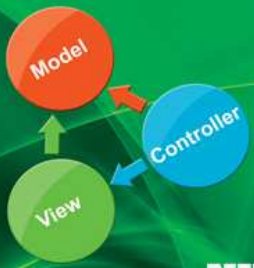
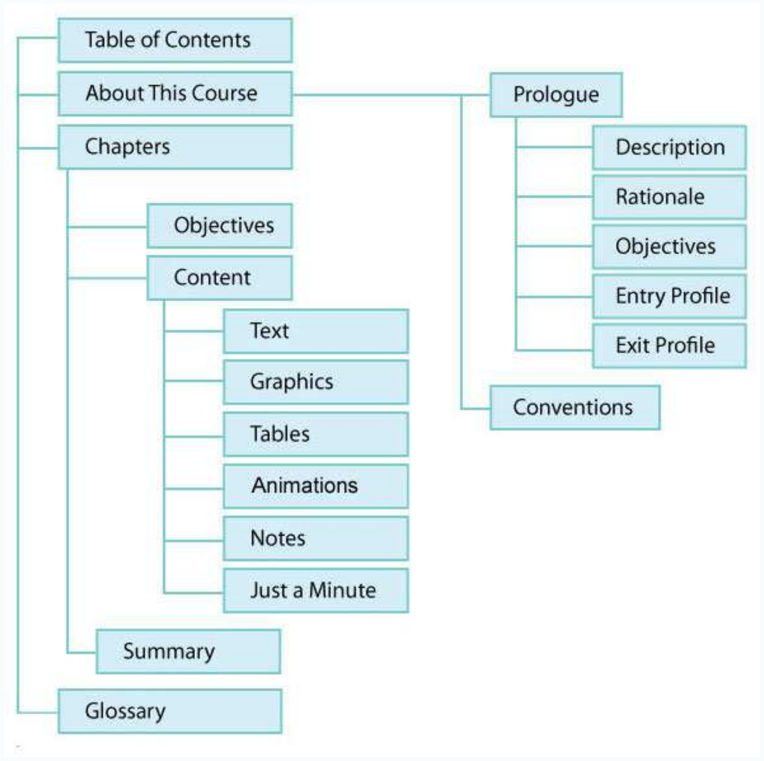
Student Guide

Developing Web Application Using ASP.NET MVC - Part II



NIIT

COURSE DESIGN - STUDENT GUIDE



About This Course

Prologue

Description

ASP.NET provides a unified Web development model, which includes the services for creating websites easily and quickly. This course discusses various features, such as master pages and themes, which enable the developers to create pages that are more consistent and offer a richer experience to the users. It also discusses how to establish communication between a Web application and a database server. In addition, it discusses how to deploy Web applications on a server.

Rationale

With the increased use of the Internet and development in the field of Information Technology (IT), application developers need to quickly create applications that are accessible over the Web or a corporate intranet. These applications should also be efficient and effective.

ASP.NET provides developers with various time-saving and code-saving features. One of its key design goals is to make programming easier and quicker by reducing the amount of code. In addition, it contains several new server controls, which eliminate the need for writing voluminous code.

Objectives

After completing this module, the student will be able to:

- ❑ Identify the fundamentals of application development
- ❑ Work with Controllers, Views, Models, and Helper Methods
- ❑ Identify data annotations and implement validation
- ❑ Identify the fundamentals of Entity Framework
- ❑ Identify the fundamentals of LINQ
- ❑ Implement a consistent look and feel using layouts
- ❑ Make a Web application responsive by using JavaScript
- ❑ Implement the partial page updates using AJAX
- ❑ Implement state management and optimize the performance of a Web application
- ❑ Implement authentication and authorization
- ❑ Deploy a Web application

Entry Profile

The students who want to take this course should be:

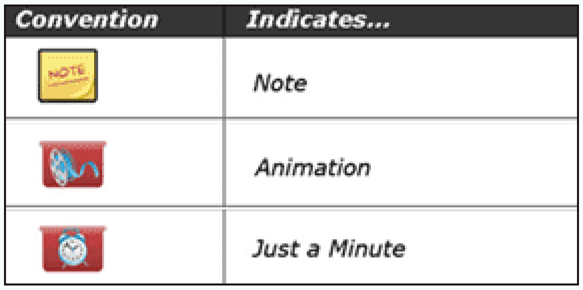
- ❑ Familiar with fundamentals of computers.
- ❑ Familiar with programming logic and techniques.
- ❑ Familiar with HTML 5, CSS, and JavaScript.
- ❑ Able to interact in English in a classroom environment because the classes will be conducted in English.

Exit Profile

After completing this course, the student should be able to:

- ❑ Identify the various phases and development models of application development life cycle.
- ❑ Design interactive Web applications.

Conventions



Convention	Indicates...
NOTE	*Note*
	Animation
⏰	*Just a Minute*

Chapter 6

Using LINQ to Access Data

While developing data-centric applications, you need to deal with two types of code, the business logic implementation code and the data manipulation code. The business logic implementation code is written by using a programming language. However, traditionally, the data manipulation code includes queries written in a data-source-specific query language. Therefore, in order to develop a data-centric application, you need to learn a programming language as well as a query language. In addition, if the data source of an application changes, you need to learn a different query language and modify the data manipulation code of your application.

To overcome these issues, Microsoft introduced Language Integrated Query (LINQ). LINQ enables you to create data-source-independent queries in an application and offers different providers to work with different data sources.

This chapter explains how to work with LINQ. Further, it discusses how to use LINQ to access data from disparate data sources, such as SQL, XML, and business objects.

Objectives

In this chapter, you will learn to:

- ❑ Identify the fundamentals of LINQ
- ❑ Access data from disparate data sources

Introduction to LINQ

Traditionally, an application that needed to access data from a database used to include database-specific queries in a string format within the application code. For example, consider the following code snippet that creates an application object for storing a query that retrieves data from the Product table:

```
SqlCommand cmd = new SqlCommand
("SELECT * FROM Product",
myConnection);
```

Queries in this format do not support compile-time checking or intellisense support. In addition, you need to learn multiple query languages to interact with the various data sources, such as SQL databases, XML documents, and various Web services.

LINQ simplifies this situation by offering a consistent model for working with data across the various kinds of data sources and formats.

LINQ is a set of APIs that enables you to create data-source-independent queries to implement data access in an application. It has a programming model that provides the standard query syntax to query different types of data sources. In LINQ, the query is written in any .NET Framework supported programming language, such as C# or VB. The LINQ query acts on a strongly-typed collection of objects with the help of language keywords and common operators. This helps developers to quickly manipulate data from the various data sources without requiring them to learn a new query language.

Using a Simple LINQ Query

LINQ offers a consistent programming model to query data from different data sources. So, the same query syntax is used to query and alter data in the various data sources. Therefore, all LINQ query operations consist of the same set of actions that act on different data sources. All LINQ query operations consist of the following three distinct actions:

1. Obtain the data source.
2. Create the query.
3. Execute the query.

Obtaining the Data Source

For executing a LINQ operation, the first step is to specify the data source. All LINQ queries operate on objects that implement the `IEnumerable<T>` or `IQueryable<T>` interface. This allows LINQ queries to operate on a variety of types, ranging from in-memory arrays and collections (that implement the `IEnumerable<T>` interface) to external databases, such as SQL server or Oracle (that implement the `IQueryableT>` interface). The `IEnumerable<T>` interface is available in the System.Collections namespace. `T` in the `IEnumerable<T>` interface represents the type of data that is being retrieved from a collection or array, whereas, the `IQueryable<T>` interface enables you to create queries to retrieve data from a specific data source. `T` in the `IQueryable<T>` interface represents the type of data in a data source. This interface is available in the `System.Linq` namespace.

Creating the Query

A query specifies the type of information to be retrieved from the data source. A query is written by using a query expression that uses the standard query syntax, and is stored in a query variable. The query variable stores the information that is required to produce the results when the query is executed.

Consider the scenario of the Newbay application wherein you need to retrieve customer details from the Customer table stored in an SQL Server database. In this application, you can use an instance of the `IQueryable<T>` interface to retrieve the customer details from the data source, as shown in the following code snippet:

```
IQueryable<Customer> q = from s in
db.customers
select s;
```

In the preceding code snippet, the `from` clause specifies the data source from where the data has to be retrieved. `db` is an instance of the data context class that provides access to the `customers` data source, and `s` is the range variable. When the query is executed, the range variable acts as a reference to each successive element in the data source.

The `select` clause in a LINQ query specifies the type of the returned elements. Using the `select` clause, you can specify whether your result will consist of complete customer objects, a subset of the members of the customer object, or some computed values. The `select` clause in the preceding LINQ query specifies that each element in the result will consist of a customer object.

Now, consider the following LINQ query:

```
var q = from s in db.customers
select s.Name;
```

The `select` clause in the preceding LINQ query specifies that the result will consist only of the names of the customers.

Further, consider the following LINQ query:

```
var q = from s in db.customers
select new {s.Name, s.Address};
```

The `select` clause in the preceding LINQ query specifies that the result will consist of the names and addresses of the customers.

Executing the Query

So far, you have seen how a query is created. The query would not retrieve the data unless it is executed. In order to execute a query, you need to iterate over the query variable in a `foreach` statement. This concept is referred to as deferred execution.

Deferred execution helps you to create a query once, and then execute it as many times as you want. It is useful in situations where the database is constantly updated. In such a situation, you can create a query variable and execute it repeatedly at some interval to retrieve the updated data.

For example, to retrieve and display the customer details from the Customers table of an SQL Server database, you need to first create a query. Next, you need to use the `foreach` statement to execute the query and retrieve the details, as shown in the following code snippet:

```
string names = "";
IQueryable<Customer> q = from s in
db.customers
select s;
foreach (var cust in q)
{
names = names+" "+cust.Name;
}
ViewBag.Name = names;
```

In the preceding code snippet, the `foreach` statement is used to retrieve the query results iteratively and the variable, `cust`, is used to hold each value one at a time. The name of each customer is appended to the string variable, `names`. Finally, the `names` variable is assigned to the `Name` property of `ViewBag` to pass the information to a view.

Queries that perform aggregate functions, such as `Max` and `Count`, over a range of source elements, execute without explicitly using the `foreach` loop and return a single value. These queries are executed at the time of their definition. This is called immediate execution. For example, to count the number of customer records in the `db.customers` data source, consider the following code snippet:

```
int customerCount = (from s in
db.customers
 select s).Count();
ViewBag.Count = customerCount;
```

In the preceding example, the `foreach` loop has not been used to execute the query. It gets executed automatically when the `Count()` function is used.

You can also force immediate execution by calling the `ToList<TSource>()` or `ToArray<TSource>()` methods.

The `ToList<TSource>()` method forces immediate query execution and returns `List<T>` that caches the query results. Consider the following example, which forces immediate query execution and returns `List<T>` that contains the query results:

```
List<string> names = (from s in
db.customers
 select s.Name).ToList();
```

The preceding code returns a list that contains the names of all the customers.

The `ToArray<TSource>()` method forces immediate query execution and returns an array that caches the query results. Consider the following example, which forces immediate query execution and returns an array that contains the query results.

```
string[] names = (from s in
db.customers
    select s.Name).ToArray();
```

The preceding code returns an array that contains the names of all the customers.

Using LINQ Queries to Filter, Sort, Group, and Join Data

In addition to selecting data, a query can perform the following operations on data:

- ❑ Filter the data

- ❑ Sort the data
- ❑ Group the data
- ❑ Join the data

Filtering the Data

Filtering refers to the operation of restricting the result set to contain only those elements that satisfy a specified condition. The filter causes the query to return only those elements for which the expression is true. In LINQ, the concept of filtering can be used effectively to filter out a set of objects from a list of objects.

Consider the scenario of Newbay wherein you need to filter only those customers who have an address in Tokyo. This requires you to retrieve the records from the `Customer` table. For this, you need to use the `from` and `where` clauses in the query, as shown in the following code snippet:

```
IQueryable<Customer> q = from s in
db.customers
               where s.City == "Tokyo"
               select s;
```

The preceding code snippet retrieves details of all the customers from Tokyo.

In addition, you can use the logical operators, such as AND (&&) and OR (||), to apply multiple filters as necessary in the `where` clause. For example, to filter the customers who live in Tokyo and have the name, Tom, you need to use the following code snippet:

```
IQueryable<Customer> q = from s in
db.customers
                   where s.City ==
 "Tokyo" && s.Name == "Tom"
                   select s;
```

In the preceding code snippet, the AND (&&)operator is used to apply multiple filters.

Sorting the Data

LINQ queries can also be used to retrieve data and display it in a sorted manner. A sorting operation orders the elements in a proper sequence based on one or more attributes. In order to implement sorting using LINQ, you need to use the `orderby` clause. The `orderby` clause specifies the order in which the result of the query should be sorted. This clause enables you to sort the result of a query either in the ascending order or in the descending order. The default sort order is ascending. However, you can explicitly use the ascending keyword to sort in the ascending order.

Consider the scenario of Newbay wherein you need to sort the names of the customers who live in Tokyo in the ascending order. This requires you to retrieve the records from the `Customer` table. For this, you need to use the `ascending` keyword along with the `orderby` clause, as shown in the following code snippet:

```
IQueryable<Customer> q = from s in
db.customers
               where s.City == "Tokyo"
               orderby s.Name
ascending
               select s;
```

The preceding query retrieves the records of customers who stay in Tokyo in the ascending order of the names.

To sort the records in the descending order, you need to explicitly use the descending keyword in the query. In order to sort the customer records in the descending order of name, you need to use the `descending` keyword in the query, as shown in the following code snippet:

```
IQueryable<Customer> q = from s in
db.customers
               where s.City == "Tokyo"
               orderby s.Name
descending
               select s;
```

Grouping the Data

The group clause enables you to group the results based on a key that you specify.

Consider the scenario of Newbay wherein you need to group the customers by the city name so that all the customers from a particular city, such as Tokyo or London, are arranged in individual groups. To group the customers according to their cities, you need to use the following code snippet:

```
var q = from s in db.customers
  group s by s.City;
```

The preceding query retrieves the records from the `Customer` table and groups the retrieved records on the basis of their city. The result takes the form of a list of lists. Each element in the list is an object that has a member named Key. In addition, it contains a list of elements that are grouped under that key. In case of the preceding code snippet, `s.City` is the key.

To execute the preceding query and to display the results, you need to use nested `foreach` loops, as shown in the following code snippet:

```
var result = "";
foreach (var group in q)
{
result += group.Key + " : ";
foreach (var entry in group)
{
result += entry.Name + " ";
}
result += " | ";
}
```

In the preceding code snippet, the outer `foreach` loop iterates through each group in the set of retrieved records. The inner `foreach` loop then iterates through the individual records in each group.

Joining the Data

In LINQ, you can use the join clause to associate the elements in one table with the elements in another table, where the two tables share a common attribute. The join clause enables you to retrieve the elements from different tables on the basis of a common attribute. While using the join operator to retrieve elements, you need to use the equals keyword to compare the data in the common attribute.

Consider the scenario of Newbay wherein you need to retrieve the records of all the male customers who have placed an order online. This requires you to retrieve information from the customers table and the orderdetails table. For this, you can use the join clause in the query, as shown in the following code snippet:

```
var customers = db.customers;
var orderdetails = db.orderdetails;
var list = (from s in customers
 join t in orderdetails on s.Name
 equals t.Customer
 where s.Gender == "M"
 select new {Customer=s.Name,
 Product=t.Product}).ToList();
var orders = "";
foreach (var order in list)
{
 orders += order.Customer + " : " +
 order.Product + " ";
}
```

The preceding code snippet retrieves the details of the customers from the customers and orderdetails tables by joining the tables using the join clause. The equals keyword is used to retrieve the records on the basis of equal values in the Name field of the Customer table and the Customer field of the orderdetails table, where both these fields contain the names of customers. The select clause of the query specifies that the query needs to retrieve the Name field from the customers table and the Product field from the orderdetails table. These fields are assigned the names, Customer and Product, respectively. The foreach loop traverses through the retrieved records and stores the details in the variable named orders.

Syntax of LINQ Queries

So far, you have seen how to write a simple LINQ query. LINQ queries in C# can be written by using two different syntaxes:

- Query syntax
- Method syntax

The syntax that you have been using until now is called the query syntax. The query syntax is the most commonly-used technique to write a LINQ query. This syntax is written in the form of a query expression. Such expressions are simple and easy to read.

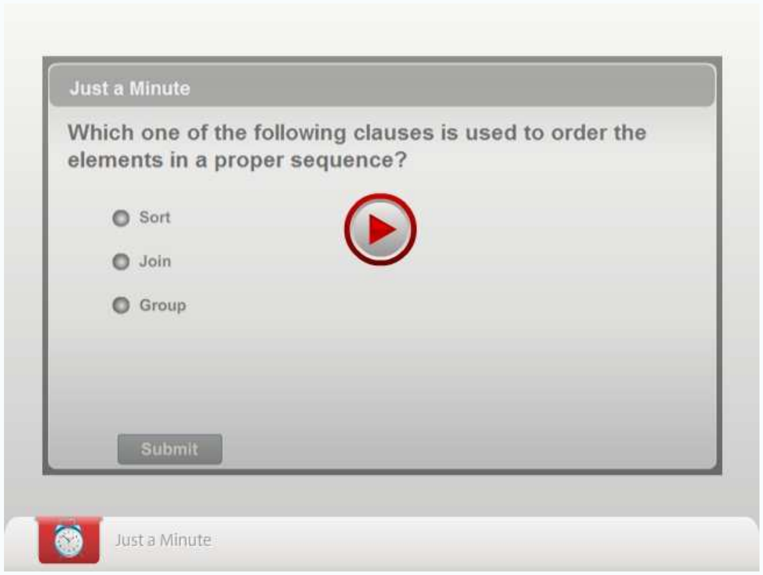
A query expression written by using the query syntax is translated into method calls at the time of compilation. The method calls finally invoke the standard query operators, such as Where, Select, and GroupBy. However, you can directly invoke the query operators by using the method syntax.

The method syntax specifies an exact order in which the methods in a query are to be called. For this, the method syntax uses the standard method invocation, using which the standard query operators, such as Where, Select, and GroupBy, are directly called. In addition, the method syntax is used in cases where a single numeric value is returned, such as Sum, Max, Min, and Average. These methods must always be called last in any query because they represent only a single value and cannot serve as the source for an additional query operation.

Let us consider an example of a LINQ query that uses the method syntax:

```
IEnumerable<Customer>          q          =
db.customers.Where(C                      =>
C.City=="Tokyo").OrderBy(C => C.Name);
 foreach (var cust in q)
 {
 names = names + " " + cust.Name;
 }
 ViewBag.Name = names;
```

In the preceding code snippet, the Where clause is expressed as a method call on the customers object. Further, the conditional expression, C => C.City=="Tokyo", is passed as an inline argument to the Where method. This inline expression is called a lambda expression.


Just a Minute

Which one of the following clauses is used to order the elements in a proper sequence?

- Sort
- Join
- Group

Activity 6.1: Using LINQ Queries

Accessing Data from Disparate Data Sources

LINQ provides a consistent programming model using which you can create the standard query syntax to query different types of data sources. However, different data sources accept queries in different formats. To solve this problem, LINQ provides the various LINQ providers. These providers help translate a LINQ query to a query that is specific to a particular data source.

Some LINQ providers are:

- ❑ LINQ to SQL
- ❑ LINQ to XML
- ❑ LINQ to Objects

LINQ to SQL

LINQ to SQL enables you to access SQL-complaint databases and makes data available as objects in an application. These objects in the application are mapped with the database objects to enable you to work with data.

The object model of the programming language and the data model of the relational database are mapped to each other. You can perform all the database operations, such as select, insert, update, and delete, on a database with the help of mapped objects.

On executing a LINQ to SQL query to retrieve the data from a database, the LINQ to SQL queries are converted into SQL queries. These queries are sent to the database where the database query execution engine executes the queries and returns the results. These results are converted back to objects by using LINQ to SQL, and then sent to the application.

During this process, LINQ to SQL uses an instance of the data context class of the application to cache the data retrieved from the database and send the data to the application. In addition, LINQ to SQL uses the instances of the model class to represent the tables of the database in the application.

The following example describes how to access the names of the customers of Newbay Store from the `Customer` table:

```
string names = "";
IQueryable<Customer> q = from s in
db.customers
 select s;
foreach (var cust in q)
{
names += " " + cust.Name;
}
```

The preceding code snippet will retrieve the names of the customers from the `Customers` table.

LINQ to XML

LINQ to XML is a LINQ-enabled, in-memory XML programming interface that enables you to work with XML from within the .NET Framework programming languages. XML is widely used to exchange data between various applications. LINQ to XML is a programming interface that enables you to access data stored in XML files. The following example shows the content of the XML file, `CustomersDetails.xml`:

```
<?xml version="1.0" encoding="utf-8"?>
<CustomersDetails>
<Customer>
<CustomerID>C001</CustomerID>
<Name>Svetlana</Name>
<City>London</City>
</Customer>
<Customer>
<CustomerID>C002</CustomerID>
<Name>Claire</Name>
<City >Paris</City>
</Customer>
<Customer>
<CustomerID>C003</CustomerID>
<Name>Cesar</Name>
<City>New York</City>
</Customer>
</CustomersDetails>
```

To access the data from `CustomersDetails.Xml` by using LINQ, you need to write the following code snippet:

```
string result = "";
XDocument xmlDoc = XDocument.Load("D:\
\CustomerDetails.xml");
var q = from c in xmlDoc.Descendants
("Customer")
 select (string)c.Element("CustomerID")
 + "-" +
 (string)c.Element("Name") + "-" +
 (string)c.Element("City");
foreach (string entry in q)
{
 result += entry + " | ";
}
```

The preceding code snippet will retrieve all the customer details from the XML file. `xmlDoc.Descendants` returns a collection of the descendant elements for the specified element in the order they are present in the XML file.

LINQ to Objects

LINQ to Objects refers to the use of LINQ queries with enumerable collections, such as List<T> or arrays. Traditionally, you had to write the `foreach` loops to retrieve data from enumerable collections. LINQ to Objects provides an easier way to retrieve data from object collections by using standard queries.

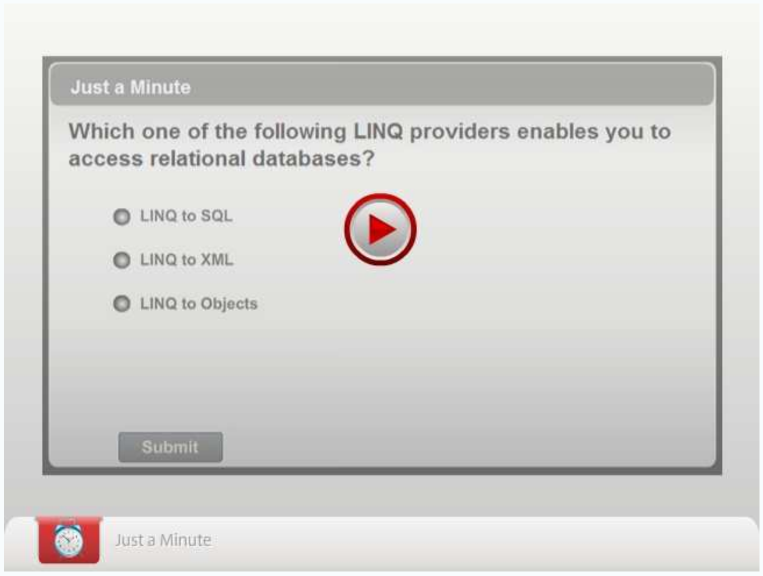
To use LINQ to query an object collection, you need to declare a range variable. The type of the range variable should match the type of the objects in the collection. For example, if you create an array that stores objects of the type, `Customer`, the range variable should also be of the type, `Customer`, as shown in the following code snippet:

```
var query = from Customer cust in arr
select cust;
```

In the preceding code snippet, the range variable, `cust`, of the type, `Customer`, is declared. By declaring a range variable, each item in the array is typecasted to a `Customer` object.

The following example describes how to access data from a string array that contains the names of students in a class:

```
string names = "";
string[] arr = new string[] { "Peter",
"Sam", "Philip"};
var query = from string name in arr
select name;
foreach (var n in query)
{
names = names + " " + n;
}
```



Which one of the following LINQ providers enables you to access relational databases?

- LINQ to SQL
- LINQ to XML
- LINQ to Objects

Activity 6.2: Using LINQ to Access XML

Summary

In this chapter, you learned that:

- ❑ LINQ offers a consistent programming model to query data from different data sources.
- ❑ All LINQ query operations consist of the following three distinct actions:
  - Obtain the data source
  - Create the query
  - Execute the query
- ❑ In addition to selecting data, a query can perform the following operations on data:
  - Filter the data
  - Sort the data
  - Group the data
  - Join the data
- ❑ LINQ queries in C# can be written by using two different syntaxes:
  - Query syntax
  - Method syntax
- ❑ LINQ provides a consistent programming model using which you can create the standard query syntax to query different types of data sources.
- ❑ Some LINQ providers are:
  - LINQ to SQL
  - LINQ to XML
  - LINQ to Objects

Chapter 7

Implementing a Consistent Look and Feel Across an MVC Web Application

You can create a website by creating several Web pages connected to each other. However, a professional website requires more than creating individual Web pages and connecting them. The look and feel of all the Web pages in a website should be consistent. All the pieces of information should appear in a standard, consistent format across all the Web pages. In addition, the Web pages should have a standard layout.
To maintain a standard layout in the MVC applicahtion, you can use layouts. In addition, you can enhance the appearance of a Web application by implementing styles on it.
This chapter discusses how to implement a consistent look and feel across all the Web pages by using layouts. In addition, it discusses how to style views.

Objectives

In this chapter, you will learn to:

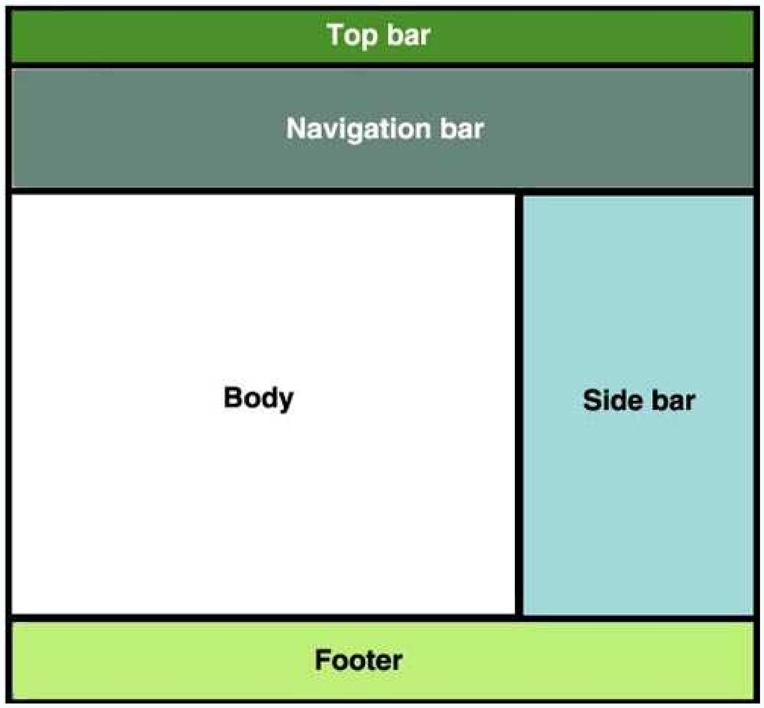
- ❑ Implement a consistent look and feel using layouts
- ❑ Style views

Implementing a Consistent Look and Feel Using Layouts

Consider the scenario of the NewBay store website where the Home page contains a navigation bar containing links to different views of the website. This navigation bar needs to appear on each view and allows easy navigation to other views. Moreover, the placement and appearance of this navigation bar should be consistent across all the views.
To implement the preceding requirement, you need to copy the navigation bar links on each view individually. You also need to ensure that the navigation bar is placed at the same position on all the views to maintain a consistent look and feel. In addition, any change in the navigation bar at a later stage will require changes to be made on each view of the website. This is a complex and time-consuming process. This process can be simplified and made efficient by using layouts.
A layout is a feature that enables you to specify a common site template to be used across the views of a website. Layout defines the structure and common ingredients of the views. You can use a layout with the selected views in a website or with all the views in a website.

