ASP.NET MVC

MVC 5

MVC Core

Eng. Basma Hussien

Model Property

- ViewPage.**Model** Property
- Gets the Model property of the associated ViewDataDictionary object.

- Any view(Ex. Index.cshtml) inherits from "ViewPage class", which have a Model property:

  Object Model { get; }

- Namespace: System.Web.Mvc
- Assembly: System.Web.Mvc.dll

- View Created Based on Model, called "***Strongly Typed View***"

HTML Helpers

HTML Helpers In ASP.NET MVC5

- HTML Helpers are classes which help to render the HTML. These classes have methods which generate HTML at runtime. We can also bind a model object to individual HTML element for displaying or retrieving values.

- One of the major differences between calling the HtmlHelper methods and using an HTML tag is that the HtmlHelper methods are designed to make it easy to bind to the View data or Model data.

- @Html is used to access the HTML helper, however, HTML is the property of HtmlHelpers which is included in the base class.

HTML Helpers Features

- HTML Helpers are methods that return a string.
- Helper class can create HTML controls programmatically. HTML Helpers are used in View to render HTML content.
- It is not mandatory to use HTML Helper classes for building an ASP.NET MVC application. We can build an ASP.NET MVC application without using them, but HTML Helpers helps in the rapid development of a view.
- HTML Helpers are more lightweight as compared to ASP.NET Web Form controls as they do not use ViewState and do not have event models.
- MVC has built-in Helpers methods
- We can create custom HTML helpers.

HTML Helpers Usage

- Using the HTML Helper class, we can create HTML Controls programmatically.
- HTML Helpers are used in View to render HTML content. HTML Helpers (mostly) is a method that returns a string.
- It is not mandatory to use HTML Helper classes for building an ASP.NET MVC application. We can build an ASP.NET MVC application without using them, but HTML Helpers helps in the rapid development of a view.
- HTML Helpers are more lightweight as compared to ASP.NET Web Form controls as they do not use ViewState and do not have event models.

HTML Helpers Types

- HTML Helpers are categorized into three types:
  - Inline HTML Helpers
  - Built-in HTML Helpers
    1. Standard HTML Helpers
    2. Strongly Typed HTML Helpers
    3. Templated HTML Helpers
  - Custom HTML Helpers

1. Inline HTML Helpers

- Inline HTML Helper is used to create a reusable Helper method by using the Razor @helper tag.
- Inline helpers can be reused only on the same view.
- We cannot use Inline helper to the different view Pages.
- We can create our own Inline helper method based on our requirements.

Example of Inline HTML helpers

```
@helper ListHelper(string[] strList)
{
    <ol>
        @foreach (var item in strList)
        {
            <li>@item</li>
        }
    </ol>
}
```

```
@{
    string[] strBooks = new string[] { "C#.NET", "ASP.NET MVC", "ASP.NET CORE", "VB.NE
}
```

```
<div id="div1" style="background-color:yellow;">
    Book Name List: @ListHelper(strBooks)
</div>
```

Book Name List:

1. C#.NET
2. ASP.NET MVC
3. ASP.NET CORE
4. VB.NET
5. WEB API

Advantages of Using Inline HTML Helpers

- It is reusable on the same view.
- It reduces the code repetition
- It is simple to create and easy to use.
- It is easy to customization the method based on the requirement.

Inline HTML helper Syntax

```
@helper HelperName(Parameters list..)
{
    // code.....
}
```

Example of Inline HTML helpers

```
@helper ListHelper(string[] strList)
{
    <ol>
        @foreach (var item in strList)
        {
            <li>@item</li>
        }
    </ol>
}
```

```
@{
    string[] strBooks = new string[] { "C#.NET", "ASP.NET MVC", "ASP.NET CORE", "VB.NE
}
```

```
<div id="div1" style="background-color:yellow;">
    Book Name List: @ListHelper(strBooks)
</div>
```

Book Name List:

1. C#.NET
2. ASP.NET MVC
3. ASP.NET CORE
4. VB.NET
5. WEB API

Example of Inline HTML helpers

We can create inline HTML Helper method having integer parameter.

```
01.  @helper AddHelper(int a, int b)
02.  {
03.       <label>Addition of two No = @(a + b)</label>
04.  }
```

This is how we use inline HTML Helper method in View.

```
01.  <div style="background-color:azure;">
02.              @AddHelper(100, 200)
03.  </div>
```

Output 1

Addition of two No = 300

2. Built-in HTML Helpers

- Built-In Html Helpers are extension methods on the HtmlHelper class. It can be divided into three categories:

1. Standard HTML Helpers
2. Strongly Typed HTML Helpers
3. Templated HTML Helpers

Standard HTML Helpers

- Standard HTML Helpers are used to render the most common type of HTML controls like TextBox, DropDown, Radio buttons, Checkbox etc.
- Extension methods of HTML Helper classes have several overloaded versions.
- We can use any one according to our requirement.

