

View trong ASP.Net MVC

- ◆ Khái niệm về View
- ◆ Sử dụng Razor trong View
- ◆ Sử dụng bộ sinh mã HTML Helper

Làm việc với View

- ◆ Để hiển thị nội dung cho người dùng, ta cần xây dựng các phương thức action trong Controller để trả về View.
- ◆ View
 - ◆ Cung cấp giao diện cho ứng dụng ASP.Net MVC
 - ◆ Được sử dụng như giao diện để hiển thị dữ liệu trong ứng dụng hoặc thu thập dữ liệu mà người dùng muốn gửi lên máy chủ qua các form
 - ◆ Giao diện có thể sử dụng các Model
 - ◆ Trong View có thể chứa cả mã HTML và các mã thực thi phía máy chủ

View Engine

- ◆ Là một phần của MVC Framework hỗ trợ chuyển đổi mã lệnh trong View thành mã HTML mà trình duyệt có thể hiển thị được
- ◆ View Engine được chia thành hai loại
 - ◆ Web form view engine: là view engine cũ hỗ trợ làm việc với Views theo cú pháp giống như trong ASP.Net web form
 - ◆ Razor view engine: là view engine ngầm định được giới thiệu bắt đầu từ MVC3. View engine này không giới thiệu ngôn ngữ mới mà thay vào đó là giới thiệu một cú pháp mới, linh hoạt giúp chuyển đổi và trộn mã HTML với ngôn ngữ nền C# trên View một cách đơn giản hơn

Tạo View cho một Action

- ◆ Khi tạo ứng dụng ASP.Net MVC, ta thường phải chỉ định một View để trình bày kết quả trả ra của một Action.
- ◆ Khi tạo một dự án ASP.Net MVC trên Visual Studio IDE, trong cấu trúc cây thư mục của dự án có một thư mục tên là Views chứa các View để trình bày kết quả trả ra của các Action cho người dùng.
- ◆ Trong một ứng dụng, nếu một Action của Controller trả về một View thì trong dự án sẽ có:
 - ◆ Một thư mục con trong thư mục Views với tên giống của Controller, thư mục này sẽ chứa tất cả các View hiển thị dữ liệu trả về bởi các Action của Controller này.
 - ◆ Một file View có phần mở rộng .cshtml đặt trong thư mục nêu trên với tên chính là tên của Action tương ứng.

Tạo View cho một Action

- ◆ Đoạn mã sau là Action Index trả về một đối tượng ActionResult thông qua việc gọi phương thức View() của lớp Controller

```
public class HomeController : Controller {  
    public ActionResult Index()  
    {  
        return View();  
    }  
}
```

- ◆ Trong đoạn mã trên, Action Index của Controller tên là HomeController trả về kết quả bằng cách gọi phương thức View(). Kết quả của phương thức View() là một đối tượng ActionResult cho phép hiển thị View.

Tạo View cho một Action

- ◆ Trong Visual Studio IDE, ta có thể tạo View cho Action một cách đơn giản
- ◆ Các bước tạo View cho một Action:
 1. Nháy chuột phải vào Action mà ta muốn tạo View.
 2. Từ menu ngữ cảnh được hiển thị ta chọn Add View. Hộp thoại Add View sẽ xuất hiện như hình bên

The screenshot shows the 'Add View' dialog box in Visual Studio. The dialog is titled 'Add View' and contains the following fields and options:

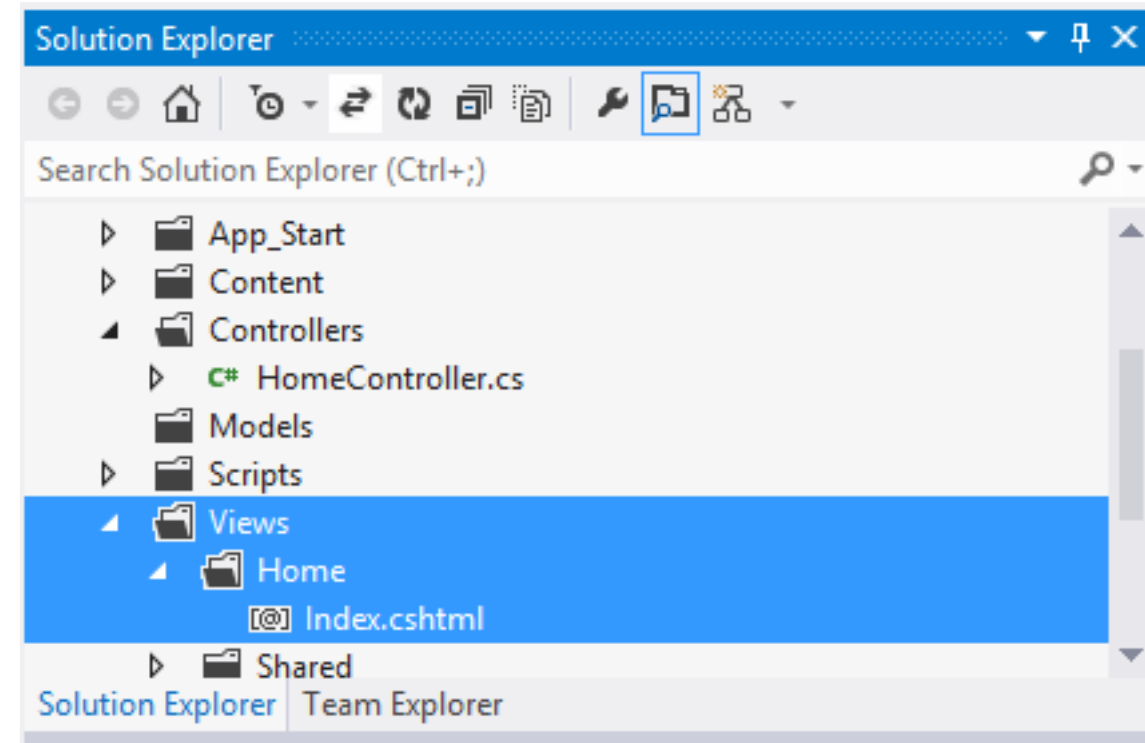
- View name:** Index
- View engine:** Razor (CSHTML)
- ☐ Create a strongly-typed view
- Model class:** (empty)
- Scaffold template:** Empty
- ☒ Reference script libraries
- ☐ Create as a partial view
- ☒ Use a layout or master page:
- ContentPlaceHolder ID:** MainContent

At the bottom right, there are 'Add' and 'Cancel' buttons.

Tạo View cho một Action

3. Click Add. Visual Studio IDE sẽ tự động tạo thư mục theo cấu trúc đã quy định và thêm file View vào thư mục đó.

Hình bên mô tả file View được tạo ra cho Action Index của lớp HomeController trong cửa sổ Solution Explorer



Tạo View cho một Action

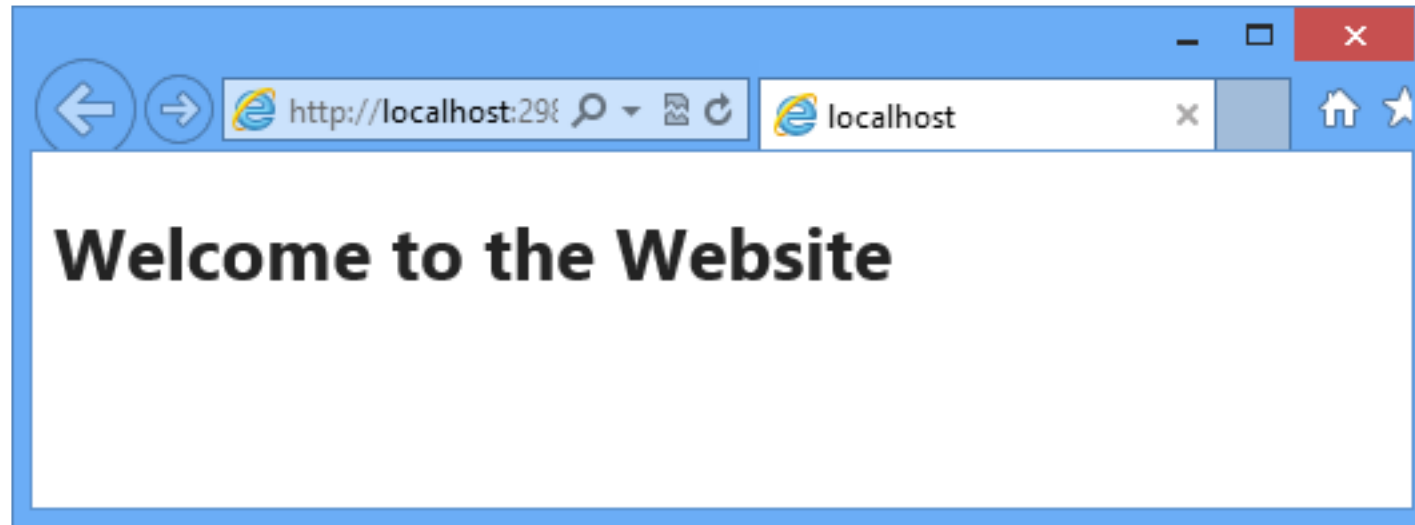
- ◆ Trong file Index.cshtml có thể thêm đoạn mã sau để hiển thị View

```
<!DOCTYPE html>
<html>
  <head>
    <title>Test View</title>
  </head>
  <body>
    <h1> Welcome to the Website </h1>
  </body>
</html>
```