Creating a Custom Layout

The process of creating a layout in an ASP.NET MVC application is similar to the process of creating a view. However, the only difference is that in case of an ASP.NET MVC application, you need to add a layout file in the **Shared** folder under the **Views** folder. As a convention, the name of the layout file is preceded by the underscore (_) character. Similar to a view, the extension of a layout file is **.cshtml**.
A layout file defines the visual components that are common across a set of views. In addition, it defines placeholders for the unique content associated with each individual page.
Consider the following figure that represents a layout.



A Layout

With reference to the preceding figure, suppose that the top bar, navigation bar, and footer contain some static content to be rendered in each view. However, the body and the side bar are specific to each view. This means that the content for the body is provided by the main content of each individual view, and the content for the sidebar is provided as the content for a specific section by each individual view.
To define the content for the body and section in a layout, you need to design a view, as shown in the following code snippet:

```
<p>This is the main content!</p>
@section Section1 {
This is Section1.
}
```

In the preceding code snippet, `This is the main content!` defines the main content of the view and the `@section` syntax is used to define the content for a section named `section1`.

To render the unique content associated with a view in the layout, the following methods can be used:

- The `@RenderBody()` method
- The `@RenderSection()` method

The @RenderBody() Method

The `@RenderBody()` method enables you to define a placeholder in the layout to insert the main content of the view. You can use only one `@RenderBody` method in a layout. For example, consider a simple layout view that contains the `@RenderBody` method:

```
<!DOCTYPE html>
<html lang="en">
  <head>
 <meta charset="utf-8" />
 <title>@ViewBag.Title - My ASP.NET MVC
 Application</title>
  </head>
  <body>
 <div id="body">
   @RenderBody()
   </div>
  </body>
</html>
```

In the preceding code, the call to the `@RenderBody()` method inside the `<div>` tag inserts the main content of the view.

The @RenderSection() Method

The `@RenderSection()` method in a layout enables you to render a section specified in the view. You can use multiple sections in a layout. For example, consider a simple layout view that contains the `@RenderSection()` method:

```
<!DOCTYPE html>
<html>
<head><title>@ViewBag.Title</title></
head>
<body>
<h1>@ViewBag.Title</h1>
<div id="main-content">@RenderBody()</
div>
<div>@RenderSection("Section1")</div>
</body>
</html>
```

The sections specified in a layout must be defined in all the views that use the layout.

However, if you do not want to use those sections in some of the views, you can use an overloaded method of the `@RenderSection()` method in your layout. The overloaded method of `@RenderSection()` uses the `required` attribute with the `false` value, as shown in the following code snippet:

```
<div>@RenderSection("Section1",
required: false)</div>
```

The preceding `@RenderSection()` method indicates that a view that uses this layout may supply content for a section named `Section1`. However, this is not mandatory. A view that uses this layout may not contain any section named `Section1`.

However, a better approach is to use some default content for the section that can be used in case a view does not provide content for the section. For this, you can use the following code in the layout:

```
<div>
@if (IsSectionDefined("Section1")) {
RenderSection("Section1");
}
else {
<span>This is the default section
content</span>
}
</div>
```

The preceding code snippet renders a section named`Section1` if it is defined in a view. However, if the section, `Section1`, is not defined in the view, the message, `This is the default section content`, is rendered in the view.

Task 7.1: Creating a Layout

Specifying a Layout for a View

You can specify a layout for either each individual view or for all the views on a website. A layout can be specified for a view by using the `Layout` property, as shown in the following code snippet:

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

In the preceding code snippet, the `MyLayout.cshtml` file is specified as the layout for the view that contains the preceding code snippet.

However, if you want to specify a particular layout for all the views of the website, then you can specify the layout to be used in the **_ViewStart.cshtml** file. This file is located inside the **Views** folder.

The code specified in the **_ViewStart.cshtml** file is executed before the code in any view, which is placed in the same directory or within a sub-directory. It applies the layout specified in the **_ViewStart.cshtml** file to all the views in the same directory/subdirectories. However, a view can override the `Layout` property by specifying a different value for the property.

When you create a default ASP.NET MVC project by using any template except the **Empty** template, an **_ViewStart.cshtml** file is already present in the **Views** folder. This file specifies the default layout to be used, as shown in the following code snippet:

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

In the preceding code snippet, the default layout is specified as `_Layout.cshtml`.

The _Layout.cshtml File

The **_Layout.cshtml** file is a default layout file that is included as part of the ASP.NET MVC project, which is created by using any template except the **Empty** template. It represents the layout that can be applied to the selected views or to all the views. This file is located in the **Shared** folder under the **Views** folder of the application.

The **_Layout.cshtml** file contains the basic structure of a view. If you create a project by using the **Internet** template, the **_Layout.cshtml** file contains the following code:

```
<!DOCTYPE html>
<html lang="en">
  <head>
 <meta charset="utf-8" />
 <title>@ViewBag.Title - My ASP.NET MVC
 Application</title>
 <link href="~/favicon.ico"
 rel="shortcut icon" type="image/x-
 icon" />
 <meta name="viewport"
content="width=device-width" />
@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("your logo
here", "Index", "Home")</p>
  </div>
  <div class="float-right">
    <section id="login">
      @Html.Partial("_LoginPartial")
    </section>
    <nav>
      <ul id="menu">
        <li>@Html.ActionLink("Home",
"Index", "Home")</li>
        <li>@Html.ActionLink("About",
"About", "Home")</li>
        <li>@Html.ActionLink
("Contact", "Contact", "Home")</li>
      </ul>
    </nav>
  </div>
  </div>
</header>
<div id="body">
  @RenderSection("featured", required:
false)
  <section class="content-wrapper
main-content clear-fix">
        @RenderSection("JavaScript",
required: false)
  @RenderBody()
  </section>
</div>
<footer>
  <div class="content-wrapper">
  <div class="float-left">
    <p>&copy; @DateTime.Now.Year - My
ASP.NET MVC Application</p>
  </div>
  </div>
</footer>
@Scripts.Render("~/bundles/jquery")
@RenderSection("scripts", required:
false)
  </body>
</html>
```

The preceding code snippet contains the `@RenderBody()` method and multiple `@RenderSection()` methods. These methods act as placeholders for the page-specific content appearing on each view. In addition, it contains the

`@Styles.Render("~/Content/css")` statement that renders the css files. In addition, it contains the `@Scripts.Render("~/bundles/modernizr")` statement that renders the script files in the modernizr library. Moreover, it contains the `Scripts.Render("~/bundles/jquery")` statement that renders the jQuery library.

You can modify the **_Layout.cshtml** file according to your need for developing an MVC application. For example, you can add/delete hyperlinks, change the layout color, and include/exclude sections.

Activity 7.1: Implementing a Consistent Look and Feel Using Layouts

Creating a Nested Layout

Layouts are used to standardize the look and feel of all the views on a website. However, sometimes, it is required that some of the views on a website should follow certain standard features, apart from the features provided by the global layout. In such situations, you can implement nested layouts.

Consider an example where a website uses a layout for displaying the navigation bar. Apart from the navigation bar provided by the layout, some of the views on the website require a sidebar to be displayed on the left side of the Web page. This can be implemented by using a nested layout.

A nested layout page refers to a layout that is derived from a parent layout. It is similar to the layout page and is responsible for defining the structure of a page. Similar to layouts, a nested layout page needs to call the `@RenderBody()` method once and can have multiple sections that can be rendered by using the `@RenderSection()` method. However, the `Layout` directive in a nested layout page contains the reference of the parent layout page. For example, consider the parent layout page named **_ParentLayout.cshtml** that contains the following code:

```
@{
  ViewBag.Title = "_ParentLayout";
}
<html xmlns="http://www.w3.org/1999/
xhtml">
<head runat="server">
  <title></title>

</head>
<body>
<div>
  <table style="width: 100%; font-
family: 'Monotype Corsiva'">
  <tr style="background-color:
darkblue">
    <td style="width: 200px">
   <img src="~/Images/logo1.jpg"
width="100" height="100"/></td>

    <td style="font-size: 50px; font-
weight: bold; color:white;
height:50px ;text-align:
center">NewBay Store&nbsp; </td>
    <td style="width: 200px">

    </td>
  </tr>
  <tr style="font-size: 22px;
background-color: lightblue; color:
black">
    <td colspan="3"
style="height:50px">
     @Html.ActionLink("Home", "Index",
"NewBayHome") |
        @Html.ActionLink("About",
"About", "NewBayHome")

    </td>
  </tr>
   <tr style="font-size: 18px;">

      @RenderBody()

  </tr>

  </table>
</div>
</body>
</html>
```

Now, you can create your nested layout page named **_SubLayout** by referring **_ParentLayout**, as shown in the following code:

```
@{
  Layout = "~/Views/Shared/
_ParentLayout.cshtml";
}
<table>
<tr>
<td style="background-color:
lightblue; color: black; height:
350px; width: 200px; vertical-align:
top">
</td>
<td style="vertical-align: top;">
@RenderBody()
```

```
  </td>
</tr>
</table>
```

In the preceding layout page code, the `_ParentLayout.cshtml` file is referred by using the `Layout` directive. In addition, the markup to create the left side bar is added and the `@RenderBody()` method is called to render the main content of the view.

Once you have created a nested layout, you can use the nested layout to define the structure of a view. To accomplish this task, you can use the following code snippet:

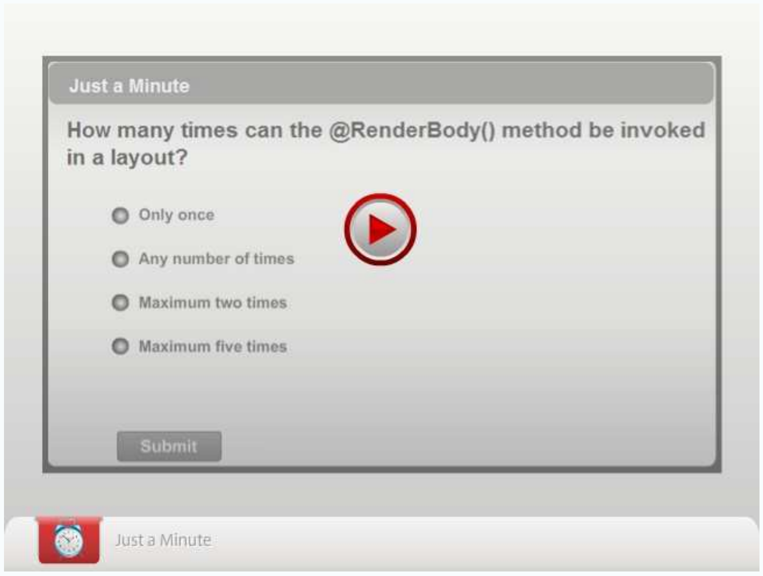
```
@{
  ViewBag.Title = "Index";
  Layout = "~/Views/Shared/
_SubLayout.cshtml";
}
Welcome to NewBay store
```

The preceding code snippet refers to the `_SubLayout.cshtml` file by using the `Layout` directive. The call to the `_SubLayout.cshtml` file, in turn, calls the `_ParentLayout.cshtml` file.

In case, the parent layout of a nested layout renders a section, you will need to define additional markup in your nested page to define and render the contents of the section. For example, suppose the parent layout defines a section named featured and the target view provides the content for this section in a section named featuredcontent. In this case, you need to include the following markup in your nested layout page:

```
  @section featured{
      @RenderSection("featuredcontent",
required:false)
  }
```

The `@section` directive defines the content for the featured section that is required by the parent layout. `@RenderSection()` method renders the content of the featuredcontent section that is defined in the target view.



How many times can the @RenderBody() method be invoked in a layout?

- Only once
- Any number of times
- Maximum two times
- Maximum five times

Activity 7.2: Implementing a Nested Layouts

Styling Views

The look and feel of a view depends upon the appearance and arrangement of the HTML elements contained in it. Therefore, you need to format the contents of a view to make it look attractive. Formatting each HTML helper individually is a tedious task. In addition, it fails to bring consistency in the appearance of all the views on the Web application. Styles provide a solution to this problem by enabling a programmer to apply consistent formatting across the entire Web application. 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.

Let us see how to use styles for applying a consistent look and feel across various views in a Web application.

Defining Styles in an MVC Application

You can define the styles to be used in an MVC application in a **.css** file.

Consider the following style definitions contained in a file named **main.css**:

```
h1
{
font-weight: bold;
font-size: 24px;
color: green;
font-family: Arial;
}
```

The preceding styles will be applied to all the `<h1>` tags in any Web page that references the style sheet, **main.css**. In the preceding style definition, `h1` is the CSS selector that specifies the elements on which the style needs to be applied.

Now, consider the following code snippet in a CSS file to display the content inside the `H1` elements in red, and the text inside the `<H2>` element as italic:

```
H1 {color:red;}
H2 {font-style: italic;}
```

In the preceding code snippet, `H1` and `H2` are the CSS selectors.

If you want to set the same attributes for more than one element, you can combine selectors as a single group. For example, to display the text of both the`H1` and `H2` elements in red, you can type the following code snippet in a CSS file:

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

In the preceding code snippet, all text displayed using the `H1` and `H2` elements get styled with the red color.

You have seen how you can use the element name as the CSS selector for styling a Web page. Apart from using an element name, you can also use other types of CSS selectors. Some of these are:

- **ID selector**: CSS styles can be applied to an element with a specific ID using the ID selector. The ID selector is used to identify an element that you need to style differently from the rest of the page. Each element on a Web page can have a unique ID. An ID selector consists of a hash symbol (#) followed by the element ID.
  Consider the following code snippet that specifies an id for the `<p>` tag:
  ```
  <p id="pname">Barbie doll</p>
  ```
  Now, consider the following code snippet to define the style specifications to be applied to the element, whose ID is `pname`:
  ```
  #pname
  {
  color:green;
  font-size:20pt;
  font-weight:bold;
  }
  ```
  The preceding code snippet selects the element with the id, `pname`, and styles it as per the specified attribute values.
- **Class selector**: The class selector is used to specify styles for a group of elements. Unlike an id selector, a class selector is used more often when there is a need to apply a style on several elements in a view. For example, suppose you have added three paragraphs on a Web page. Now, you want to specify same formatting for the first two paragraphs and different formatting for the third paragraph. You can accomplish this task by using the class selector. Unlike unique ids, multiple elements can belong to the same class. Hence, you can set a particular style for multiple elements with the same class.
  The class selector uses the HTML class attribute and is preceded with ".". Consider the following markup:
  ```
  <P class="red">This paragraph
  will appear in red color</P>
  <P class="red">This paragraph
  will appear in red color</P>
  <P class="green">This paragraph
  will appear in green color</P>
  ```
  Now, consider the following code snippet to style the first and second paragraphs of the Web page:
  ```
  .red
  {
  color:red;
  }
  .green
  {
  color:green;
  }
  ```
  In the preceding code snippet, the text color of the first two paragraphs will be red, and that of the third paragraph will be green.
  Further, consider the following code snippet to specify that only specific HTML elements should be affected by a class:
  ```
  p.center {text-align:center;}
  p.left {text-align:left;}
  ```
  In the preceding code, all the `<P>` elements with the class,`center`, will be center-aligned, and all `<P>` elements with the class, `left`, will be left aligned. The other `<P>` elements, which are used without any class inside a document, will not be affected. Similarly, all other types of elements with the class, `center` or `left`, will not be affected.
- **Universal selector**: The universal selector is used to select and specify styles for all the elements in a particular area on the Web page or the whole Web page. The universal selector is defined by using the `*` symbol. For example, to specify styles for all the elements inside the `<div>` tag on the Web page, you can use the universal selector. Consider the following code snippet that specifies the `<div>` tag containing some elements:
  ```
  <div class="products">
  <p id="product1">Mobile</p>
  <p id="product2">Laptop</p>
  </div>
  ```

You can apply styles to all the elements in the preceding <div> tag by using the universal selector, as shown in the following code snippet:

```
div.products *
{
color:pink;
}
```

Importing Styles in an MVC Application

Once created, the .css file needs to be linked with HTML documents on which the styles are to be implemented. This can be done by using the `<link>` tag.

The `<link>` tag is intended to link together the Web page with the style sheet. It is added to the `<head>` tag of the Web page. For example, consider the following code snippet:

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

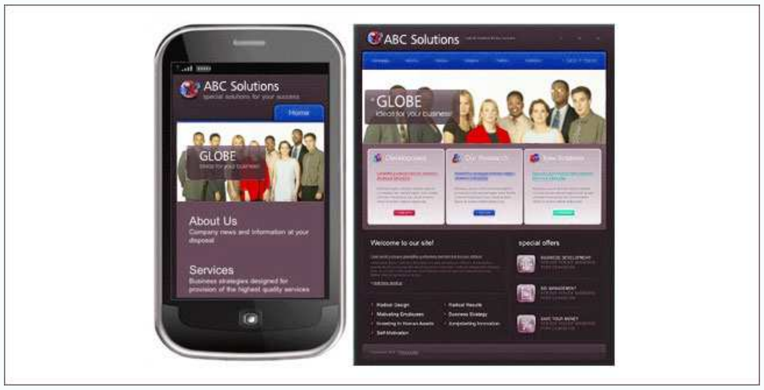
In the preceding code snippet, the main.css file is referred to by using the `href` attribute of the `<link>` tag. The attribute, `rel="stylesheet"`, specifies that the document will use a style sheet, and the attribute, `type="text/css"`, specifies that the MIME type of the linked document is text/css.

Creating an Adaptive User Interface

Now-a-days, Web applications can be accessed from a variety of devices, such as tabs, laptops, or mobile phones having different page widths and browser capabilities. Because the size of the screen in different devices may vary, the page width and the visual appearance of the Web application may change accordingly. This can hamper the functionality of the website and the user's experience.

By implementing adaptive rendering, the MVC model allows you to handle such a situation. Adaptive rendering not only refers to automatically scaling down the page, but also to redrawing the page according to a device screen. This enables the creation of an adaptive user interface.

For example, you must have seen a website on a desktop that displays a navigation bar as a horizontal list containing links. However, when you visit the same website on a mobile device, the navigation bar is rendered vertically to present the same functionality on a smaller device. This is possible by using adaptive rendering. For example, consider the following figure that depicts adaptive rendering:



An Example of Adaptive Rendering

The preceding figure depicts how a Web page adapts its look and feel as per the available screen size.

To implement adaptive rendering and creating an adaptive user interface, the following features are used:

- ❑ The viewport meta tag
- ❑ CSS media queries

The Viewport Meta Tag

Now-a-days, the browsers of mobile devices can easily render Web pages and scale them so that they properly fit inside the phone browser's visible screen area. The Web pages are rendered in a virtual window known as a viewport, which is usually wider than the actual width of the mobile device. Consider a mobile device that has a viewport of 980px. You can think of it as pretending to be a desktop browser with a width of 980px. Web pages are rendered in a viewport of width 980px, and then scaled down to fit into the actual dimensions of the device screen. Therefore, users need to zoom in on Web pages to view the content.

However, if you want to control the size and scaling of the Web page, you can use the viewport `<meta>` tag in the head section of the view.

You can control the size of the viewport by using the viewport `<meta>` tag, as shown in the following code snippet:

```
<meta name="viewport"
content="width=220">
```

In the preceding code snippet, the `width` property is set to `220` pixels, which sets the width of the viewport to 220 pixels. Web pages will now be rendered on a canvas of width 220 pixels, and then scaled properly inside the available screen area.

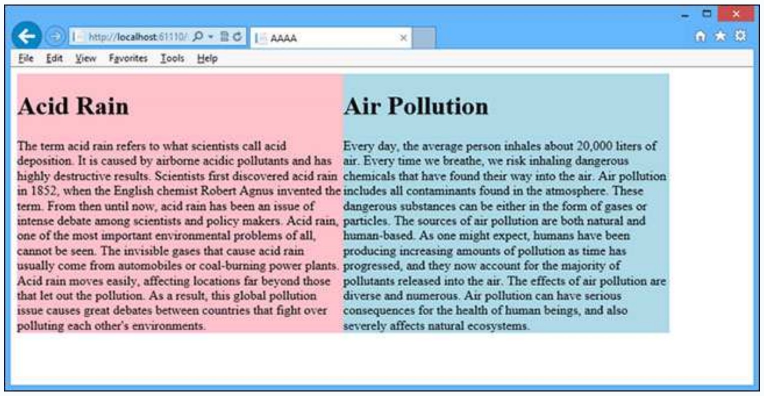
However, a better practice is to set the width of the viewport to the native screen size of the browser by using the `device-width` keyword. This helps you to create the content that you can adjust according to the device specifications. For example, consider the following code snippet:

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

In the preceding code snippet, the `width` property is set to the `device-width` value, which specifies the width of the device screen. When you set the width of the viewport to the width of the device, no scaling is required.

CSS Media Queries

CSS media queries allow you to apply different styles in your application to support different browsers and devices with different sizes. CSS media queries are special selectors that begin with `@media`. These queries help in applying conditional CSS styles based on the device conditions or browser capabilities. For example, consider that you need to design a Web page, as shown in the following figure.



Acid Rain

The term acid rain refers to what scientists call acid deposition. It is caused by airborne acidic pollutants and has highly destructive results. Scientists first discovered acid rain in 1852, when the English chemist Robert Agnus invented the term. From then until now, acid rain has been an issue of intense debate among scientists and policy makers. Acid rain, one of the most important environmental problems of all, cannot be seen. The invisible gases that cause acid rain usually come from automobiles or coal-burning power plants. Acid rain moves easily, affecting locations far beyond those that let out the pollution. As a result, this global pollution issue causes great debates between countries that fight over polluting each other's environments.

Air Pollution

Every day, the average person inhales about 20,000 liters of air. Every time we breathe, we risk inhaling dangerous chemicals that have found their way into the air. Air pollution includes all contaminants found in the atmosphere. These dangerous substances can be either in the form of gases or particles. The sources of air pollution are both natural and human-based. As one might expect, humans have been producing increasing amounts of pollution as time has progressed, and they now account for the majority of pollutants released into the air. The effects of air pollution are diverse and numerous. Air pollution can have serious consequences for the health of human beings, and also severely affects natural ecosystems.

A Web Page

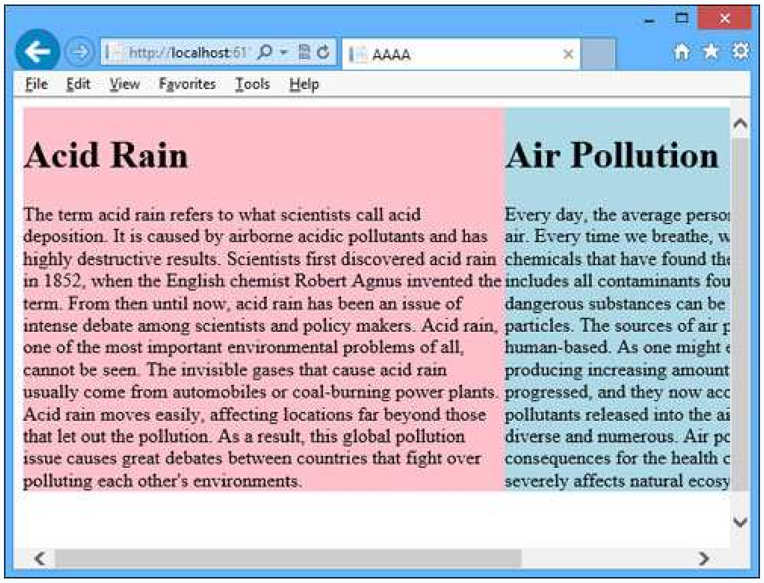
For designing the preceding Web page, you have used the following markup:

```
<html>
<head>
  <title>AAAA</title>
<link href="~/content/main.css"
rel="stylesheet" type="text/css" />
</head>
@{
  ViewBag.Title = "Index";
  Layout = null;
}
<body>
<div id="container">
<div id="article1">
  <h1> Acid Rain</h1> The term acid
rain refers to what scientists call
acid deposition. It is caused by
airborne acidic pollutants and has
highly destructive results.
Scientists first discovered acid rain
in 1852, when the English chemist
Robert Agnus invented the term. From
then until now, acid rain has been an
issue of intense debate among
scientists and policy makers.
Acid rain, one of the most important
environmental problems of all, cannot
be seen. The invisible gases that
cause acid rain usually come from
automobiles or coal-burning power
plants.
Acid rain moves easily, affecting
locations far beyond those that let
out the pollution. As a result, this
global pollution issue causes great
debates between countries that fight
over polluting each other’s
environments.
</div>
<div id="article2">
  <h1>Air Pollution</h1> Every day,
the average person inhales about
20,000 liters of air. Every time we
breathe, we risk inhaling dangerous
chemicals that have found their way
into the air.
Air pollution includes all
contaminants found in the atmosphere.
These dangerous substances can be
either in the form of gases or
particles.
The sources of air pollution are both
natural and human-based. As one might
expect, humans have been producing
increasing amounts of pollution as
time has progressed, and they now
account for the majority of pollutants
released into the air.
The effects of air pollution are
diverse and numerous. Air pollution
can have serious consequences for the
health of human beings, and also
severely affects natural ecosystems.
</div>
</div>
 </body>
</html>
```

The preceding markup uses the **main.css** file, which contains the following code:

```
#container {
  float: left;
  width: 800px;
}
#article1 {
  float: left;
  width: 400px;
  background: pink;
}
#article2 {
  float: right;
  width: 400px;
  background: lightblue;
}
```

When you view the preceding Web page, the output is displayed correctly. However, when you reduce the dimensions of the window, the look and feel of the Web page is hampered. Here, the Web page is displayed, as shown in the following figure.



Acid Rain

The term acid rain refers to what scientists call acid
deposition. It is caused by airborne acidic pollutants and has
highly destructive results. Scientists first discovered acid rain
in 1852, when the English chemist Robert Agnus invented the
term. From then until now, acid rain has been an issue of
intense debate among scientists and policy makers. Acid rain,
one of the most important environmental problems of all,
cannot be seen. The invisible gases that cause acid rain
usually come from automobiles or coal-burning power plants.
Acid rain moves easily, affecting locations far beyond those
that let out the pollution. As a result, this global pollution
issue causes great debates between countries that fight over
polluting each other's environments.

