

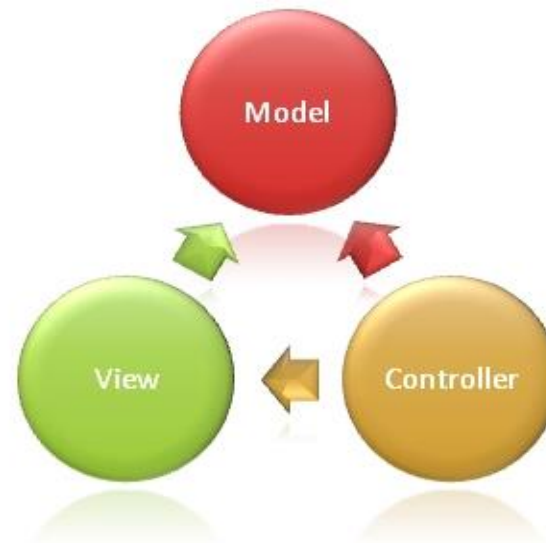
Lập trình WEB

Mô hình MVC

GIỚI THIỆU

✓ Khái niệm MVC

- Là một mô hình phát triển phần mềm mới bằng ASP.NET
- Là framework được xây dựng dựa trên mô hình MVC (Model View Controller)
- Mẫu kiến trúc Model – View – Controller được sử dụng nhằm chi ứng dụng thành ba thành phần chính: model, view và controller



GIỚI THIỆU



Model – View – Controller

- **Model:** Các đối tượng Model là một phần của ứng dụng, các đối tượng này thiết lập logic của phần dữ liệu của ứng dụng.
- **Views:** Views là các thành phần dùng để hiển thị giao diện người dùng (UI). Thông thường, view được tạo dựa vào thông tin dữ liệu model.
- **Controllers:** Controller là các thành phần dùng để quản lý tương tác người dùng, làm việc với model và chọn view để hiển thị giao diện người dùng.



GIỚI THIỆU



Lợi ích của việc sử dụng MVC

- **Mẫu MVC** cho phép tạo các ứng dụng mà chúng phân tách rạch ròi các khía cạnh của ứng dụng (logic về nhập liệu, logic xử lý tác vụ và logic về giao diện). Mẫu MVC chỉ ra mỗi loại logic kể trên nên được thiết lập ở đâu trên ứng dụng.
 - Logic giao diện (UI logic) thuộc về views.
 - Logic nhập liệu (input logic) thuộc về controller.
 - Logic tác vụ (Business logic – là logic xử lý thông tin, mục đích chính của ứng dụng) thuộc về model.
- **Mẫu MVC** giúp cho chúng ta có thể kiểm thử ứng dụng dễ dàng hơn hẳn so với khi áp dụng mẫu **Web Forms**.



GIỚI THIỆU



Lợi ích của việc sử dụng MVC

- **MVC** dễ dàng quản lý sự phức tạp của ứng dụng bằng cách chia ứng dụng thành ba thành phần model, view, controller.
- **MVC** không sử dụng view state hoặc server-based form. Điều này tốt cho những lập trình viên muốn quản lý hết các khía cạnh của một ứng dụng.
- **MVC** sử dụng mẫu Front Controller, mẫu này giúp quản lý các requests (yêu cầu) chỉ thông qua một Controller. Nhờ đó bạn có thể thiết kế một hạ tầng quản lý định tuyến.
- Hỗ trợ tốt hơn cho mô hình phát triển ứng dụng hướng kiểm thử
- Hỗ trợ tốt cho các ứng dụng được xây dựng bởi những đội có nhiều lập trình viên và thiết kế mà vẫn quản lý được tính năng của ứng dụng