Standard HTML Helpers Example

TextBox: The TextBox Helper method renders a textbox in View that has a specified name. We can also add attributes like class, placeholder etc. with the help of overloaded method in which we have to pass objects of HTML Attributes.

```
@Html.TextBox("txtName", null, new { @class = "textbox", placeholder = "Enter Name" })
```

▲ 3 of 7 ▼ (extension) MvcHtmlString HtmlHelper.TextBox(string name, object value, object htmlAttributes)
Returns a text input element by using the specified HTML helper, the name of the form field, the value, and the HTML attributes.

Output

```
<input class="textbox" id="txtName" name="txtName" placeholder="Enter Name" type="text" value="" />
```

Enter Name

We can also set the value In Textbox by passing a value in the TextBox extension method.

```
@Html.TextBox("txtName", "Anoop", new { @class = "textbox", placeholder = "Enter Name" })
```

```
<input class="textbox" id="txtName" name="txtName" placeholder="Enter Name" type="text" value="Anoop" />
```

Anoop

Standard HTML Helpers Example

Label: The Label method of HTML helper can use for generating label element. Label extension method have 6 overloaded versions.

```
@Html.Label("Name","Name")
```

▲ 3 of 6 ▼ (extension) MvcHtmlString HtmlHelper.Label(string expression, **string labelText**)

Returns an HTML label element and the property name of the property that is represented by the specified expression using the label text.

labelText: *The label text to display.*

Output

```
<label for="Name">Name</label>
```

Standard HTML Helpers Example

In the above example, did you notice an additional input element? In case if you unchecked the checkbox or checkbox value not selected then you will get the value from the hidden field.

DropDownList: The DropDownList helper renders a drop down list.

```
@Html.DropDownList("Country", new List<SelectListItem> {
    new SelectListItem{Text="India",Value="0"},
    new SelectListItem{Text="SriLanka",Value="1"},
    new SelectListItem{Text="Bangladesh",Value="2"}
})
```

```
<select id="Country" name="Country">
    <option value="0">India</option>
    <option value="1">SriLanka</option>
    <option value="2">Bangladesh</option>
</select>
```

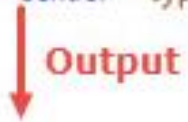
Output

Standard HTML Helpers Example

RadioButton: RadioButton can be rendered in the view using the RadioButton Helper method. In the simplest form, RadioButton Helper method takes three parameters i.e. name of the control, the value, and the Boolean value for selecting the value initially.

```
Male:@Html.RadioButton("Gender", "Male", true)
Female:@Html.RadioButton("Gender", "Female", false)
```

```
Male:<input checked="checked" id="Gender" name="Gender" type="radio" value="Male" />
Female:<input id="Gender" name="Gender" type="radio" value="Female" />
```



Output

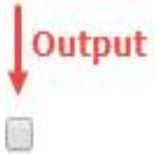
Male: Female:

Standard HTML Helpers Example

CheckBox: The CheckBox helper method renders a checkbox and has the name and id that you specify.

```
@Html.CheckBox("IsMarried",false)
```

```
<input id="IsMarried" name="IsMarried" type="checkbox" value="true" />
<input name="IsMarried" type="hidden" value="false" />
```

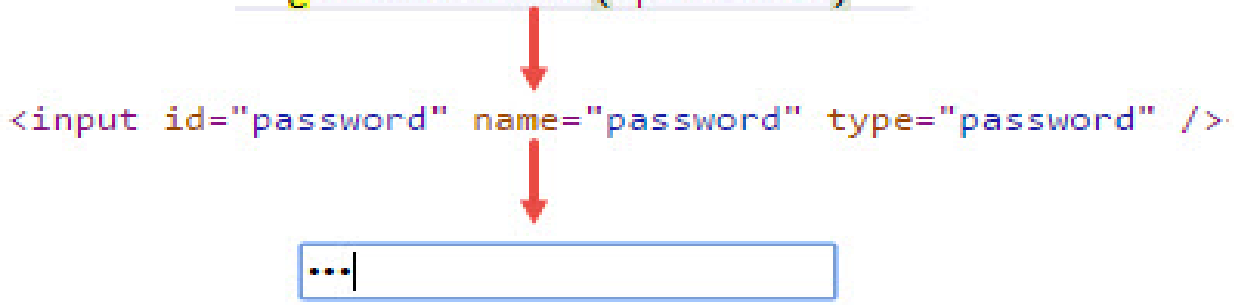


In the above example, did you notice an additional input element? In case if you unchecked the checkbox or checkbox value not selected then you will get the value from the hidden field.