Air Pollution

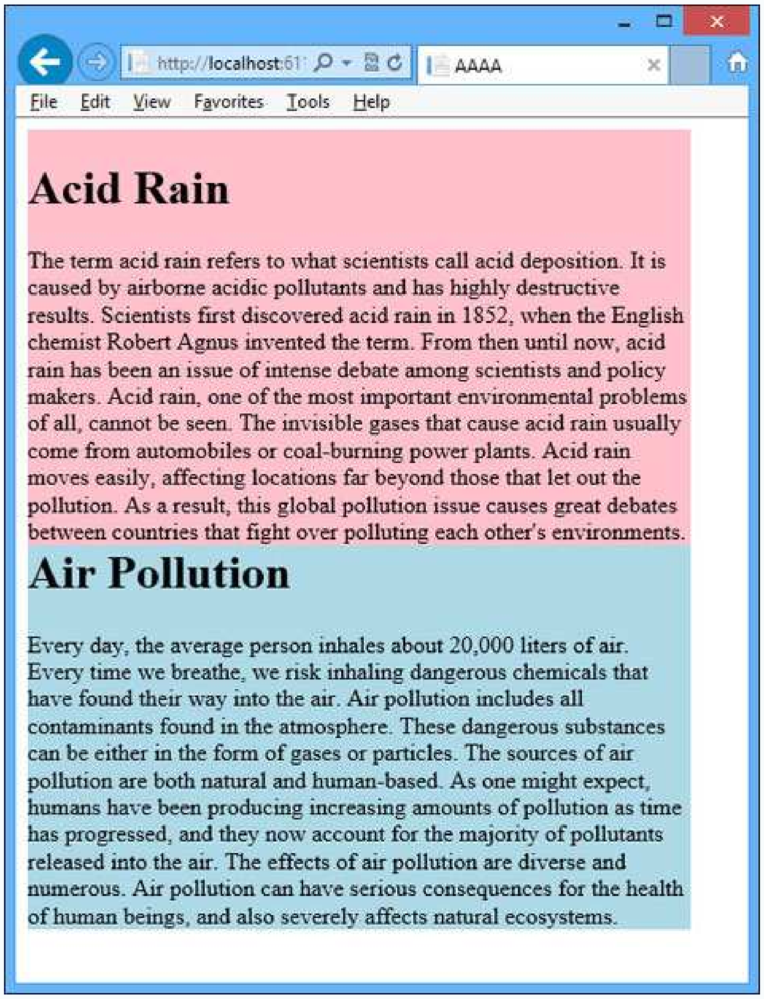
Every day, the average perso
air. Every time we breathe, w
chemicals that have found th
includes all contaminants fou
dangerous substances can be
particles. The sources of air p
human-based. As one might e
producing increasing amount
progressed, and they now acc
pollutants released into the ai
diverse and numerous. Air pc
consequences for the health c
severely affects natural ecosy

A Web Page After Reducing Dimiensions

In such a case, you can use media queries to add styles for reduced dimensions in the css file, as shown in the following code snippet:

```
@media screen and (max-width:800px) {
  #container {
float: left;
width: 600px;
  }
  #article1 {
float: left;
width: 600px;
  }
  #article2 {
float: none;
width: 600px;
  }
}
@media screen and (max-width:600px) {
  #container {
float: left;
width: 450px;
  }
  #article1 {
float: left;
width: 450px;
  }
  #article2 {
float: none;
width: 450px;
  }
}
```

The preceding code snippet defines two media queries by using the `@media` syntax. The first one defines the styles for the screens that have maximum width of 800px, and the second one defines the styles for the screens that have maximum width of 600px by using the `max-width` property. When you reduce the width of your browser window, the Web page is rendered according to the styles specified for reduced screen width, as shown in the following figure.



Acid Rain

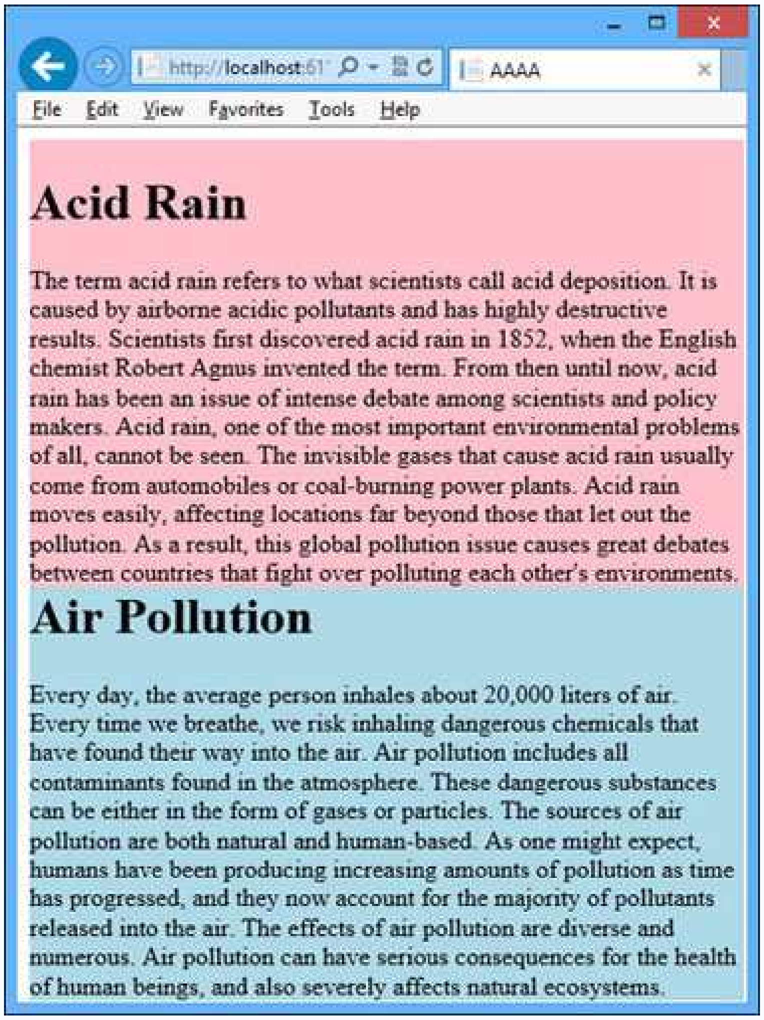
The term acid rain refers to what scientists call acid deposition. It is caused by airborne acidic pollutants and has highly destructive results. Scientists first discovered acid rain in 1852, when the English chemist Robert Agnus invented the term. From then until now, acid rain has been an issue of intense debate among scientists and policy makers. Acid rain, one of the most important environmental problems of all, cannot be seen. The invisible gases that cause acid rain usually come from automobiles or coal-burning power plants. Acid rain moves easily, affecting locations far beyond those that let out the pollution. As a result, this global pollution issue causes great debates between countries that fight over polluting each other's environments.

Air Pollution

Every day, the average person inhales about 20,000 liters of air. Every time we breathe, we risk inhaling dangerous chemicals that have found their way into the air. Air pollution includes all contaminants found in the atmosphere. These dangerous substances can be either in the form of gases or particles. The sources of air pollution are both natural and human-based. As one might expect, humans have been producing increasing amounts of pollution as time has progressed, and they now account for the majority of pollutants released into the air. The effects of air pollution are diverse and numerous. Air pollution can have serious consequences for the health of human beings, and also severely affects natural ecosystems.

The Web Page After Reducing the Dimensions of the Window

When you reduce the dimensions of the Window further, the Web page is rendered, as shown in the following figure.



Acid Rain

The term acid rain refers to what scientists call acid deposition. It is caused by airborne acidic pollutants and has highly destructive results. Scientists first discovered acid rain in 1852, when the English chemist Robert Agnus invented the term. From then until now, acid rain has been an issue of intense debate among scientists and policy makers. Acid rain, one of the most important environmental problems of all, cannot be seen. The invisible gases that cause acid rain usually come from automobiles or coal-burning power plants. Acid rain moves easily, affecting locations far beyond those that let out the pollution. As a result, this global pollution issue causes great debates between countries that fight over polluting each other's environments.

Air Pollution

Every day, the average person inhales about 20,000 liters of air. Every time we breathe, we risk inhaling dangerous chemicals that have found their way into the air. Air pollution includes all contaminants found in the atmosphere. These dangerous substances can be either in the form of gases or particles. The sources of air pollution are both natural and human-based. As one might expect, humans have been producing increasing amounts of pollution as time has progressed, and they now account for the majority of pollutants released into the air. The effects of air pollution are diverse and numerous. Air pollution can have serious consequences for the health of human beings, and also severely affects natural ecosystems.

The Web Page After Further Reducing the Dimension of a Window

In addition to `max-width`, the following table describes some more properties that can be used with a media query.

Properties	***Description***
`Width`	*It specifies the width of the display area that represents the browser window.*
`Height`	*It specifies the height of the display area.*
`device-width`	*It specifies the width of the device screen.*
`device-height`	*It specifies the height of the device screen.*
`aspect-ratio`	*It specifies the ratio of the width and height properties.*

The Media Query Properties

Detecting Browser Capabilities

Web developers use browser detection techniques to ensure that the websites display properly when viewed with specific browsers. To detect the capabilities of a browser in an MVC Web application, the `HttpBrowserCapabilities` object is used, which provides the information about the browser during an HTTP request. The information includes the type and level of support the browser offers. To expose this information, the `HttpBrowserCapabilities` object uses strongly typed properties and a generic name-value dictionary. Some properties of the `HttpBrowserCapabilities` object are:

Properties	***Description***
`Browser`	*Gets the browser string, if any, sent by the browser.*
`Browsers`	*Gets ArrayList of the browsers in the Capabilities dictionary.*
`Cookies`	*Gets a value that specifies whether the browser supports cookies.*
`Version`	*Gets the full version number of the browser as a string.*
`Tables`	*Gets a value that specifies whether the browser supports HTML tables.*

The Properties of an HttpBrowserCapabilities Object

The capabilities of a browser are determined through the `Browser` property of the ASP.NET intrinsic `Request` object.

For example, consider the following code snippet:

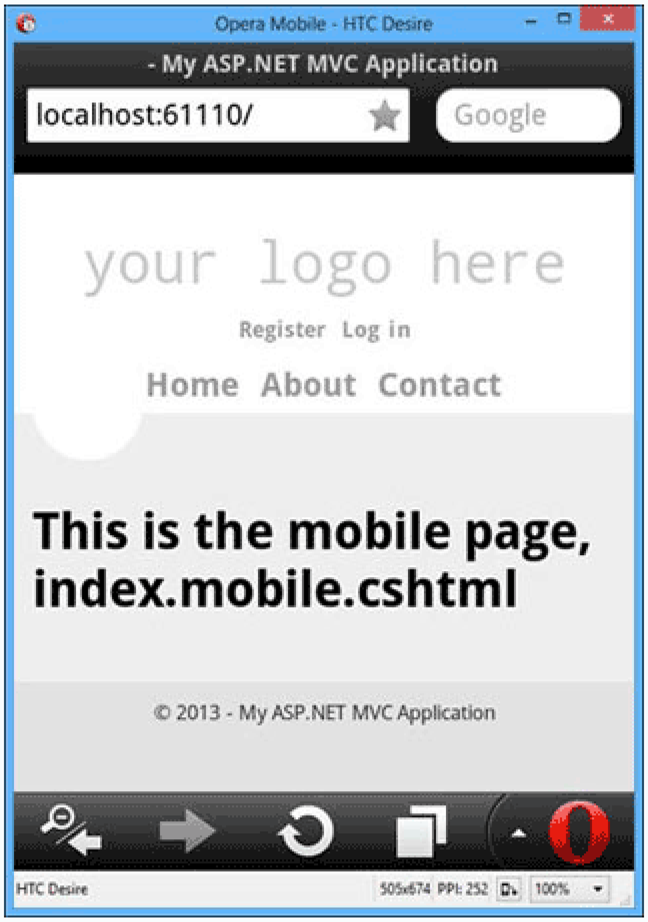
```
if  ((Request.Browser.Browser  ==  "IE")
&&       ((float)System.Convert.ToSingle
(Request.Browser.Version) < 7.0))
   {
   ViewBag.Message = "Please upgrade
 your browser";
   }
```

The preceding code snippet uses the `Browser` property of the ASP.NET intrinsic `Request` object and checks whether the browser is IE and the version is below 7. If yes, the message, `Please upgrade your browser`, is rendered on the view by using the `ViewBag` object.

Creating Mobile Specific Views

At times, modifying only css to manage the look and feel of a view according to the device is not sufficient. Here, you need to create views specific for the device. ASP.NET MVC enables you to globally override views for mobile devices. Upon receiving a request from a mobile browser, an ASP.NET MVC application first looks for views with the naming convention of `[view].mobile.cshtml`.

If a view matches the naming convention, ASP.NET MVC renders it, as shown in the following figure.

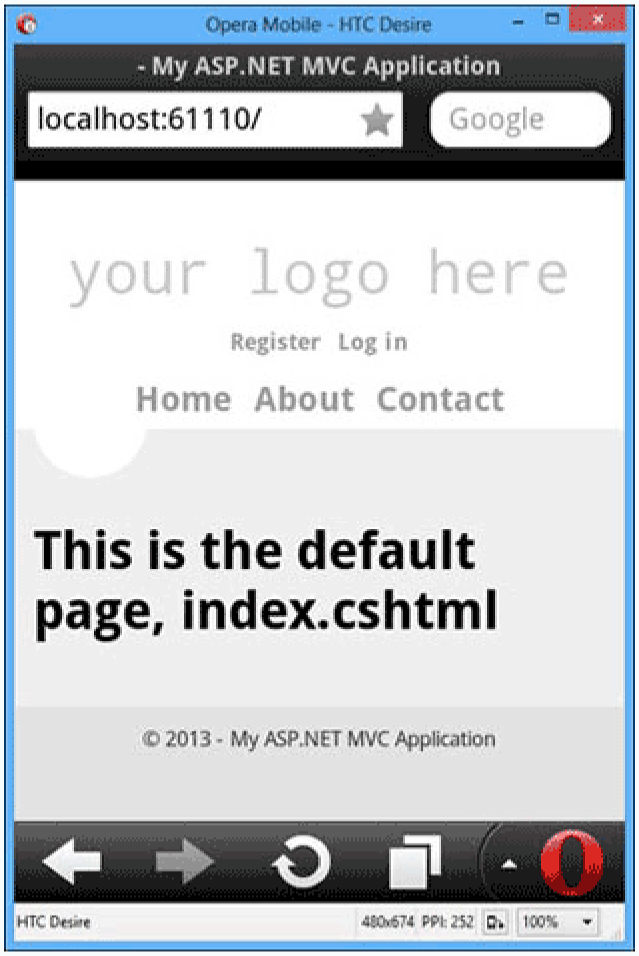


This is the mobile page, index.mobile.cshtml

© 2013 - My ASP.NET MVC Application

The Mobile Specific View

Otherwise, the standard view (`[view].cshtml`) is rendered, as shown in the following figure.



This is the default page, index.cshtml

© 2013 - My ASP.NET MVC Application

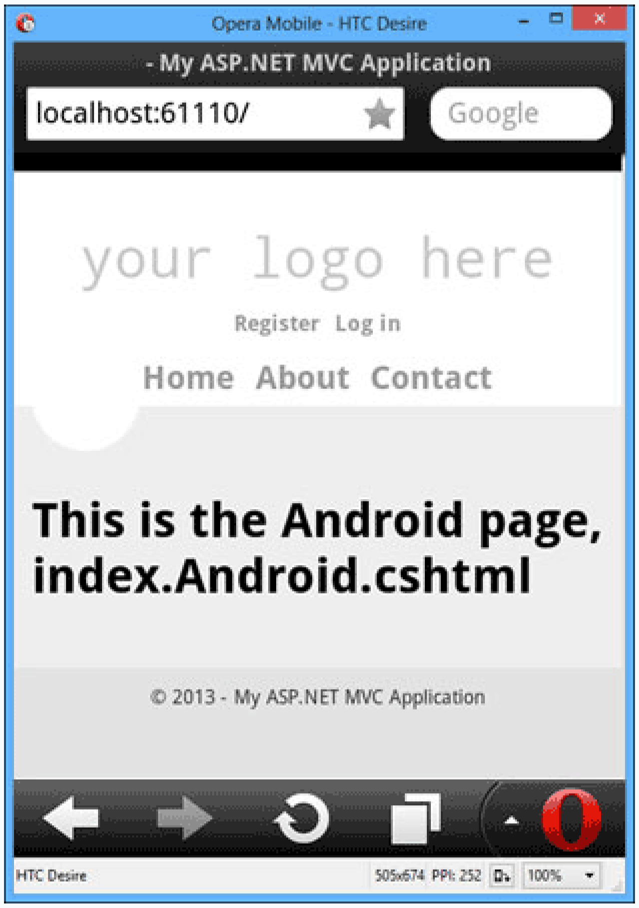
The Standard View

MVC automatically renders the mobile specific view when it receives a request from a mobile browser. However, now-a-days, a large number of mobile platforms are available in the market. These mobile platforms use their own browsers. Therefore, a Web page that appears properly on one mobile platform may appear differently on another platform. To solve this problem and render platform-specific views, you can define display modes.

You need to define display modes in the `Application_Start()` method of the Global.asax file. For example, consider that your Web application includes a layout that is specific to a browser for an Android mobile. In this situation, you can create browser-specific views for that browser. However, before creating browser-specific views, you need to create a new display mode named Android and write the code for examining the `UserAgent` string received from a browser along with a request. The `UserAgent` string helps in identifying the type of browser. Consider the following code snippet:

```
DisplayModeProvider.Instance.Modes.Ins
ert(0, new DefaultDisplayMode
("Android")
{
ContextCondition = (context =>
context.GetOverriddenUserAgent
().IndexOf
("Android",
StringComparison.OrdinalIgnoreCase) >=
0)
});
```

In the preceding code snippet, a new display mode named `Android` is defined. This mode will be matched against each incoming request. If the incoming request is from an Android browser, ASP.NET MVC will render the view, whose name contains the suffix, `Android`, as shown in the following figure.



This is the Android page, index.Android.cshtml

© 2013 - My ASP.NET MVC Application

The Android Specific View

Which one of the following CSS selectors is prefixed with #?

- Class selector
- ID selector
- Universal selector

Summary

In this chapter, you learned that:

- ❑ A layout is a feature that enables you to specify a common site template to be used across the views of a website.
- ❑ As a convention, the name of the layout file is preceded by the underscore (_) character.
- ❑ A layout file defines the visual components that are common across a set of views.
- ❑ To render the unique content associated with a view in the layout, the following methods can be used:
  - The `@RenderBody()` method
  - The `@RenderSection()` method
- ❑ The `@RenderBody()` method enables you to define a placeholder in the layout to insert the main content of the view.
- ❑ The `@RenderSection()` method in a layout enables you to render a section specified in the view.
- ❑ The **_Layout.cshtml** file is a default layout file that is included as part of the ASP.NET MVC project, which is created by using any template except the **Empty** template.
- ❑ Layouts are used to standardize the look and feel of all the views on a website.
- ❑ A nested layout page refers to a layout that is derived from a parent layout.
- ❑ You can define the styles to be used in an MVC application in a .css file.
- ❑ An ID selector consists of a hash symbol (#) followed by the element ID.
- ❑ The class selector uses the HTML class attribute and is preceded with ".".
- ❑ The universal selector is defined by using the * symbol.
- ❑ The `<link>` tag is intended to link together the Web page with the style sheet.
- ❑ To implement adaptive rendering and creating an adaptive user interface, the following two-browser features are used:
  - The viewport meta tag
  - CSS media queries
- ❑ The Web pages are rendered in a virtual window known as a viewport, which is usually wider than the actual width of the mobile device.
- ❑ CSS media squeries allow you to apply different styles in your application to support different browsers and devices with different sizes.
- ❑ To detect the capabilities of a browser in an MVC Web application, the `HttpBrowserCapabilities` object is used,

which provides the information about the browser
during an HTTP request.

Chapter 8

Enhancing Web Applications Using JavaScript

The success of an application largely depends on how easily and effectively it allows a user to interact with it. For an application to be successful, its interface should be responsive and easy-to-use. Such an interface can be designed by adding JavaScript to a Web application. You can use various predefined JavaScript libraries, such as jQuery, to make the application interactive.

The response time of an application is an important factor that impacts the user experience. You can speed up the response time of an application by implementing partial page updates using *Asynchronous JavaScript and XML (AJAX)*.

This chapter provides an overview of how an application is made responsive using JavaScript. In addition, it discusses how the partial page updates are implemented using AJAX.

Objectives

In this chapter, you will learn to:

- ❑ Make a Web application responsive by using JavaScript
- ❑ Implement the partial page updates using AJAX

Making a Web Application Responsive by Using JavaScript

While developing a dynamic, interactive, and rich Web application, you need to ensure that the following functionalities have been implemented:

- ❑ The UI is easy to use.
- ❑ Users get quick response for their requests.
- ❑ The application runs on all the browsers.

By using client-side scripts, the preceding functionalities can be implemented. The client-side scripts enable you to develop dynamic Web pages that can respond to the user input without interacting with a Web server. This helps in reducing the network traffic because interaction with a Web server is not required in providing a dynamic response to the user interaction. This, in turn, helps in improving the response time of a Web application because the processing happens at the client side and round-trips to the server are not required.

JavaScript is a client-side scripting language that allows you to develop dynamic, interactive, and rich Web applications. JavaScript is an easy-to-learn scripting language. Therefore, programmers are able to quickly incorporate JavaScript functionalities into a Web page. However, to develop a Web application that provides quick user interactions using JavaScript, you need to write a good amount of code because the JavaScript functions vary from browser to browser. Thus, to simplify the task of developing such Web applications with fewer lines of code, JavaScript provides various libraries, such as jQuery and jQuery UI.

Let us discuss some concepts related to JavaScript and JavaScript libraries.

Unobtrusive JavaScript

In the earlier days of Web application development, it was a common practice to include JavaScript code and HTML inside the same file. It was even normal to put the JavaScript code inside an HTML element as the value of an attribute. For example, the `onclick` event handler of a div element was represented in the markup, as shown in the following code snippet:

```
<div onclick="javascript:alert
('click');">Testing, testing</div>
```

The preceding code snippet was used because there was no easier approach for handling click events. However, the embedded JavaScript code was messy. To overcome this problem, Unobtrusive JavaScript can be used to separate the JavaScript code from HTML.

Unobtrusive JavaScript is a general approach to implement JavaScript in Web pages. In this approach, the JavaScript code is separated from the HTML markup. In an ASP.NET MVC application, the JavaScript code is separated from the markup by storing the JavaScript code in a `.js` file in the **Script** folder of the root directory of an application.

To refer to the .js file in a view, you need to use the `<script>` tag, as shown in the following code snippet:

```
<script src= "@Url.Content("~/Scripts/
Myscript.js")"                type="text/
javascript"></script>
```

NOTE *In the preceding code snippet, the Url.Content() helper method has been used to convert a relative URL to an absolute URL.*

Keeping all scripts in a separately downloadable file can give your site a performance boost because the browser can cache the script files locally. In addition, Unobtrusive JavaScript allows you to use a strategy known as progressive enhancement for your site.

Progressive enhancement is a technique that allows you to deliver the content progressively. It uses a layered approach to deliver the content that allows everyone to access the basic content and functionality of a view, while providing an enhanced version of the view to users whose device or browser supports features, such as scripts and style sheets.

The **_Layout.cshtml** file that is included with most ASP.NET MVC project templates includes a section named scripts, which specifies the placeholder for the scripts in a view. Therefore, a view that needs to refer to a custom script can include the `<script>` tag for the custom JavaScript file within the scripts section definition, as shown in the following code snippet:

```
@section scripts
{
<script src= '@Url.Content("~/Scripts/
Myscript.js")' type='text/
javascript'></script>
}
```

Introduction to JavaScript Libraries

A JavaScript library is a set of prewritten JavaScript code, which helps in developing a JavaScript-based application easily.

In most default ASP.NET MVC project templates, such libraries are present in the **Scripts** folder, which is present in the root directory of the application. One of the commonly used JavaScript libraries present in this directory is the jQuery library. jQuery is a cross-browser JavaScript library that helps you easily perform various tasks, such as DOM traversal, event handling, and animating elements. It provides an easy-to-use API that allows you to quickly provide complex functionality by writing very little code. This API works across a variety of browsers and eliminates the need to write different versions of code for different browsers.

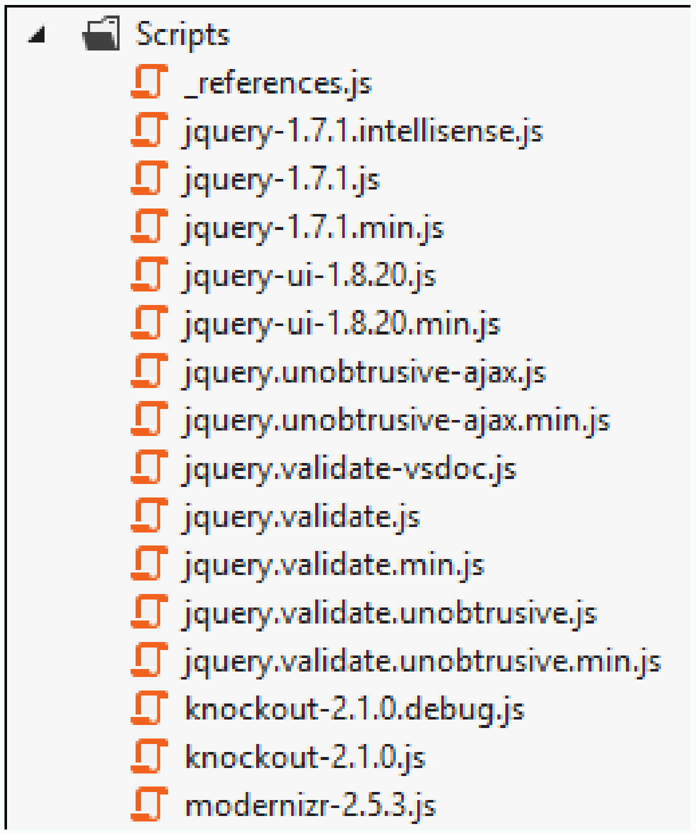
Apart from the jQuery library, the Scripts folder also contains jQuery plugins, such as jQuery UI and jQuery Validation. These plugins add additional capabilities to the core jQuery library. Additional plug-ins such as jQuery Mobile can also be downloaded and used in an application.

jQuery UI is an organised set of user interface interactions, effects, widgets, and themes built on top of the jQuery JavaScript library. Thus, you can build highly interactive Web applications using jQuery UI.

jQuery validation makes the client-side form validation trivial. It is bundled with a set of validation methods, such as email and URL validation that helps you to easily implement client-side validation.

jQuery Mobile is the mobile version of the jQuery library. It helps you to develop applications for mobile devices.

In addition to the .js files for jQuery and its plugins, the **Scripts** folder contains several other files, as shown in the following figure.



```
Scripts
    _references.js
    jquery-1.7.1.intellisense.js
    jquery-1.7.1.js
    jquery-1.7.1.min.js
    jquery-ui-1.8.20.js
    jquery-ui-1.8.20.min.js
    jquery.unobtrusive-ajax.js
    jquery.unobtrusive-ajax.min.js
    jquery.validate-vsdoc.js
    jquery.validate.js
    jquery.validate.min.js
    jquery.validate.unobtrusive.js
    jquery.validate.unobtrusive.min.js
    knockout-2.1.0.debug.js
    knockout-2.1.0.js
    modernizr-2.5.3.js
```

The Different Files in the Scripts Folder

In the preceding figure, you will find the files containing **min** in their name. These files are the minified versions of their corresponding script files. Minification is the process of removing unnecessary characters, such as white spaces, new lines, and comments from the JavaScript code, without changing the functionality provided by the code. Minified files are smaller in size as compared to their corresponding script files. This helps reduce the amount of data to be transferred over the Internet.

Further, the **Scripts** folder contains some files containing **vsdoc** in their name. These files are annotated to help Visual Studio provide better intellisense. The **Scripts** folder also contains some files that include the word, **unobtrusive**, in their names. These files are written by Microsoft. These files integrate with jQuery and the MVC framework to provide the unobtrusive JavaScript features. Further, the **Scripts** folder contains a JavaScript file containing **modernizr** in its name. This file helps you take benefit of the evolving Web technologies, such as HTML 5 and CSS3, and still have control over the older Web browsers that do not support these technologies. In addition, the **Scripts** folder contains some files that include the word, knockout, in their names. These files provide the data binding capabilities.

Using jQuery

jQuery helps in finding, traversing, and manipulating HTML elements inside an HTML document. In addition, it helps you to animate HTML elements, handle events, and make your applications rich and interactive.

You can make use of the jQuery library present in the Scripts folder of the Web application, in a view by using the `<script>` tag in the head section of the view. The following syntax is used to include a jQuery library in a view:

```
<script type= "text/javascript"
src="@Url.Content("~/Scripts/
jquery-1.7.1.js")"> </script>
```

In the preceding syntax:

- ❑ The `script` tag instructs the browser that the HTML document uses a script.
- ❑ The `type` tag specifies the type of scripting used.
- ❑ The `src` tag specifies the name of the jQuery library used.
- ❑ `@Url.Content()` is a helper method that converts a relative URL to an absolute URL.

The default `_Layout.cshtml` file that is included with most ASP.NET MVC project templates already includes a reference to the jQuery library. Therefore, in such cases, you do not need to explicitly add a reference to the jQuery library.

The execution of the jQuery code should occur only after the Web page has been fully loaded. To ensure that a Web page is fully loaded before the jQuery code is executed on it, jQuery provides the `document.ready()` function. Within this function, you can write the jQuery code to be executed on the HTML elements, after the Web page has been fully loaded in the browser.