- ◆ Đoạn mã trên tạo View với tiêu đề là <title> và một thông điệp hiển thị trong thẻ <h1>

Tạo View cho một Action

- ◆ Khi truy cập Action Index của HomeController từ trình duyệt thì View Index.cshtml sẽ hiển thị.
- ◆ Để truy cập tới Action Index của HomeController ta sử dụng URL sau
<http://localhost:1267/Home/Index>
- ◆ Hình sau thể hiện View Index.cshtml được hiển thị trên trình duyệt



- ◆ Ta có thể hiển thị các View khác nhau từ phương thức Action
- ◆ Để trả về một View khác, ta truyền tên View vào làm tham số
- ◆ Đoạn mã dưới đây mô tả cách trả về một View với tên TestIndex từ Action

```
public class HomeController : Controller
{
    public ActionResult Index()
    {
        return View("TestIndex");
    }
}
```

- ◆ Theo đoạn mã này, View sẽ được tìm kiếm trong thư mục /Views/Home nhưng trả về View tên là TestIndex thay vì View Index như trong ví dụ trước

- ◆ Ta cũng có thể trả về View ở thư mục khác thư mục ngầm định gắn với Controller bằng cách chỉ đường dẫn tới View đó
- ◆ Đoạn mã sau sẽ trả về View có tên Welcome.cshtml trong thư mục /Views/Demo

```
public class HomeController : Controller
{
    public ActionResult Index()
    {
        return View("~/Views/Demo/Welcome.cshtml");
    }
}
```

- ◆ Đoạn code trên sẽ hiển thị View tên là Welcome.cshtml được đặt trong thư mục /Views/Demo

Truyền dữ liệu từ Controller về View

- ◆ Trong ứng dụng ASP.Net MVC, Controller thường thực thi các logic nghiệp vụ rồi gửi trả kết quả về cho người dùng thông qua View.
- ◆ Ta có thể sử dụng các đối tượng sau để truyền dữ liệu giữa Controller và View
 - ◆ ViewData
 - ◆ ViewBag
 - ◆ TempData

Truyền dữ liệu từ Controller về View

◆ ViewData

- ◆ Truyền dữ liệu từ Controller về View
- ◆ Là một từ điển các đối tượng thuộc lớp dẫn xuất từ lớp ViewDataDictionary
- ◆ Một số đặc điểm của ViewData
 - ◆ Vòng đời của đối tượng ViewData chỉ tồn tại trong request hiện tại
 - ◆ Giá trị của ViewData sẽ là null nếu request bị redirect (điều hướng)
 - ◆ ViewData cần chuyển kiểu khi ta dùng các kiểu dữ liệu phức hợp
- ◆ Cú pháp sử dụng ViewData
- ◆ Trong đó:
 - ◆ Key: kiểu String để xác định chính xác phần tử trong ViewData
 - ◆ Value: là giá trị của đối tượng trong ViewData, có thể kiểu String hoặc các kiểu khác ví dụ như DateTime.

```
ViewData[<key>] = <Value>;
```

Truyền dữ liệu từ Controller về View

- ◆ Đoạn mã sau mô tả sử dụng ViewData với 2 cặp key-value trong Action Index của lớp HomeController

```
public class HomeController : Controller {  
    public ActionResult Index()  
    {  
        ViewData["Message"] = "Message from ViewData";  
        ViewData["CurrentTime"] = DateTime.Now;  
        return View();  
    }  
}
```

- ◆ Trong đoạn mã trên, ViewData được tạo với 2 cặp key-value.
 - ◆ Khóa thứ nhất là “Message” lưu giá trị kiểu String là “Message from ViewData”
 - ◆ Khóa thứ hai là “CurrentTime” lưu giá trị kiểu DateTime được trả về bởi DateTime.Now

Truyền dữ liệu từ Controller về View

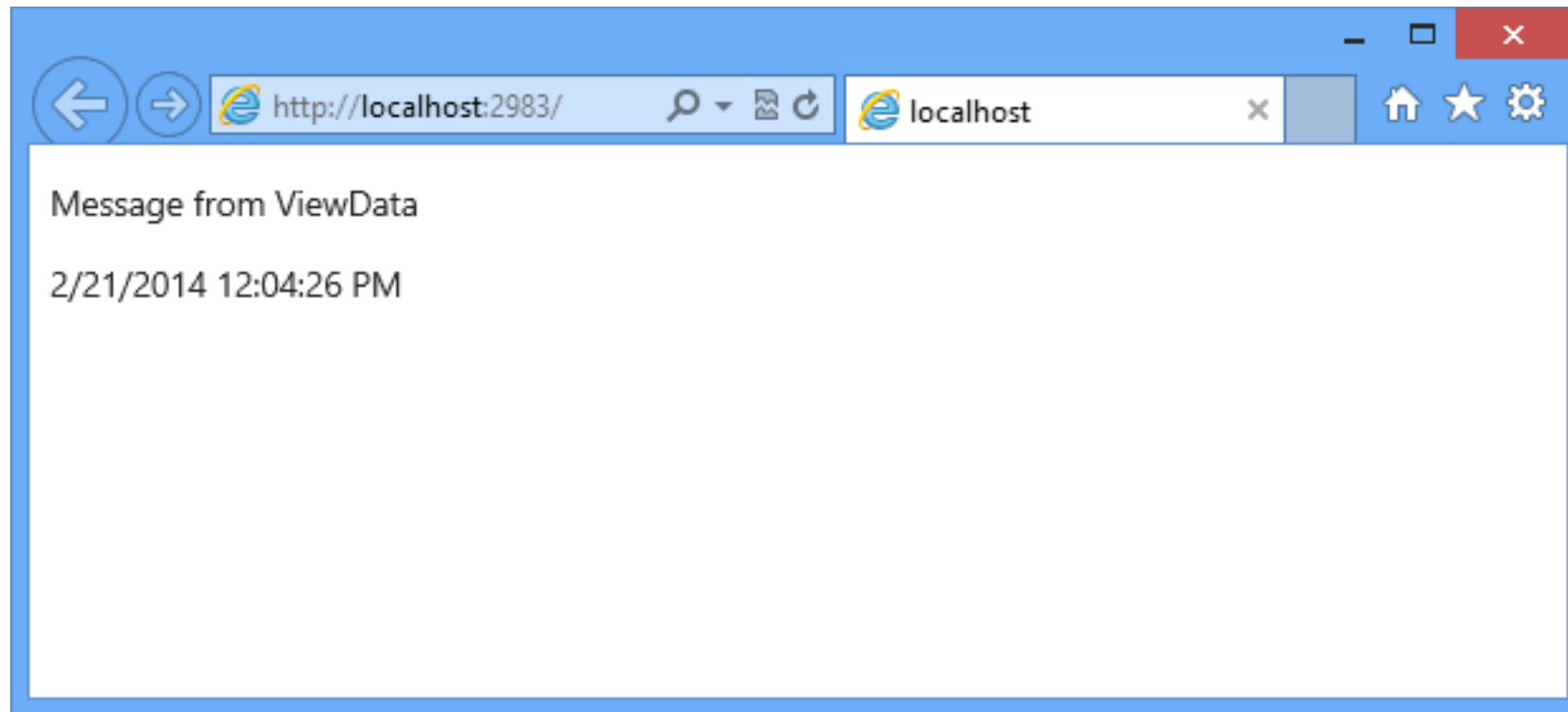
- ◆ Đoạn mã sau mô tả cách lấy dữ liệu từ ViewData

```
<!DOCTYPE html>
<html>
  <head>
    <title>Index View</title>
  </head>
  <body>
    <p> @ViewData["Message"] </p>
    <p> @ViewData["CurrentTime"] </p>
  </body>
</html>
```

- ◆ Trong đoạn mã trên, ViewData được sử dụng để hiển thị giá trị của khóa “Message” và “CurrentTime”

Truyền dữ liệu từ Controller về View

- ◆ Kết quả được thể hiện như sau



Truyền dữ liệu từ Controller về View

◆ ViewBag

- ◆ Được sử dụng để thay thế ViewData
- ◆ Cũng chỉ tồn tại trong request hiện tại và bị rỗng (null) khi request bị redirected
- ◆ Là một tính năng linh động được giới thiệu bắt đầu từ C# 4.0
- ◆ Không cần chuyển kiểu (typecasting) khi sử dụng các kiểu dữ liệu phức hợp
- ◆ Cú pháp sử dụng ViewBag
- ◆ Trong đó:
 - ◆ Property: kiểu String là tên thuộc tính của ViewBag
 - ◆ Value: là giá trị của thuộc tính đó trong ViewBag

```
ViewBag.<Property> = <Value>;
```