Standard HTML Helpers Example

Password: The Password Helper method renders the input type as password.

```
@Html.Password("password")
```



```
<input id="password" name="password" type="password" />
```

Hidden: The Hidden Helper method renders a Hidden field.

```
@Html.Hidden("StudentID")
```

```
<input id="StudentID" name="StudentID" type="hidden" value="" />
```

Standard HTML Helpers Example

Html.ActionLink

Html.ActionLink creates a hyperlink on a view page and the user clicks it to navigate to a new URL. It does not link to a view directly, rather it links to a controller's action. Here are some samples of Html.ActionLink.

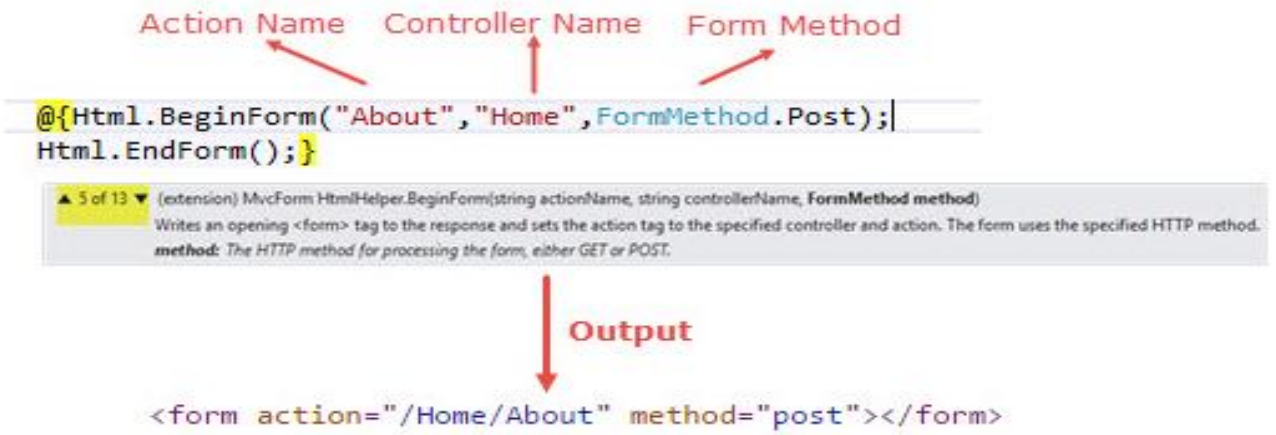
```
@Html.ActionLink("Edit", // <-- Link text
                 "Edit", // <-- Action Method Name
                 new { id=item.CustomerID }, // <-- Route arguments
                 new {@class="ui-btn"} // <-- htmlArguments
                )
```

If you want to navigate to a different controller's action method, use the one given below. You can even avoid typing "null" for the route value and htmlArguments. Here also, razor will assume the first param as link text, the second param as the action method name and the third param as the controller name, if it finds three parameters.

```
01. @Html.ActionLink("Click here", // <-- Link text
02.                  "About", // <-- Action Method Name
03.                  "Home", // <-- Controller Name
04.                  null, // <-- Route value
05.                  null // <-- htmlArguments
06.                  )
```

Standard HTML Helpers Example

Form: For creating the Form element, we can use BeginForm() and EndForm() extension method.



