ASP.NET MVC

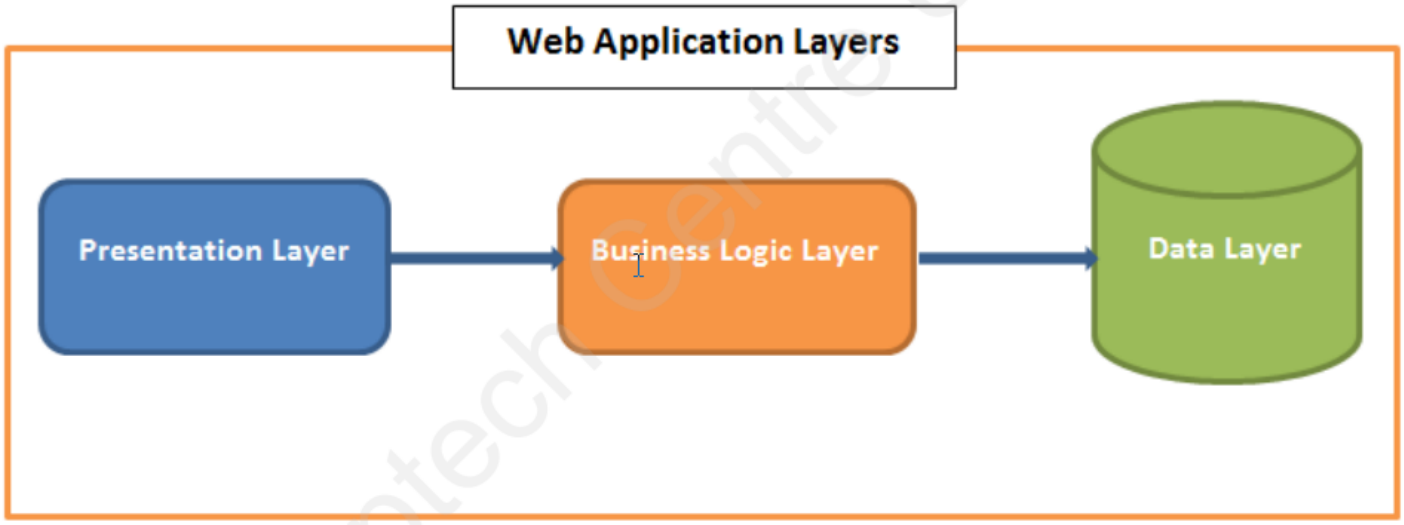
Methods and Access Specifier

- Định nghĩa và miêu tả các lớp của ứng dụng web
- Cấu trúc ứng dụng ASP.NET MVC

Ứng dụng web

- Là chương trình thực hiện trên Web server và truy cập trên Web browser
- Cho phép bạn chia sẻ các thông tin truy cập thông qua Internet trên toàn thế giới, bất cứ giờ nào
- Cho phép thực hiện thương mại hóa qua các ứng dụng thương mại

- Ứng dụng web được chia thành
  - Tầng biểu diễn: cho phép tương tác với ứng dụng
  - Tầng chức năng: điều khiển luồng thực thi và tương tác giữa hai tầng
  - Tầng dữ liệu: cung cấp dữ liệu cho tầng chức năng



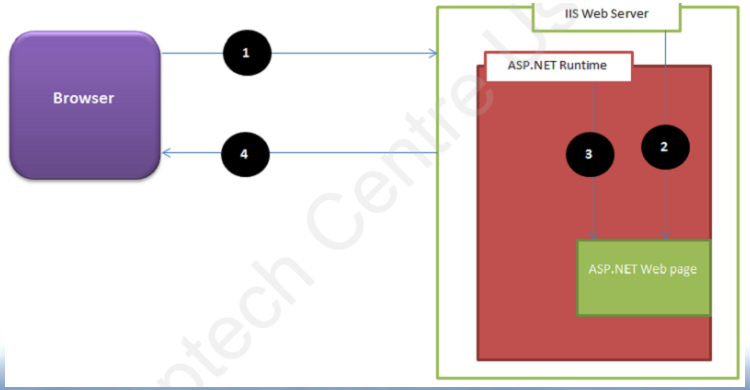
- Mỗi tầng phân biệt với các tầng khác
- Một ứng dụng dựa trên một trong các kiểu kiến trúc sau:
  - Một lớp: ba lớp tích hợp với nhau và cài đặt trên một máy tính đơn
  - Hai lớp: ba lớp được chia thành 2 lớp, một client và một server
  - Ba lớp: ứng dụng được phân bố ở ba máy tính khác nhau
  - Nhiều lớp: Các thành phần của ba lớp được phân lập nhiều nơi

- Ứng dụng web gồm nhiều trang web
- ASP.NET MVC là framework cho việc phát triển web động sử dụng framework .NET
- Trước khi có ASP.NET MVC, web động dựa trên .NET Framework được xây dựng dựa trên
  - ASP.NET Web Form
  - ASP.NET Web Page