Truyền dữ liệu từ Controller về View

- ◆ Đoạn mã sau mô tả sử dụng ViewBag với 2 thuộc tính trong Action Index của lớp HomeController

```
public class HomeController : Controller
{
    public ActionResult Index()
    {
        ViewBag.Message = "Message from ViewBag";
        ViewBag.CurrentTime = DateTime.Now;
        return View();
    }
}
```

- ◆ Trong đoạn mã trên, đối tượng ViewBag được tạo với 2 thuộc tính
 - ◆ Thuộc tính thứ nhất là “Message” lưu giá trị kiểu String là “Message from ViewBag”
 - ◆ Thuộc tính thứ hai là “CurrentTime” lưu giá trị kiểu DateTime được trả về bởi DateTime.Now

Truyền dữ liệu từ Controller về View

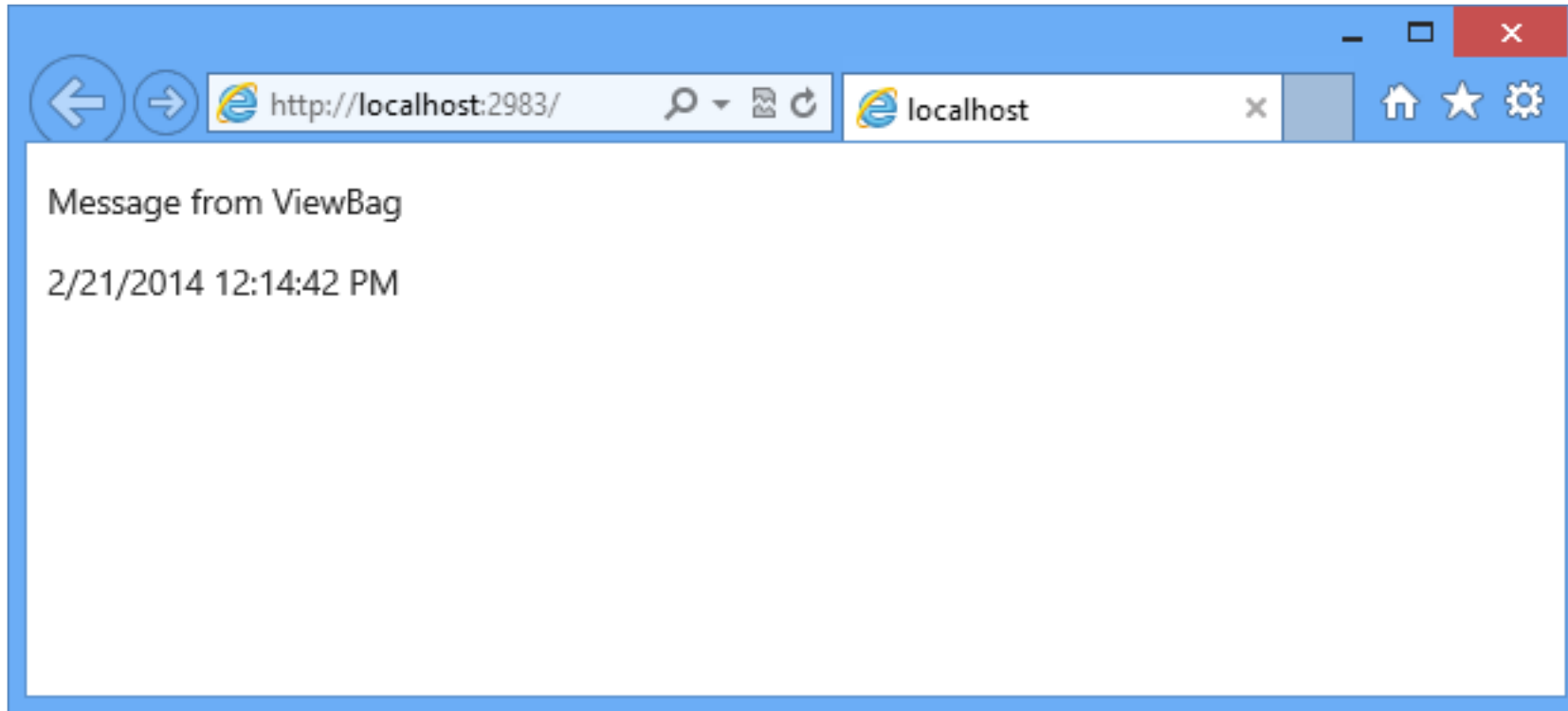
- ◆ Đoạn mã sau mô tả cách lấy dữ liệu từ ViewBag

```
<!DOCTYPE html>
<html>
  <head>
    <title>Index View</title>
  </head>
  <body>
    <p>@ViewBag.Message</p>
    <p>@ViewBag.CurrentTime</p>
  </body>
</html>
```

- ◆ Trong đoạn mã trên, ViewBag được sử dụng để hiển thị giá trị của thuộc tính “Message” và “CurrentTime”

Truyền dữ liệu từ Controller về View

- ◆ Kết quả được thể hiện như sau



- ◆ Khi sử dụng ViewBag để lưu trữ thuộc tính và giá trị trong Action thì trong View thuộc tính đó có thể được truy cập bởi cả ViewBag và ViewData

Truyền dữ liệu từ Controller về View

- ◆ Đoạn mã sau mô tả lưu trữ dữ liệu bằng ViewBag trong Action

```
public class HomeController : Controller
{
    public ActionResult Index()
    {
        ViewBag.CommonMessage = "Common message accessible to both
                                ViewBag and ViewData";

        return View();
    }
}
```

- ◆ Trong đoạn mã trên, đối tượng ViewBag được tạo với thuộc tính tên là CommonMessage

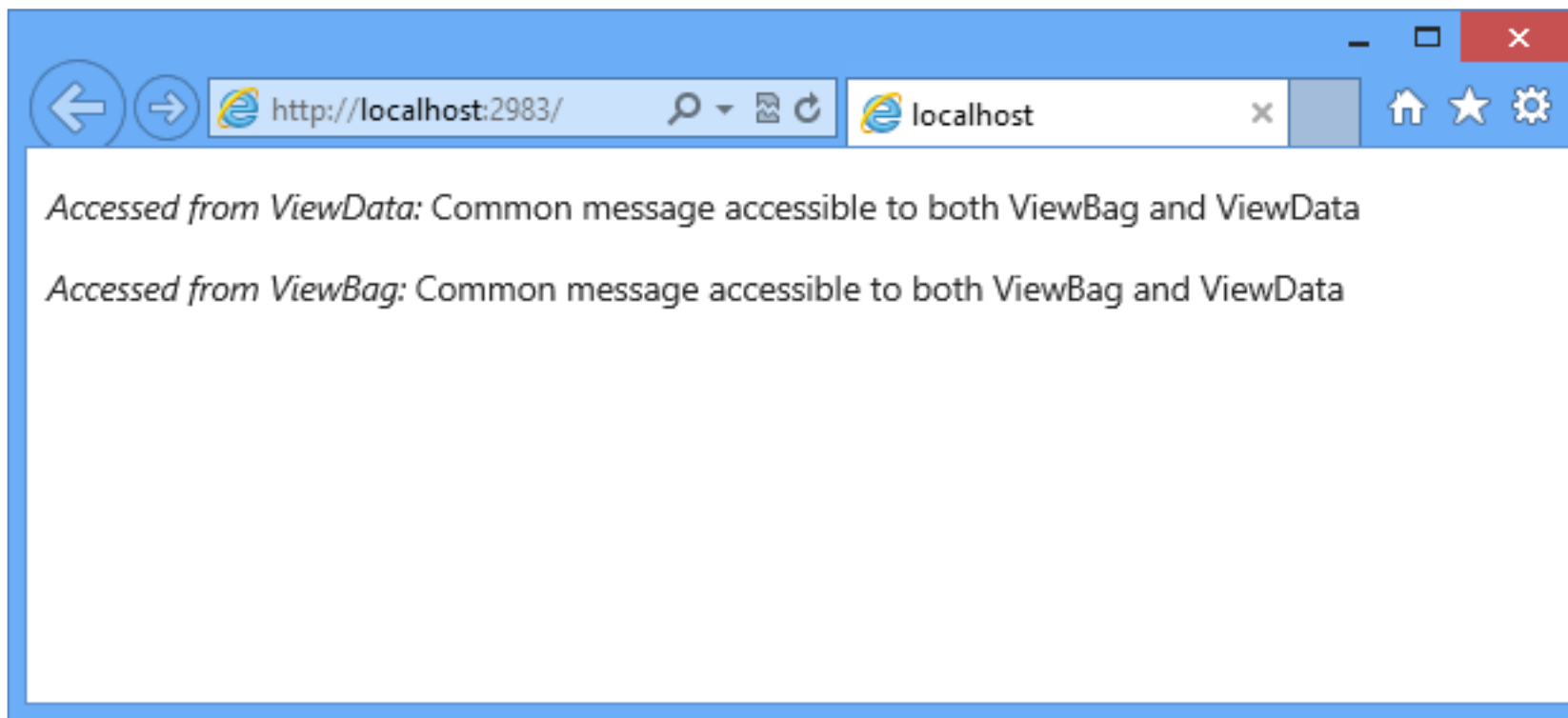
Truyền dữ liệu từ Controller về View

- ◆ Đoạn mã sau mô tả cách lấy dữ liệu từ ViewBag
- ◆ Trong đoạn mã trên, cả hai đối tượng ViewBag và ViewData được dùng để hiển thị dữ liệu lưu trữ trong thuộc tính CommonMessage của ViewBag

```
<!DOCTYPE html>
<html>
  <head>
    <title>Index View</title>
  </head>
  <body>
    <p>
      <em>Accessed from ViewData:</em>
      @ViewData["CommonMessage"]
    </p>
    <p>
      <em>Accessed from ViewBag:</em>
      @ViewBag.CommonMessage
    </p>
  </body>
</html>
```