The following syntax is used to specify the `document.ready()` function:

```
$(document).ready(function()
{
// jQuery code...
});
```

In the preceding syntax, the dollar sign ($) represents the start of a jQuery code block. The `(document).ready ()` function is useful for checking the readiness of the actions performed on an HTML element. For example, if the jQuery code for hiding an image executes before the document is ready, it will lead to failure as the image that the code is trying to hide has not yet rendered on the Web page.

Therefore, any code that needs to be executed after a page loads must be written inside the `(document).ready ()` function.

To add the customized client-side functionality to your website, you can write custom JavaScript/jQuery code in a **.js** file and add it to the **Scripts** folder contained in the application's root directory.

Now, to invoke the code written in your **.js** file from your application, you will need to refer to this file by using the `<script>` tag, as given in the following code syntax:

```
<script type= "text/javascript"
src="~/Scripts/><customJavaScriptName>
.js"> </script>
```

However, if your custom script uses the jQuery library, you need to ensure that the `<script>` tag for the jQuery library appears before the `<script>` tag for the custom script in the rendered document. If a custom script contains functionality that needs to be used in the entire application, you can include the `<script>` tag in the **_Layout.cshtml** view. However, if the functionality is required on a particular view, the `<script>` tag can be included in the same view.

jQuery can be used to select and manipulate HTML elements. It can also be used to handle events related to HTML elements and apply effects to HTML elements. Let us see how to perform these operations with jQuery.

Selecting and Manipulating HTML Elements

jQuery allows you to select the HTML elements and perform the required actions on them. For this, you need to use the following jQuery syntax:

```
$(selector).action()
```

In the preceding syntax,

- ❑ `$`: Is used to define or access jQuery.
- ❑ `selector`: Is used to find an HTML element or a group of HTML elements on the basis of their names, ids, classes, or attributes. A selector always starts with a dollar sign followed by a parenthesis. Depending on the need, you can use any jQuery selector. Some of the jQuery selectors are:
  - **Element selector**: Searches an HTML element or a group of HTML elements on the basis of their names. For example, to search all the h1 elements, you can use the following statement:
    `$("h1")`
  - **ID selector**: Searches the elements on the basis of their ids. For example, to search an element with the id, `blue`, you can use the following statement:
    `$("#blue")`
  - **Class selector**: Searches the elements on the basis of their class names. For example, to search all elements with the class, `blue`, you can use the following statement:
    `$(".blue")`
- ❑ **action()**: Is used to specify the action to be performed on the HTML element(s). jQuery provides many built-in actions, such as hide, fadeout, show, and slideup, which can be applied to the HTML elements.

Consider a scenario where you want to hide the h1 elements on a mouse-click. For this, you can use the following code snippet:

```
$(document).ready(function()
{$("h1").click(function()
 {$(this).hide();});});
```

In the preceding code snippet, all the h1 elements are selected and an anonymous function is assigned to click event of every h1 element. Within the anonymous function, the h1 element, which is clicked by a user, is selected by using the this keyword and its `hide()` method is called to hide the element.

In the preceding code snippet, the `<h1>` elements are referred to by the name, `h1`. However, HTML elements can also be selected using a class or id. For example, an HTML element with the id, `name`, can be selected by using the following syntax:

```
$("#name")
```

Similary, HTML elements with the class, `myclass`, can be selected by using the following syntax:

```
$(".myclass")
```

The jQuery library provides predefined functions to perform modifications on the content of the HTML elements. The following table lists some of the functions used to modify the content of HTML elements.

Function	**Description**	**Syntax**
html()	*Changes or retrieves the content of HTML elements, such as text in <p> and <div> tags.*	*$(selector).html (content)*
append()	*Enables a user to append the specified content to the content already contained in the selected element.*	*$(selector).append (content)*
prepend()	*Enables a user to add the specified content before the content already contained in the selected element.*	*$(selector).prepen d(content)*

The jQuery Functions

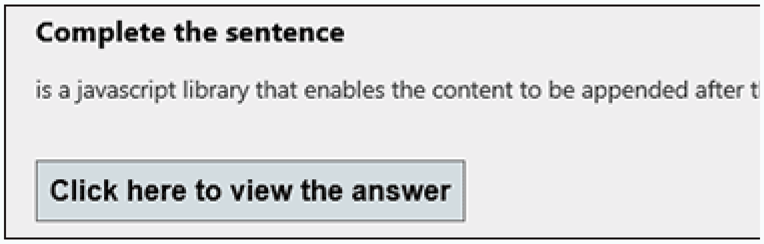
Consider the following markup for a view:

```
<html>
<head>
<script type= "text/javascript"
src="@Url.Content("~/Scripts/
jQuery-1.7.1.js")"> </script>
 <script src= "@Url.Content("~/Scripts/
 Myscript.js")" type="text/
 javascript"></script>
</head>
<body>
<h3>Complete the sentence</h3>
<p id="para">is a javascript library
that enables the content to be
appended after the inner content in
the selected element using the</p><br>
<button id="b">Click here to view the
answer.</button>
</body>
</html>
```

Now, consider the script file,**Myscript.js**, containing the following code:

```
$(document).ready(function(){
$("#b").click(function(){
 $("#b").html("You are viewing the
 answer");
 $("#para").prepend(" <b>jQuery </b>");
 $("#para").append(" <b>append()
 function</b>.");
});
});
```

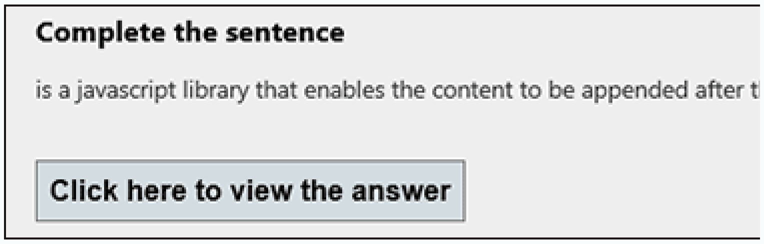
In the preceding code, when a user clicks the **Click here to view the answer** button, the text on the button is changed to, **You are viewing the answer**. This is done by using the `html()` function in the **jQuery** click event handler for the button. In addition, the text,**JjQuery**, is added before the content in the `<p>` tag that has the id, `para`, by using the `prepend()` function. Similarly, the content, `append()` function, is added after the content in the `<p>` tag by using the `append()` function. Before the button,**Click here to view the answer**, is clicked, the Web page appears, as shown in the following figure.


Complete the sentence

is a javascript library that enables the content to be appended after t

The Web Page Before the Button is Clicked

After the button, **Click here to view the answer**, is clicked, the Web page appears, as shown in the following figure.


Complete the sentence

is a javascript library that enables the content to be appended after t

The Web Page After the Button is Clicked

Handling Events

In jQuery, events are handled by using functions or predefined event methods. An event method is used to detect an event and trigger a function when that event occurs. You can also handle events for manipuling HTML elements.

The following table lists some of the event methods provided by jQuery.

Event Method	Description
`$(document).ready (function)`	*Used to execute a function after the document has finished loading.*
`$(selector).click (function)`	*Used to execute a function on the click event of the selected elements.*
`$(selector).dblclick(function)`	*Used to execute a function on the double-click event of the selected elements.*
`$(selector).mouseover(function)`	*Used to execute a function when the mouse pointer is moved over the selected elements.*
`$(selector).mouseout(function)`	*Used to execute a function when the mouse pointer is moved out of the selected elements.*
`$(selector).keydown (function)`	*Used to execute a function when a key is pressed on the selected elements.*

The Event Methods in jQuery

Consider the following markup for a view:

```
<html>
<head>
<script type= "text/javascript"
src="@Url.Content("~/Scripts/
jQuery-1.7.1.js")"> </script>
 <script src= "@Url.Content("~/Scripts/
 Myscript.js")" type="text/
 javascript"></script>
</head>
<body>
<h1>I am the header</h1>
<hr/>
<input type="button"
id="change_header" value='Change the
header'/></body>
</html>
```

Now, consider the script file, Myscript.js, containing the following code:

```
$(document).ready(function(){
$("#change_header").click(function(){
$("h1").html("I am the new
header"); });
});
```

In the preceding code,JjQuery recognizes a click event when the user clicks the **Change the header** button. After the event is recognized by jQuery, the function associated with the click event of the button is executed and the text inside the `<h1>` element is changed.

Adding Effects

With the help of jQuery, amazing visual effects can be applied to the elements of a Web document. These visual effects help enrich the browsing experience of a user. Some of the predefined jQuery effects that can be used to add visual appeal to a Web page are:

- ❑ Hide effect
- ❑ Show effect
- ❑ Toggle effect
- ❑ Fade effect

Hide Effect

The `hide()` function is used to make an element disappear when an event, such as click or double-click, occurs.

The following syntax is used to hide the selected elements:

`$(selector).hide(speed)`

In the preceding syntax, speed is an optional parameter that denotes speed at which an element disappears. The values that can be used for the speed attribute are:

- ❑ slow
- ❑ fast
- ❑ Duration in milliseconds

While specifying the speed attribute as duration in milliseconds, the value must not be enclosed within quotes. However, while specifying the speed as slow or fast, the value must be enclosed within quotes.

Consider the following markup for a view:

```
<html>
<head>
<script type= "text/javascript"
src="@Url.Content("~/Scripts/
jQuery-1.7.1.js")"> </script>
 <script src= "@Url.Content("~/Scripts/
 Myscript.js")" type="text/
 javascript"></script>
</head>
<body>
<h1>Hide me</h1>
<button>Click to hide</button>
</body>
</html>
```

Now, consider the script file, Myscript.js, containing the following code:

```
$(document).ready(function(){
 $(" button").click(function(){
   $(" h1").hide("slow");
});
});
```
In the preceding code, the text, **Hide me**, is displayed above the `Click to hide` button. When the `Click to hide` button is clicked, the text, **Hide me**, slowly disappears from the browser window.

Show Effect

The `show()` function is used to make a hidden element visible when an event occurs. The following syntax is used to show the selected elements if they are already hidden:

```
$(selector).show(speed)
```
In the preceding syntax, `speed` is an optional parameter used to specify the speed with which the hidden element reappears. The values that this parameter can take are slow, fast, and duration in milliseconds.

Consider the following markup for a view:

```
<html>
<head>
<script type= "text/javascript"
src="@Url.Content("~/Scripts/
jQuery-1.7.1.js")"> </script>
<script src= "@Url.Content("~/Scripts/
Myscript.js")" type="text/
javascript"></script>
</head>
<body>
<h3> This text can be hidden and
shown. </h3> <br/>
<button class="view">Click to view</
button>
<button class="conceal">Click to
hide</button>
</body>
</html>
```
Now, consider the script file, Myscript.js, containing the following code:

```
$(document).ready(function(){
$(".view").click(function(){
$("h3").show(2000);
});
 $(".conceal").click(function(){
 $("h3").hide();
});
});
```
In the preceding code snippet, the `<h3>` tag gets hidden in the browser window when the `Click to hide` button is clicked. When the `Click to view` button is clicked, the `<h3>` tag is shown again in 2000 milliseconds.

Toggle Effect

The `toggle()` function can be used to switch between the show effect and the hide effect of an element. This event can be used to hide or show the element alternatively when an event occurs.

The following syntax is used to show or hide the selected elements:

```
$(selector).toggle(speed)
```
In the preceding syntax, `speed` is an optional parameter. It is used to specify the speed at which the selected element appears and disappears. The values that this parameter can take are slow, fast, and duration in milliseconds.

Consider the following markup for a view:

```
<html>
<head>
<script type= "text/javascript"
src="@Url.Content("~/Scripts/
jQuery-1.7.1.js")"> </script>
 <script src= "@Url.Content("~/Scripts/
 Myscript.js")" type="text/
 javascript"></script>
</head>
<body>
<h3> This is some text.</h3>
<hr/>
<button class="view">Show/Hide</
button>
</body>
</html>
```
Now, consider the script file, Myscript.js, containing the following code:

```
$(document).ready(function(){
$(".view").click(function(){
$("h3").toggle("fast");
});
});
```
In the preceding code snippet, the `toggle()` function hides and shows the `<h3>` element alternately.

Fade Effect

The jQuery fade effect is used to gradually reduce the opacity of the selected elements. There are various fade functions, such as `fadeOut()`and `fadeIn()`, which can be applied on the selected elements.

The following table lists these functions to produce the fade effect.

Functions	**Description**	**Syntax**
`fadeout()`	*Is used to fade the appearance of the selected elements.*	*$(selector).fadeOut(speed)*
`fadeIn()`	*Is used to restore the appearance of a selected faded element.*	*$(selector).fadeIn (speed)*

The Fade Effect Functions

Consider the following markup for a view:

```
<html>
<head>
<script type= "text/javascript"
```

```
src="@Url.Content("~/Scripts/
jQuery-1.7.1.js")"> </script>
 <script src= "@Url.Content("~/Scripts/
 Myscript.js")" type="text/
 javascript"></script>
</head>
<body>
 <p>CLICK HERE TO FADE ME AWAY!</p>
 <br />
 <button>Restore</button>
</body>
</html>
```

Now, consider the script file, Myscript.js, containing the following code:

```
$(document).ready(function () {
 $("p").click(function () {
   $(this).fadeOut(1200);
 });
 $("button").click(function () {
   $("p").fadeIn(1200);
 });
 });
```

In the preceding code, the <p> tag is referred using its name. In the click event handler of the <p> tag, the keyword, `this`, refers to the current HTML element, which is the <p> tag. When the user clicks the text, CLICK HERE TO FADE ME AWAY!, the text fades away. When the `Restore` button is clicked, the text, CLICK HERE TO FADE ME AWAY!, reappears.

The values that can be used for the speed attribute are:

slow

fast

Duration in milliseconds
While specifying the speed attribute as duration in milliseconds, the value must not be enclosed within quotes.

Using jQuery UI

jQuery UI library provides a set of widgets, such as date pickers, spinners, dialog boxes, and autocomplete text boxes, which allow you to design a user interface and apply various effects to the UI elements. In addition, it allows you to apply themes to the widgets to integrate their colors and styles with the website.

To use the jQuery UI library in a view, you need to include a reference to the jQuery UI script file in your view, as given in the following code snippet:

```
<script    type=    "text/javascript"
src="@Url.Content("~/Scripts/jQuery-
ui-1.8.20.js")">  </script>Let  us  see
how  to  add  effects  and  widgets  to  a
view by using jQuery UI.
```

Using jQuery UI Effects

You can add various effects to the various UI elements in your application. Some ways to implement these effects in an application are:

- ❑ Add/Remove/Toggle classes
- ❑ Effect
- ❑ Show/Hide/Toggle

Add/Remove/Toggle Classes

Using this technique, you can add, remove, or toggle class (es) for the HTML elements while applying the style animations on them. Consider the following markup for a view:

```
<html>
<head>
<link rel="stylesheet"
href="@Url.Content("~/Content/
main.css")">
<script type= "text/javascript"
src="@Url.Content("~/Scripts/
jQuery-1.7.1.js")"> </script>
<script type= "text/javascript"
src="@Url.Content("~/Scripts/jQuery-
ui-1.8.20.js")"> </script>
<script src= "@Url.Content("~/Scripts/
Myscript.js")" type="text/
javascript"></script>
</head>
<body>
 <div id="div1"></div>
 <button id="button1">Add Class</
 button>
 <button id="button2">Remove Class</
 button>
 <button id="button3">Toggle Class</
 button>
</body>
</html>
```

The preceding markup generates a view consisting of a `div` element and three buttons titled `Add Class`, `Remove Class`, and `Toggle Class`. The view uses a stylesheet named `main.css`.

Now, consider the following style definitions included in the `main.css` file:

```
div {
 width:300px;
 height: 200px;
 background-color:yellow;
}
.myclass {
width : 600px;
height: 400px;
background-color:green;
}
```

In addition, the markup refers to a JavaScript file named Myscript.js. The Myscript.js file includes the following code:

```
$(document).ready(function () {
$("#button1").click(function() {
  $("#div1").addClass("myclass", 1000);
});
$("#button2").click(function () {
  $("#div1").removeClass("myclass",
1000);
});
$("#button3").click(function () {
  $("#div1").toggleClass("myclass",
1000);
});
});
```

When a user clicks the button titled, `Add Class`, the `myclass` class defined in the `main.css` file is applied to the `div` element. Similarly, when a user clicks the button titled, `Remove Class`, the `myclass` class applied to the `div` element is removed. When the user clicks the button titled,`Toggle Class`, the `myclass` class is applied to the `div` element if it is not already applied. However, if the `myclass` class is already applied to the `div` element, the class is removed from the `div` element.

If your view uses the default _Layout.cshtml layout included in most ASP.NET MVC project templates, you need to add the script elements within the scripts section, as shown in the following code snippet:

```
@section scripts{
<script type= "text/javascript"
src="@Url.Content("~/Scripts/
jQuery-1.7.1.js")"> </script>
<script type= "text/javascript"
src="@Url.Content("~/Scripts/jQuery-
ui-1.8.20.js")"> </script>
<script src= "@Url.Content("~/Scripts/
Myscript.js")" type="text/
javascript"></script> }
```

Effect

Using this technique, you can apply various animation effects to an element, such as appear, bounce, blind, clip, and fade. Consider the following markup for a view:

```
<html>
<head>
<script type= "text/javascript"
src="@Url.Content("~/Scripts/
jQuery-1.7.1.js")"> </script>
<script type= "text/javascript"
src="@Url.Content("~/Scripts/jQuery-
ui-1.8.20.js")"> </script>
<script src= "@Url.Content("~/Scripts/
Myscript.js")" type="text/
javascript"></script>
</head>
<body>
 <div id="div1" style="width:300px;
 height: 200px; background-
 color:yellow;"></div>
 <button id="button1">Apply Blind
 Effect</button>
</body>
</html>
```

The preceding markup generates a view containing a div element and a button titled, `Apply Blind Effect`. The preceding view refers to a script file, `Myscript.js`, which contains the following code:

```
$(document).ready(function () {
 $("#button1").click(function () {
   $("#div1").effect("blind", 1000);
 });
 });
```

When a user clicks the button titled, `Apply Blind Effect`, the `div` element disappears in 1000 milliseconds with a blind effect. Similarly, you can apply other effects, such as bounce and clip, to various HTML elements. For example, you can use the following code to apply the bounce effect to a `div` element with the id, div1:

```
$("#div1").effect("bounce", 1000);
```

To apply the clip effect to a div element with the id, div1, you can use the following code:

```
$("#div1").effect("clip", 1000);
```

Show/Hide/Toggle

Using this technique, you can display or hide elements using custom jQuery effects. Consider the following markup for a view:

```
<html>
<head>
<script type= "text/javascript"
src="@Url.Content("~/Scripts/
jQuery-1.7.1.js")"> </script>
<script type= "text/javascript"
src="@Url.Content("~/Scripts/jQuery-
ui-1.8.20.js")"> </script>
<script src= "@Url.Content("~/Scripts/
Myscript.js")" type="text/
javascript"></script>
</head>
<body>
<div id="div1" style="width:300px;
```

```
height: 200px; background-
color:yellow;"></div>
 <button id="button1">Hide</button>
 <button id="button2">Show</button>
</body>
</html>
```

The preceding markup generates a view consisting of a `div` element and two buttons titled, `Hide` and `Show`. The markup refers to a custom script file,`Myscript.js`, which contains the following code:

```
$(document).ready(function () {
 $("#button1").click(function () {
   $("#div1").hide("blind", 1000);
 });
 $("#button2").click(function () {
   $("#div1").show("clip", 1000);
});});
```

When the user clicks the `Hide` button, the `div` element hides with a blind effect. When the user clicks the `Show` button, the `div` element appears with a clip effect.

Using jQuery UI Widgets

You can easily add different types of widgets to your pages by using jQuery UI. Some widgets provided by jQuery UI are:

- Autocomplete
- Datepicker
- Accordion

Autocomplete

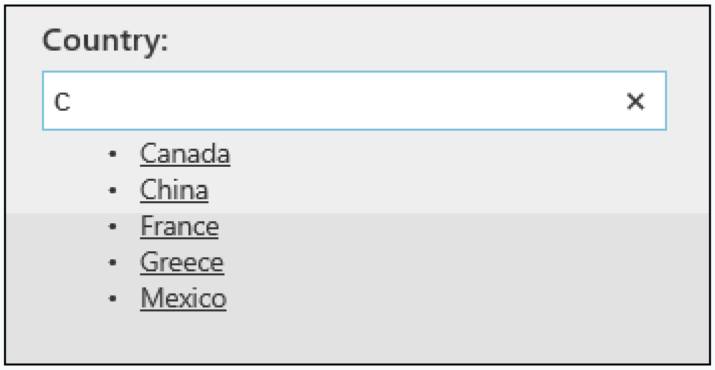
The `Autocomplete` widget adds an autocomplete text box to a Web page. This widget provides suggestions while you type into the text box. Consider the following markup for a view:

```
<html>
<head>
<script type= "text/javascript"
src="@Url.Content("~/Scripts/
jQuery-1.7.1.js")"> </script>
<script type= "text/javascript"
src="@Url.Content("~/Scripts/jQuery-
ui-1.8.20.js")"> </script>
<script src= "@Url.Content("~/Scripts/
Myscript.js")" type="text/
javascript"></script>
<link rel="stylesheet" href="http://
code.jQuery.com/ui/1.10.3/themes/
smoothness/jQuery-ui.css" />
</head>
<body>
<div>
 <label for="country">Country: </label>
 <input id="country" />
</div>
</body>
</html>
```

The preceding markup generates a view consisting of a `div` tag, which contains a label and an input field with the id, `country`. The markup refers to a style sheet named jQuery-ui.css that defines styles to display jQuery UI widgets. In addition, it refers to a custom script file named Myscript.js, which contains the following code:

```
$(document).ready(function () {
var countries = [
 "Alaska",
 "Argentina",
 "Bahrain",
 "Bhutan",
 "Canada",
 "China",
 "Denmark",
 "Egypt",
 "France",
 "Germany",
 "Greece",
 "Hong Kong",
 "India",
 "Japan",
 "Mexico",
 "New Zealand",
 "Singapore",
 "United Kingdom",
 "United States"
];
$("#country").autocomplete({source:
countries});
});
```

The preceding code snippet declares an array named `countries`, which defines the suggestions for the autocomplete widget. It then calls the `autocomplete ()` function to convert the input element with the id, `country`, into an Autocomplete widget. While invoking this function, it passes the `countries` array as the source for the Autocomplete widget. As the user types in the autocomplete widget, matching `country` names are shown to the user in the form of suggestions, as shown in the following figure.



The Autocomplete Widget Showing Suggestions

Datepicker

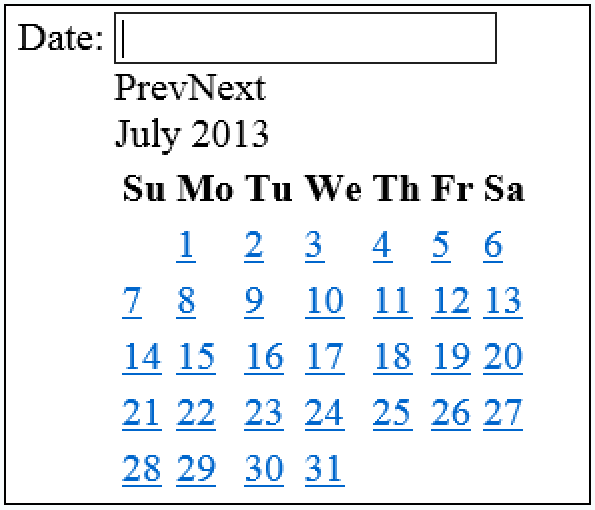
The Datepicker widget allows a user to select a date from a calendar. Consider the following markup for a view:

```
<html>
<head>
<script type= "text/javascript"
src="@Url.Content("~/Scripts/
jQuery-1.7.1.js")"> </script>
<script type= "text/javascript"
src="@Url.Content("~/Scripts/jQuery-
ui-1.8.20.js")"></script>
<script src= "@Url.Content("~/Scripts/
Myscript.js")" type="text/
javascript"></script>
<link rel="stylesheet" href="http://
code.jQuery.com/ui/1.10.3/themes/
smoothness/jQuery-ui.css" />
</head>
<body>
<div>
<p>Date: <input type="text"
id="datepicker" /></p>
</div>
</body>
</html>
```

The preceding markup creates a view that contains a text box with the id, `datepicker`. The preceding markup refers to a custom script file, Myscript.js, which contains the following code to convert the text box into a Datepicker widget:

```
$(document).ready(function () {
 $("#datepicker").datepicker();
});
```

When the user clicks the text box, a Datepicker widget is displayed, as shown in the following figure.


Date:

PrevNext

July 2013

Su	Mo	Tu	We	Th	Fr	Sa
	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30	31			

The Datepicker Widget

Accordion

The Accordion widget displays collapsible content panels for presenting information in a limited amount of space. For example, you can expand/collapse content broken into logical sections. To use accordion, you need to include a reference to the jQuery UI css file, as shown in the following markup:

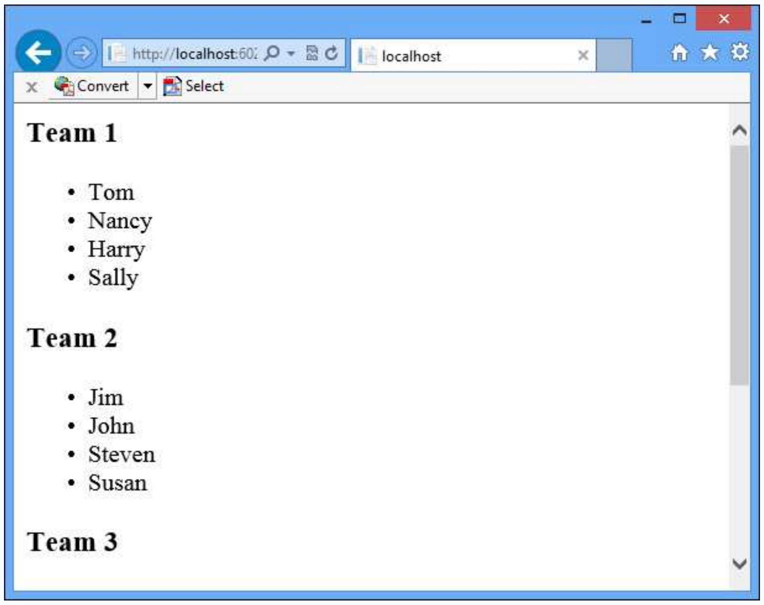
```
<link rel="stylesheet" href="http://
code.jQuery.com/ui/1.10.3/themes/
smoothness/jQuery-ui.css" />
```

Consider the following markup for a view named Accordion.cshtml:

```
<html>
<head>
<link rel="stylesheet" href="http://
code.jQuery.com/ui/1.10.3/themes/
smoothness/jQuery-ui.css" />
<script type= "text/javascript"
src="@Url.Content("~/Scripts/
jQuery-1.7.1.js")"> </script>
<script type= "text/javascript"
src="@Url.Content("~/Scripts/jQuery-
ui-1.8.20.js")"> </script>
<script src= "@Url.Content("~/Scripts/
Myscript.js")" type="text/
javascript"></script>
</head>
<body>
<div id="div1">
<h3>Team 1</h3>
 <div>
  <ul>
    <li>Tom</li>
    <li>Nancy</li>
    <li>Harry</li>
    <li>Sally</li>
  </ul>
 </div>
<h3>Team 2</h3>
 <div>
  <ul>
    <li>Jim</li>
    <li>John</li>
    <li>Steven</li>
    <li>Susan</li>
  </ul>
 </div>
<h3>Team 3</h3>
 <div>
  <ul>
    <li>Mark</li>
    <li>Ben</li>
    <li>Kelly</li>
    <li>Jane</li>
  </ul>
 </div>
<h3>Team 4</h3>
 <div>
```

```
  <ul>
   <li>George</li>
   <li>Lina</li>
   <li>Amanda</li>
   <li>Ken</li>
  </ul>
 </div>
</div>
</body>
</html>
```

The preceding markup generates a view, as shown in the following figure.



