design the Index.cshtml view.

Adaptive User Interface 6-11

- Following code snippet shows applying styles to design the Index.cshtml view:

Snippet

```
h1 {
    text-align: center;
    color:blue;
}
#container {
    float: left;
    width: 830px;
}
#article1 {
    float: right;
    width: 410px;
    background: lightblue;
}
#article2 {
    float: left;
    width: 410px;
    background: lightblue;
}
```

Adaptive User Interface 7-11

- The preceding code applies styles in the Site.css file to design the look and feel of the Index.cshtml view.
- Following figure shows the Index.cshtml view when access it on a browser:

Adaptive User Interface 8-11

- When you reduce the dimensions of the browser window, the look and feel of the view is affected.
- As a solution, you can use media queries in the Site.css file to add styles that allow reducing the dimensions of a view.
- Following code shows how to use media queries:

Snippet

```
@media screen and (max-width:700px) {
    #container {
        float: left;
        width: 500px;  }
    #article1 {
        float: left;
        width: 500px;
         background: lightblue;   }
    #article2 {
        float: none;
        width: 500px;
background: lightblue;    }
}
```

Adaptive User Interface 9-11

- Following code shows how to use media queries:

Snippet

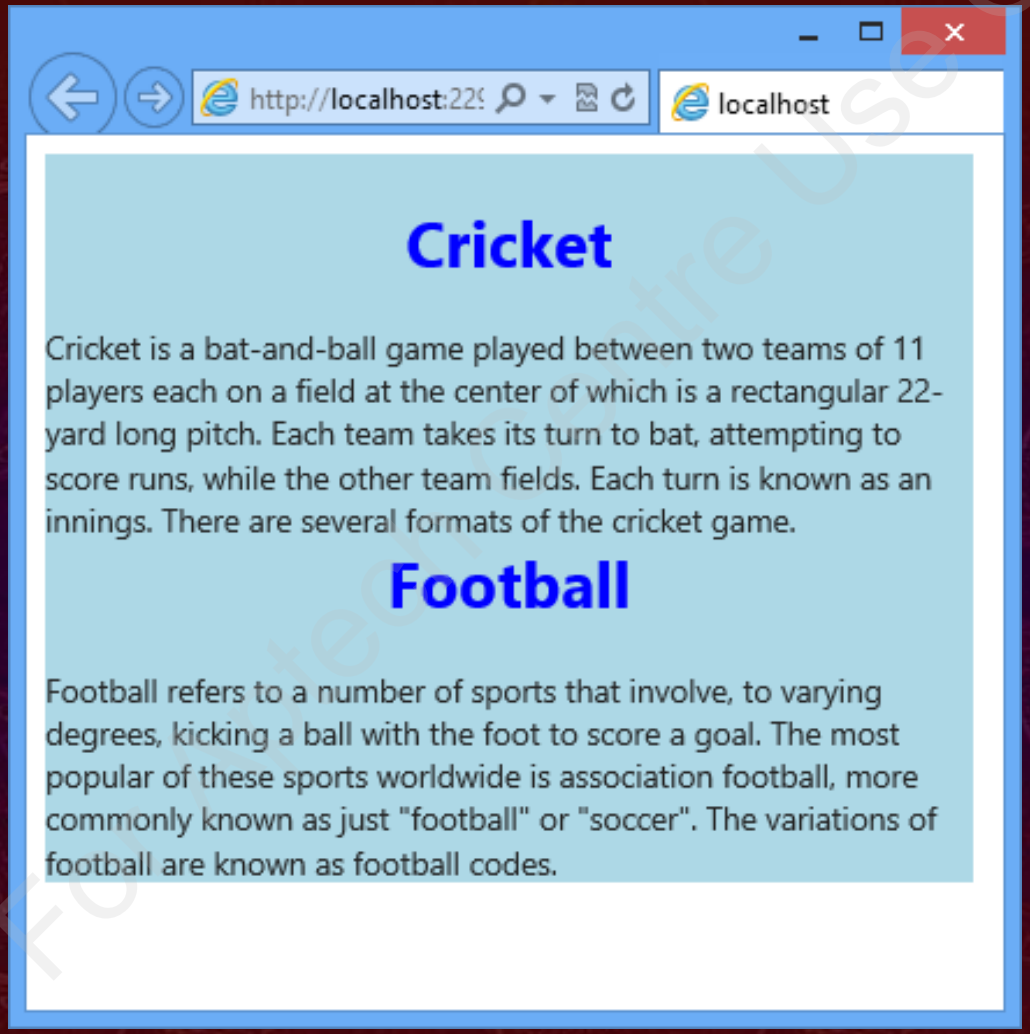
```
@media screen and (max-width:500px) {
    #container {
        float: left;
        width: 400px; }
    #article1 {
        float: left;
        width: 400px;
        background: lightblue;
    }
    #article2 {
        float: none;
        width: 400px;
        background: lightblue;
    }
}
```

Adaptive User Interface 10-11

- This code defines two media queries by using the @media syntax:
  - The first query defines the styles for the screens that have maximum width of 700px.
  - The second query defines the style for the screens that have maximum width of 500px.
- When you reduce the width of the browser, the view will appear according to the styles specified for reduced screen width.

Adaptive User Interface 11-11

- Following figure shows the Index.cshtml view with reduced screen width:



Summary

- ASP.NET MVC Framework allows you to use layouts to simplify the process of maintaining consistent look and feel in an application.
- The _Layout.cshtml file represents the layout that you can apply to the views of an application.
- You can specify a layout for a view by using the Layout property.
- A nested layout refers to a layout that is derived from a parent layout and is responsible for defining the structure of a page.
- Styles are used to define a set of formatting options, which can be reused to format different HTML helpers on a single view or on multiple views.
- The Site.css file allows you to define styles for a particular view or for all the views available in the application.
- Adaptive rendering enables automatic scaling down of a page and redrawing the page according to a device screen.