ASP.NET

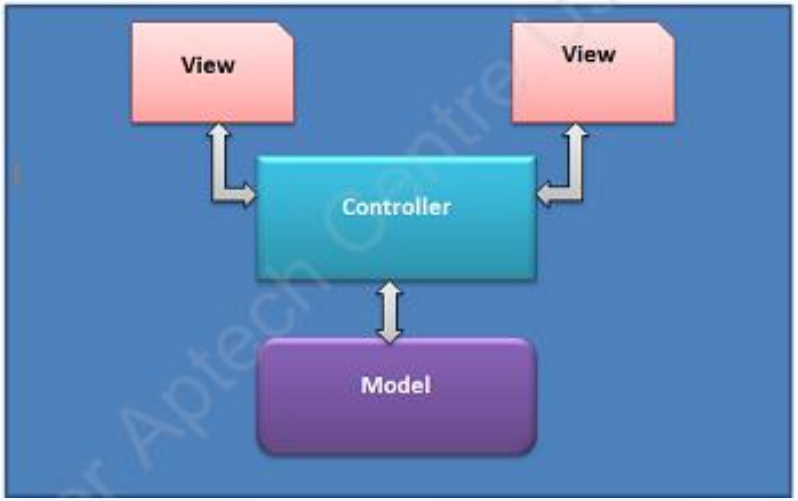
- Là công nghệ phía server
- Phối hợp các trang Web .aspx ở cả hai phía client và server
- Dựa trên Webserver như IIS

- Tiến trình yêu cầu - đáp ứng của ASP.NET:
  - Trình duyệt gửi yêu cầu cho trang ASP.NET
  - Khi nhận được yêu cầu, IIS phân loại các yêu cầu, lấy các file được yêu cầu, chuyển tiếp đến ASP.NET runtime để xử lý
  - ASP.NET runtime chứa các engine xử lý các yêu cầu và sinh các đáp ứng
  - IIS server gửi các đáp ứng tới những web server yêu cầu



- Các ứng dụng ASP.NET truyền thống tiến triển dần thành web form để dễ dàng phát triển ứng dụng web động
- Trong ASP.NET web form
  - Kéo thả các User control (UI) để thiết kế
  - Xác định cách thức form và các điều khiển sẽ đáp ứng ở thời điểm chạy
- ASP.NET Web Page
  - Sử dụng kết hợp các thẻ HTML, điều khiển server, mã lệnh server, cho phép người dùng yêu cầu thông qua trình duyệt
  - Không yêu cầu nền code quá khó
  - Cho phép sử dụng CSS, môi trường tự sinh HTML

- ASP.NET MVC dựa trên giải pháp MVC phát triển phần mềm
- Thiết kế MVC
  - Cho phép thiết kế web với các thành phần ghép cặp lỏng
  - Phân biệt truy cập dữ liệu, chức năng và trình bày
- MVC cho ứng dụng web được chia thành 3 loại:
  - Model: tượng trưng cho thông tin về dữ liệu ứng dụng
  - View: trình bày logic chuẩn bị dữ liệu cho Model
  - Controller: tượng trưng cho sự điều phối và đáp ứng giữa hai lớp view và model



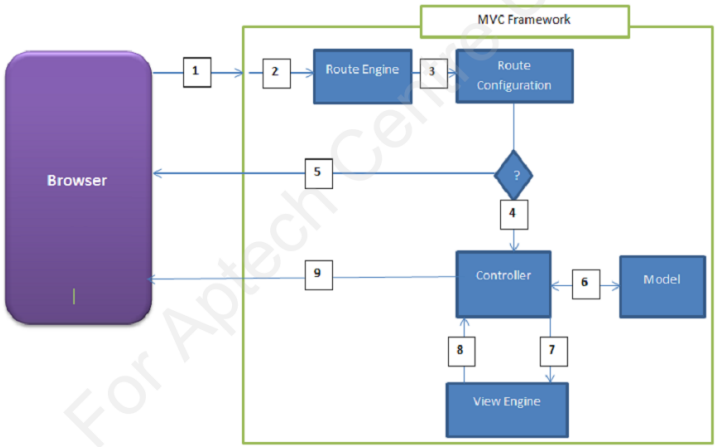
- Ưu điểm khi xây dựng ASP.NET MVC dựa trên MVC
  - Phân biệt các quan hệ: cho phép các ứng dụng quan hệ trong các thành phần phần mềm khác nhau và độc lập với nhau
  - Dễ dàng thử nghiệm và bảo trì: Cho phép thử nghiệm cũng như bảo trì từng phần tử riêng biệt
  - Có khả năng mở rộng: Cho phép các model gồm cả các thành phần độc lập có thể dễ dàng chỉnh sửa hay thay thế dựa trên yêu cầu ứng dụng

- Các phiên bản
  - MVC 1: 3/2009, .NET Framework 3.5
  - MVC 2: 3/2010, .NET Framework 3.5 và 4
  - MVC 3: 1/2011, .NET Framework 4
  - MVC 4: 8/2012, .NET Framework 4 và 4.5
  - MVC 5: 10/2013, .NET Framework 4.5 và 4.5.1

Kiến trúc ASP.NET MVC

- Kiến trúc cơ bản của ASP.NET MVC gồm các thành phần:
  - MVC Framework
  - Route engine
  - Route configuration
  - Controller
  - Model
  - View engine
  - View