Truyền dữ liệu từ Controller về View

- ◆ Kết quả được thể hiện như sau



Truyền dữ liệu từ Controller về View

◆ TempData

- ◆ Là một đối tượng từ điển dẫn xuất từ lớp TempDataDictionary
- ◆ Lưu trữ dữ liệu dưới dạng các cặp key-value (khóa-giá trị)
- ◆ Cho phép truyền dữ liệu từ request hiện thời tới request tiếp theo khi bị điều hướng (redirected)
- ◆ Cú pháp sử dụng TempData
- ◆ Trong đó:
 - ◆ Key: kiểu String là khóa cho phép định danh đối tượng dữ liệu được lưu trong TempData
 - ◆ Value: là giá trị của đối tượng dữ liệu được lưu trong TempData

```
TempData[<Key>] = <Value>;
```

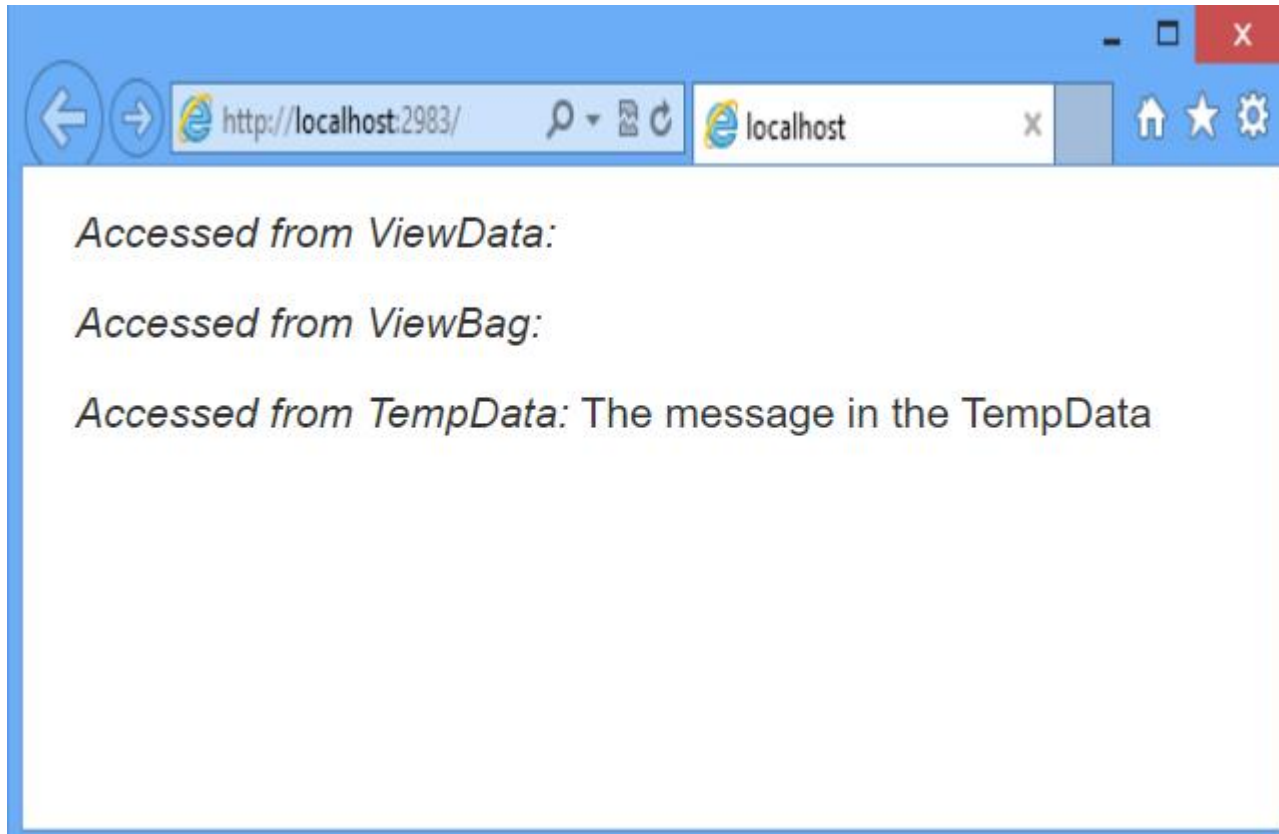
Truyền dữ liệu từ Controller về View

- ◆ Đoạn mã sau mô tả sử dụng TempData để truyền dữ liệu từ View này tới View khác thông qua điều cơ chế điều hướng (redirect)
- ◆ Đoạn mã trên tạo 2 Action
 - ◆ Index Action lưu trữ dữ liệu bằng cả đối tượng ViewData, ViewBag và TempData. Sau đó, điều hướng tới Action About bằng việc gọi phương thức Redirect("Home/About")
 - ◆ About Action trả về View tương ứng đó là View.cshtml

```
public class HomeController : Controller {  
    public ActionResult Index() {  
        ViewData["Message"] = "The message in the  
        ViewData";  
        ViewBag.Message = "The message in the  
        ViewBag";  
        TempData["Message"] = "The message in the  
        TempData";  
        return Redirect("Home/About");  
    }  
    public ActionResult About() {  
        return View();  
    }  
}
```

Truyền dữ liệu từ Controller về View

◆ Kết quả được thể hiện như sau



```
<!DOCTYPE html>
<html>
  <head>
    <title>About View</title>
  </head>
  <body>
    <p>
      <em>Accessed from ViewData: </em>
      @ViewData["Message"]
    </p>
    <p>
      <em>Accessed from ViewBag: </em>
      @ViewBag.Message
    </p>
    <p>
      <em>Accessed from TempData: </em>
      @TempData["Message"]
    </p></body>
</html>
```

Sử dụng Partial View

- ◆ Partial View:
 - ◆ Hiển thị một phần của một View chính
 - ◆ Cho phép các View khác nhau dùng lại các phần nội dung chung (Layout)
- ◆ Để tạo Partial View ta làm các bước sau:
 1. Nháy chuột phải vào thư mục Views/Shared trong cửa sổ Solution Explorer và chọn Add -> View. Hộp thoại Add View sẽ xuất hiện
 2. Trong hộp thoại Add View, nhập tên Partial View muốn tạo vào mục View Name
 3. Tích vào checkbox Create as a partial view
 4. Chọn Add

The screenshot shows the 'Add View' dialog box with the following configuration:

- View name:** _TestPartialView
- View engine:** Razor (CSHTML)
- ☐ Create a strongly-typed view
- Model class:** (empty)
- Scaffold template:** Empty
- ☒ Reference script libraries
- ☒ Create as a partial view
- ☒ Use a layout or master page:
 - Text field: (empty)
 - Text: (Leave empty if it is set in a Razor _viewstart file)
- ContentPlaceHolder ID:** MainContent
- Buttons:** Add, Cancel

Sử dụng Partial View

- ◆ Trong Partial View ta có thể chèn mã HTML mà ta muốn nó hiển thị như phần dùng chung trong các View chính. Ví dụ chèn mã sau

```
<h3> Content of partial view. </h3>
```

- ◆ Cú pháp để chèn Partial View trong View chính như sau:

```
@Html.Partial(<partial_view_name>)
```

- ◆ Trong đó, <partial_view_name> là tên của Partial View bỏ đi phần mở rộng .cshtml