GIỚI THIỆU



Asp.net MVC5

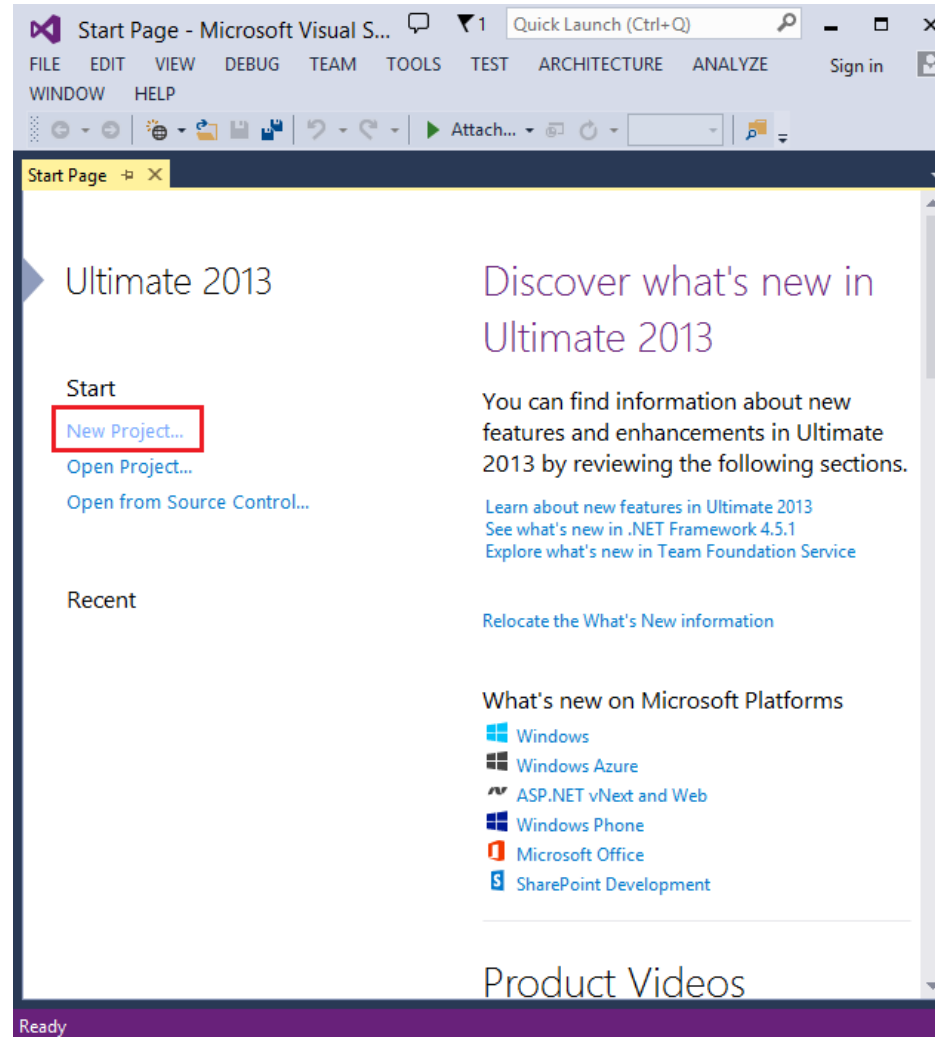
- **Asp.net MVC5** là phiên bản kế tiếp của MVC4 được xây dựng trên nền .net 4.5
- **Asp.net MVC5** kế thừa các tính năng mạnh từ các phiên bản trước là MVC3, MVC4 và tích hợp thêm 1 số tính năng về xử lý đa tiến trình, tương tác với Web API 2.0 tốt hơn MVC4.