- Các bước xử lý yêu cầu:
  - Trình duyệt gửi yêu cầu đến trang web
  - MVC framework chuyển yêu cầu cho routing engine
  - Routing engine kiểm tra cấu hình và cho phép controller phù hợp xử lý
  - Nếu controller không được tìm thấy, routing engine thông báo để MVC Framework trả lỗi cho trình duyệt
  - Khi controller được tìm thấy
  - Controller kết nối với model
  - Controller yêu cầu view engine đưa ra trang trình bày dựa trên dữ liệu của model
  - View engine trả kết quả cho controller
  - Cotroller gửi kết quả cho trình duyệt



- ASP.NET MVC hỗ trợ nhiều công nghệ:
  - JavaScript
  - Jquery
  - Asynchronous JavaScript and XML (AJAX)
  - IIS
  - Windows Azure

Tạo ứng dụng ASP.NET MVC

- Mở VS 2015
- File/ New/ Project
- Web/ ASP.NET Web Application
- Chọn tên và đặt thư mục chứa/ OK
- Chọn MVC/ OK
- Chạy ứng dụng: Debug/ Start

- ASP.NET MVC chứa các thư mục:
  - Controller: chứa các lớp điều khiển xử lý yêu cầu
  - Model: chứa các lớp tượng trưng cho thao tác dữ liệu và đối tượng chức năng
  - Views: chứa các file giao diện sẽ sinh mã HTML
  - Scripts: chứa các file thư viện js
  - Images: Chứa ảnh
  - Content: Chứa CSS và nội dung khác
  - Filters: Chứa mã lọc
  - App_Data: chứa file dữ liệu cần đọc, viết
  - App_Start: Chứa các file chứa mã cấu hình để sử dụng các tính năng như Routing, Bundling, Web API

Controller trong ASP.NET MVC

- Controller là gì
- Cách thức làm việc của controller

Controller

- Quản lý dòng ứng dụng
- Chịu trách nhiệm phân loại các yêu cầu và thực thi mã lệnh tương ứng
- Kết nối các model của ứng dụng và chọn view phù hợp để đáp ứng yêu cầu
- Trong lớp C# lớp controller có namespace là System.Web.Mvc
- Phân biệt chức năng và trình bày

- Controller đáp ứng:
  - Định vị method phù hợp cho yêu cầu đầu vào
  - Chuẩn hóa dữ liệu yêu cầu trước khi gọi method yêu cầu
  - Tìm dữ liệu cần thiết và chuyển cho method
  - Điều khiển mọi vật cản (exception)
  - Hỗ trợ xuất view phù hợp

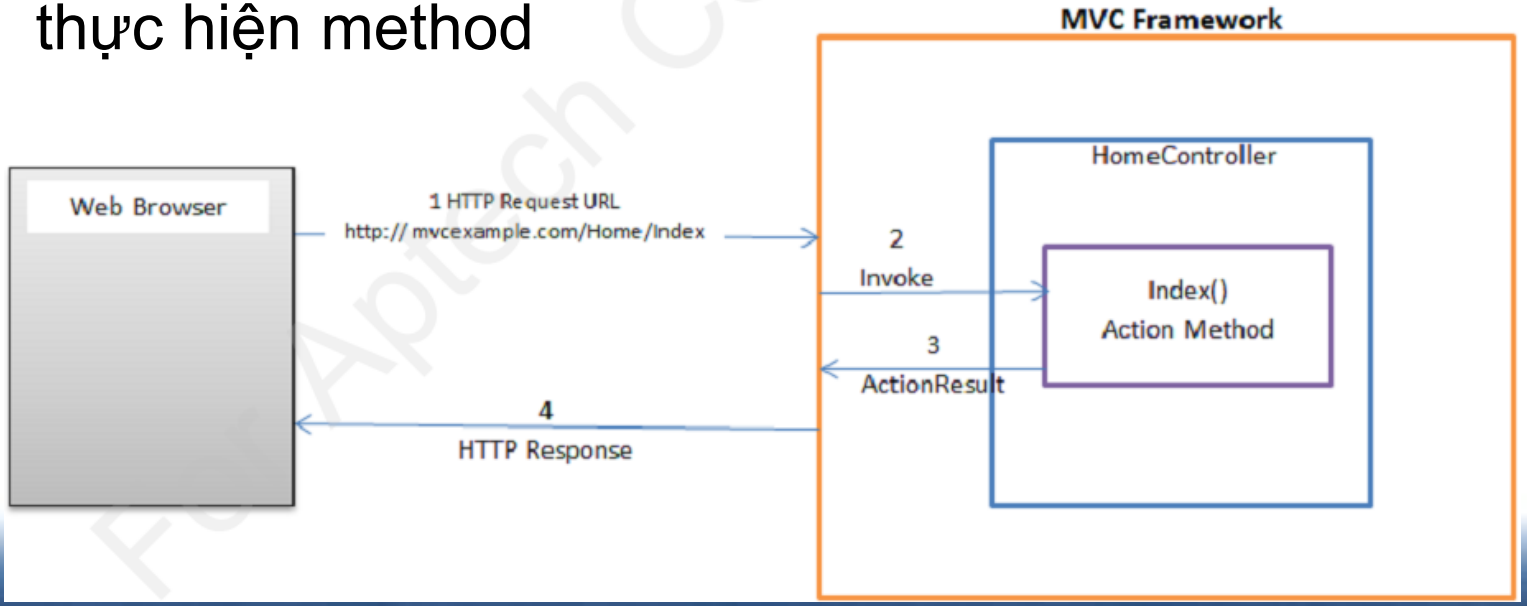
- Lớp ControllerBase của System.Web.Mvc là lớp cơ sở cho mọi controller
- Lớp Controller là mở rộng của lớp ControllerBase
- Để tạo controller sử dụng lớp Controller
- Có thể sử dụng IDE để tạo controller