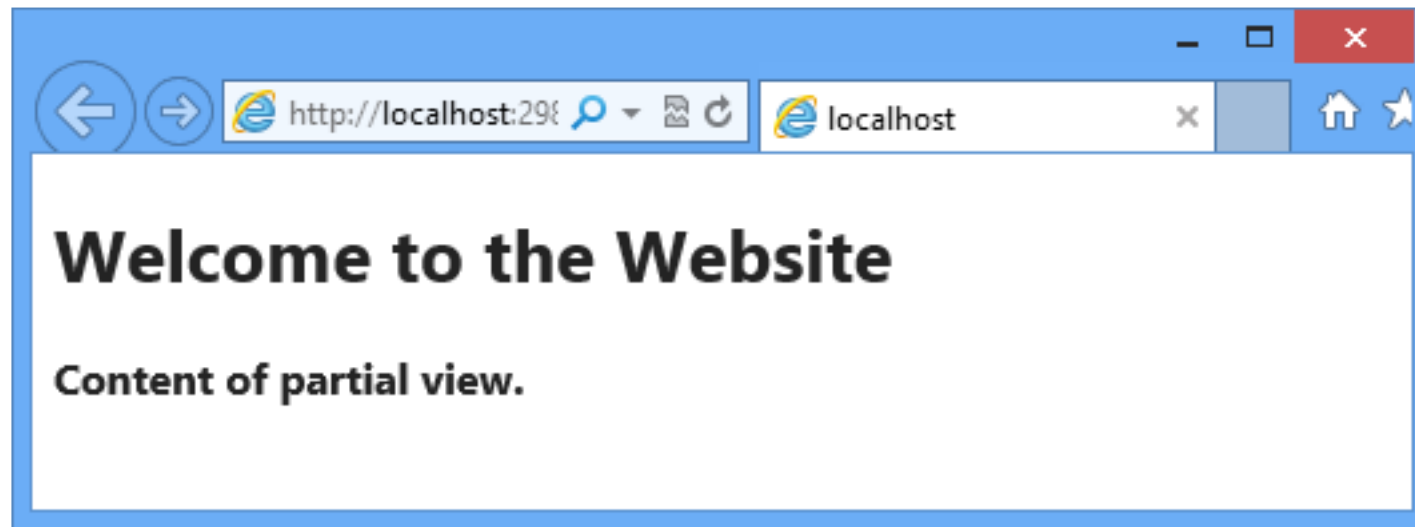
Sử dụng Partial View

- ◆ Đoạn mã sau thể hiện View chính tên là Index.cshtml hiển thị Partial View tên là _TestPartialView

```
<!DOCTYPE html>
<html>
  <head>
    <title>Index View</title>
  </head>
  <body>
    <h1>
      Welcome to the Website
    </h1>
    <div>@Html.Partial("_TestPartialView")</div>
  </body>
</html>
```

Sử dụng Partial View

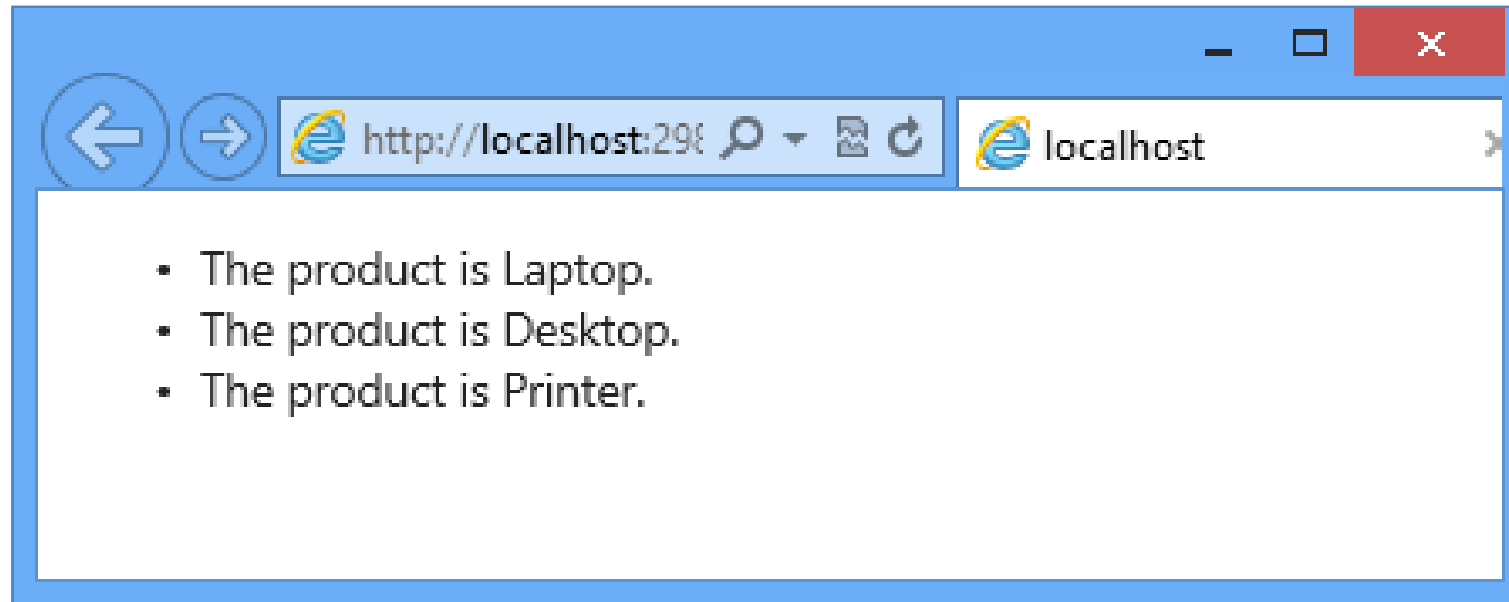
- ◆ View Index.cshtml sẽ được hiển thị như sau



- ◆ Là cú pháp dựa trên ASP.Net Framework cho phép tạo View.
- ◆ Dễ hiểu hơn khi người lập trình phải trộn cả mã HTML với các mã viết bằng ngôn ngữ nền C# hoặc VB.Net
- ◆ Trong đoạn mã ở bên
 - ◆ Mảng string[] được khai báo và khởi tạo bằng cách sử dụng cú pháp Razor
 - ◆ Sau đó cú pháp Razor @foreach để lặp qua từng phần tử của mảng và hiển thị
 - ◆ Các phần mã còn lại là mã HTML hiển thị title, body và các phần tử trong danh sách

```
@{  
var products = new string[] {"Laptop",  
                             "Desktop", "Printer"};  
}  
<html>  
  <head><title>Test View</title></head>  
  <body>  
    <ul>  
      @foreach (var product in products)  
      {  
        <li>The product is @product.</li>  
      }  
    </ul>  
  </body>  
</html>
```


◆ Kết quả hiển thị như sau



Razor engine

- ◆ Trong MVC Framework, view engine chuyển đổi view thành mã HTML để hiển thị trên trình duyệt.
- ◆ Razor engine:
 - ◆ Được sử dụng như view engine ngầm định của MVC Framework
 - ◆ Biên dịch view khi view được yêu cầu (request) lần đầu tiên
 - ◆ Không phải là một ngôn ngữ lập trình mới, mà cung cấp cú pháp để tách biệt mã đánh dấu HTML và ngôn ngữ nền để lập trình trong View.
 - ◆ Hỗ trợ Test Driven Development (TDD) cho phép kiểm thử các View của một ứng dụng một cách độc lập

◆ Razor

- ◆ Trước tiên, Razor cần phân tách mã phía máy chủ (server-side script) khỏi mã đánh dấu để thông dịch những mã này khi chúng được nhúng bên trong View.
- ◆ Sử dụng ký tự @ để tách mã phía máy chủ (server-side script) khỏi mã HTML (client-side script).
- ◆ Khi sử dụng Razor trong View ta cần xem xét các nguyên tắc sau:
 - ◆ Khối lệnh được viết trong vùng giữa @{ và }
 - ◆ Biểu thức nội dòng được bắt đầu bằng @
 - ◆ Các biến được khai báo với từ khóa var
 - ◆ Các chuỗi được bao bằng cặp dấu nháy “
 - ◆ Mỗi lệnh trong Razor kết thúc bằng dấu ;

Cú pháp Razor

- ◆ Razor hỗ trợ viết các khối mã lệnh server-side trong View. Các khối mã lệnh này có thể được viết bằng ngôn ngữ C# hoặc VB.Net
 - ◆ Cú pháp chung của Razor `@{ <code>}`
 - ◆ Trong đó, <code> là khối mã lệnh được viết bằng ngôn ngữ C# hoặc VB.Net
 - ◆ Đoạn mã sau đây có hai khối Razor, mỗi khối chỉ có một lệnh đơn
- ```
@{ var myMessage = "Hello World"; }
@{ var num = 10;}
```
- ◆ Hai khối Razor mỗi khối chỉ có một lệnh đơn để định nghĩa biến myMessage và num. Dấu @{ bắt đầu khối và dấu } kết thúc khối

# Cú pháp Razor

- ◆ Razor hỗ trợ viết nhiều lệnh trong một khối
- ◆ Cú pháp chung của Razor
- ◆ Trong đó, <code> là khối mã lệnh được viết bằng ngôn ngữ C# hoặc VB.Net
- ◆ Đoạn mã sau đây có một Razor bao gồm hai lệnh đơn khai báo hai biến myMessage và num như ví dụ trên

```
@{
 var myMessage = "Hello World";
 var num = 10;
}
```

# Cú pháp Razor

- ◆ Cũng giống như khối lệnh, Razor cũng sử dụng dấu @ để thể hiện một biểu thức nội dòng
- ◆ Đoạn mã sau đây có sử dụng biểu thức nội dòng

```
@{
 var myMessage = "Hello World";
 var num = 10;
}
@myMessage is numbered @num.
```

- ◆ Dòng cuối cùng trong đoạn mã trên sử dụng hai biểu thức nội dòng để đọc giá trị của hai biến myMessage và num được khai báo trong khối Razor bên trên
- ◆ Mỗi khi Razor đọc được dấu @, ngay lập tức nó thông dịch phần tên biến liền sau đó như mã phía máy chủ với ý nghĩa là đọc giá trị của biến đó.