Team 1

- Tom
- Nancy
- Harry
- Sally

Team 2

- Jim
- John
- Steven
- Susan

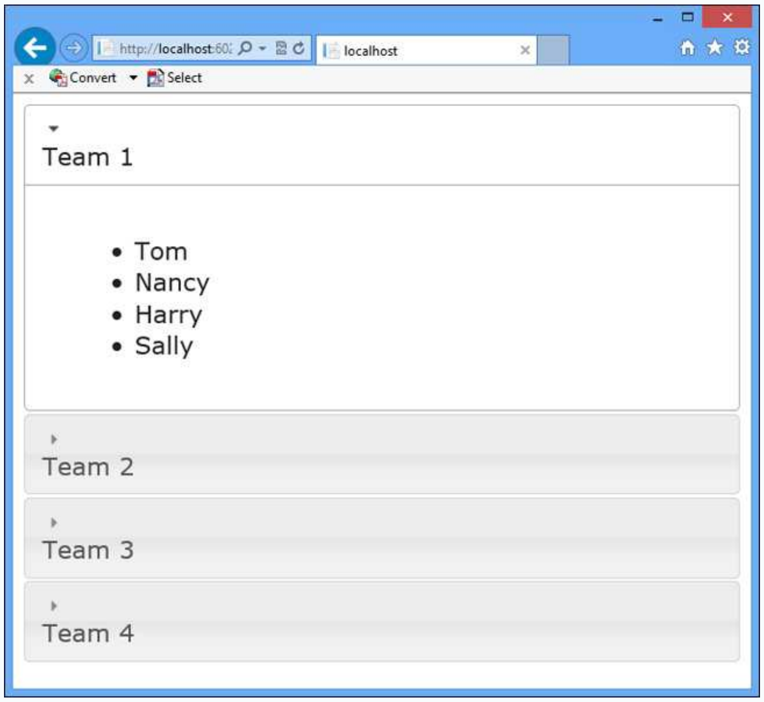
Team 3

The Accordion.cshtml View

Now, to display the data of the various teams under different tab-like structures, you need to convert the `div` tag with the id, `div1`, into an accordion. For this, you need to add the following code in a script file associated with the view:

```
$(document).ready(function () {
 $("#div1").accordion();
});
```

Once the preceding code in a script file is associated with the view, the `Accordion.cshtml` view appears, as shown in the following figure:



Team 1

- Tom
- Nancy
- Harry
- Sally

Team 2

Team 3

Team 4

The Accordion.cshtml View After Adding the Accordion

In the preceding view, a user can click any of the section headers to view the details of that section.

Using Content Delivery Networks for JavaScript Libraries

You have seen how to include the script files for JavaScript libraries, such as jQuery and jQuery UI, in your application. Although it is perfectly acceptable to serve jQuery scripts from your own server, a better approach is to use a `<script>` tag that references jQuery from a Content Delivery Network (CDN).

A CDN is a network of servers containing shared code libraries.

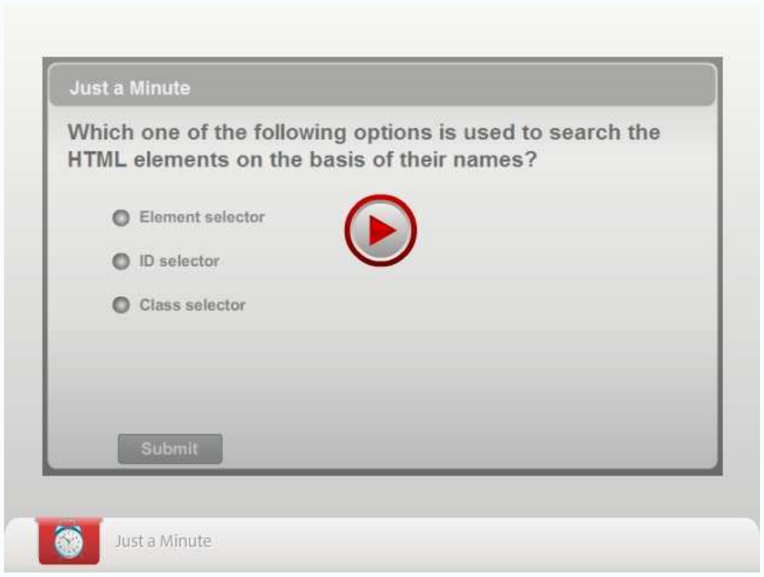
The contents of the CDN are cached on servers located around the world. This helps improve the download speed of the scripts at the client end. In addition, a number of sites may refer to jQuery from the same CDN. If a user visits a website that refers to jQuery from a particular CDN, the script can be cached locally on the client. Now, if the user visits another website that refers to jQuery from the same CDN, the cached script will be used, thereby improving the performance and saving the cost of downloading scripts.

There are various CDN providers, such as Google and Microsoft, which provide free CDNs for a number of JavaScript libraries.

To use a JavaScript library from a CDN, you need to provide the URL of the CDN as the value of the `src` attribute of the `<script>` tag, as shown in the following code snippet:

```
<script src="https://
ajax.googleapis.com/ajax/libs/
```

```
jQuery/1.8.3/jQuery.min.js">
< /script>
```


Just a Minute

Which one of the following options is used to search the HTML elements on the basis of their names?

- Element selector
- ID selector
- Class selector

Activity 8.1: Using jQuery

Implementing Partial Page Updates Using AJAX

In traditional Web applications, the client and the server interact synchronously. This means that the processing of the Web applications happens sequentially, one after the other.

Consider the example of a Web application that contains a Registration page. This page contains the drop-down lists for selecting the country and the state in the selected country. When a user selects a country from the drop-down list, the states for the selected country need to be populated in the state drop-down list. For this, a request is sent to the server along with all the data contained in the Web page. The server processes the request, retrieves the names of states in the selected country, and sends the entire Web page back to the client machine. This process is called a postback.

This approach leads to the following drawbacks:

- ❑ The user interaction with the application is interrupted every time a postback is made.
- ❑ The user has to wait during each postback.
- ❑ The full page is rendered and transferred to the client after each postback, which is time-consuming and traffic intensive.

To overcome such drawbacks, a technology called AJAX can be implemented in Web applications. By implementing AJAX in Web applications, the interaction between the client and the server becomes asynchronous. This allows the users to interact with the Web application while waiting for a response from the server. AJAX implementation enables partial updates in Web applications. This means that instead of loading the entire Web page again, only the portion that needs to be updated is reloaded.

AJAX is a Web development technique used to create dynamic and interactive applications. It enables Web applications to retrieve data from the server, asynchronously in the background, without interfering with the display and behavior of the existing page. AJAX-enabled Web applications provide various advantages, such as:

- ❑ **Quick response to a user's request**: This is due to a partial-page updates feature that refreshes only those parts of a Web page that have changed or updated. In addition, the partial-page updates feature enables an application to use less bandwidth because only those parts of a Web page that have changed or updated are sent to the server.
- ❑ **Asynchronous communication**: It allows a user to interact with the rest of the Web page while the application is processing the changed or updated parts of the Web page.
- ❑ **Web browser support**: AJAX is supported by all the widely used Web browsers, such as Microsoft Internet Explorer, Mozilla Firefox, and Apple Safari.

AJAX provides asynchronous communication to the server and partial updates of the Web pages. Therefore, it can be used in the Web applications where a portion of the Web page needs to be updated, instead of updating the entire page. For example, the weather forecasting websites need to be updated frequently to provide the current temperature and weather updates. Therefore, AJAX can be used in these websites.

Let us learn how to use AJAX for making your websites responsive.

Introduction to AJAX Helpers

You have seen how to use HTML helpers to create forms and links that point to controller actions. In the same way, there is a set of AJAX helpers, such as `Ajax.ActionLink()` and `Ajax.BeginForm()`, supported by ASP.NET MVC. These AJAX helpers behave asynchronously and create forms and links that point to controller actions. To work with such helpers, you need to use the `jQuery.unobtrusive-ajax` script. Ajax helpers depend on unobtrusive MVC extensions for jQuery. Therefore, before using Ajax, you need to prepare your project for Unobtrusive Ajax.

Preparing a Project for Unobtrusive AJAX

To use unobtrusive AJAX in an application, you need to update the Web.config file in the root folder of the application. For this, you need to set the `UnobtrusiveJavaScriptEnabled` property in the Web.config file to `true`.

The highlighted portion of the following code snippet illustrates how to enable the unobtrusive AJAX feature in the Web.config file:

```
...
<configuration>
<appSettings>
<add key="webpages:Version"
value="2.0.0.0" />
<add key="webpages:Enabled"
value="false" />
<add key="PreserveLoginUrl"
value="true" />
<add key="ClientValidationEnabled"
value="true" />
<add
key="UnobtrusiveJavaScriptEnabled"
value="true" />
</appSettings>
</configuration>
...
```

In addition, you need to add references to the jQuery JavaScript libraries that implement the unobtrusive AJAX functionality. You can refer to the libraries from individual views. However, if you want to refer to the libraries from all the views, then you can refer to it from the `_Layout.cshtml` view using the `<script>` tag, as shown in the following code snippet:

```
<script type= "text/javascript"
src="@Url.Content("~/Scripts/
jQuery-1.7.1.js")">
</script>
<script src="@Url.Content("~/Scripts/
jQuery.unobtrusive-ajax.js")"></
script>
```

In the preceding code snippet, the `jQuery-1.7.1.min.js` file contains the core jQuery library, and the `jQuery.unobtrusive-ajax.min.js` file contains the AJAX library, which relies on the main jQuery library.

Using AJAX ActionLinks

The `Ajax.ActionLink` helper creates an anchor tag with asynchronous behavior.

Consider a scenario where you want to add a link, Best Deal, on the Home page of the Newbay Store. You want that when a user clicks this link, the details of the best deal should be retrieved from the server and displayed on the Home page. However, this should happen through a partial page update, and the entire page should not be posted back to the server. To implement this functionality, you can add the following markup in the view corresponding to the Home page:

```
<div id="msg">
@Ajax.ActionLink("Click here to see
best deal for today!",
"BestDeal",
new AjaxOptions{
UpdateTargetId="BestDeal",
InsertionMode=InsertionMode.Replace,
HttpMethod="GET"
})
</div>
<div id="BestDeal"></div>
```

In the preceding markup, the first parameter to the `ActionLink` method specifies the link text, and the second parameter is the name of the action you want to invoke asynchronously. The `AjaxOptions` parameter specifies how to send the request, and what would be its effect. The `UpdateTargetId` property specifies that the response of the Ajax request should be displayed in the element with the id, `BestDeal`. The `InsertionMode` property specifies that the content of the target element will be replaced with the contents of the Ajax response. The `HttpMethod` property specifies that the asynchronous request should be sent to the server by using the HTTP Get method.

To handle the request at the server, you need to create the `BestDeal` action method in the appropriate controller, as shown in the following code snippet:

```
public ActionResult BestDeal()
{
 ProductDBContext db = new
 ProductDBContext();
 /* Retrieve product details ordered by
 price and store the first record from
 the result set in the variable,
 product. */
 var product = db.Products.OrderBy(a =>
 a.price).First();
 return PartialView("_BestDeal",
 product);
}
```

The preceding code snippet will return a partial view, `_BestDeal.cshtml`. This partial view will be rendered within the div element with the id, `BestDeal`, on the Home page. You need to add the following markup in the `_BestDeal.cshtml` file:

```
@model MvcApplication1.Models.Product
<p>
```

```
Product:
@Model.name <br />
Price:
@Model.price
</p>
```

In the preceding markup, `MvcApplication1` is the name of the application. This markup renders the details of the product passed to it by the controller.

Once the user clicks the Ajax link, an asynchronous request is sent to the `BestDeal` action method. Once the action returns a partial view, the script renders the partial view in the existing `BestDeal` element on the home page.

Using AJAX Forms

Consider the scenario of the Home page of the Newbay store. You want to give the user the ability to search for a product. As the user types in the search criteria, the details of the matching records should be retrieved from the server and displayed on the same page, without having to post the page back to the server. This can be easily implemented by sending an Ajax request to the server. To implement this functionality, you must place an asynchronous form element on the Home page, as shown in the following code snippet:

```
@using (Ajax.BeginForm("Search",
"Home",
new AjaxOptions
{
 InsertionMode = InsertionMode.Replace,
 HttpMethod = "GET",
 OnFailure = "searchFailed",
 LoadingElementId = "ajax-loader",
 UpdateTargetId = "searchresults",
}))
{
<input type="text" name="q" />
<input type="submit" value="search" />
<img id="ajax-loader"
src="@Url.Content("~/Content/Images/
ajax-loader.gif")"
style="display:none"/>
}
<div id="searchresults"></div>
```

The preceding code snippet renders a textbox and button element on the screen. In addition, its adds a `div` element with the id, `searchresults`, to the page. The first parameter of the `Ajax.BeginForm()` method specifies the action method that will handle the Ajax request on the server. The second parameter specifies the controller that contains the action method. The third parameter specifies various options for the Ajax request. In the preceding code snippet, the third parameter specifies that:

- ❑ The contents of the Ajax response should replace the existing content of the target element.
- ❑ The HTTP method, Get, should be used for sending the request to the server.
- ❑ If the Ajax request fails, the `searchfailed()` function should be executed.
- ❑ While the asynchronous request is in progress, animated gif specified by the element id, `ajax-loader` should be displayed to indicate progress.
- ❑ The response from the server should be displayed in an element with the id, `searchresults`.

When the user clicks the submit button, the browser sends an asynchronous GET request to the `Search` action of the `Home` controller. The `Search` action method can be defined, as shown in the following code snippet:

```
public ActionResult Search(string q)
{
 ProductDBContext db = new
 ProductDBContext();
 var products = db.Products.Where
 (a=>a.name.Contains(q));
 return PartialView("_searchresults",
 products);
}
```

The preceding code retrieves product records that match the specified search string from the database, and returns a partial view named `_searchresults`, containing the retrieved records. This partial view is then rendered in the `searchresults` element on the Home page.

The `_searchresults` partial view contains the following markup to render the details of the products that match the search criteria:

```
@model
IEnumerable<MvcApplication1.Models.Pro
duct>
<table>
<tr>
    <th>
      @Html.DisplayNameFor(model →
model.name)
    </th>
    <th>
      @Html.DisplayNameFor(model =>
model.price)
    </th>
    <th> </th>
</tr>
@foreach (var item in Model) {
<tr>
    <td>
      @Html.DisplayFor(modelItem =>
item.name)
    </td>
    <td>
```

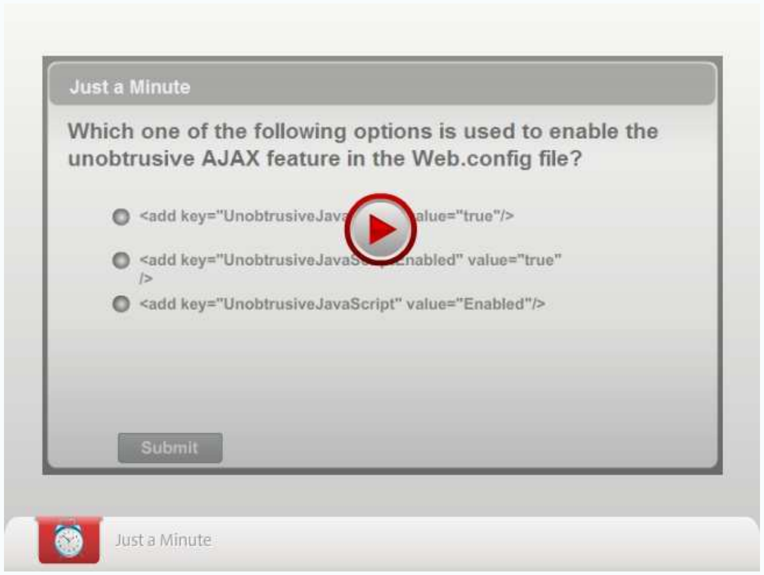
```
        @Html.DisplayFor(modelItem =>
 item.price)
     </td>
 </tr>
 }
</table>
```

In the preceding markup, `MvcApplication1` is the name of the application.

If the Ajax request fails, a function named `searchFailed` is invoked. The `searchFailed()` function is defined in a custom JavaScript file referenced by the view, as shown in the following code snippet:

```
function searchFailed() {
$("#searchresults").html("Sorry, there
was a problem with the search.");
}
```

Developing Responsive Web Applications Using JavaScript and AJAX
NIIT
Animation


Just a Minute

Which one of the following options is used to enable the unobtrusive AJAX feature in the Web.config file?

- <add key="UnobtrusiveJava alue="true"/>
- <add key="UnobtrusiveJavaS nabled" value="true" />
- <add key="UnobtrusiveJavaScript" value="Enabled"/>

Submit

Just a Minute

Activity 8.2: Implementing Partial Page Updates Using AJAX

Summary

In this chapter, you learned that:

- ❑ JavaScript is a client-side scripting language that allows you to develop dynamic and interactive rich Web applications.
- ❑ Unobtrusive JavaScript is a general approach to implement JavaScript in Web pages.
- ❑ A JavaScript library is a set of prewritten JavaScript code, which helps in developing a JavaScript-based application easily.
- ❑ jQuery helps in finding, traversing, and manipulating HTML elements inside an HTML document.
- ❑ With the help of jQuery, amazing visual effects can be applied to the elements of a Web document.
- ❑ Some of the predefined jQuery effects that can be used to add visual appeal to a Web page are:
  - Hide effect
  - Show effect
  - Toggle effect
  - Fade effect
- ❑ jQuery UI library provides a set of widgets, such as date pickers, spinners, dialog boxes, and autocomplete text boxes, which allow you to design a user interface and apply various effects to the UI elements.
- ❑ You can add various effects to the various UI elements in your application. Some ways to implement these effects in an application are:
  - Add/Remove/Toggle classes
  - Effect
  - Show/Hide/Toggle
- ❑ You can easily add different types of widgets to your pages by using jQuery UI. Some widgets provided by jQuery UI are:
  - Autocomplete
  - Datepicker
  - Accordion
- ❑ A CDN is a network of servers containing shared code libraries.
- ❑ AJAX is a Web development technique used to create dynamic and interactive applications.
- ❑ AJAX-enabled Web applications provide various advantages, such as:
  - Quick response to a user's request
  - Asynchronous communication
  - Web browser support

Chapter 9

Managing State and Optimizing Performance

An ASP.NET MVC application uses the HTTP protocol to communicate between a Web browser and a Web server. HTTP is a stateless protocol and cannot automatically indicate whether the sequential requests are coming from the same or different clients. Therefore, each time a Web page is posted to the server, a new instance of the Web page is created. As a result, there is a loss of information associated with the page and the controls placed on it. To overcome this limitation, ASP.NET MVC provides the various state management features.

A Web application must provide quick response to its users. Response time is a direct measure of the performance of a Web application. ASP.NET MVC provides the various optimization techniques to enhance the performance of a Web application.

This chapter discusses the various state management options provided by ASP.NET MVC. In addition, it discusses the various techniques to enhance the performance of a Web application, such as caching, bundling, and minification.

Objectives

In this chapter, you will learn to:

- ❑ Implement state management
- ❑ Optimize the performance of a Web application

Implementing State Management

While developing a Web application, you may want to implement functionality that requires information to be retained across multiple requests to the server.

Consider the example of the Newbay Web application where a customer needs to place an order online. While doing this, the customer selects a product category from a drop-down list on the Web page. As soon as the customer selects the category, a request is sent to the server to retrieve a list of product names belonging to the selected category, which needs to be populated in another drop-down list on the same Web page. However, as HTTP is a stateless protocol, when the request is sent to the server, a new instance of the Web page is created, and therefore, all the information previously entered on the Web page is lost. This problem can be solved by using state management.

Some options available to implement state management are:

- ❑ Cookies
- ❑ QueryString
- ❑ TempData
- ❑ Application State
- ❑ Session State

Cookies

Cookies are small pieces of information that are stored on the client's computer. The purpose of storing this information is to offer a personalized experience to the user. For example, if a user has logged on to a website for the first time, the name of the user can be stored in the form of a cookie. When a browser requests the same page the next time, the website sends the cookie along with the requested information. The Web server reads the cookie and extracts its value. It then processes the Web page according to the information contained in the cookie and renders it on the Web browser.

For example, if the name of the user is stored in a cookie, the user's name will be sent to the server along with the request for a Web page. The name of the user can then be displayed with a greeting message on the Web page when it is rendered on the browser.

Types of Cookies

Cookies can be either temporary or persistent. Temporary cookies exist in the memory space of a browser. When the browser is closed, all temporary cookies added to the browser are lost. Temporary cookies are also known as session cookies.

A temporary cookie is useful for storing information required for only a short time. Temporary cookies are also useful for storing information that should not be written to the disk on a client computer for security reasons. For example, a temporary cookie can be created to store the user's login state. This cookie is sent with every subsequent request to the same website to establish the user's login credentials. This enables the user to login just once during a session, and then access the various pages of the website without logging in again. However, when the user closes the browser or logs out, the user's session ends, and the cookie is removed. This is done as a security measure to prevent a user from remaining logged in unintentionally.

A persistent cookie is saved as a text file in the file system of the client computer. Persistent cookies are used when you want to store information for a longer time period. For example, persistent cookies can be used to store customized user settings for a website. This helps in customizing the website when the user revisits the website.

While creating a persistent cookie, you can set the expiration date of the cookie, which defines the duration for which the cookie will be stored on the client computer. If you do not set the expiration date of a cookie, the cookie is maintained as a temporary cookie, as a part of the user's session information. Once the user session is finished, the cookie is discarded.

Creating Cookies

Persistent cookies can be created, as shown in the following example:

```
Response.Cookies ["userName"].Value =
"Peter";
Response.Cookies ["userName"].Expires
= DateTime.Now.AddDays(2);
Response.Cookies ["lastVisited"].Value
= DateTime.Now.ToString();
Response.Cookies
["lastVisited"].Expires =
DateTime.Now.AddDays(2);
```

In the preceding example, two cookies, `userName` and `lastVisited`, are created. The user name, `Peter`, is stored in the first cookie, and the date and time when the user, `Peter`, last visited the site are stored in the second cookie. The expiry period for both the cookies is set as `2` days. If you do not set the expiry period for the cookies, the cookies will be created as temporary cookies.

You can also store multiple values in a single cookie. For example, instead of creating separate cookies for storing the user name and last visited information, you can create a single cookie that can hold both the values. Such a cookie can be created, as shown in the following example:

```
Response.Cookies["userInfo"]
["userName"] = "Peter";
Response.Cookies["userInfo"]
["lastVisited"] =
DateTime.Now.ToString();
Response.Cookies["userInfo"].Expires =
DateTime.Now.AddDays(2);
```

In the preceding example, the cookie, `userInfo`, with two subkeys, `userName` and `lastVisited`, is created.

Accessing Cookies

You can access the value of a cookie by using the `Request` built-in object. However, if you want to modify a cookie, you need to use the `Response` built-in object of ASP.NET.

The following code snippet reads a cookie that stores a single value:

```
if (Request.Cookies
["userName"].Value != null)
{
string Name = Request.Cookies
["userName"].Value;
}
```

In the preceding code snippet, if the value of the cookie, `userName`, is not null, the value is assigned to the string, `Name`.

Similarly, you can read the values from a cookie that stores multiple values. The following example reads a cookie that stores multiple values:

```
if (Request.Cookies["userInfo"] !=
null)
{
string Name = Request.Cookies
["userInfo"]["userName"];
string VisitedOn = Request.Cookies
["userInfo"]["lastVisited"];
}
```

In the preceding code snippet, if the value of the cookie, `userInfo`, is not null, the value of the subkeys, `userName` and `lastVisited`, are assigned to the strings, `Name` and `VisitedOn`, respectively.

Advantages and Disadvantages of Cookies

Cookies have the following advantages:

- ❑ A cookie is stored on the client computer and can be read by the server after a request for a page is posted. Therefore, there is no server resource involved in maintaining the cookie.
- ❑ A cookie is a text-based data structure that contains key-value pairs. Therefore, it is easy to create and manipulate cookies.
- ❑ A cookie can expire when the browser session ends or can exist for a specified time on the client computer, as per your requirement.

Cookies have the following limitations:

- ❑ Cookies that are stored on the client computer have a limited size. Therefore, you cannot store a large amount of data in a cookie.
- ❑ A user can disable cookies to prevent them from being stored on the hard disk of the computer. If a user denies permission for cookies, an ASP.NET Web application cannot store cookies on the client computer.
- ❑ Users can tamper with cookies because cookies are stored on the client computer.

Query String

Sometimes, it is required to transfer information from one page to another. For example, on the Newbay shopping website, the product details page displays the list of categories of products. When the user selects a category, another page with the detailed information about the selected category is displayed. This process requires you to transfer the information about the selected category to another page so that the relevant details can be displayed. You can perform this task by using a query string.

A query string provides a simple way to pass information from one page to another. It is the part of a URL that appears after the Question mark (`?`) character. Consider the following URL that passes information by using a query string:

```
http://www.Newbay.com/search?
Category=Electronics
```

In the preceding URL, the query string defines the variable, `Category`, which contains the string, `Electronics`.

You can pass data from one page to another page in the form of a query string by using the `Response.Redirect` method, as shown in the following statement:

```
Response.Redirect(Url.Action
("Contact", "Home") + "?
Category=Electronics");
```

In the preceding statement, the string, `Electronics`, is stored in the variable, `Category`, and is sent to the `Contact` action method of the `Home` controller. The `Contact()` action method can retrieve the information stored in the variable, `Category`, by using the following statement:

```
String cat = Request.QueryString
["Category"];
```

The preceding statement stores the string, `Electronics`, in the `cat` variable.

Instead of using the preceding statement, the `Contact()` action method can also accept the query string as a parameter, as shown in the following code snippet:

```
public ActionResult About(string
Category)
{
  String cat = Category;
  ...
  return View();
}
```

You can pass multiple values from one page to another by using the query string. For this, the values must be separated with the ampersand (`&`) symbol, as shown in the following statement:

```
Response.Redirect(Url.Action
("Contact", "Home") + "?
Category=Electronics&Description=Affor
dable");
```

NOTE *There should be no blank space before and after the ampersand (&) symbol.*

In the preceding statement, the strings, `Electronics` and `Affordable`, are stored in the variables, `Category` and `Description`, respectively. The information stored in these variables can be retrieved by using the following code snippet:

```
String Cat = Request.QueryString
["Category"];
String Det = Request.QueryString
["Description"];
```

The preceding code snippet stores the strings, `Electronics` and `Affordable`, in the variables,`Cat` and `Det`, respectively.

Advantages and Disadvantages of QueryString

Query strings provide the following advantages:

- ❑ A query string is contained in the HTTP request for a specific URL. Therefore, there is no server resource involved in storing a query string.
- ❑ All browsers support query strings.

Query strings have the following limitations:

- ❑ The information in a query string is directly visible to the users in the browser window. Therefore, query strings are not useful for storing sensitive data.
- ❑ Some browsers impose a limit of 2083 characters on the URL length. This restricts the amount of information that you can pass from one page to another by using query strings.

TempData

TempData is a dictionary for storing temporary data in the form of key-value pairs. It can be used to store values between one request and another. Values can be added to TempData by adding them to the TempData collection. This information is preserved for a single request only and is designed to maintain data across a Web page redirect. Any data stored in TempData will be available during the current and subsequent requests only, or until the item is removed explicitly. This is useful in a situation when you want to pass data to a view during a Web page redirect.

For example, consider the following code snippet:

```
public class HomeController :
Controller
{
public ActionResult Index()
{
  TempData["text"] = "Hello";
  return RedirectToAction("About");
}
public ActionResult About()
{
  return View();
}
  }
```

In the preceding code snippet, if you send a request for the `Index()` action method, the value, Hello, is stored in the `TempData` key named `text`, and you are automatically redirected to the `About()` action method, which renders a view.

Now, consider the following code included in the view associated with the `About()` action method:

```
@{var msg = TempData["text"] as
String;}
@msg
```

The preceding view is able to access the data stored for it in `TempData` and display it to the user. However, if you now navigate to another view, the data stored in `TempData` by the `Index()` action method will not be available there.

Advantages and Disadvantages of TempData

TempData provides the following advantages:

- TempData can be used to pass an error message to an error page easily.
- TempData is used to pass the data from the current request to the subsequent request in case of redirection.