- Tạo controller bằng IDE
  - Chuột phải controller
  - Add/ controller/ MVC 5 Controller-Empty/ Add
  - Đặt tên/ Add
- Tạo controller bằng câu lệnh:

```
using System.Web.Mvc;
public class <Controller_Name>Controller:Controller
{
//Some code
}
```

Làm việc với Action Method

- Một controller có thể chứa một hoặc nhiều action method, còn gọi là controller action
- Action method:
  - Xử lý một yêu cầu được gửi đến
  - Thường trả một đối tượng ActionResult là kết quả thực hiện method



Làm việc với Action Method

- Tạo action

```
public ActionResult <ActionMethod_Name>()
{
/*Code to execute logic and return the result as ActionResult*/
}
```

Làm việc với Action Method

- VD:

```
using System.Web.Mvc;
public class HomeController : Controller
{
public ActionResult Index()
{
/*Code to execute logic and return the result as ActionResult*/
}
public ActionResult About()
{
/*Code to execute logic and return the result as ActionResult*/
}
}
```

Action Result

- Action Result
  - Là một lớp nền tảng cho các lớp cung cấp các kiểu kết quả khác nhau
  - Gộp các mã HTML cùng với script hai phía server và client để đáp ứng các tương tác của người dùng

Các kiểu ActionResult

- ViewResult
- PartialViewResult
- EmptyResult
- RedirectResult
- JavascriptResult
- ContentResult
- FileContentResult
- FileStreamResult
- FilePathResult

Lời gọi Action Method

http:// <domain_name> /<controller_name>/<actionmethod_name>

- <domain_name>
- <controller_name>
- <actionmethod_name>

- VD:

  http:// mvcexample.com/Home/Registration

Truyền tham số

- VD:
  http://www.mvcexample.com/student/details?Id=006
- Details:

```
public ActionResult Details(string Id)
{
        /*Return student records based on the Id
        parameter as an ActionResultobject*/
}
```

Routing Requests

- Định nghĩa cách ứng dụng xử lý và đáp ứng yêu cầu từ HTTP
- Miêu tả controller action cần cho yêu cầu gửi tới
- Routing là tiến trình ánh xạ yêu cầu đến controller action
  - Xây dựng URL đầu ra tương ứng với controller action
- Cấu hình mẫu route:
  - Tạo mẫu route
  - Ghi mẫu vào bảng route của MVC Framework

Route mặc định

- Một ứng dụng MVC yêu cầu 1 route để điều khiển yêu cầu người dùng
- Khi tạo một ứng dụng MVC trong VS2015, 1 route tự động được cấu hình trong RouteConfig.cs
- MapRoute() method:

```
routes.MapRoute(
name: "Default",
url: "{controller}/{action}/{id}",
defaults: new { controller = "Home", action = "Index", id = UrlParameter.Optional }
);
```

Ghi thông tin route mặc định

- Trong ứng dụng ASP.NET MVC , tệp Global.asax:
  - Khởi tạo ứng dụng với các tính năng của MVC framework khi bắt đầu ứng dụng
  - Chứa lớp MVCApplication với method: Application_Start() khi đăng ký route mặc định

- Global.asax file:

```
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Http;
using System.Web.Mvc;
using System.Web.Optimization;
using System.Web.Routing;

namespaceUrlsAndRoutes {
public class MvcApplication : System.Web.HttpApplication {
protected void Application_Start() {
RouteConfig.RegisterRoutes(RouteTable.Routes);
/*Code for registering other MVC components*/
}
}
}
```

URL

- Được xác định khi tạo một route
- Được đối chiếu với URL của yêu cầu bởi route engine của MVC
- Chứa các giá trị ngữ nghĩa và placeholder phân cách bởi dấu “/”

  "{controller}/{action}/{id}“
- Tuân theo mẫu trước: http://www.mvcexample.com/student/records/36
- Khi routing engine kiểm tra sự phù hợp giữa URL trước và phần URL, sẽ thực hiện :
  - Ấn định student cho {controller}
  - Ấn định records cho {action}
  - Ấn định 36 cho {id}
- Một URL có thể kết hợp giá trị và placeholder: "student/{action}/{id}"

Sắp xếp routes

- Đôi khi cần đăng ký nhiều routes cho một ứng dụng ASP.NET MVC
- Route engine ánh xạ URL với mẫu URL

```
routes.MapRoute(
name: "general",
url: "{controller}/{action}",
defaults: new { controller = "Home", action =
"Index"});
   routes.MapRoute(
   name: "manager",
   url: "Manager/{action}",
   defaults: new { controller = "Manager",
action = "Browse"}
);
```

Ràng buộc route

- Routing engine cho phép áp dụng các ràng buộc giá trị placeholder

```
routes.MapRoute(
"Product",
"{controller}/{action}/{id}",
new { controller = "Product", action =
"Browse", id =
UrlParameter.Optional },
new { id = "(|Jewellery|Jeans|Mobile)" }
);
```