# Cú pháp Razor

- ◆ Theo cú pháp của Razor, dấu @ được sử dụng để chỉ định mã lệnh server-side. Vậy trong trường hợp ta muốn hiển thị dấu @ thì sao? Ta cần loại bỏ ý nghĩa tạo Razor của dấu @ bằng cách nhân bản nó lên.
- ◆ Ví dụ dòng lệnh sau hiển thị dấu @ trong địa chỉ email

```
<h3>The email ID is: john@@mvcexample.com </h3>
```

# Biến (Variables)

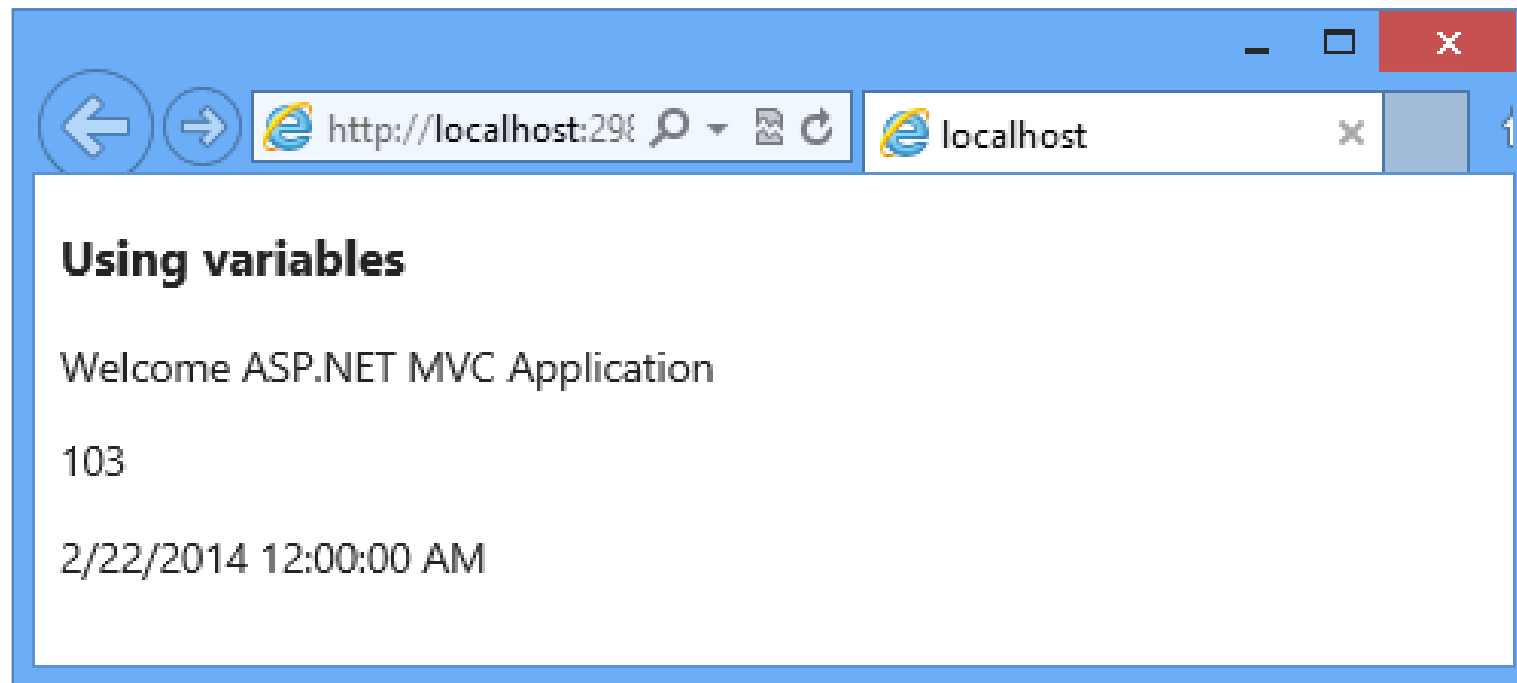
- ◆ Được sử dụng để lưu dữ liệu tạm thời.
- ◆ Trong Razor, ta có thể khai báo và sử dụng biến như khi lập trình với C#
- ◆ Đoạn mã bên cho thấy cách khai báo và sử dụng biến trong Razor

```
<!DOCTYPE html>
<html>
 <body>
 @{
 var heading = "Using variables";
 string greeting = "Welcome ASP.NET MVC Application";
 int num = 103;
 DateTime today = DateTime.Today;
 <h3>@heading</h3>
 <p>@greeting</p>
 <p>@num</p>
 <p>@today</p>
 }
 </body>
</html>
```



# Biến (Variables)

- ◆ Đoạn mã ở trên khai báo 4 biến là heading, greeting, num và today
- ◆ Hình sau đây là kết quả hiển thị



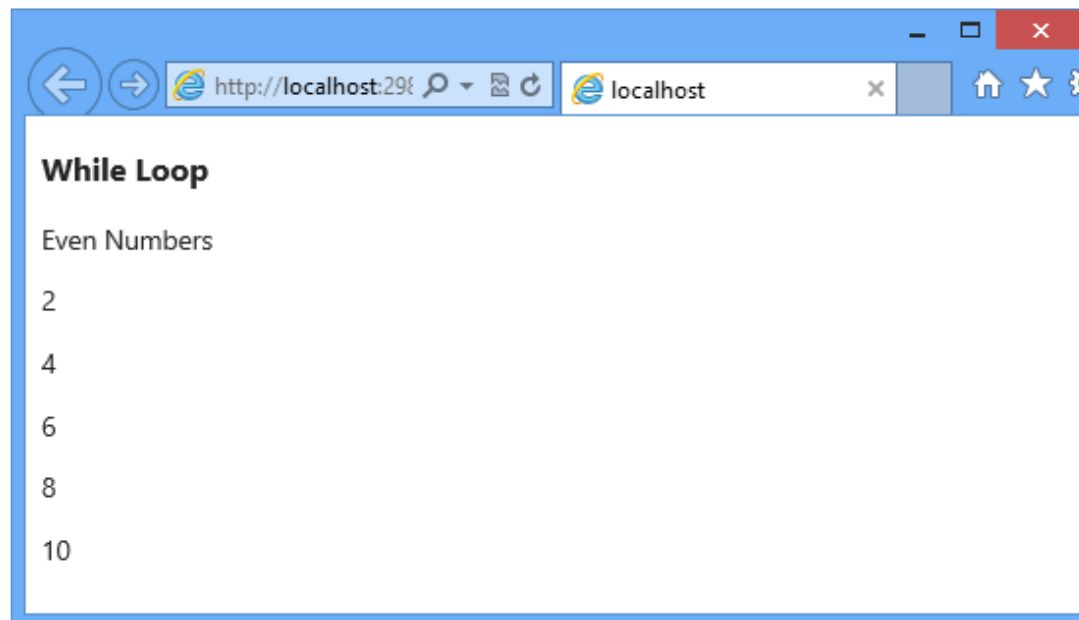
# Vòng lặp

- ◆ Khi phát triển ứng dụng ASP.Net MVC, có thể dùng vòng lặp để thực hiện các công việc giống nhau lặp đi lặp lại liên tiếp.
- ◆ C# hỗ trợ các cấu trúc lặp sau:
  - ◆ Vòng lặp while
  - ◆ Vòng lặp for
  - ◆ Vòng lặp do...while
  - ◆ Vòng lặp foreach

# Vòng lặp

## ◆ Vòng lặp while

- ◆ Được sử dụng để thực hiện các công việc lặp đi lặp lại cho khi nào điều kiện lặp vẫn là true.
- ◆ Cú pháp sử dụng từ khóa while đặt trước cặp dấu { }
- ◆ Đoạn code bên sử dụng Razor với vòng lặp while để in ra các số từ 1 tới 10



```
<!DOCTYPE html>
<html>
 <body>
 @{ var b = 0;
 while (b < 10)
 {
 b += 1;
 <p>Text @b</p>
 }
 }
 </body>
</html>
```