The BeginForm helper implements the IDisposable interface, which enables us to use the **using** keyword.

```
@using(Html.BeginForm("About","Home",FormMethod.Post))
{

}
```

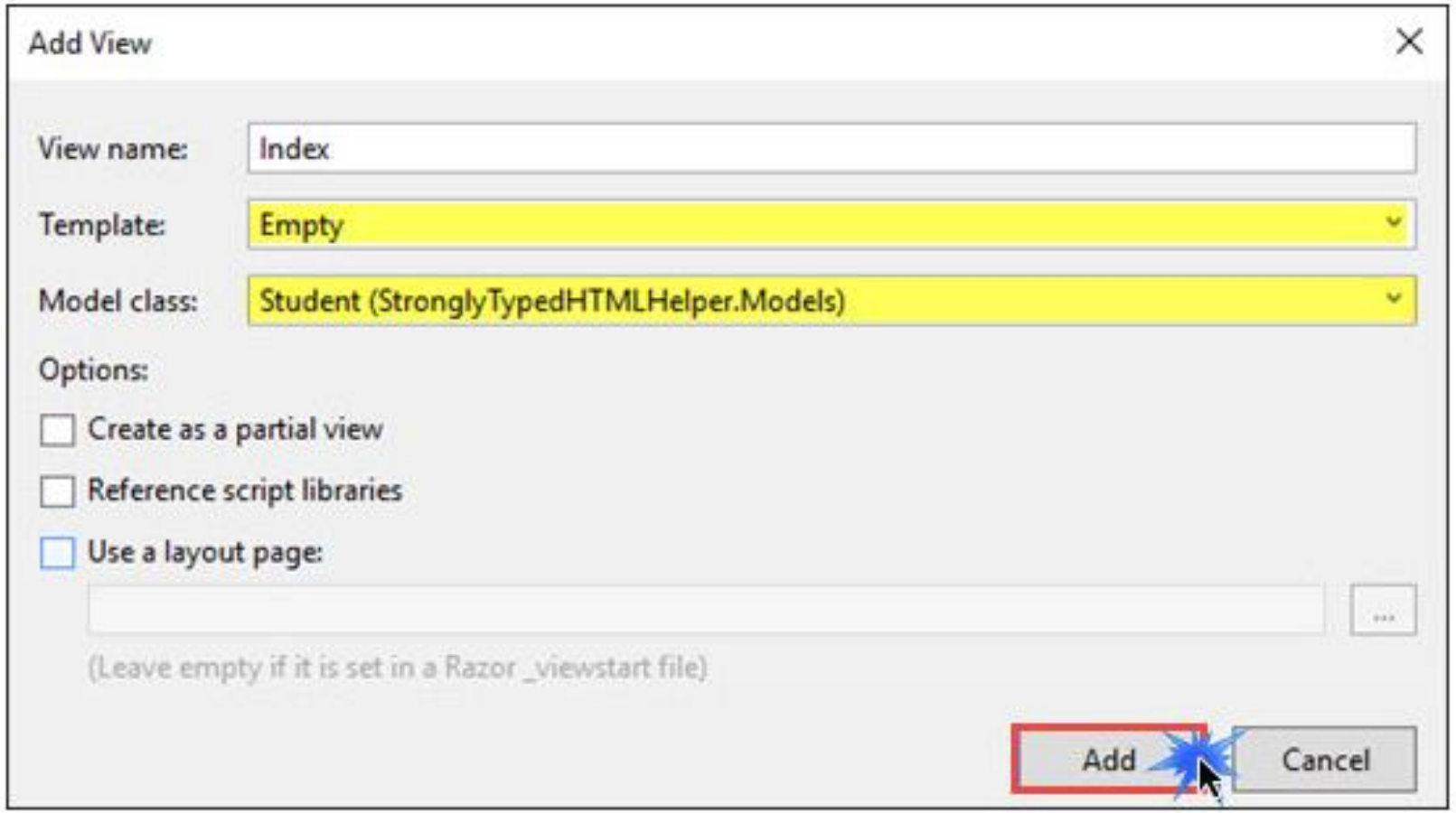
Strongly Typed Helper Method

- Html.TextBoxFor(), Html.TextAreaFor(), Html.DropDownListFor(), Html.CheckboxFor(), Html.RadioButtonFor(), Html.ListBoxFor(), Html.PasswordFor(), Html.HiddenFor(), Html.LabelFor(), etc.

- The strongly typed HTML helpers work on ***lambda expression***. The ***Model*** object is passed as a value to lambda expression, and you can select the field or property from model object to be used to set the id, name and value attributes of the HTML helper.

- To use Strongly Typed Helper method, we first have to make ***Strongly Typed View.***

Strongly Typed View



Template Helper Method

- These helpers figure out what HTML elements are required to render based on properties of your model class. This is a very flexible approach for displaying data to the user, although it requires some initial care and attention to set up. To setup proper HTML element with Templated HTML Helper, make use of DataType attribute of DataAnnitation class.

- Ex: EditorFor Helper method will generate TextArea element if we have declared MultiLine Datatype on Address property.

- ***DisplayFor*** and ***EditorFor*** are the examples of Template Helper method.