Ràng buộc route

- VD: id có giá trị integer

```
routes.MapRoute(
name: “Product”,
url: “{controller}/{action}/{id}”,
defaults: new { controller =
“Product”, action = “Browse”,
id=UrlParameter.
Optional},
constraints: new { id = @”\d*” }
);
```

Bỏ qua route

- Phương thức IgnoreRoute() của lớp RouteTable xác định route được bỏ qua

```
routes.IgnoreRoute("{resource}.axd/
{*pathInfo}");
```

Views trong ASP.NET MVC

- Views là gì?
- Razor engine

Làm việc với Views

- Để hiển thị nội dung HTML, ta có thể chỉ dẫn hành xử controller để đưa ra view
- View
  - Cung cấp giao diện người dùng
  - Hiển thị nội dung của ứng dụng và nhận đầu vào của người dùng
  - Sử dụng mô hình dữ liệu để tạo UI
  - Chứa cả HTML và code khi chạy trên Web server

View Engines

- Chuyển code trong view sang HTML để các trình duyệt có thể đọc được
- Có hai loại
  - Web Form view engine:
  - Zaror view engine

Xác định view cho action

- Mọi view được lưu mặc định trong thư mục view
- Nếu một action controller trả về một view, khi đó phần view có:
  - Một thư mục cho controller cùng tên với controller nhưng không có hậu tố Controller
  - Một file trong thư mục Home có cùng tên với action

Xác định view cho action

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

- Action Index trả về một view
- Tên controller: HomeController
- Demo cách tạo một view

Xác định view cho action

- Có thể trả về một view khác trong phương thức action bằng cách đặt tên của view là một thông số

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

- Đoạn code trên sẽ tìm view trong thư mục Views/Home và sinh ra Testdext thay vì Index

Xác định view cho action

- Render một view trong thư mục khác: cần xác định đường dẫn cho view

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

Chuyển dữ liệu từ controller sang view

- Controller thực hiện chức năng, trả kết quả người dùng thông qua view
- Chuyển dữ liệu giữa controller và view qua:
  - ViewData
  - ViewBag
  - TempData

Chuyển dữ liệu từ controller sang view

- ViewData
  - Chuyển dữ liệu từ controller đến view
  - Là thư viện gồm các đối tượng của lớp ViewDataDictionary
  - Các tính chất của ViewData:
    - Đối tượng ViewData chỉ tồn tại khi được gọi và là null khi chuyển hướng hay thay đổi URL (redirected)
    - ViewData yêu cầu mẫu khi sử dụng dữ liệu phức tạp để tránh lỗi
  - Cú pháp: `ViewData[<key>] = <Value>;`

Chuyển dữ liệu từ controller sang view

- VD: ViewData kết nối HomeController và Index

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

- ViewData được tạo với 2 cặp giá trị
  - Message là chuỗi
  - CurrentTime chứa dữ liệu DateTime.Now

Chuyển dữ liệu từ controller sang view

- Đoạn code sau lấy các giá trị của ViewData

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

- ViewData trong đoạn code trên: Hiển thị giá trị của Message và CurrentTime

Chuyển dữ liệu từ controller sang view

- ViewBag
  - Bao phủ ViewData
  - Chỉ tồn tại cho yêu cầu hiện tại và có giá trị null khi yêu cầu được chuyển hướng
  - Không yêu cầu mẫu khi sử dụng dữ liệu phức tạp
  - Cú pháp: `ViewBag.<Property> = <Value>;`
    - Property: chuỗi giá trị thể hiện thuộc tính của ViewBag
    - Value: giá trị của thuộc tính ViewBag

Chuyển dữ liệu từ controller sang view

- VD: Đoạn code sau cho thấy 2 thuộc tính trong Index của lớp HomeController

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

Chuyển dữ liệu từ controller sang view

- Đoạn code hiển thị giá trị biểu diễn trong ViewBag

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

- Khi sử dụng ViewBag để lưu thuộc tính và giá trị trong action thì thuộc tính đó có thể được truy cập từ cả hai: ViewBag và ViewData

Chuyển dữ liệu từ controller sang view

- Đoạn code sau cho thấy một controller action lưu thuộc tính ViewBag

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

Chuyển dữ liệu từ controller sang view

- Đoạn code sau cho thấy cả ViewData và ViewBag đều truy cập đến thuộc tính CommonMessage trong ViewBag

```
<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>
```

Chuyển dữ liệu từ controller sang view

- TempData
  - Là đối tượng Dictionary của lớp TempDataDictionary
  - Lưu trữ dữ liệu dạng key-value
  - Cho phép chuyển dữ liệu từ yêu cầu hiện tại đến yêu cầu sau khi chuyển hướng
  - Cú pháp: `TempData[<Key>] = <Value>;`
    - Key; chuỗi nhận dạng đối tượng hiện tại trong TempData
    - Value: là đối tượng biểu diễn trong TempData

Xác định view cho action

- Đoạn code sau cho thấy sử dụng TempData để chuyển giá trị từ view này đến view khác khi chuyển hướng

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

View bộ phận (Partial Views)