TempData has the following limitations:

- Its life is short and it lives only until the time the target view is fully loaded.
- It can be used to store only one time messages, such as error messages and validation messages.

Application State

ASP.NET provides application state as a means of storing application-specific information. The information in the application state is stored as key-value pairs.

Application state is created the first time a user accesses any URL resource in an application. After the application state is created, the application-specific information is stored in it. The information stored in the application state is shared among all the pages and user sessions of the Web application by using the `HttpApplicationState` class. The `HttpApplicationState` class is accessed by using the `Application` property of the `HttpContext` object.

Application state variables are global for an ASP.NET application. However, the values stored in the application state variables are accessible only from the code running within the context of the originating application. Other applications running on the system cannot access or modify the values.

Application State and Application Events

The application state is generally initialized and manipulated on the occurrence of certain application events. These application events are:

- `Application.Start`: This event is raised when the application receives the first request for a Web page. It is also raised on the next request after the server, Web service, or website has been restarted. The event handler for this event usually contains code for initializing the application variables.
- `Application.End`: This event is raised when the server, Web service, or website is stopped or restarted. The event handler for this event usually contains code to clean up the resources that the application has used.
- `Application.Error`: This event is raised when an unhandled error occurs.

The handlers for the preceding events are defined in the `Global.asax` file. The code to manipulate the application state can be written inside these event handlers.

Storing Information in the Application State

You can add application-specific information to the application state by creating variables and objects and adding them to the application state. Variables and objects added to the application state are global to an ASP.NET application. In an ASP.NET application, the code for initializing application variables and objects is usually written within the `Application_Start()` function in the `Global.asax` file. For example, you can create the variable named `MyVariable` with the value, `Hello`, and store it in the application state by using the following statement:

```
HttpContext.Application ["MyVariable"]
= "Hello";
```

The value stored in this variable can now be accessed across all the pages and user sessions of the Web application.

Retrieving Information from the Application State

After the application in which you declared `MyVariable` is executed, any page in the application can retrieve the value of `MyVariable`. To read the value of `MyVariable`, you can use the following statement:

```
string val = (string)
HttpContext.Application["MyVariable"];
```

The preceding statement retrieves the value contained in the application variable, MyVariable, and stores it in the local variable named val.

Removing Information from the Application State

Apart from adding and retrieving variables and objects to the application state, you can also remove an existing object or variable from the application state. For example, you can remove the application variable, `MyVariable`, from the application state by using the following statement:

```
HttpContext.Application.Remove
("MyVariable");
```

You can also remove all the application state variables and objects by using the following statement:

```
HttpContext.Application.RemoveAll();
```

Once added, an object remains in the application state until the application is shut down, the `Global.asax` file is modified, or the item is explicitly removed from the application state.

Synchronizing the Application State

Multiple pages and sessions within an ASP.NET Web application can simultaneously access the values stored in the application state. This can result in conflicts and deadlocks. For example, you can add the variable named `PageCounter` in the application state to keep track of the number of times a page has been requested. If two users access a Web page simultaneously, there will be an attempt to update the value of the variable, `PageCounter`, by both the users, which could result in data inconsistency. To avoid such situations, the `HttpApplicationState` class provides two methods, `Lock()` and `Unlock()`. These methods allow only one thread at a time to access the application state variables and objects.

Each browser's request for a Web page initiates a new thread on the Web server.

Calling the `Lock()` method on an `Application` object causes ASP.NET to block attempts by the code running on other worker threads to access anything in the application state. These threads are unblocked only when the thread that called the `Lock()` method calls the corresponding `Unlock()` method on the `Application` object. Consider the following code snippet that locks and unlocks the variable, `PageCounter`:

```
HttpContext.Application.Lock();
if (HttpContext.Application
["PageCounter"] == null)
{
HttpContext.Application["PageCounter"]
=1;
}
else
{
HttpContext.Application["PageCounter"]
=(int) HttpContext.Application
["PageCounter"
]+1;
}
HttpContext.Application.UnLock();
```

In the preceding code snippet, the `Lock()` method is called before accessing the value of the `PageCounter` variable. This ensures that the variable cannot be modified simultaneously by any other thread. After calling the `Lock()` method, the value of the `PageCounter` variable is modified. Once modified, the `UnLock()` method is called to release the imposed lock on the`PageCounter` application state variable. Then, the variable can be modified by other threads.

The `Lock()`method locks all the items in the application state object. In other words, you cannot selectively lock items in the application state.

Advantages and Disadvantages of Application State

The application state has the following advantages:

- ❑ The application state is easy to use and consistent with other .NET Framework classes.
- ❑ Storing information in the application state involves maintaining only a single copy of the information.

The application state has the following limitations:

- ❑ The data stored in the application state is lost when the Web server containing the application state fails due to server crash, upgrade, or shutdown.
- ❑ The application state requires server memory and can affect the performance of the server and the scalability of the Web application, if it occupies excess memory space.

You can implement the application state to increase the performance of the Web application. For example, storing relatively static data that is required across pages/sessions in the application state can enhance the performance of the Web application by reducing the overall number of data requests to a database server. However, it is important to remember that application state variables, which contain large blocks of information, can reduce the Web server performance as the server load increases. In addition, the memory occupied by a variable stored in the application state is not released until the value is explicitly removed or replaced. Therefore, it is preferred to use the application state variables with small and less frequently-changed data.

Session State

In ASP.NET, session state is used to store session-specific information for a Web application. Unlike application state, the scope of session state is limited to the current browser session. If multiple users are accessing a Web application, each user will have a different session state. If a user exits from a Web application and returns later, the user will have a different session state.

Session state is structured as a key-value pair for storing session-specific information that needs to be maintained between server round trips and requests for pages.

The session state has a built-in support in ASP.NET. The built-in session state feature automatically performs the following actions:

- ❑ Identify and classify requests coming from a browser into a logical session on the server.
- ❑ Store session-specific data on the server.
- ❑ Raise session lifetime-related events, such as `Session.Start` and `Session.End`, which can be handled in the `Global.asax` file.
- ❑ Automatically release session data if a browser does not access an application within a specified timeout period.

When a user first requests a page from an ASP.NET website, the website automatically adds a session cookie to the client browser to store the current session ID. This cookie is sent to the server along with all the subsequent requests within the session.

Identifying a Session

Each active ASP.NET session is identified and tracked by a unique session IDstring containing American Standard Code for Information Interchange (ASCII) characters. The session ID strings are communicated across client/server requests by using either an HTTP cookie or a modified URL containing the embedded session ID string. The `SessionStateModule` class is responsible for generating or obtaining session ID strings.

Similar to the application state, you can store objects and variables in the session state. By default, a session variable is active for 20 minutes without any user interaction. In an ASP.NET Web application, the objects stored in the session state are stored on the server.

Using the Session State

The method of adding and retrieving items from the session state is basically the same as that of adding and retrieving items from the application state. The following statement adds the variable, `MyVariable`, with the value, `HELLO`,in the session state:

```
Session["MyVariable"]="HELLO";
```

You can retrieve the value of the variable, `MyVariable`, by using the following statement:

```
string val = (string) Session
["MyVariable"];
```

The preceding statement retrieves the value of `MyVariable` and stores it in the variable, `val`.

You need to consider the following issues when adding variables and objects to the session state:

- ❑ Any variable or object that you add to the session state is available until the user closes the browser window. The variables and objects are automatically removed from the session state if the user does not request a page for more than 20 minutes, which is the default duration for session timeout.
- ❑ Any variable or object added to the session state is related to a particular user. For example, you can store different values for `MyVariable` for two different users accessing the Web page, and users can access only the value that is assigned to the copy of the session variable associated with them.

Session state is global to the entire application for the current user. Session state is not lost if the user revisits a Web page or visits a different page by using the same browser window. However, session state can be lost in the following ways:

- ❑ When the user closes and restarts the browser.
- ❑ When the user accesses the same Web page through a different browser window.
- ❑ When the session times out because of inactivity.
- ❑ When the `Session.Abandon()` method is called within the Web page code.

Similar to application state, you can remove an object or a variable added to the session state by using the`Remove()` method or the `RemoveAll()` method based on the requirement.

Handling the Session Events

Session state is generally initialized and manipulated on the occurrence of certain events. These events are:

- ❑ `Session.Start`: This event is raised when a user requests the first page from a website and is useful for initializing session variables.
- ❑ `Session.End`: This event is raised when a user session expires or when the `Session.Abandon()` method is called.

The handlers for the preceding events are defined in the `Global.asax` file. The code to initialize and manipulate session state can be written inside these event handlers.

Configuring the Session State

Session state can be configured through the `Web.config` file for the application. This file allows you to set the advanced options, such as the timeout and the session state mode.

The following markup shows some important options that can be set for the `<sessionState>` element:

```
<?xml version="1.0" encoding="utf-8" ?
>
<configuration>
<system.web>
<sessionState
  cookieless="UseCookies"
cookieName="ASP.NET_SessionID"
  timeout="20"
  mode="InProc" />
```

```
</system.web>
</configuration>
```

The preceding markup uses the following session state attributes:

- cookieless
- timeout
- mode

The cookieless Attribute

The `cookieless` attribute specifies whether a cookie should be used or not. The following table lists the values that the `cookieless` attribute can have, along with their description.

Value	***Description***
`UseCookies`	*Specifies that cookies are always used.*
`UseUri`	*Specifies that cookies are never used and the session ID is inserted in the URL that goes to the server.*
`UseDeviceProfile`	*Specifies ASP.NET to choose whether to use cookies by examining the BrowserCapabilities object. It does not take into account that the user may have disabled cookies in a browser that supports them.*
`AutoDetect`	*Specifies that cookies should be used if the browser supports them. ASP.NET determines whether the browser supports cookies by attempting to set and retrieve a cookie.*

The Values of the cookieless Attribute

The timeout Attribute

The `timeout` attribute is an important attribute of the `<sessionState>` element. It specifies the number of minutes that ASP.NET will wait, without receiving a request, before abandoning the session.

The value for the `timeout` attribute should be set in such a way that it is short enough to allow the server to reclaim valuable memory after a client stops using the application. At the same time, the value should be long enough to allow a client to pause and continue a session without losing it.

The `timeout` attribute can also be set programmatically, as shown in the following statement:

```
Session.Timeout = 15;
```

The mode Attribute

The `mode` attribute specifies where the session state values are to be stored. The `mode` attribute can have the following values:

- `Custom`: Specifies a custom data store to store the session-state information.
- `InProc`: Specifies that the information is to be stored in the same process as the ASP.NET application. This is the only mode in which the `Session.End` event is raised.
- `Off`: Specifies that the session state is disabled.
- `SQLServer`: Specifies that the session state is to be stored by using an out-of-process SQL Server database to store the state information.
- `StateServer`: Specifies that the session state is to be stored in an out-of-process ASP.NET state service running on the Web server or a dedicated server. It does not store information on Internet Information Services (IIS).

Advantages and Disadvantages of Session State

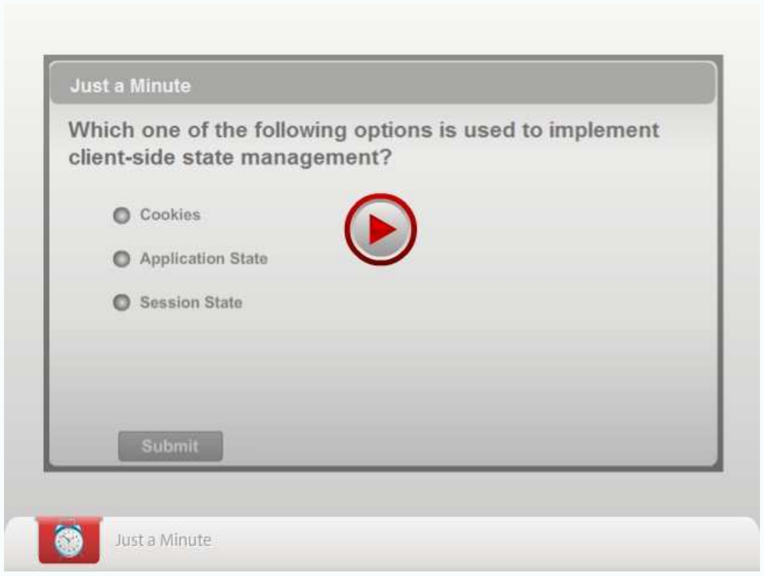
Session state has the following advantages:

- Access of data is fast as it stores session data in a memory object of the current application domain.
- It ensures platform scalability as it can work in a multiprocess configuration.

Session state has the following limitations:

- As the session state data is stored in server memory, it is not advisable to use session state when you are working with a large volume of data.
- It affects the performance of memory, because the session state variable stays in memory until you destroy the state.

NOTE *In addition to the state management techniques discussed here, there are various other techniques, such as View State, Control State, and Hidden Fields. These techniques are supported by ASP.NET Web forms applications and are not available in ASP.NET MVC applications.*


Just a Minute

Which one of the following options is used to implement client-side state management?

- Cookies
- Application State
- Session State

Submit

Just a Minute

Activity 9.1: Implementing State Management

Optimizing the Performance of a Web Application

Nowadays, Web applications are used for various purposes, such as discussion forums, online shopping, and social networking. The performance of such Web applications is dependent on how quickly they respond to the user request.

To improve the overall performance of your Web application, you need reduce the number of server interactions with the database. ASP.NET MVC provides the various techniques to optimize the performance of a Web application. Caching is one of the techniques that can be used to optimize the performance of a Web application. Caching reduces the number of server interactions with the database by storing the data in the cache memory for a specific period of time.

You can also improve the performance of a Web application by reducing the number of requests and the size of the requests between the client and the server. This can be done by techniques, such as bundling and minification. In addition, you can adopt some best practices to further optimize the performance of a Web application.

Enhancing Performance by Using Output Caching

Consider the scenario of the Newbay application where you want the application to display the latest products available with their prices on the Home page. For this, you need to retrieve the updated list of products from the database so that your Web page always displays the current list of products with their prices.

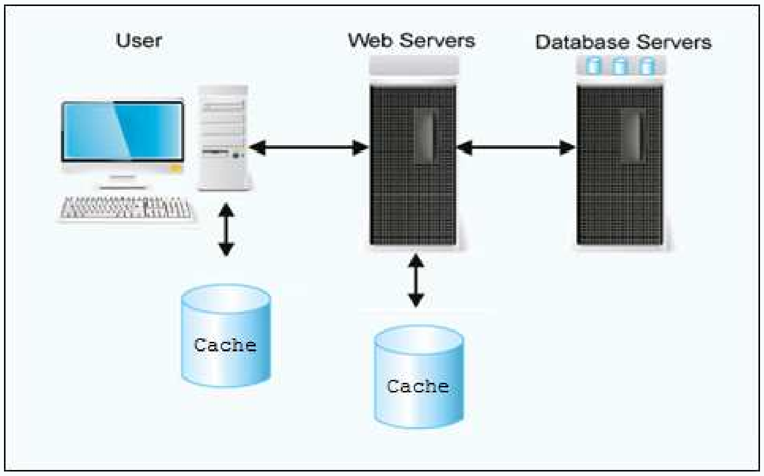
In such a situation, where you frequently need to query the database for the updated list, the interaction with the database becomes one of the most important factors that can affect the performance of an application. This is because executing a query against a database takes a substantial amount of time. Moreover, the same data may need to be fetched multiple times from the database, when the data is requested by multiple clients.

In such a situation, you can use the output caching technique to improve the performance of a Web application. Output caching allows you to cache the content returned by a controller action. As a result, the same content does not need to be generated each time the controller action is invoked.

Before implementing output caching, you need to consider the various factors, such as:

- Identify whether output caching is relevant for your application. If the content of your application changes frequently, output caching may not be useful.
- Identify the duration for which the output must be cached. This would depend on how frequently data associated with the output changes.
- Identify the location where you need to cache the output. You can cache the output on locations, such as client machines, the Web server, or both.

The following figure depicts the functionality of caching.



The Functionality of Caching

Caching is an important and extensively-used concept to enhance the application performance. The major benefits of using caching are:

- **Reduced access time**: When most of the requests are processed by the cache itself and need not be sent to the database server, the access time of the Web page is reduced significantly.
- **Less bandwidth required**: Caching can be

implemented on the client computer or on the server. If caching is implemented at the client computer, most of the requests and responses do not require data transfer over the network between the client and the server. This, in turn, reduces the bandwidth required for the communication. Implementing caching at the server side will also reduce the bandwidth requirement as the data transfer between the server and the database would be reduced.

- **Lesser load on server**: Caching avoids executing the same code repetitively to create the same result. This reduces the CPU usage at the server, and in the process, lowers the load on the server.

Output caching improves the performance of an ASP.NET MVC application by enabling you to cache the pages of the application for a specified duration. When the user requests the same page within the specified period of time, instead of rerendering the page, ASP.NET MVC serves the cached page to the user.

In an ASP.NET MVC application, you can implement caching at the controller action level. You can apply the output cache attribute to a controller action or the entire controller you decide to cache. By doing this, you can cache the result returned from a controller action. This will result in getting the same content each and every time a new user invokes the same action. The following code snippet shows how to enable caching by adding an output cache attribute to a controller action.

```
public class HomeController :
Controller
{
   [OutputCache]
   public ActionResult Index()
   {
   //some code
   }
}
```

In the preceding code snippet, the `[OutputCache]` attribute is added to the `Index` action method of the `Home` controller.

Configuring the Cache Duration and Location

Consider the example of the Newbay store website where the Products page gets the details of the products from a database table. In this case, if there are a thousand users requesting for the same page at the same time, the Web server needs to raise a thousand queries to the same database to produce the same output.

You can avoid such a situation by caching the output for a specified duration. The duration for which a Web page is cached will depend on the frequency at which the underlying data changes. If the data rarely changes, the Web page can be cached for a long duration. However, if the data associated with the Web page changes too frequently, caching the Web page for too long may result in serving stale data to the users.

Therefore, if the data associated with a particular Web page changes frequently and there are thousands of users requesting the same page per second, you can cache the output for a couple of seconds. Caching the output for a small duration of time is called micro caching. When the traffic on the Web application is high, micro caching significantly reduces the number of queries to access the database.

So far, you have seen how to enable output caching by adding the `[OutputCache]` attribute either to an individual controller action or to an entire controller class. However, you can specify the duration and location of the cache attribute. In order to see the effect of the output caching, consider the following code snippet:

```
public class HomeController :
Controller
 {
 [OutputCache(Duration=3)]
 public ActionResult Index()
 {
   ViewBag.Message="This page was
 rendered at " + DateTime.Now;
    return View();
 }
 }
```

In the preceding code snippet, the output is cached for 3 sec. This means that the time displayed on the Web page will get refreshed every 3 seconds. However, if you want to cache the output of a controller action for one day, you can specify the cache duration of 86400 seconds.

If you invoke the `Index()` action multiple times by entering URL/ Home/ Index in the address bar of your browser and hitting the Refresh/ Reload button in your browser repeatedly, the time displayed by the Index view would not change for 3 seconds. The same time is displayed because the view is cached. This view is cached for everyone who visits your application. Anyone who invokes the `Index()` action will get the same cached version of the Index view. This means that the amount of work that the Web server must perform to serve the Index view is dramatically reduced.

In addition to the duration of the cache, you can also define the location where you want the data to be cached. When you use the `[OutputCache]` attribute, content is cached at three locations: the Web server, any proxy server, and the Web browser. However, you can set your own location by modifying the `Location` property of the `[OutputCache]` attribute. The `[OutputCache]` attribute has the `Location` property that can be used to specify the location of the output to be cached. The `Location` property can take the following values:

- `Any`: The output cache can be located on the browser, a proxy server participating in the request, or the server where the request was processed.
- `Client`: The output cache is located on the browser where the request originated.
- `Downstream`: The output cache can be stored in any HTTP cache-capable device other than the original server, such as the proxy servers and the client that made the request.
- `Server`: The output cache is located on the Web server where the request was processed.
- `None`: The output cache is disabled for the requested page.

NOTE *In an enterprise environment, client machines generally obtain access to the Internet through a proxy server. A proxy server is a server that sits between a client application, such as a Web browser and the Internet. When a client requests an online resource, the request is intercepted by the proxy server, and then forwarded to the target Web server.*

By default, the `Location` property has the value, `Any`. However, there are situations in which you might want to cache content only on the browser or the server.

For example, if you are caching information that is personalized for each user, you should not cache the information on the server. However, you might want to cache the personalized content in the browser cache to improve performance. If you cache content in the browser, and a user invokes the same controller action multiple times, the content can be retrieved from the browser cache instead of the server. In order to see the effect of setting the `Location` property, consider the following code snippet:

```
public class UserController :
Controller
  { [OutputCache
(Duration=36000,Location=System.Web.UI
.OutputCacheLocation.Client)]
   public string GetOrderDetails()
   {
//Code to display order details
   }
 }
```

In the preceding code snippet, the `[OutputCache]` attribute helps in caching the output of the `GetOrderDetails()` action method. The `Location` property is set to the value, `System.Web.UI.OutputCacheLocation.Client`. This helps in caching the content only on the browser in such a way that when multiple users invoke the `GetOrderDetails()` method, each person gets its own order details.

Varying the Output Cache Based on Request Parameters

Consider the scenario of Newbay where the Product page displays the list of products under the various categories. When you select a product, you get the details of that particular product. If you cache the details page, the details for the same product will be displayed even if you select a different product. The first product selected by the first user will be displayed to all the future users.

To avoid such a situation, you can take advantage of the `VaryByParam` property of the `[OutputCache]` attribute. This property enables you to create different cached versions of the same content. For example, consider the `Product()` action method that returns a list of products, and the `Details()` action method that returns the details for the selected product, as shown in the following code snippet:

```
[OutputCache(Duration=3600,
VaryByParam="none")]
public ActionResult Product()
{
ViewData.Model = (from m in
db.products select m).ToList();
return View();
}
[OutputCache(Duration=3600,
VaryByParam ="id")]
public ActionResult Details (int id)
{
ViewData.Model =
db.products.SingleOrDefault(m => m.Id
== id);
  return View();
}
  }
}
```

In the preceding code snippet, the `Product()` action includes the `VaryByParam` property with the value, `none`. The `VaryByParam` attribute specifies how many copies of the page need to be saved. In this example, the value of the `VaryByParam` attribute is set to `none`, which indicates that only one copy of the page is to be cached. When the `Product()` action is invoked, the same cached version of the Product view is returned. The `Details()` action includes the `VaryByParam` property with the value, `id`. The `SingleorDefault` extension method returns the only element of the sequence that satisfies a specified condition or a default value if no such element exists. When different values of the `id` parameter are passed to the controller action, different cached versions of the Details view are generated. In addition, you can set the `VaryByParam` property to the following values:

- `*`: Create a different cached version whenever a form or query string parameter varies.
- `None`: Never create different cached versions.
- `Semicolon separated list of parameters`: Create different cached versions whenever any of the specified form or query string parameters in the list varies.

Output Cache Profile

So far you have seen how to configure output cache properties by modifying properties of the `[OutputCache]` attribute. However, you can create a cache profile in the `Web.config` file.

Creating a cache profile has the following advantages:

- You can apply the cache profile to several controllers and controller actions and control how they cache contents from one central location.
- You can modify the `Web.config` file without recompiling the application.

For example, to create a cache profile, you need to include the following code snippet in the `<system.web>` section of a Web configuration file:

```
<caching>
<outputCacheSettings>
  <outputCacheProfiles>
 <add name="Cache1Hour" duration="3600"
 varyByParam="none"/>
  </outputCacheProfiles>
</outputCacheSettings>
</caching>
```

In the preceding code snippet, the `Cache1Hour` profile is applied to a controller action with the `[OutputCache]` attribute. For example, in order to implement the `Cache1Hour` profile to the `Index()` action method of the `Home` controller, consider the following code snippet:

```
public class HomeController :
Controller
{
   [OutputCache
(CacheProfile="Cache1Hour")]
   public ActionResult Index()
   {
   //some code
   }
}
```

In the preceding code snippet, the `Cache1Hour` profile is applied to the `Index()` action method of the `Home` controller.

Donut Hole Caching

So far, you have seen how to cache an entire Web page. However, there are situations where you need to cache only a part of the page. In most of the Web applications, there are parts of pages that do not change. The donut hole caching is useful in scenarios where most of the elements in your page are dynamic, except few sections that rarely change but are expensive to generate. The most common example is a dynamic navigation structure that has some associated values in the database.

Consider a scenario where the Product categories are retrieved from the database and shown as links on each and every page of the Newbay Web application. Retrieving the product categories from the database and rendering them on a view as links is a time-consuming activity. Moreover, this activity needs to be repeated for every page in the Web application. Therefore, the code to retrieve the categories needs to be repeated in each and every controller action in the Web application.

To avoid these problems, you can retrieve the categories just once and cache the resulting HTML by using donut hole caching.

You can implement donut hole caching by making use of the child action methods provided by ASP.NET MVC. A child action method is a controller action that is only meant to be invoked during the execution of other controller actions. Child actions act as any other controller action and their output can be cached as well.

In order to implement donut hole caching, you need to create a `Controller` action and include the logic, as shown in the following code snippet:

```
[ChildActionOnly]
[OutputCache(Duration=60)]
public ActionResult Categories()
{
 var db = new ProductDBContext();
var Categories = db.Products.select
(p=>p.Category).Distinct();
ViewBag.Categories =
Categories.ToArray();
return PartialView();
}
```

NOTE *In the preceding code snippet, the Distinct() method used in the LINQ query to retrieve category details ensures that only distinct categories are retrieved.*

In the preceding code snippet, the `[ChildActionOnly]` attribute creates a child action that returns the product categories from the database. It then returns the partial view named Categories that contains the following markup:

```
<ul id="navigation">
 @foreach (string category in
 ViewBag.Categories)
 {
     <li> @Html.ActionLink(category,
 category, "Products")</li>
 }
</ul>
```

You can then use the `Html.Action()` helper method in any view to tell ASP.NET MVC to execute the child action method and render the partial view associated with it at a particular place within the view.

For example, consider the following code snippet written in a view which calls the child action method, `Categories()`.

```
@Html.Action("Categories", "Home")
```

In the preceding code snippet, `Home` is the name of the controller that contains the child action method, `Categories`. Including the `Action()` helper method in each view of your Web application will ensure that links corresponding to all the product categories appear on all the views. In addition, as the `[OutputCache]` attribute has been specified for the child action method, once the product categories are retrieved from the database and the partial view has been generated, the HTML for the partial view will be cached and used for all the subsequent requests to the website.

Enhancing Performance by Using Data Caching

Most Web pages in a Web application typically contain dynamic data that needs to be retrieved from a database. However, executing a database query each time a user places a request can slow down the performance of the application. This problem can be solved by using a concept called data caching.

Data can be cached in an ASP.NET MVC application by using the `MemoryCache` class.

Consider the following code snippet that illustrates how to cache data in an application by using the `MemoryCache` class:

```
Customer dtCustomer =
System.Runtime.Caching.MemoryCache.Def
ault.
AddOrGetExisting("CustomerData",
getCustomer(),
System.DateTime.Now.AddHours(1));
```

In the preceding example, the `AddOrGetExisting()` function enables the application to refresh and retrieve data in one line of code. This function retrieves the data from the cache, if the cache contains the relevant data. However, if the cache does not contain the relevant data, the `AddOrGetExisting()` function allows adding the data to the cache by invoking the `getCustomer()` function.

The first parameter of the `AddOrGetExisting()` function specifies the unique identifier of the object that needs to be stored in the memory cache. The second parameter specifies the object that needs to be stored in the cache, and the third parameter specifies the time when the cache should expire.

The data cache may fetch the outdated content, which is stored in memory, instead of taking the latest from the database. To overcome this problem, you can reduce the amount of time for storing data in the cache. This helps in providing updated content to the users.

Enhancing Performance by Using HTTP Caching

You can implement HTTP caching in the browser cache and the proxy cache. Storing data in the browser cache helps remove the need to repeatedly download content from the server. Web browsers frequently check content for updates. If the content is updated in the server, Web browsers download the content from the server to attend to user requests. Otherwise, Web browsers render content from the local cache.