Extension Method	Strongly Typed Method	Html Control
Html.ActionLink()	NA	<a>
Html.TextBox()	Html.TextBoxFor()	<input type="textbox">
Html.TextArea()	Html.TextAreaFor()	<input type="textarea">
Html.CheckBox()	Html.CheckBoxFor()	<input type="checkbox">
Html.RadioButton()	Html.RadioButtonFor()	<input type="radio">
Html.DropDownList()	Html.DropDownListFor()	<select> <option> </select>
Html.ListBox()	Html.ListBoxFor()	multi-select list box: <select>
Html.Hidden()	Html.HiddenFor()	<input type="hidden">
Html.Password()	Html.PasswordFor()	<input type="password">
Html.Display()	Html.DisplayFor()	HTML text: ""
Html.Label()	Html.LabelFor()	<label>
Html.Editor()	Html.EditorFor()	Generates Html controls based on data type of specified model property e.g. textbox for string property, numeric field for int, double or other numeric type.

3. Custom Html Helpers

- You can also create your own custom helper methods by creating an extension method on the HtmlHelper class or by creating static methods with in a utility class.

```
public static class CustomHelpers
{
 //Submit Button Helper
 public static MvcHtmlString SubmitButton(this HtmlHelper helper, string
 buttonText)
 {
 string str = "<input type=\"submit\" value=\"" + buttonText + "\" />";
 return new MvcHtmlString(str);
 }
 //Readonly Strongly-Typed TextBox Helper
 public static MvcHtmlString TextBoxFor<TModel, TValue>(this
 HtmlHelper<TModel> htmlHelper, Expression<Func<TModel, TValue>>
 expression, bool isReadonly)
 {
 MvcHtmlString html = default(MvcHtmlString);

 if (isReadonly)
 {
 html = System.Web.Mvc.Html.InputExtensions.TextBoxFor(htmlHelper,
 expression, new { @class = "readOnly",
 @readonly = "read-only" });
 }
 else
 {
 html = System.Web.Mvc.Html.InputExtensions.TextBoxFor(htmlHelper,
 expression);
 }
 return html;
 }
}
```

Passing Data in MVC

Model Property

- ViewPage.**Model** Property
- Gets the Model property of the associated ViewDataDictionary object.
- Property Value: Object

- Namespace: System.Web.Mvc
- Assembly: System.Web.Mvc.dll

ViewBag, ViewData And TempData In MVC

- ViewBag, ViewData, and TempData all are objects in ASP.NET MVC and these are used to pass the data in various scenarios.

- The following are the scenarios where we can use these objects:
  1. Pass the data from Controller to View.
  2. Pass the data from one action to another action in the same Controller.
  3. Pass the data in between Controllers.
  4. Pass the data between consecutive requests.

ViewBag, ViewData And TempData

- ViewBag and ViewData are used for the same purpose to pass the data from Controller action to View

- TempData is used to pass the data from action to another action or one Controller to another Controller.

ViewData

- ViewData is a dictionary object to pass the data from Controller to View where data is passed in the form of key-value pair “it is accessible by string as key.”.
- ViewData is a property of controller that exposes an instance of the ViewDataDictionary class.
- Typecasting is required to read the data in View if the data is complex and we need to ensure null check to avoid null exceptions.
- The scope of ViewData is similar to ViewBag and it is restricted to the current request and the value of ViewData will become null while redirecting.

ViewData Example

```
01. //Controller Code
02. public ActionResult Index()
03. {
04.         List<string> Student = new List<string>();
05.         Student.Add("Jignesh");
06.         Student.Add("Tejas");
07.         Student.Add("Rakesh");
08.
09.         ViewData["Student"] = Student;
10.         return View();
11. }
12. //page code
13. <ul>
14.     <% foreach (var student in ViewData["Student"] as List<string>)
15.             { %>
16.     <li><%: student%></li>
17.     <% } %>
18. </ul>
```

ViewBag

- ViewBag is a dynamic object to pass the data from Controller to View. And, this will pass the data as a property of object ViewBag.
- ViewBag is able to set and get value dynamically and able to add any number of additional fields without converting it to strongly typed. ViewBag is just a wrapper around the ViewData.
- We have no need to typecast to read the data or for null checking.
- The scope of ViewBag is permitted to the current request and the value of ViewBag will become null while redirecting.