- Partial View:
  - View con của view chính
  - Cho phép dùng lại các thẻ của view khác
- Tạo Partical View
  - Kích chuột phải thư mục Views/Shared, Add/View
  - Đánh dấu hộp kiểm: Create as a partial view

Partial Views

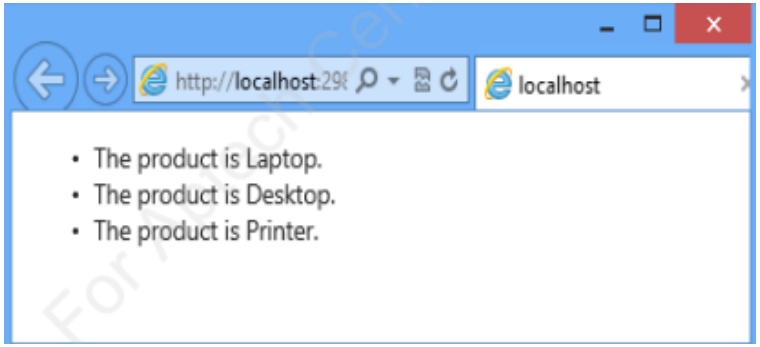
- Cú pháp: @Html.Partial(<partial_view_name>
  - partial_view_name: tên view cục bộ không kèm phần mở rộng .cshtml

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

Razor

- Dựa trên ASP.NET tạo ra view
- Dễ hiểu cho người lập trình: ngôn ngữ lập trình tương tự C#.NET và VB.NET

```
@{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>
```



Razor Engine

- Là engine sinh view mặc định của MVC
- Biên dịch view cho lần gọi đầu
- Cung cấp view được biên dịch cho lần gọi sau
- Không cung cấp ngôn ngữ lập trình mới mà đưa ra cú pháp tích hợp HTML
- Hỗ trợ Test Driven Development (TDD) cho phép kiểm tra view độc lập với ứng dụng

Cú pháp Razor

- @{ <code> }
  - Code: cú pháp c# hoặc vb# được thực hiện trên server
- Biến được khai báo với từ khóa var
  - `@{ var myMessage = "Hello World"; }`
  -
```
@{
  var myMessage = "Hello World";
  var num = 10;
  }
```
- Chuỗi để trong ngoặc kép
- Dòng lệnh kết thúc bởi (;)
- Dùng .cshtml để lưu kiểu file c#

Biến - Cấu trúc lệnh

- Khai báo biến, sử dụng cấu trúc lệnh tương tự như trong C#

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

```
<!DOCTYPE html>
<html><body>
    @{ var b = 0;
       while (b < 7) {
          b += 1;
          <p>Text @b</p>}
    }
</body></html>
```

```
<!DOCTYPE html>
@{var mark=60;}
<html><body>
    @if (mark<80)  {
       <p>You have failed in
              the exam.</p>}
    else  {
       <p>You have passed
              the exam.</p>  }
</body></html>
```

```
<!DOCTYPE html>
<html><body>
    <h1>Even Numbers</h1>
    @{ var num=1;
       for (num = 1; num <= 11; num++){
          if ((num % 2) == 0)
          { <p> @num</p> }
       } }
</body></html>
```

```
<!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;
                break; }
        <p>@msg</p>
    </body> </html>
```

```
<!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 Methods

- Là phương thức của lớp HtmlHelper, chỉ được gọi từ view
- Đơn giản hóa thi hành view
- Cho phép sinh HTML và tái sử dụng
- Một số phương thức thường dùng:
  - Html.ActionLink()
  - Html.BeginForm() và Html.EndForm()
  - Html.Label()
  - Html.TextBox()
  - Html.TextArea()
  - Html.Password()
  - Html.CheckBox()

HTML Helper Methods

- Html.ActionLink(): cho phép sinh ra một hyperlink dựa trên action method của controller class

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

- VD:

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

HTML Helper Methods

- Html.BeginForm()
  - Cho phép đánh dấu điểm bắt đầu của form
  - Phối hợp với routing engine để sinh URL
  - Quản lý thẻ <form>
  - Cú pháp:

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

  - Khi sử dụng HTML.BeginForm, cần sử dụng HTML.EndForm

HTML Helper Methods

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

HTML Helper Methods

- Html.Label()
  - Cho phép hiển thị nhãn trong form
  - Cho phép kết nối đến các thẻ khác
  - Cú pháp: `@Html.Label(<label_text_name>)`
- VD:

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

HTML Helper Methods

- Html.TextBox():
  - Cho phép hiển thị thẻ đầu vào
  - Sử dụng chấp nhận đầu vào của người dùng
  - Cú pháp:

```
@Html.TextBox("textbox_text_name”)
```

- VD:

```
<!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 Methods

- Html.TextArea()
  - Cho phép hiển thị thẻ <textarea>
  - Cho phép xác định số cột, hàng hiển thị
  - VD:

```
<!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 Methods

- Html.Password(): hiển thị trường mật khẩu

```
<!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>
   @Html.Label("Password:")</br>@Html.Passwo
   rd("password")</br></br>
   <input type="submit" value="Submit">
   @{Html.EndForm();}
</body></html>
```

HTML Helper Methods

- Html.CheckBox(): hiển thị checkbox