# Vòng lặp

## ◆ Vòng lặp for

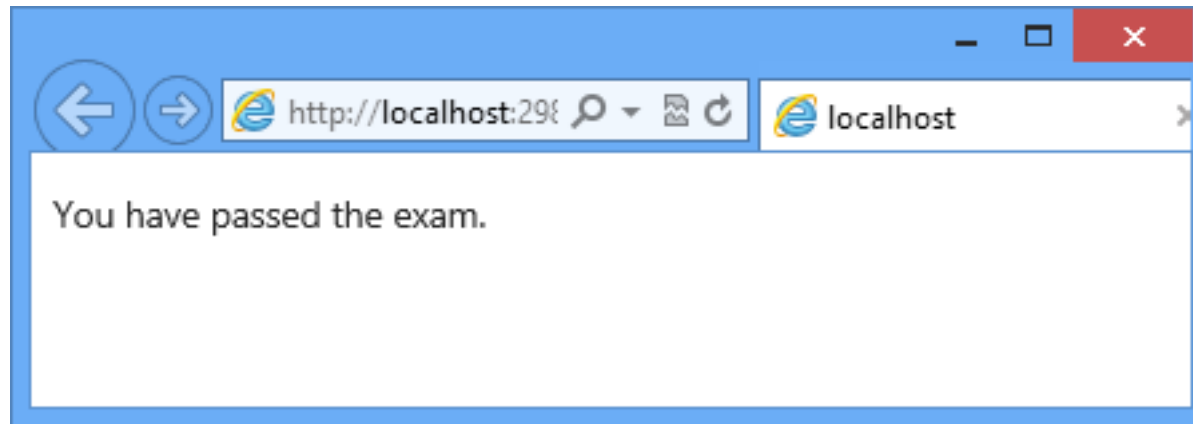
- ◆ Khi biết trước số lần lặp ta có thể dùng vòng lặp for
- ◆ Đoạn code bên sử dụng Razor với vòng lặp for để in ra các số chẵn từ 1 tới 10

```
<!DOCTYPE html>
<html>
 <body>
 <h1>Even Numbers</h1>
 @{ var num=1;
 for (num = 1; num <= 10; num++){
 if ((num % 2) == 0)
 {
 <p> @num</p>
 }
 }
 }
 </body>
</html>
```

# Câu lệnh rẽ nhánh

## ◆ Cấu trúc if ... else

- ◆ Cho phép kiểm tra điều kiện, tùy thuộc vào kết quả kiểm tra điều kiện đúng – true hay sai – false mà công việc tương ứng được thực hiện
- ◆ Đoạn code bên sử dụng cấu trúc if ... else với Razor

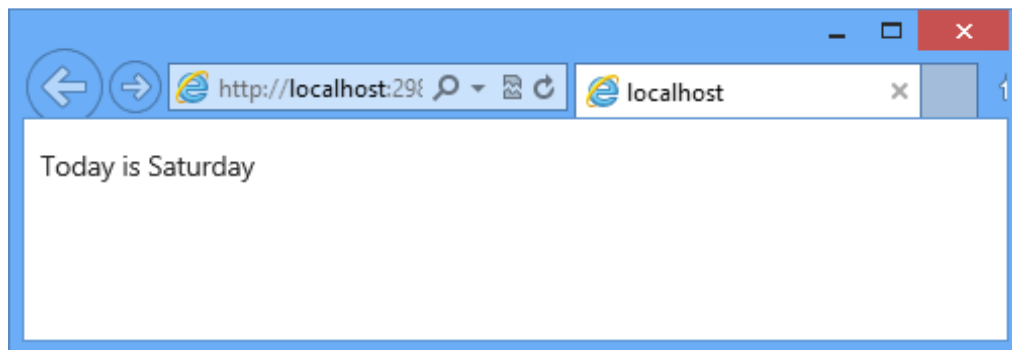


```
<!DOCTYPE html>
@{var mark=60;}
<html>
 <body>
 @if (mark<60) {
 <p>You have failed in the exam.</p> }
 else {
 <p>You have passed the exam.</p> }
 </body>
 </html>
```

# Câu lệnh rẽ nhánh

## ◆ Cấu trúc switch...case

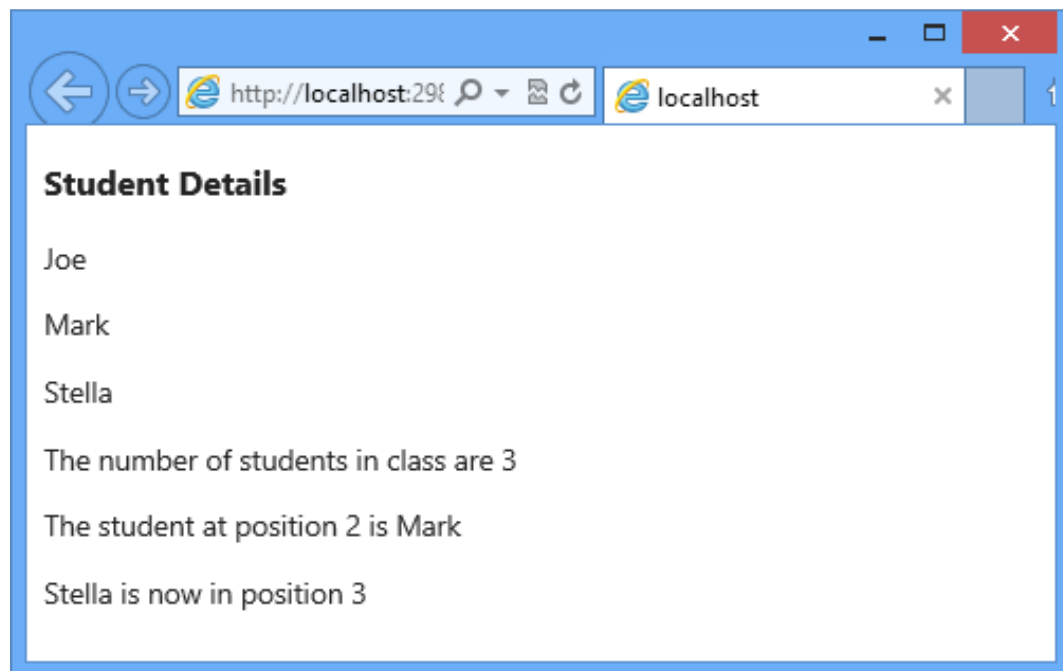
- ◆ Cho phép kiểm tra lần lượt một tập các điều kiện, với mỗi điều kiện đúng xảy ra, công việc tương ứng được thực hiện
- ◆ Đoạn code bên sử dụng cấu trúc switch ... case với Razor
- ◆ Kết quả sẽ hiển thị hôm nay là ngày thứ mấy tùy vào lịch



```
<!DOCTYPE html>
@{
 var day=DateTime.Now.DayOfWeek.ToString();
 var msg=""; }
<html>
 <body>
 @switch(day) {
 case "Monday":
 msg="Today is Monday, the first working day.";
 break;
 case "Friday":
 msg="Today is Friday, the last working day.";
 break;
 default:
 msg="Today is " + day;
 }
 <p>@msg</p>
 </body>
</html>
```

# Mảng

- ◆ Mảng là tập hợp các phần tử cùng kiểu dữ liệu
- ◆ Đoạn mã bên khai báo một mảng members với 03 phần tử



```
<!DOCTYPE html>
@{
 string[] members = {"Joe", "Mark", "Stella"};
 int i = Array.IndexOf(members, "Stella")+1;
 int len = members.Length;
 string x = members[2-1];
}
<html>
<body>
 <h3>Student Details</h3>
 @foreach (var person in members) {
 <p>@person</p>
 }
 <p>The number of students in class are @len</p>
 <p>The student at position 2 is @x</p>
 <p>Stella is now in position @i</p>
</body>
</html>
```

# HTML Helper

- ◆ HTML Helper là một phần của MVC Framework, hỗ trợ sinh mã HTML để việc viết mã HTML, tạo View dễ dàng hơn.
- ◆ Là các phương thức mở rộng chỉ có thể được gọi trong View
- ◆ Đơn giản hóa việc tạo View
- ◆ Cho phép sinh mã HTML có thể sử dụng lại trong ứng dụng ASP.Net MVC
- ◆ Một số phương thức HTML Helper:
  - ◆ `Html.ActionLink()`
  - ◆ `Html.BeginForm()` and `Html.EndForm()`
  - ◆ `Html.Label()`
  - ◆ `Html.TextBox()`
  - ◆ `Html.TextArea()`
  - ◆ `Html.Password()`
  - ◆ `Html.CheckBox()`



# HTML Helper

- ◆ Phương thức `Html.ActionLink()` cho phép tạo hyperlink tới một Action trong một Controller
- ◆ Cú pháp sử dụng `Html.ActionLink` được mô tả như sau

```
@Html.ActionLink(<link_text>,<action_method>,<optional_controller>)
```