The functionality of the proxy cache is similar to the functionality of the browser cache. However, many users can access the cache in a proxy server, while only one user can access the browser cache at a time.

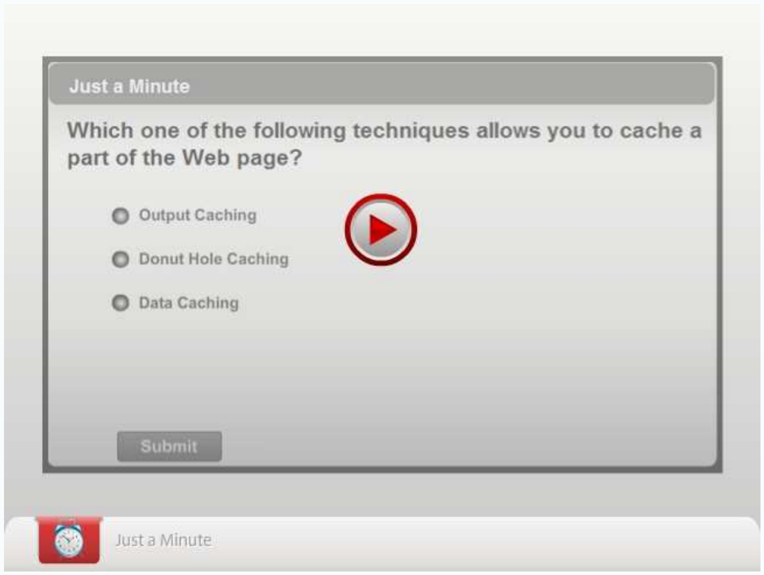
Preventing Caching

Consider a Web application that involves frequent content updates. If you cache the output of such Web applications, it will prevent the users from viewing the content updates. In order to overcome this problem, you can implement an HTTP header called Cache-Control.

The Cache-Control header indicates to the Web browser how to handle the local cache or proxy server cache. All HTTP clients, such as browsers and proxy servers, respond to the instructions provided in the Cache-Control header to determine how to handle the local cache of a Web application.

You can use the `HttpCachePolicy.SetCacheability()` method to specify the value of the Cache-Control header. To prevent caching in your Web application, you should set the value of the Cache-Control header to `NoCache`, as shown in the following code snippet:

```
Response.Cache.SetCacheability
(HttpCacheability.NoCache);
```



Which one of the following techniques allows you to cache a part of the Web page?

- Output Caching
- Donut Hole Caching
- Data Caching

Activity 9.2: Optimizing Performance of a Web Application

Introduction to Bundling and Minification

Bundling and Minification are the two techniques provided by ASP.NET MVC, which improve the load time of a Web page by reducing the number of requests to the server and the size of the requested resource, such as CSS or JavaScript.

Bundling

Bundling is a technique that allows you to combine multiple files, such as CSS and JavaScript, into a single file. Combining such files into a single file reduces the number of requests between the client and the server. Most modern browsers restrict the number of open connections to any given hostname to six, which is independent of the type of files getting downloaded. This means that while six requests are being processed, any additional requests for resources on a host will need to wait until another request completes.

When there are a number of files, such as style sheets and scripts, which need to be downloaded, at times, a lot of time is spent in waiting for a request to complete, while the actual download does not take much time. You can solve this problem by using bundling. Bundling allows you to combine multiple files into a bundle that can be downloaded as a single entity.

In order to implement bundling in a Web application, you must define your bundles or files that you would like to combine together. In an ASP.NET MVC application, this happens in the `BundleConfig` class in the `App_Start` folder. You need to create a bundle within the `RegisterBundles()` method present in the `BundleConfig` class, as shown in the following code snippet:

```
public static void RegisterBundles
(BundleCollection bundles)
     {
bundles.Add(new ScriptBundle("~/
bundles/jqueryval").Include(
        "~/Scripts/jquery.unobtrusive*",
        "~/Scripts/jquery.validate*"));
bundles.Add(new StyleBundle("~/
Content/css").Include("~/Content/
site.css", "~/Content/styles.css"));
}
```

In the preceding code snippet, the `RegisterBundles()` method is used to create, register, and configure bundles. A new instance of the `ScriptBundle` class has been created to create a bundle of scripts. The virtual path, ~/bundles/jqueryval, has been assigned to the bundle. The `Include()` method is used on the bundle object to specify the files that should be included in that bundle. * is used to indicate that the bundle should include all the script files in the Scripts folder whose name begins with `jquery.unobtrusive` or `jquery.validate`.

Similarly, a new instance of the `StyleBundle` class has been created to create a bundle of styles. The virtual path,`~/Content/css`, has been assigned to the bundle. The`Include()` method is used on the bundle object to specify the files that should be included in that bundle.

In order to include these bundled files in a view/layout, you can use the `@Styles.Render` and `@Scripts.Render` methods in the view or layout file. Then, you need to provide the virtual path that you configured in the bundle configuration as a parameter to these methods, as shown in the following code snippet:

```
@Scripts.Render("~/bundles/jqueryval")
@Styles.Render("~/Content/css")
```

Minification

Minification is a technique that helps in reducing the size of a file by removing unnecessary whitespaces and comments, and shortening the variable names. Minified code is especially useful for interpreted languages, such as JavaScript, which are deployed and transmitted over the Internet because it reduces the amount of data that needs to be transferred.

By implementing minification, you can drastically reduce the size of a file, thereby reducing the time required to get it from the server and load it into the browser.

Best Practices for Optimizing Performance

While building an ASP.NET MVC application, there are different approaches which can be used to improve the performance of the application. Some of the most commonly-used approaches are:

- ❑ Make fewer HTTP requests.
- ❑ Use Content Delivery Network (CDN).
- ❑ Make scripts and styles external.
- ❑ Minify Javascript and CSS.
- ❑ Avoid redirects.
- ❑ Compress HTTP responses.

Make Fewer HTTP Requests

To optimize the performance of your Web application, you need to send fewer HTTP requests between the client and the server. You can reduce the number of HTTP requests by reducing the number of components that are required to render a page, such as images, style sheets, scripts, and flash. Such components can be combined into a single file. For example, you can combine all the scripts into a single script and, similarly, combine all CSS files into a single style sheet.

Use a Content Delivery Network

To enhance the performance of a Web application, you can host the content used by your Web application, such as images, style sheets, and scripts, on a Content Delivery Network (CDN). A CDN is a collection of Web servers distributed across multiple locations. CNDs help deliver content more efficiently to the users. This is because a user's request is served from a server with the fewest network hops or the fastest response time.

Make Scripts and Styles External

JavaScript and CSS that are embedded in HTML documents get downloaded every time the HTML document is requested. This reduces the number of HTTP requests that are needed, but increases the size of the HTML document. To overcome this problem, you can put JavaScript and CSS in external files. Using external files generally produces faster page loads because the JavaScript and CSS files are cached by the browser. Thus, it reduces the size of the HTML document without increasing the number of HTTP requests.

Minify Javascript and CSS

Minification is the practice of removing unnecessary comments, white spaces, and characters from the files to reduce their size, thereby improving the load time of the Web page. JavaScript and CSS can be minified to improve the response time of the Web application.

Avoid Redirects

Redirects help user to point to a new page from an old page. In addition, it guides the user to a correct page. However, inserting a redirect slows down user experience. This is because each redirect creates an additional HTTP request and, thus, increases the round-trip time.

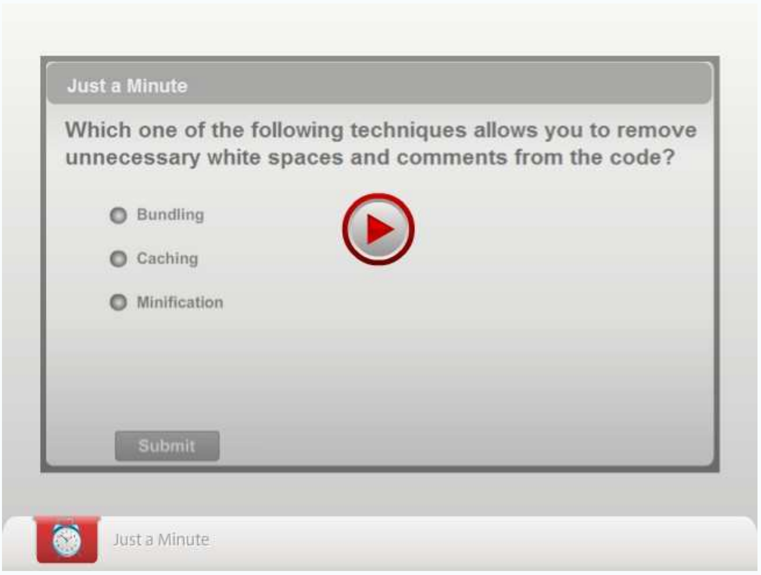
Compress HTTP Responses

The response time of a Web application can be improved by using compression to reduce the size of the HTTP response that is sent to the client. However, this can work only if the client provides support for compression.

When a Web browser sends a request to the server, it indicates the support for compression in its Accept-Encoding header, as shown in the following example:

```
Accept-Encoding: gzip, deflate
```

If the Web server receives this header in the request, it can send a compressed HTTP response to the client. For compressing the response, it can use one of the methods, such as gzip or deflate, as indicated by the client.



Which one of the following techniques allows you to remove unnecessary white spaces and comments from the code?

- Bundling
- Caching
- Minification

Summary

In this chapter, you learned that:

- ❑ Some options available to implement state management are:
  - Cookies
  - QueryString
  - TempData

    - Application State
    - Session State
- ❑ Cookies are small pieces of information that are stored on the client’s computer.
- ❑ Cookies can be either temporary or persistent.
- ❑ A query string provides a simple way to pass information from one page to another.
- ❑ TempData is a dictionary for storing temporary data in the form of key-value pairs.
- ❑ Application state is created the first time a user accesses any URL resource in an application.
- ❑ Session state is structured as a key-value pair for storing session-specific information that needs to be maintained between server round trips and requests for pages.
- ❑ Caching is an important and extensively-used concept to enhance the application performance.
- ❑ You can implement donut hole caching by making use of the child action methods provided by ASP.NET MVC.
- ❑ Data can be cached in an ASP.NET MVC application by using the MemoryCache class.
- ❑ You can implement HTTP caching in the browser cache and the proxy cache.
- ❑ Bundling is a technique that allows you to combine multiple files, such as CSS and JavaScript, into a single file.
- ❑ Minification is a technique that helps in reducing the size of a file by removing unnecessary whitespaces and comments, and shortening the variable names.
- ❑ While building an ASP.NET MVC application, there are different approaches which can be used to improve the performance of the application. Some of the most commonly-used approaches are:
    - Make fewer HTTP requests.
    - Use Content Delivery Network.
    - Make scripts and styles external.
    - Minify Javascript and CSS.
    - Avoid redirects.
    - Compress HTTP responses.

Chapter 10

Controlling Access to a Web Application

At times, you need to ensure that some or all the pages of a Web application should be accessible only to specific users. In addition, the information on the Web application, which is intended for a specific user, should not be visible to other users. Therefore, before a user can access the restricted Web pages, you need to verify the identity of the user. You can accomplish this task by using a mechanism known as authentication.
Moreover, you need to verify the rights given to a user for accessing specific resources on the website. A Web application accomplishes this task by using a mechanism known as authorization.
This chapter discusses how to implement authentication and authorization in a Web application.

Objectives

In this chapter, you will learn to:

- ❑ Implement authentication
- ❑ Implement authorization

Implementing Authentication

Consider the scenario of the NewBay store website that allows users to view and buy products. On this website, anybody can view the products. However, to buy a product, the user must be a member of the website. In this case, a user who wishes to buy a product on the website needs to be verified. By using a mechanism known as authentication, the identity of the user can be verified. Authentication is the process of identifying an individual, usually based on the username and password provided by the user.

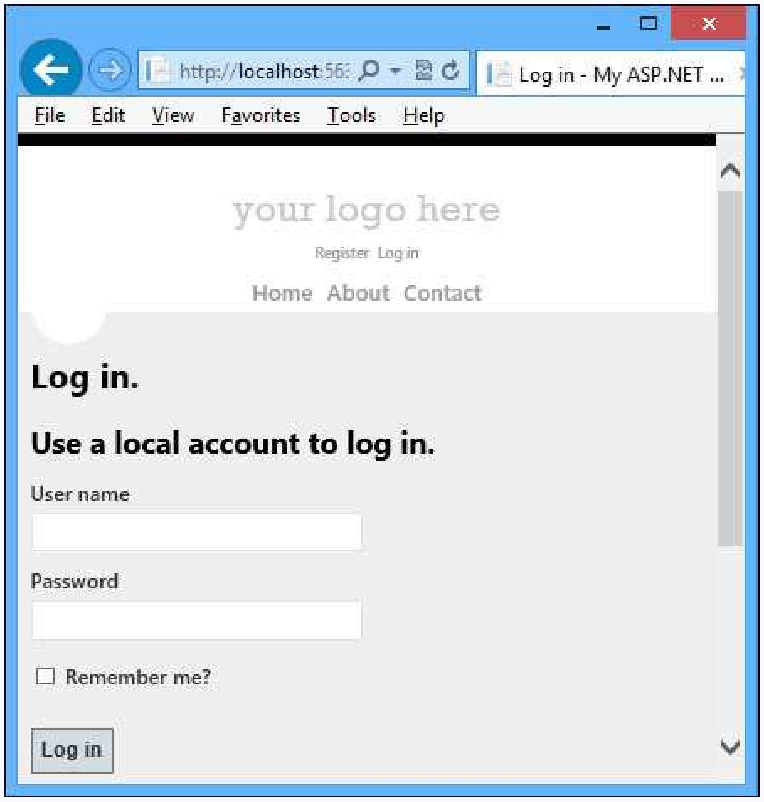
Using Authentication Modes

MVC allows you to implement authentication by using any of the following authentication modes:

- ❑ Forms Authentication
- ❑ Windows Authentication
- ❑ OpenID/OAuth

Forms Authentication

While using forms authentication, the website itself is responsible for implementing authentication. Here, the website can authenticate the users by providing a login form on the Web page, where the users can provide their username and password, as shown in the following figure.



A Login Form

In the preceding figure, when the users click the login button, the authentication mechanism ensures that the users have entered a valid username and password. Forms authentication uses a session cookie to authenticate user requests.
In websites that use forms authentication, if a user tries to access a restricted page, such as a Member page, ASP.NET redirects the user to the login page.
When you create an ASP.NET MVC application by using the **Internet Application** template, the MVC application implements the forms authentication and provides the basic functionality that allows users to authenticate themselves to the website. The **Internet Application** template uses the following statement between the `<system.web>` and `</system.web>` tags in the **Web.config** file to configure forms authentication:

```
<authentication mode="Forms">
</authentication>
```

In the preceding `<authentication>` element, the attribute mode is set to `Forms`. This specifies that the application uses forms authentication.
An application created using the **Internet Application** template allows the users to register and login to the website and enables them to change their password. This functionality comes out of the box and the code to implement this functionality is included in the `Account` controller and the corresponding views and models.
Forms authentication is the default authentication type

enabled for MVC applications created using the **Internet Application** template.

Windows Authentication

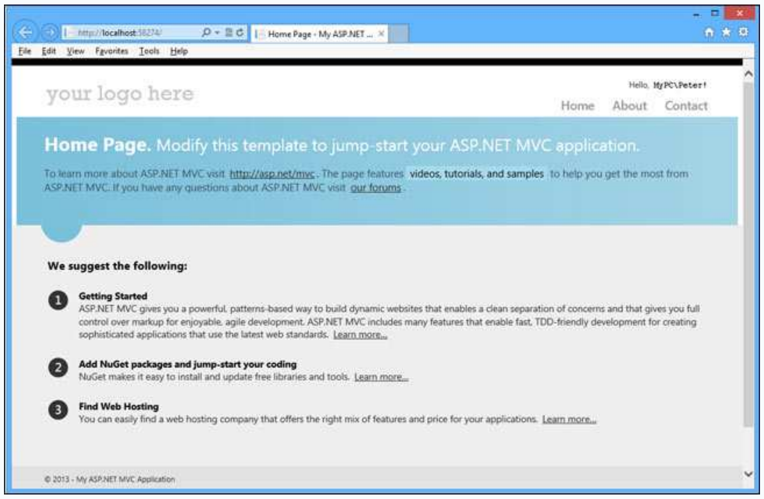
Windows authentication is typically used for intranet applications that run across a company's firewall. In Windows authentication, the users are allowed to use their standard Windows user name and password for accessing the website. Once the users are logged on to the desktop, Window automatically authenticates them to the application.

Visual Studio provides the **Intranet Application** template that implements Windows authentication. The **Intranet Application** template uses the following statement between the `<system.web>` and `</system.web>` tags in the **Web.config** file to configure Windows authentication:

```
<authentication mode="Windows" />
```

In order to use the **Intranet Application** template, you need to enable the **Windows authentication** property of the project.

When you run an application created by using the **Intranet Application** template, the Home page will be displayed, as shown in the following figure.



http://localhost:58274/
Home Page - My ASP.NET ...

File Edit View Favorites Tools Help

your logo here

Hello, MyPC\Peter!

Home About Contact

Home Page. Modify this template to jump-start your ASP.NET MVC application.

To learn more about ASP.NET MVC visit http://asp.net/mvc. The page features videos, tutorials, and samples to help you get the most from ASP.NET MVC. If you have any questions about ASP.NET MVC visit our forums.

We suggest the following:

1. **Getting Started**
   ASP.NET MVC gives you a powerful, patterns-based way to build dynamic websites that enables a clean separation of concerns and that gives you full control over markup for enjoyable, agile development. ASP.NET MVC includes many features that enable fast, TDD-friendly development for creating sophisticated applications that use the latest web standards. Learn more...
2. **Add NuGet packages and jump-start your coding**
   NuGet makes it easy to install and update free libraries and tools. Learn more...
3. **Find Web Hosting**
   You can easily find a web hosting company that offers the right mix of features and price for your applications. Learn more...

© 2013 - My ASP.NET MVC Application

The Home Page

In the preceding figure, the message, Hello MyPC\Peter!, refers to the Windows account name. This message is displayed by using the following code in the layout:

```
Hello, <span
class="username">@User.Identity.Name</
span>!
```

NOTE *The Windows account name displayed on the Home page will depend on the Windows account used for logging on to the machine.*

OpenID/OAuth

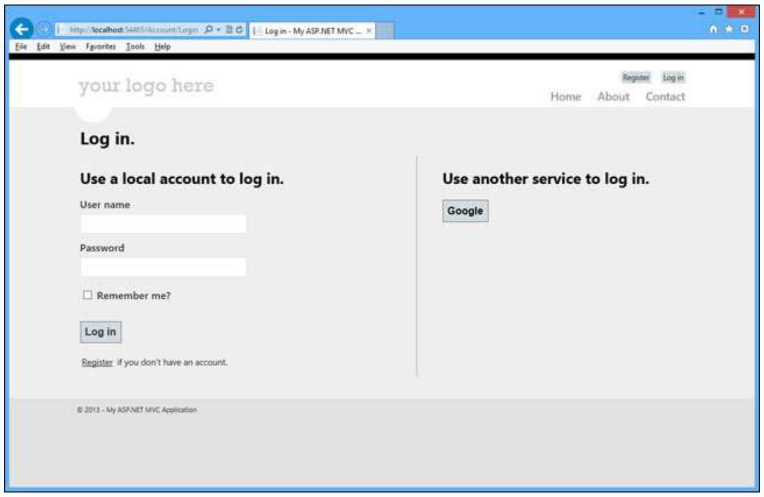
OAuth and OpenID are authorization protocols. They enable users to logon to a website by using the credentials for third party authentication providers, such as Google, Twitter, Facebook, and Microsoft.

When you create an MVC application using the **Internet Application** template, the third party authentication providers need to be configured in the **App_Start \AuthConfig.cs** file in the application folder. The **AuthConfig.cs** file contains the commented code, as shown in the following code snippet:

```
public static class AuthConfig
{
public static void RegisterAuth()
{
// To let users of this site log in
using their accounts from
// other sites such as Microsoft,
Facebook, and Twitter,
// you must update this site. For more
information visit
// http://go.microsoft.com/fwlink/?
LinkID=252166
//
OAuthWebSecurity.RegisterMicrosoftClie
nt(
// clientId: "",
// clientSecret: "");
//
OAuthWebSecurity.RegisterTwitterClient
(
// consumerKey: "",
// consumerSecret: "");
//
OAuthWebSecurity.RegisterFacebookClien
t(
// appId: "",
// appSecret: "");
//
OAuthWebSecurity.RegisterGoogleClient
();
}
}
```

The preceding code is commented because before you can use the services provided by the external login providers, you have to register the application with them. For example, if you want to use Facebook as an external service provider, you first need to register the application with Facebook. Once you register the application, Facebook provides you appId and appSecret. Then, you can use appId and appSecret to register yourself as a Facebook client. Similar to Facebook, you can use the services provided by Microsoft and Twitter. However, to use the services of Google, you need not register. Here, you just need to uncomment the call to `OAuthWebSecurity.RegisterGoogleClient()` in the preceding code.

When you run the application and view the login page, the login page will be displayed, as shown in the following figure.


http://localhost:54465/Account/Login
Log in - My ASP.NET MVC ...
File Edit View Favorites Tools Help

your logo here

Register Log in

Home About Contact

Log in.

Use a local account to log in.

User name

Password

☐ Remember me?

Log in

Register if you don't have an account.

Use another service to log in.

Google

© 2013 - My ASP.NET MVC Application

The Login Page

When you click the Google button shown in the preceding figure, the login page of the Google website is displayed, which allows you to logon to the application.

Task 10.1: Using Windows Authentication

Task 10.2: Using Forms Authentication

Using the Authorize Attribute to Require Login

To ensure the security of a Web application, a user must be logged on to access specific URLs within the application. This can be done by using the `Authorize` action filter. Action filters are attributes that let you add behavior that can be executed either before an action method is called or after an action method is executed. The `Authorize` attribute is the default authorization filter provided with ASP.NET MVC. You can use the `Authorize` attribute to:

- ❑ Secure a controller action.
- ❑ Secure a controller.
- ❑ Secure an entire application.

Securing a Controller Action

Consider the example of the NewBay store website where you need to implement the functionality to display the list of products on the website. In addition, you need to enable a user to buy products online. Though any user can access the list of products on the website, yet only authenticated users should be allowed to buy a product online. To enable a user to buy a product, you need to create the `BuyProduct()` action method.

To ensure that only authenticated users are allowed to access the `BuyProduct()` action method, you need to add the `Authorize` attribute on the `BuyProduct()` method, as shown in the following code snippet:

```
[Authorize]
  public ActionResult BuyProduct()
{
    return View();
}
```

Once you have added the `Authorize` attribute on the `BuyProduct()` action method, the access to the corresponding view is restricted. Therefore, whenever any user tries to access this view, a page to authenticate the user is displayed. To configure the page to be displayed for user authentication, the following code snippet is used in the **Web.config** file:

```
<authentication mode="Forms">
<forms loginUrl="~/Account/Login"
timeout="2880" />
</authentication>
```

In the preceding code snippet, the `<authentication>` element contains a `<forms>` element that specifies the login URL for the application. Whenever a user tries to access a restricted resource, the user is redirected to the login URL. The `timeout` attribute of the `<forms>` element specifies the amount of time in minutes, after which the authentication cookie expires. It is set to 2880 minutes. Its default value is 30 minutes.

When a user is redirected to the login URL, the return URL is appended to the login URL in the form of a query string. The return URL specifies the URL of the restricted page that the user tried to access when he/she was redirected to the login page. When the user enters the valid details in the login form, the user is redirected to the return URL.

Once the user is successfully logged in, an authentication cookie is set at the client's end. The name of this cookie is .ASPXAUTH. The browser sends this cookie along with every subsequent request. Therefore, the user need not log in again. The .ASPXAUTH cookie exists until the user closes the browser. However, if the user selects the **Remember me** checkbox on the Login page, then the cookie will persist between the browsing sessions.

Securing a Controller

You have seen how to add the `Authorize` attribute to a specific action method within a controller. You can also secure an entire controller by adding the `Authorize` attribute on the controller, as shown in the following code snippet:

```
[Authorize]
public class ShoppingCartController :
Controller
{
…
}
```

The preceding code snippet will authorize all the action methods defined in the `ShoppingCartController` controller.

However, if you don't want to secure certain action methods within the controller, you need to use the `AllowAnonymous` attribute on those action methods, as shown in the following code snippet:

```
[AllowAnonymous]
public ActionResult Index()
{
return View();
}
```

In the preceding code snippet, the `AllowAnonymous` attribute on the `Index()` action method specifies that the access to the `Index()` action method does not require authentication.

Securing an Entire Application

At times, almost the entire Web application requires authentication. In such a situation, you can configure the `Authorize` attribute as a global filter. To register the `Authorize` attribute as a global filter, you need to add the same to the global filters collection in the `RegisterGlobalFilters()` method in the **FilterConfig.cs** file, as shown in the highlighted portion of the following code snippet:

```
public static void
RegisterGlobalFilters
(GlobalFilterCollection filters) {
filters.Add(new
System.Web.Mvc.AuthorizeAttribute());
filters.Add(new HandleErrorAttribute
());
}
```

The preceding code will apply the `Authorize` attribute to all controller actions in the application. To allow access to specific controllers or action methods, you need to use the `AllowAnonymous` attribute on them.

Introduction to Membership

An ASP.NET MVC 4 application created by using the **Internet Application** template provides some useful features for implementing authentication and authorization. To provide these features, the application template uses a membership provider called SimpleMembership.

When you create an MVC application by using the **Internet Application** template, the Visual Studio project wizard automatically creates the Membership database with `connectionstring` specified in the **web.config** file.

The Membership API provided by the `SimpleMemberShipProvider` class can be used to populate the database with the details of users and roles.

The `SimpleMembershipProvider` class works with the SQL server and handles membership tasks, such as creating and deleting user accounts.

The Internet template defines certain components to bootstrap SimpleMembership. These include:

❑ **The \Models\AccountModels.cs file**: This file defines a model for a basic user account. It includes the definition of the `UserProfile` class that provides the details of a user to be stored in the membership database. The `UserProfile` class is defined, as shown in the following code snippet:

```
public class UserProfile
  {
 [Key]
 [DatabaseGeneratedAttribute
 (DatabaseGeneratedOption.Identit
 y)]
 public int UserId { get; set; }
 public string UserName { get;
 set; }
  }
```

The preceding code snippet defines the `UserId` and `UserName` properties of the `UserProfile` class. You can modify the `UserProfile` class by adding more properties according to your requirement.

❑ **The \Filters \InitializeSimpleMembershipAttribute.cs file**: This file creates a membership database for the account model, and then calls the `WebMatrix.WebData.WebSecurity.InitializeDatabaseConnection()` method to initialize the database connection for the membership, as shown in the following code snippet:

```
WebSecurity.InitializeDatabaseCon
nection("DefaultConnection",
"UserProfile", "UserId",
"UserName", autoCreateTables:
true);
```

The preceding code snippet ensures that the database connection is initialized with the required schema. The `InitializeDatabaseConnection()` method specifies the name of the connection string, `DefaultConnection`, to connect to the database. It also specifies that the name of the table is `UserProfile`, which contains the user information. The `UserProfile` table contains the columns, `UserId` and `UserName`. To use the `WebSecurity` class, you need to include the namespace named `WebMatrix.WebData`.

- **The \Controllers\AccountController.cs file**: This file defines the Account controller, which contains various action methods, such as those for user registration and login. The Account controller is decorated with the `[InitializeSimpleMembership]` attribute that initializes the SimpleMembership provider.

When you create an MVC application by using the **Internet Application** template, `AccountController` and the corresponding views and models provide all the basic functionality to authenticate the user. The **Internet Application** templateuses forms authentication to implement authentication. Forms authentication is customizable. Therefore, you can change the `AccountController` class and corresponding views to manage the look and feel of the login form. You can also customize the information to be stored in the database.