```
<!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>
      @Html.Label("Password:")</br>
      @Html.Password("password")</br></br>
      @Html.Label("I need updates on my mail:")
      @Html.CheckBox ("checkbox1")</br> </br>
      <input type="submit" value="Submit">
      @{Html.EndForm();}
</body> </html>
```

HTML Helper Methods

- Html.DropDownList(): sinh thẻ <select>
- Cú pháp:

```
@Html.DropDownList("myList",
new SelectList(new []
{<value1>,<value2>, <
value3>}), "Choose")
```

  - Value1, value2, value3: các giá trị danh sách
  - Choose: giá trị đầu danh sách

HTML Helper Methods

- VD:

```
<!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>
      @Html.Label("Password:")</br>@Html.Password
                        ("password")</br></br>
      @Html.Label("I need updates on my mail:")
      @Html.CheckBox("checkbox1")</br> </br>
      @Html.Label("Select your city:")
      @Html.DropDownList("myList", new SelectList(new []
            {"New York","Philadelphia", "California"}),
                  "Choose")</> </br></br>
      <input type="submit" value="Submit">
      @{Html.EndForm();}
</body></html>
```

HTML Helper Methods

- Html.RadioButton(): tạo nút lựa chọn
  `@Html.RadioButton("name", "value", isChecked)`
- Url.Action(): sinh url
  `@Url.Action(<action_name>, <controller_name>)`

Models trong ASP.NET MVC

- Models là gì?
- Tạo model
- Chuyển model dữ liệu từ controller đến view

Models là gì?

- Là lớp chứa các thuộc tính biểu diễn dữ liệu
- Tượng trưng cho dữ liệu kết nối với ứng dụng
- MVC định nghĩa 3 kiểu models:
  - Data model: lớp tương tác với dữ liệu. Là tập các lớp hoặc theo phương pháp database-first hoặc theo phương pháp code-first
  - Business model: lớp tượng trưng thực hiện các chức năng
  - View model: lớp tương tác giữa controller và view

Tạo model

- Để tạo model MVC
  - Tạo lớp public
  - Khai báo thuộc tính public cho mỗi thông tin của model
- VD: Khai báo một lớp model có tên User

```
public class User
{
    public long Id { get; set; }
    public string name { get; set; }
    public string address { get; set; }
    public string email { get; set; }
}
```

Truy cập model từ controller

- Mọi yêu cầu được trả về bởi action
- Action được sử dụng để truy cập các model chứa dữ liệu
- Để truy cập dữ liệu, cần tạo một đối tượng của lớp model nhằm lấy hoặc gán giá trị
- VD:

```
public ActionResult Index()
{
    var user = new MVCModelDemo.Models.User();
    user.name = "John Smith";
    user.address = "Park Street";
    user.email = "john@mvcexample.com";
    return View();
}
```

Chuyển dữ liệu model từ controller đến view

- Khi truy cập dữ liệu trong controller, cần chuyển model cho view khi gọi view, như vậy view mới có thể hiển thị dữ liệu cho người dùng
- Có hai các mô hình
  - Đối tượng đơn lẻ
  - Tập đối tượng
- Trong action method, tạo một đối tượng model sau đó chuyển qua view bằng cách sử dụng `ViewBag`

Chuyển dữ liệu model từ controller đến view

- VD

```
public ActionResult Index()
{
    var user = new MVCModelDemo.Models.User();
    user.name = "John Smith";
    user.address = "Park Street";
    user.email = "john@mvcexample.com";
    ViewBag.user = user;
    return View();
}
```

Chuyển dữ liệu model từ controller đến view

- Truy cập dữ liệu lưu trong ViewBag

```
<!DOCTYPE html>
<html> <body>
    <p>User Name: @ViewBag.user.name</p>
    <p>Address: @ViewBag.user.address</p>
    <p>Email: @ViewBag.user.email</p>
</body> </html
```

Chuyển dữ liệu model từ controller đến view

```
public ActionResult Index(){
    var user = new List<User>();
    var user1 = new User();
       user1.name = "Mark Smith";
       user1.address = "Park Street";
       user1.email = "Mark@mvcexample.com";
    var user2 = new User();
       user2.name = "John Parker";
       user2.address = "New Park";
       user2.email = "John@mvcexample.com";
    var user3 = new User();
       user3.name = "Steave Edward ";
       user3.address = "Melbourne Street";
       user3.email = "steave@mvcexample.com";
    user.Add(user1);
    user.Add(user2);
    user.Add(user3);
    ViewBag.user = user; return View();
}
```

Chuyển dữ liệu model từ controller đến view

```
<!DOCTYPE html>
<html> <body>
    <h3>User Details</h3>
    @{ var user = ViewBag.user; }
    @foreach (var p in user){
      @p.name<br />
      @p.address<br />
      @p.email<br />
      <br /> }
</body> </html>
```

Chuyển dữ liệu model từ controller đến view

```
<!DOCTYPE html>
<html> <body>
   <h3>User Details</h3>
   @{ var user = ViewBag.user; }
   @foreach (var p in user) {
      @p.name<br />
      @p.address<br />
      @p.email<br /><br /> }
</body> </html>
public ActionResult Index() {
  var user = new List<User>(); var user1 = new User();
   user1.name="Mark Smith"; user1.address="Park Street";
   user1.email = "Mark@mvcexample.com";
  var user2 = new User(); user2.name = "John Parker";
   user2.address = "New Park";
   user2.email = "John@mvcexample.com";
  var user3 = new User(); user3.name = "Steave Edward ";
   user3.address = "Melbourn Street";
   user3.email = "steave@mvcexample.com";
  user.Add(user1); user.Add(user2); user.Add(user3);
return View(user); }
```

Sử dụng strong typing

- Trong trường hợp view không nhận chính xác kiểu dữ liệu, gõ chính xác