TẠO ỨNG DỤNG ASP.NET MVC

✓ Khởi động Visual Studio 2013 (MVC_Main_vd1)

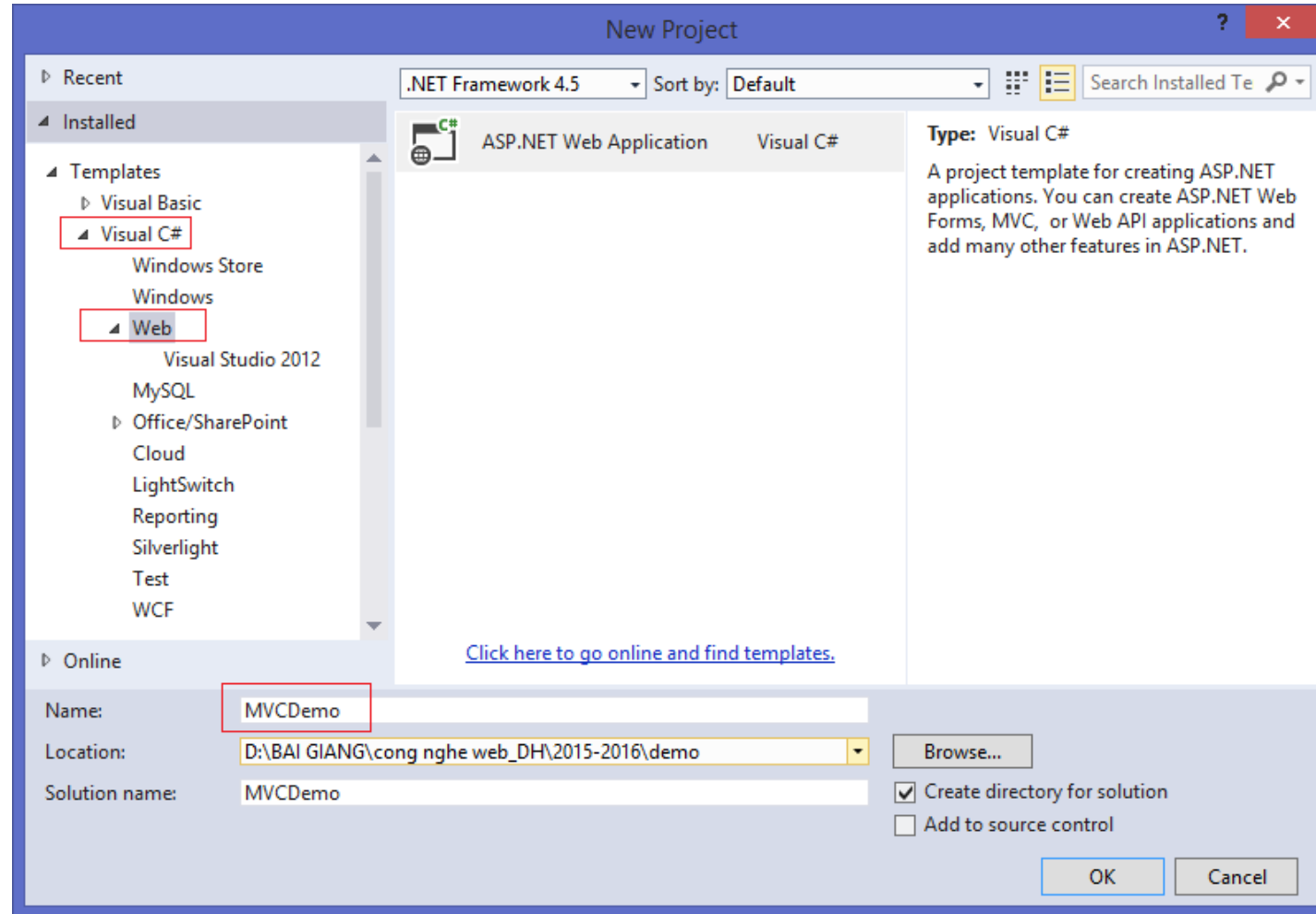
- Khởi động Visual Studio và tạo mới 1 project như sau:
File -> New Project