The `AccountController` controller uses the `WebSecurity` class that comes from the Microsoft library named `WebMatrix`. The `WebSecurity` class provides security and authentication features. It includes the ability to perform authentication tasks, such as create user accounts, log users in and out, and reset or change passwords. The `WebSecurity` class, in turn, refers to the `SimpleMemberShipProvider` class, which is responsible for storing the data in a database.

The `AccountController` class contains the `Register()` action method that returns the form for registration. In addition, it contains the HttpPost version of the `Register()` action method, which creates the record in the database, and is defined, as shown in the following code snippet:

```
[HttpPost]
public ActionResult Register
(RegisterModel model)
{
  if (ModelState.IsValid)
  {
  // Attempt to register the user
  try
  {
    WebSecurity.CreateUserAndAccount
(model.UserName, model.Password);
    WebSecurity.Login(model.UserName,
model.Password);
    return RedirectToAction("Index",
"Home");
  }
  catch (MembershipCreateUserException
e)
  {
    ModelState.AddModelError("",
ErrorCodeToString(e.StatusCode));
  }
  }
```

In the preceding code snippet, the `CreateUserAndAccount()` method creates a user record in the database. Further, the `Login()` method logs the user on to the application.

The `AccountController` class also contains a `Login()` action method that returns the login form. The HttpPost version of the `Login()` method is called when the user clicks the login button on the login form. The `Login()` action method is defined, as shown in the following code snippet:

```
[HttpPost]
  public ActionResult Login(LoginModel
model, string returnUrl)
{
  if (ModelState.IsValid &&
WebSecurity.Login(model.UserName,
model.Password, persistCookie:
model.RememberMe))
  {
  return RedirectToLocal(returnUrl);
  }
  // If we got this far, something
failed, redisplay form
  ModelState.AddModelError("", "The
user name or password provided is
incorrect.");
  return View(model);
}
```

In the preceding code snippet, the `Login()` action method accepts the username and password entered by the user as the properties of the model object that it receives as a parameter. It also accepts the return URL as a parameter. The return URL is the URL of a restricted page that the user tried to access when he/she was redirected to the Login page. The preceding `Login()` action method checks whether the model state is valid and attempts to log on to the site by using the `WebSecurity.Login()` method. If the login is successful, the user is redirected to the return URL. Otherwise, an error is added to the model state.

Identifying Membership Providers

Membership helps you to manage users. ASP.NET MVC provides membership providers to implement membership. Some membership providers provided by ASP.NET MVC are:

- **ActiveDirectoryMembershipProvider**: This provider class is defined in the **System.Web.Security** namespace. It enables you to manage membership information for an ASP.NET application in Active Directory.
- **SqlMembershipProvider**: This provider class is defined in the **System.Web.Security** namespace. It enables you to manage membership information for an ASP.NET application in SQL database. This provider can work only with a specified table schema.
- **SimpleMembershipProvider**: This membership provider works with different versions of SQL Server, such as SQL server and SQL Compact Editions.
- **UniversalProviders:** This provider works with any database that Entity Framework supports.

NOTE

This course focusses mainly on `SimpleMembershipProvider`.

You can access the current membership provider, such as `SimpleMembershipProvider`, by using the property named `Provider`, as shown in the following code snippet:

```
var membership =
(SimpleMembershipProvider)
Membership.Provider;
```

In the preceding code snippet, the `Provider` property of the `Membership` class is used to access the `SimpleMembershipProvider` providers. The preceding code snippet will work only when you include the `System.Web.Security` namespace.

Configuring Membership

To configure a Web application to use `SimpleMembershipProvider`, you need to add the following code snippet between the `<system.web>` and `</system.web>` tags in the **Web.config** file:

```
<membership
defaultProvider="SimpleMembershipProvi
der">
<providers>
  <clear />
  <add name="SimpleMembershipProvider"
    type="WebMatrix.Webdata.SimpleMemb
ershipProvider,WebMatrix.WebData"
 />
</providers>
</membership>
```

Building a Custom Membership Provider

If you need to add custom logic for authentication in an application, you need to create your own custom membership provider. For example, consider the following code snippet:

```
public class CustomMembershipProvider
: SimpleMembershipProvider
{
//code
}
```

In the preceding code snippet, a custom membership provider class named `CustomMembershipProvider` is created. This class inherits the `SimpleMemberShipProvider` class.

Once you have implemented the custom membership provider class, you need to override the `ValidateUser()` method defined in the `SimpleMembershipProvider` class. Consider the following code snippet to override the `ValidateUser()` method defined in `SimpleMembershipProvider`:

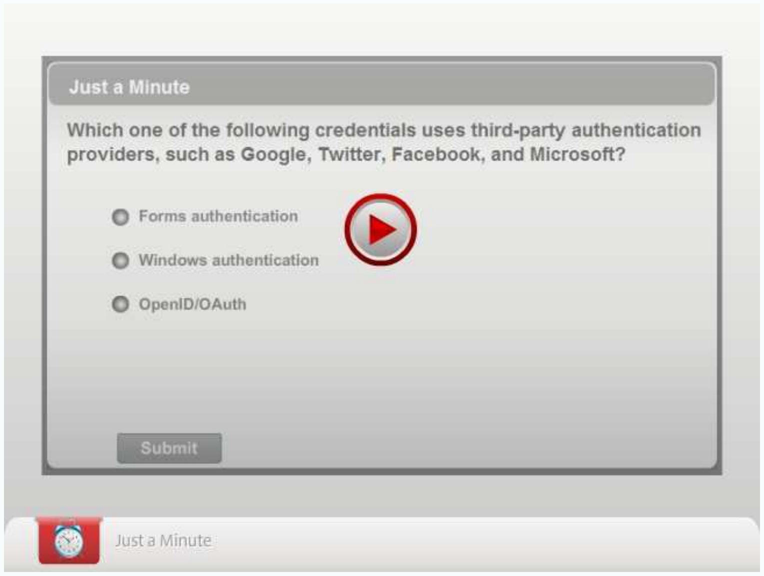
```
public override bool ValidateUser
(string username, string password)
{
//code
}
```

You can write your own custom validation logic in the `ValidateUser()` method.

Then, you need to modify the **Web.config** file to apply the custom provider class to the application, as shown in the following code snippet:

```
<membership
defaultProvider="CustomMemberProvider"
```

```
>
<providers>
<clear/>
<add name="CustomMemberProvider"
type="CustomAdminMembershipProvider"/>
</providers>
</membership>
```


Just a Minute

Which one of the following credentials uses third-party authentication providers, such as Google, Twitter, Facebook, and Microsoft?

- Forms authentication
- Windows authentication
- OpenID/OAuth

Activity 10.1: Implementing Authentication

Implementing Authorization

Consider the scenario of the NewBay store website where only the administrator is allowed to add the product details in a database. In this case, whenever another user tries to access the interface for adding the product details, he/she should not be permitted to do so. For verifying the permissions given to an authenticated user for accessing certain resources in a Web application, you need to implement authorization. Authorization is a mechanism that determines the rights given to the users for accessing the available resources.

Let us see how to implement authorization in ASP.NET MVC.

Authorizing Specific Users and Roles

You can authorize specific users and roles by using the `Authorize` attribute. Consider the following code snippet to authorize specific users:

```
[Authorize(Users="Smith, James")]
public class
ManageStoreController:Controller
{
...
}
```

In the preceding code snippet, only the users named `Smith` and `James` are the authorized users to access the `ManageStore` controller.

In addition, you can also assign roles to different users, and then implement authorization at the role level, as shown in the following code snippet:

```
[Authorize(Roles="Administrator")]
public class ManageStoreController :
Controller
{
...
}
```

The preceding code snippet will restrict the access of the `ManageStore` controller to users who belong to the `Administrator` role.

The `Roles` parameter can also take more than one role, as shown in the following code snippet:

```
[Authorize(Roles="Administrator,
SuperAdmin")]
public class
ManageStoreController:Controller
{
...
}
```

You can also use a combination of roles and users, as shown in the following code snippet:

```
[Authorize(Roles="Manager",
Users="Smith,James")]
public class
ManageStoreController:Controller
{
...
}
```

The preceding code snippet allows the users Smith and James, and all users with the role, Manager to access the `ManageStore` controller.

Identifying Role Providers

Roles help you to assign specific rights to users. ASP.NET MVC provides role providers to implement roles. Some role providers provided by ASP.NET MVC are:

- **ActiveDirectoryRoleProvider**: This provider class is defined in the **System.Web.Security** namespace. It enables you to manage role information for an ASP.NET application in Active Directory.
- **SqlRoleProvider**: This provider class is defined in the **System.Web.Security** namespace. It enables you manage role information for an ASP.NET application in SQL database.

- **SimpleRoleProvider**: This role provider works with different versions of SQL Server, such as SQL server and SQL Compact Editions.
- **UniversalProviders:** This provider works with any database that Entity Framework supports.

NOTE *This course focusses mainly on `SimpleRoleProvider`.*

You can access the current role provider, such as `SimpleRoleProvider`, by using the property named `Provider`, as shown in the following code snippet:

```
var roles = (SimpleRoleProvider)
Roles.Provider;
```

In the preceding code snippet, the `Provider` property of the `Roles` class is used to access the `SimpleRoleProvider` provider. The preceding code snippet will work only when you include the `System.Web.Security` namespace.

Defining Roles

When you create an MVC application by using the **Internet Application** template, the Visual Studio project wizard automatically creates the membership database with the `connectionstring` specified in the **web.config** file.

You can populate this database with the required roles by seeding the membership database. For this, you can add the following code in the `Seed()` method of the**Configuration.cs** file.

```
   var roles = (SimpleRoleProvider)
Roles.Provider;
if (!roles.RoleExists("Admin"))
   {
   roles.CreateRole("Admin");
   }
```

In the preceding code snippet, you first retrieve a reference to the simple role provider. Then, you use this reference to check whether the role Admin exists in the database. If it does not exist, the `CreateRole()` method creates the role, `Admin`. The `Seed()` method will be executed when you use EF code-first migrations to update the database.

Adding User Accounts to Roles

Once you have created a role in the database, you can add users to that role. For example, to assign the role, Admin, to a user named Smith_Thompson, the following code snippet can be used:

```
if (!roles.GetRolesForUser
("Smith_Thompson").Contains("admin"))
   {
   roles.AddUsersToRoles(new[]
{ "Smith_Thompson" }, new[]
{ "admin" });
   }
```

In the preceding code snippet, if the role, `admin`, is not already assigned to a user named `Smith_Thompson`, the `AddUserstoRoles()` method adds the role, `admin`, to the user that has the username, `Smith_Thompson`.

Once the roles and users are created in the database, you can use the `Authorize` attribute on the action method that needs to be restricted. For example, consider the following action method that can only be accessed by a user with the admin role:

```
[Authorize(Roles = "admin")]
public ActionResult AdminHome()
{
   return View();
}
```

When any user tries to access the `AdminHome` page, the login page will be displayed where the user needs to provide the valid credentials for a user with the admin role. If the user provides valid credentials, he/she can access the `AdminHome` page.

When a user logs in, he/she should be shown only those links that are linked to resources that the user has rights to access. For example, the `AdminHome` link should be visible only to the administrator. To prevent this link from being displayed to other users, you can use the following code in the relevant view:

```
@if(User.IsInRole("admin"))
{
@Html.ActionLink
("AdminHome","AdminHome")
}
```

The preceding code snippet displays the `AdminHome` link only when the user logs in with an account that has the `admin` role.

Configuring Roles

To configure a Web application to use `SimpleRoleProvider`, you need to add the following code snippet between the `<system.web>` and `</system.web>` tags in the **Web.config** file:

```
<roleManager enabled="true"
defaultProvider="SimpleRoleProvider">
<providers>
  <clear/>
  <add name="SimpleRoleProvider"
 type="WebMatrix.Webdata.SimpleRoleProv
 ider,WebMatrix.WebData" />
</providers>
</roleManager>
```

Building a Custom Role Provider

ASP.NET MVC also enables you to create custom role providers. By using custom role providers, you can implement role management that uses your own database schema and logic. To build a custom role provider, you need to create a class that inherits the `RoleProvider` class, as shown in the following code snippet:

```
public class CustomRoleProvider :
RoleProvider
{
}
```

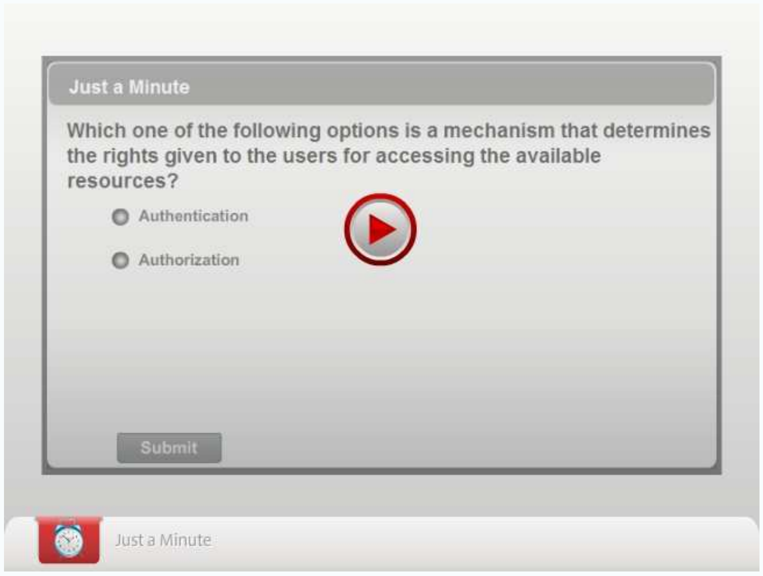
In the `RoleProvider` class, you need to implement the `GetRolesForUser()` function, as shown in the following snippet:

```
public override string[]
GetRolesForUser(string username)
{
//code to return a list of roles for
users
}
```

In the preceding code snippet, the `GetRolesForUser()` function takes the user name as the input. This function returns a list of roles to which the user belongs.

Once you have implemented the `GetRolesForUser()` function, you need to apply the custom role provider to the application by modifying the **Web.config** file, as shown in the following code snippet:

```
<roleManager
defaultProvider="CustomRoleProvider"
enabled="true"
cacheRolesInCookie="false">
<providers>
<clear />
<add name="CustomRoleProvider"
type="CustomRoleProvider" />
</providers>
</roleManager>
```


Just a Minute

Which one of the following options is a mechanism that determines the rights given to the users for accessing the available resources?

- Authentication
- Authorization

Activity 10.2: Implementing Authorization

Summary

In this chapter, you learned that:

- Authentication is the process of identifying an individual, usually based on the username and password provided by the user.
- MVC allows you to implement authentication by using any of the following authentication modes:
  - Forms Authentication
  - Windows Authentication
  - OpenID/OAuth
- Forms authentication is the default authentication type enabled for MVC applications created using the Internet Application template.
- Windows authentication is typically used for intranet applications that run across a company's firewall.
- Action filters are attributes that let you add behavior that can be executed either before an action method is called or after an action method is executed.
- The `Authorize` attribute is the default authorization filter provided with ASP.NET MVC.
- You can use the `Authorize` attribute to:
  - Secure a controller action.
  - Secure a controller.
  - Secure an entire application.
- The `SimpleMembershipProvider` class works with the SQL server and handles membership tasks, such as creating and deleting

user accounts.

- Some membership providers provided by ASP.NET MVC are:
  - ActiveDirectoryMembershipProvider
  - SqlMembershipProvider
  - SimpleMembershipProvider
  - UniversalProviders
- Authorization is a mechanism that determines the rights given to the users for accessing the available resources.
- Roles help you to assign specific rights to users.
- Some role providers provided by ASP.NET MVC are:
  - ActiveDirectoryRoleProvider
  - SqlRoleProvider
  - SimpleRoleProvider
  - UniversalProviders

Chapter 11

Deploying a Web Application

Once a Web application is developed, you need to make it accessible to the users over the Internet. In order to let the users access the application through the Internet, you need to deploy it on a Web server.

This chapter explains how to prepare a Web application for deployment. In addition, it discusses how to host a Web application on IIS.

Objectives

In this chapter, you will learn to:

- ❑ Prepare a Web application for deployment
- ❑ Host a website on IIS

Preparing a Web Application for Deployment

In today's scenario, the Internet is the most used medium for getting information on any topic. The information available on the Internet is contained in different Web applications. To enable users to access the information contained in Web applications, these applications need to be installed on a Web server. The process of installing a Web application on a Web server is known as deployment.

While developing a Web application by using Visual Studio, when you execute the application, the application is automatically deployed on Internet Information Server (IIS) Express. IIS Express makes deployment simple because it runs with our identity, and allows us to start and stop the Web server whenever required.

However, to make the Web application accessible over the Internet or an intranet, you need to host it on a full version of IIS or any other Web server.

Before deploying a Web application on a Web server, you need to prepare the application for deployment. This involves certain activities, such as:

- ❑ Identifying the files and folders to be copied to the destination server
- ❑ Configuring the Web.config transformations
- ❑ Precompiling the Web application
- ❑ Testing the application in an environment similar to production

Identifying the Files and Folders to be Copied to the Destination Server

During the development phase, a Web application is saved in a folder on the computer on which it is developed. However, once the application is developed, it needs to be deployed on a Web server. When you deploy the application on a Web server, such as IIS, you need to identify the files and folders to be copied to the destination server.

By default, Visual Studio deploys only the files that are required to run the application. However, in some scenarios, you might have special requirements. For example, some of the database files present in the App_Data folder are only required while developing a Web application. In such a case, you can identify these database files and exclude them at the time of deployment. For this, you need to configure the deployment settings in the Project properties.

Configuring Web.config Transformations

When an ASP.NET application is deployed on a Web server, there are some settings in the deployed application's `Web.config` file, which need to be different from the `Web.config` file in the development environment. For example, you might want to disable debug options and change connection strings so that they point to different databases. These groups of settings need to be set up in the `Web.config` transformation file.

A transformation file refers to an XML file that allows you to specify how the `Web.config` file should be changed when it is deployed. The transformation actions can be specified in the `Web.config` file by using `XML` attributes.

A transform file is associated with a build configuration. By default, Visual Studio creates the Debug and Release build configurations by the names, `Web.Debug.config` and `Web.Release.config`. However, you can also create custom build configurations.

The `Web.release.config` file stores the changes that Microsoft Visual Studio applies to the `Web.config` file, when you compile the application in the Release mode.

The `Web.debug.config` file stores changes that Microsoft Visual Studio applies to the `Web.config` file, when you compile the application in the Debug mode.

Before you publish the application by using the release configuration, you need to remove the debug attribute from the `<compilation>` element in the `Web.config` file. For this, the following markup is included in the `Web.release.config` file:

```
<system.web>
<compilation
xdt:Transform="RemoveAttributes
(debug)" />
```

```
</system.web>
```

In the preceding markup, the `xdt:Transform` attribute is used to remove the `debug` attribute from the `Web.config` file.

You can also add additional elements to the transformed `Web.config` file. For example, the following code shows how to insert a new connection string setting in the transformed `Web.config` file.

```
<connectionStrings>
<add name="DemoConnStr"
connectionString="Data
Source=|DataDirectory|demo.mdf"
providerName="System.Data.SqlServerCe.
4.0" xdt:Transform="Insert"/>
</connectionStrings>
```

In the preceding code snippet, the value, `Insert`, has been used for the `xdt:Transform` attribute to insert a new connection string in the transformed `Web.config` file.

Precompiling the Web Application

The most basic technique for deployment is to copy the application components to the hard drive of a Web server. In this technique, the various application components are deployed to the Web server in the uncompiled form. This type of deployment has the following issues:

- ❑ The application is deployed even if it contains compilation errors.
- ❑ The source code of the application is exposed.
- ❑ The application loads slowly initially because it is compiled when it is accessed for the first time.

These problems can be resolved by precompiling an application before deploying it to a Web server.

Precompiling a Web application involves compilation of the source code into DLL assemblies before deployment. Precompiling a Web application provides the following advantages:

- ❑ It provides faster response time because the Web application need not be compiled the first time it is accessed.
- ❑ It helps in identifying the bugs that can occur when a Web page is requested. These bugs are removed at the time of compiling the Web application. Then, the compiled and error-free Web application is deployed on to the server and is rendered to a user.
- ❑ It hides and secures the source code of a Web application from malicious users.

Testing the Website in an Environment Similar to Production

Before you deploy a Web application, you need to make sure that it is running properly. The errors occurring while the application is running need to be diagnosed and fixed. You also need to ensure that there are no performance-related issues with your Web application. As a developer, you are focused on creating a Web application that runs smoothly after its deployment. Once an application is deployed, you need to ensure that the application is working properly and is not giving unexpected results. For this, you need to keep track of the execution of your Web application. This will enable you to track any error or performance-related issue in your application.

ASP.NET provides you with the tracing feature that enables you to track the program execution, thereby ensuring that your Web application runs properly. You can use this feature to view the diagnostic information about a particular Web page. This information includes the execution time of page methods, the time spent rendering page controls, and the content of various collections, such as the query string, HTTP header, and session state.

In addition to the standard diagnostic information, tracing can also be used to display the custom tracing information about the execution of an application. Tracing information can also be written to an event log or a text file by using trace listeners.

After deploying a Web application, you, as a system administrator, need to constantly monitor it for its proper functioning. Many unexpected problems, such as website experiencing heavy load, may occur while the application is running in the real-world environment. By monitoring a Web application, you can detect the problems occurring in the application and troubleshoot them.

Hosting a Web Application on IIS

Consider the scenario where you have developed a Web application and it is ready to be made available to the users. To enable the users to access the application, the Web application needs to be deployed on a Web server, such as Internet Information Services (IIS).

IIS is a Web server that provides a comprehensive platform, which helps you develop, host, and manage Web applications. IIS is proprietary of Microsoft and is packaged along with the Windows operating system. By default, IIS is turned off when Windows is installed. Therefore, before deploying an application on IIS, you need to configure IIS on the Windows operating system.

Task 11.1: Configure IIS on Windows

Deploying a Web Application to an IIS Server

Instead of using a deployment package, you can directly deploy a website to a target server, such as:

- ❑ The IIS server on your development computer
- ❑ An IIS server on your network
- ❑ A server owned by a service provider
- ❑ Windows Azure

For deploying a Web application on any of the preceding target servers, you can use the Publish Web wizard. While using the Publish Web wizard, you need to perform the following tasks:

1. Create a publish profile.
2. Configure the connection settings.
3. Configure the publish and database settings.
4. Publish the project.

Creating a Publish Profile

The first step to deploy a Web application is to configure a publish profile. A publish profile represents the various deployment options, such as:

- ❑ The target server to be used for deployment
- ❑ The credentials needed to log on to the server
- ❑ The databases to be deployed

If you are deploying your Web application to Windows Azure or a target server owned by a service provider, the service provider will provide you a .publishsettings file. You can automatically create the publish profile by importing this .publishsettings file while using the Publish Web wizard.

Configuring the Connection Settings

After creating a publish profile, you need to configure the connection settings. The first step for this is to select a publish method. Some publish methods that can be used are:

- ❑ **File System**: This method publishes the Web application in a folder that you specify. This option can be used for servers to which you have direct network access. You can also use this option to first publish a Web application to a local folder, and then copy the files to the target server.
- ❑ **Web Deploy**: This method allows you to specify database settings, such as connection strings. These settings will override the settings in the Web.config file of the application, and a new version of the Web.config file will be created for the target environment. Web Deploy also helps you to automatically update the schema of the database.

Once you have selected a publish method, you need to specify the connection details of the target server.

Configuring the Publish and Database Settings

After specifying the connection settings, you need to specify the publish and database settings. For example, you can specify whether to use the release or debug build configuration. While deploying a Web application, you normally use a release build. The debug build is not efficient and is used only when you are deploying to a test environment, where debugging may be required.

In addition to specifying the build configuration, you can also specify the various file publish options such as:

- ❑ Whether to remove files on the destination server that have no matching files in the Web application on your development computer.
- ❑ Whether to precompile the project before publishing.
- ❑ Whether to prevent the files in the App_Data folder from being deployed on the target server.

Further, you need to specify the connection strings to the target database servers that your application needs to use after deployment. In addition, you can specify whether to run code first migrations on application start.

Publishing the Project

Once all the settings are done, you can use an option available in the Publish Web wizard to preview the files that will be copied to the target server. After ensuring that the correct files are being copied, you can publish the project.

Activity 11.1: Deploying a Web Application

Creating and Installing a Deployment Package

To deploy a Web application on a Web server, you can create a deployment package. You can use the Publish Web wizard to configure one or more publish profiles for creating a deployment package. A publish profile represents the deployment options, such as the server on which you want to deploy a Web application and the databases that are required to be deployed.

Once you have created a deployment package, you can install the package on the destination server by using the `<projectname>.deploy.cmd` file that Visual Studio creates with the package.

Task 11.2: Creating a Deployment Package

Summary

In this chapter, you learned that:

- ❑ While developing a Web application using Visual Studio, when you execute the application, the application is automatically deployed on Internet Information Server (IIS) Express.
- ❑ Before deploying a Web application on a Web server, you need to prepare the application for deployment. This involves certain activities, such as:
  - Identifying the files and folders to be copied to the destination server
  - Configuring Web.config transformations
  - Precompiling the Web application
  - Testing the application in an environment similar to production
- ❑ A transformation file refers to an XML file that allows you to specify how the `Web.config` file should be changed when it is deployed.
- ❑ IIS is a Web server that provides a comprehensive platform, which helps you develop, host, and manage Web applications.
- ❑ While using the Publish Web wizards, you need to perform the following tasks:
  1. Create a publish profile
  2. Configure the connection settings
  3. Configure the publish and database settings
  4. Publish the project

Glossary

B

Bundling

Bundling is a technique provided by ASP.NET MVC that allows you to combine multiple files, such as CSS and JavaScript, into a single file.

Business logic layer

Consists of the components of the application that control the flow of execution and communication between the presentation and data layers.

C

Client-side scripting

Client-side scripting enables you to develop Web pages that can dynamically respond to user input without having to interact with a Web server.

Controller

Refers to a set of classes that handle communication from the user and the overall application flow.

Cookies

Cookies are small pieces of information that are stored on the client&#8217;s computer.

D

Data layer

Consists of components that expose the application data stored in databases to the business logic layer

Dynamic Web page

A Web page whose content is generated dynamically by a Web application or that responds to user input and provides interactivity is called a dynamic Web page.

F

Fat client and thin server

The architecture in which the business logic layer resides on the client tier is referred to as the fat client and thin server architecture. In this architecture, the client accepts user requests and processes these requests on its own.

Fat server and thin client

The architecture in which the business logic layer resides on the server is referred to as the fat server and thin client architecture. In this architecture, the client accepts requests from the users and forwards the same to the server.

J

JavaScript library

AJavaScript library is a set of prewritten JavaScript code, which helps in developing a JavaScript-based application easily.

jQuery

jQuery is a cross-browser JavaScript library that helps you easily perform various tasks, such as DOM traversal, event handling, and animating elements.

jQuery UI

jQuery UI is an organised set of user interface interactions, effects, widgets, and themes built on top of the jQuery JavaScript library.

L

LINQ

LINQ offers a consistent programming model to query data from different data sources.

M

Model

Refers to a set of classes that describes the data that the application works with.

N

Nested layout

A nested layout page refers to a layout that is derived from a parent layout.

P

Presentation layer

Consists of the interface through which the users interact with the application.

R

Razor

Razor is a markup syntax that allows you to embed server-side code (written in C# or VB) in an HTML markup.

Routing

Routing is a feature that enables you to develop applications with comprehensible and searchable URLs.

S

Scaffolding

MVC provides a scaffolding feature that provides a quick way to generate the code for commonly used operations in a standardized way.

Server-side scripting

Server-side scripting provides users dynamic content that is based on the information stored at a remote location, such as a back-end database.

Static Web Page

A Web page that contains only static content and is delivered to the user as it is stored is called a static Web page.

U

Unobtrusive JavaScript

Unobtrusive JavaScript is a general approach to implement JavaScript in Web pages. In this approach, the JavaScript code is separated from the HTML markup.

V

View

Refers to the components that define an application's user interface.

ViewBag

ViewBag is a dynamic object that allows passing data between a controller and a view.

ViewData

ViewData is a dictionary of objects derived from the ViewDataDictionary class.