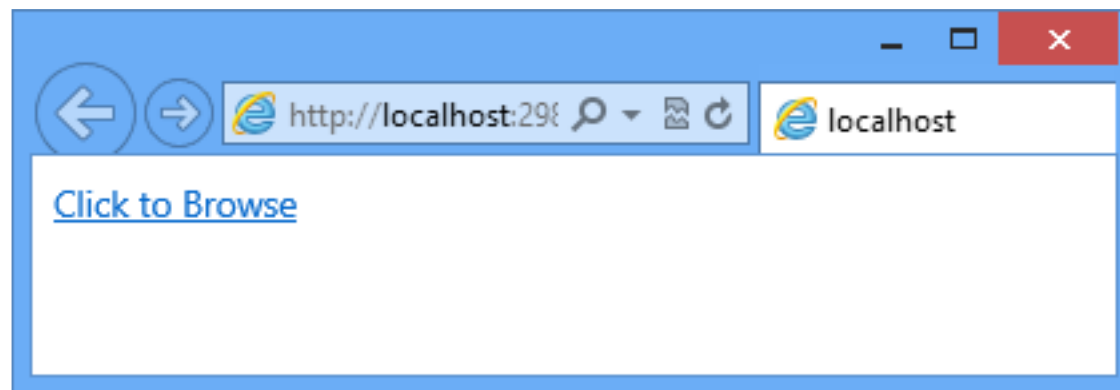
- ◆ Trong đó:
  - ◆ `<link_text>`: là nhãn sẽ hiển thị trong hyperlink
  - ◆ `<action_method>`: là tên Action sẽ được triệu gọi khi nhấn vào đường hyperlink
  - ◆ `<optional_controller>`: là tên Controller chứa Action trên. Tham số này có thể bỏ trống nếu ta muốn link tới Action trong cùng Controller

# HTML Helper

- ◆ Đoạn mã sau là ví dụ sử dụng Html.ActionLink

```
<!DOCTYPE html>
<html>
<body>
@Html.ActionLink("Click to Browse", "Browse", "Home")
</body>
</html>
```

- ◆ Nhấn “Click to Browse” sẽ xuất hiện dưới dạng một hyperlink
- ◆ Action tên là Browse trong Controller Home sẽ được gọi khi nhấn vào hyperlink



- ◆ Phương thức `Html.BeginForm()`

- ◆ Đánh dấu điểm bắt đầu của một HTML form
- ◆ Phối hợp với bộ định tuyến (routing) để tạo URL
- ◆ Được sử dụng để mở một thẻ `<form>`

- ◆ Cú pháp của `Html.BeginForm()`

```
@{Html.BeginForm(<action_method>,<controller_name>);}
```

- ◆ Trong đó:

- ◆ `<action_method>`: là tên Action sẽ xử lý dữ liệu khi form được submit
  - ◆ `<controller_name>`: là tên Controller chứa Action đó
- ◆ Mỗi lần gọi phương thức `Html.BeginForm()` để mở một thẻ `<form>` thì cần đóng thẻ `<form>` này bằng cách gọi phương thức `Html.EndForm()`

# HTML Helper

- ◆ Đoạn mã sau là ví dụ sử dụng `Html.BeginForm()` và `Html.EndForm()` theo cặp

```
<!DOCTYPE html>
<html>
 <body>
 @{Html.BeginForm("Browse","Home");}
 <p>Inside Form</p>
 @{Html.EndForm();}
 </body>
</html>
```

- ◆ Trong đoạn mã trên, phương thức `Html.BeginForm()` chỉ định Action `Browse` trong Controller `Home` sẽ được gọi khi form được submit
- ◆ Để ngắn gọn hơn ta có thể lược bỏ gọi phương thức `Html.EndForm()` bằng việc sử dụng lệnh `@using` phía trước `Html.BeginForm()`

- ◆ Phương thức `Html.Label()`
  - ◆ Cho phép hiển thị một nhãn văn bản trong form
  - ◆ Gắn nhãn văn bản trên với các phần tử nhập liệu trên form, chẳng hạn như các input text, làm tăng khả năng truy nhập của ứng dụng.
- ◆ Cú pháp của `Html.Label()` `@Html.Label(<label_text>)`
- ◆ Trong đó `<label_text>`: là nhãn văn bản được hiển thị

```
@Html.Label("name")
<!DOCTYPE html>
<html>
 <body>
 @{Html.BeginForm("Browse","Home");}
 @Html.Label("User Name:")</br>
 @{Html.EndForm();}
 </body>
</html>
```

- ◆ Phương thức `Html.TextBox()`
  - ◆ Cho phép hiển thị một thẻ input text trong form
  - ◆ Được sử dụng để tiếp nhận dữ liệu từ người dùng
- ◆ Cú pháp của `Html.TextBox()` `@Html.TextBox("textbox_text")`
- ◆ Trong đó `<textbox_text>`: là tên của input text sẽ tạo ra

```
<!DOCTYPE html>
<html>
 <body>
 @{Html.BeginForm("Browse","Home");}
 @Html.Label("User Name:")</br>
 @Html.TextBox("textBox1")</br></br>
 <input type="submit" value="Submit">
 @{Html.EndForm();}
 </body>
</html>
```

# HTML Helper

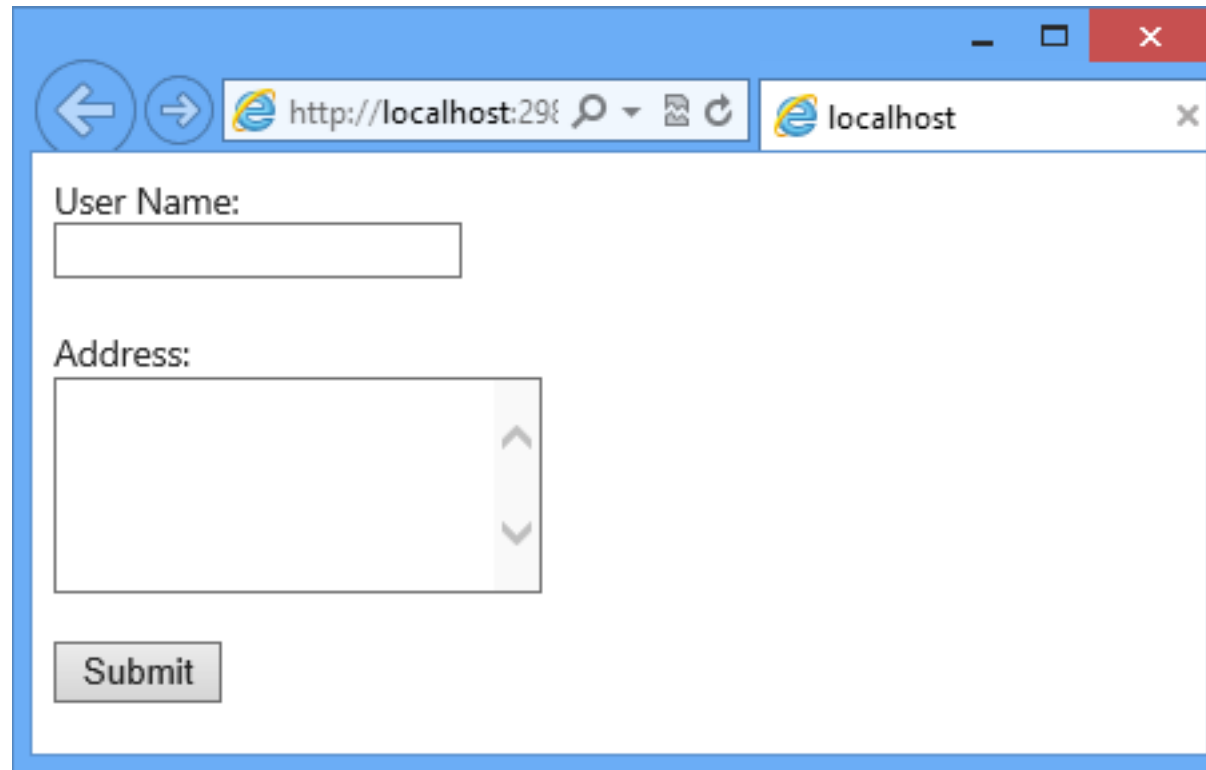
- ◆ Phương thức `Html.TextArea()`
  - ◆ Cho phép tạo một thẻ textarea để tiếp nhận một đoạn văn bản nhiều dòng trong form
  - ◆ Có thể chỉ rõ số dòng, số cột cho phù hợp với dữ liệu nhập
- ◆ Cú pháp của `Html.TextBox()`

`@Html.TextArea("textarea_name")`
- ◆ Trong đó `<textarea_name>`: là tên của textarea sẽ tạo ra

```
<!DOCTYPE html>
<html>
 <body>
 @{Html.BeginForm("Browse","Home");}
 @Html.Label("User Name:")</br>
 @Html.TextBox("textBox1")</br></br>
 @Html.Label("Address:")</br>
 @Html.TextArea("textarea1")</br></br>
 <input type="submit" value="Submit">
 @{Html.EndForm();}
 </body>
</html>
```

# HTML Helper

- ◆ Hình sau là kết quả của form hiển thị trên trình duyệt



The image shows a screenshot of a web browser window. The address bar displays "http://localhost:2981" and the page title is "localhost". The form contains two input fields: "User Name:" with a single-line text box, and "Address:" with a multi-line text box. A "Submit" button is located at the bottom left of the form area.

User Name:

Address:

Submit