```
<html> <body>
 <h3>User Details</h3>
 @{var user=Model as MVCModelDemo.Models.User;}
 @user.name <br/>
 @user.address<br/>
 @user.email<br/>
</body> </html
```

- Sử dụng strong typing
- Cú pháp: `@model <model_name>`
- Khai báo @model có thể truy cập các thuộc tính của model trong view

Sử dụng strong typing

```
@model MVCModelDemo.Models.User
<html><body>
    <h3>User Details</h3>
    @Model.name <br/>
    @Model.address<br/>
    @Model.email<br/>
</body> </html>
```

- Chuyển một tập các đối tượng cho view
  - VD: `@model IEnumerable<MVCModelDemo.Models.User`

Sử dụng strong typing

```
@model IEnumerable<MVCModelDemo.Models.User>
<html><body>
    <h3>User Details</h3>
    @{var user = Model;}
    @foreach(var u in user){
      @u.name <br/>
      @u.address<br/>
      @u.email<br/><br/>}
</body></html>
```

Các phương thức HTML Helper trong Strongly Types

- MVC cho phép
  - Kết nối trực tiếp với các thuộc tính của model chỉ ở dạng strongly types
    - Html.LabelFor()
    - Html.DisplayNameFor()
    - Html.DisplayFor()
    - Html.TextBoxFor()
    - Html.TextAreaFor()
    - Html.EditorFor()
    - Html.PasswordFor()
    - Html.DropDownListFor()

```
@model MVCModelDemo.Models.User
@{ViewBag.Title = "User Form";}
<h2>User Form</h2>
@using (Html.BeginForm()) {
    @Html.ValidationSummary(true)
    <div>@Html.LabelFor(model => model.name)</div>
    <div>@Html.EditorFor(model => model.name)</div>
    <div>@Html.LabelFor(model =>model.address)</div>
    @Html.EditorFor(model =>model.address)
    <div>@Html.LabelFor(model =>model.email)</div>
    <div>@Html.EditorFor(model =>model.email) </div>
    <p><input type="submit" value="Create" /></p>
}
```

Model Binder

- Khi người dùng submit thông tin trên form trong strongly typed view, MVC tự động kiểm tra HttpRequest và ánh xạ thông tin gửi đến trường trong model
- Tiến trình ánh xạ này gọi là model binding
- Một số lợi ích của model binding:
  - Tự động trích dữ liệu từ HttpRequest
  - Tự động chuyển kiểu dữ liệu
  - Tạo dữ liệu hợp lệ dễ dàng

Model Binder

- DefaultModelBinder là lớp model binder của MVC
- Model binder:
  - Yêu cầu giá trị nguyên thủy
  - Yêu cầu đối tượng

Yêu cầu giá trị nguyên thủy

- Tạo lớp
- Tạo view kết nối đến lớp
- VD: tạo lớp login

```
public class Login{
    public string userName { get; set; }
    [DataType(DataType.Password)]
    public string password { get; set; }
}
```

Yêu cầu giá trị nguyên thủy

- Tạo view index.cshtml

```
@model ModelDemo.Models.Login
@{ ViewBag.Title = "Index";}
<h2>User Details</h2>
@using (Html.BeginForm()) {
@Html.ValidationSummary(true)
<div>@Html.LabelFor(model =>model.userName)</div>
<div>@Html.EditorFor(model=>model.userName)</div>
<div>@Html.LabelFor(model =>model.password)</div>
<div>@Html.EditorFor(model=>model.password)</div>
<div><input type="submit" value="Submit" /></div>
}
```

- Sau đó tạo controller chứa action method: index() để hiển thị view

```
public class HomeController : Controller  {
public ActionResult Index(){return View();}
[HttpPost]
public ActionResult Index(string userName,
string password) {
if (userName == "Peter" && password ==
"pass@123")  {
     string msg = "Welcome " + userName;
     return Content(msg);}
     else  {return View();}
     }
}
```

Yêu cầu đối tượng

```
public class HomeController : Controller {
public ActionResult Index() {return View();}
[HttpPost]
   public ActionResult Index(Login login) {
   if (login.userName == "Peter" &&login.password ==
   "pass@123") {
   String msg = "Welcome " + login.userName;
   return Content(msg);  }
   else  {
   return View(); }  }
   }
```

- Index() đầu tiên sinh view để hiển thị login form
- Index() thứ 2 chuyển dữ liệu tự HttpRequest và đẩy vào Login object

Yêu cầu đối tượng

- Khi người dùng submit dữ liệu login, phương thức Index() kiểm tra username và password được chuyển trong đối tượng login
- Nếu thành công, view hiển thị thông điệp chào
- Khi truy cập ứng dụng từ trình duyệt, Index.cshtml hiển thị login form
- Gõ Peter và pass@123 trong login form
- Nhấn Submit hiển thị thông điệp chào “Welcome Peter”