ViewBag Example

```
01.  //Controller Code
02.  public ActionResult Index()
03.  {
04.        List<string> Student = new List<string>();
05.        Student.Add("Jignesh");
06.        Student.Add("Tejas");
07.        Student.Add("Rakesh");
08.
09.        ViewBag.Student = Student;
10.        return View();
11.  }
12.  //page code
13.  <ul>
14.       <% foreach (var student in ViewBag.Student)
15.            { %>
16.       <li><%: student%></li>
17.       <% } %>
18.  </ul>
```

TempData

- TempData is a dictionary object which is derived from TempDataDictionary class used to pass the data from one action to other action in the same Controller or different Controllers.
- TempData is stored data just like live session for short time. TempData keeps data for the time of HTTP Request, which means that it holds data between two consecutive requests. Note that TempData is only work during the current and subsequent request.
- Usually, TempData object will be stored in a session object.
- Tempdata is also required to typecast and for null checking before reading data from it.
- TempData generally used to store one time messages, its scope is limited to the next request and if we want Tempdata to be available even further, we should use TempData.Keep() and peek.

TempData Example

```
01. //Controller Code
02. public ActionResult Index()
03. {
04.     List<string> Student = new List<string>();
05.     Student.Add("Jignesh");
06.     Student.Add("Tejas");
07.     Student.Add("Rakesh");
08.
09.     TempData["Student"] = Student;
10.     return View();
11. }
12. //page code
13. <ul>
14.     <% foreach (var student in TempData["Student"] as List<string>)
15.         { %>
16.     <li><%: student%></li>
17.     <% } %>
18. </ul>
```

ViewData	ViewBag	TempData
It is Key-Value Dictionary collection	It is a type object	It is Key-Value Dictionary collection
ViewData is a dictionary object and it is property of ControllerBase class	ViewBag is Dynamic property of ControllerBase class.	TempData is a dictionary object and it is property of controllerBase class.
ViewData is Faster than ViewBag	ViewBag is slower than ViewData	NA
ViewData is introduced in MVC 1.0 and available in MVC 1.0 and above	ViewBag is introduced in MVC 3.0 and available in MVC 3.0 and above	TempData is also introduced in MVC1.0 and available in MVC 1.0 and above.
ViewData also works with .net framework 3.5 and above	ViewBag only works with .net framework 4.0 and above	TempData also works with .net framework 3.5 and above
Type Conversion code is required while enumerating	In depth, ViewBag is used dynamic, so there is no need to type conversion while enumerating.	Type Conversion code is required while enumerating
Its value becomes null if redirection has occurred.	Same as ViewData	TempData is used to pass data between two consecutive requests.
It lies only during the current request.	Same as ViewData	TempData only works during the current and subsequent request

Demo

MVC Applications With EF Model

Demo

RAZOR ENGINE

BUILDING VIEW PAGES USING RAZOR LANGUAGE

Razor Engine

- A new view-engine
- Optimized around HTML generation
- Code-focused templating approach

Razor Markup

- Razor is a simple **markup syntax** for embedding server code (C# or VB) into ASP.NET web pages.

Layout with Razor

- Layout views are “master pages” for razor
- Use inherited methods to specify content areas
  - RenderBody
  - RenderSection

```
<!DOCTYPE html>
<html>
<head>
    <title>@ViewBag.Title</title>
    <script src="@Url.Content("~/Scripts/jquery-1.4.4.min.js")"
            type="text/javascript"></script>
</head>

<body>
    @RenderBody()
</body>
</html>
```

Organization and Consistency

- Use layouts to ensure consistent page structure
- Layout methods
  - `RenderBody()`
    - Renders anything in a view not in a section
  - `RenderSection(name, required)`
    - Allow views to add specific sections
      - Scripts
      - Banners
      - Sidebars
    - Use `@section name` to create section in view
      - Note the casing

Layout

- Naming convention for layouts is to prefix the file with an underscore (_) character
- Layout files are placed in the /Views/Shared folder
- All views applied this Layout by default, through the /Views/_ViewStart.cshtml file
- @{Layout=null} property

- Microsoft Tutorial:
  - https://docs.microsoft.com/en-us/aspnet/mvc/
  - https://www.asp.net/mvc
- Introduction into mvc
  - https://app.pluralsight.com/player?author=scott-allen&name=mvc3-building-intro&mode=live&clip=0&course=aspdotnet-mvc3-intro
- Advanced Topics
  - https://app.pluralsight.com/player?author=scott-allen&name=aspdotnet-mvc5-fundamentals-m1-introduction&mode=live&clip=0&course=aspdotnet-mvc5-fundamentals