TẠO ỨNG DỤNG ASP.NET MVC



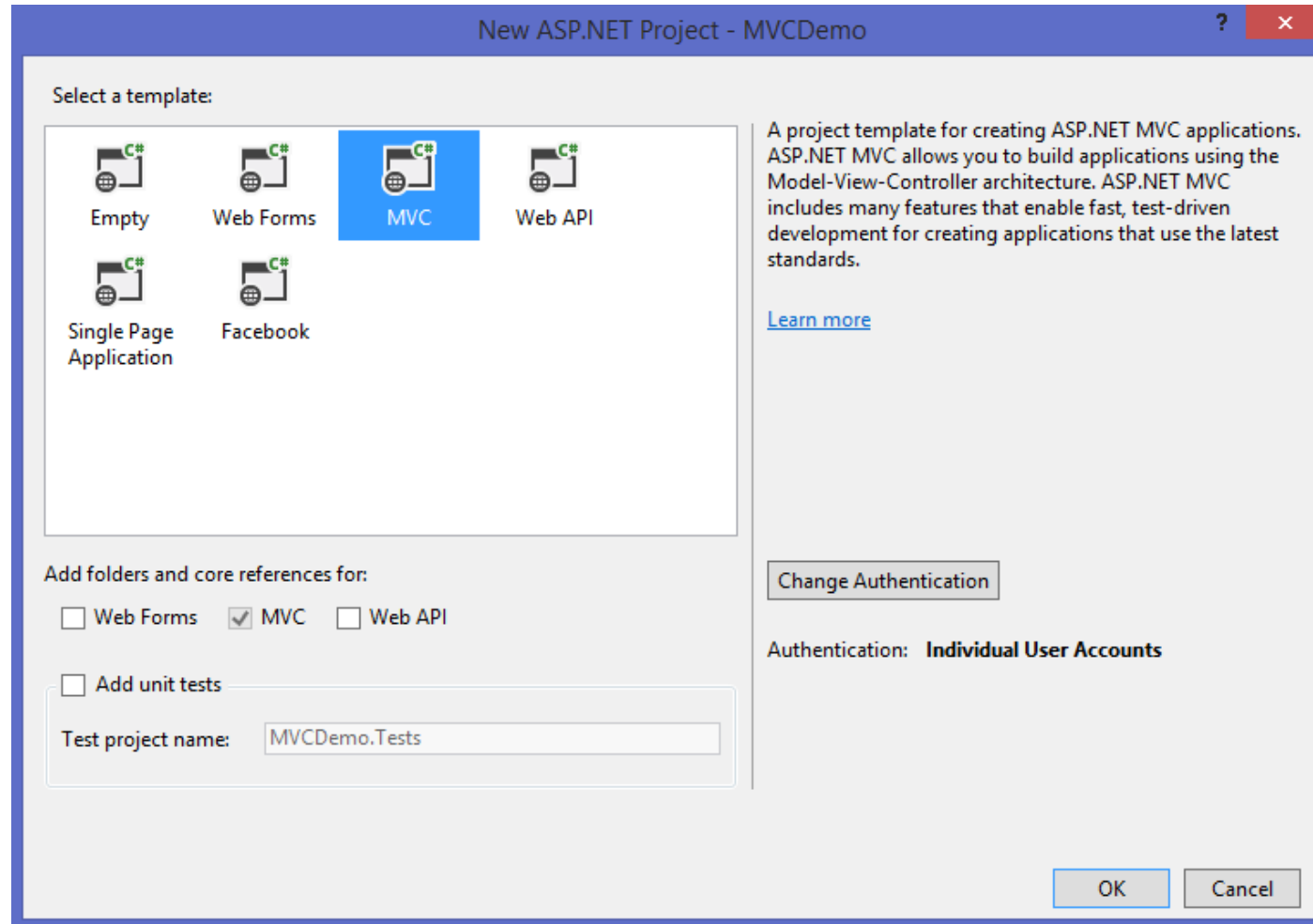
Khởi động Visual Studio 2013



TẠO ỨNG DỤNG ASP.NET MVC



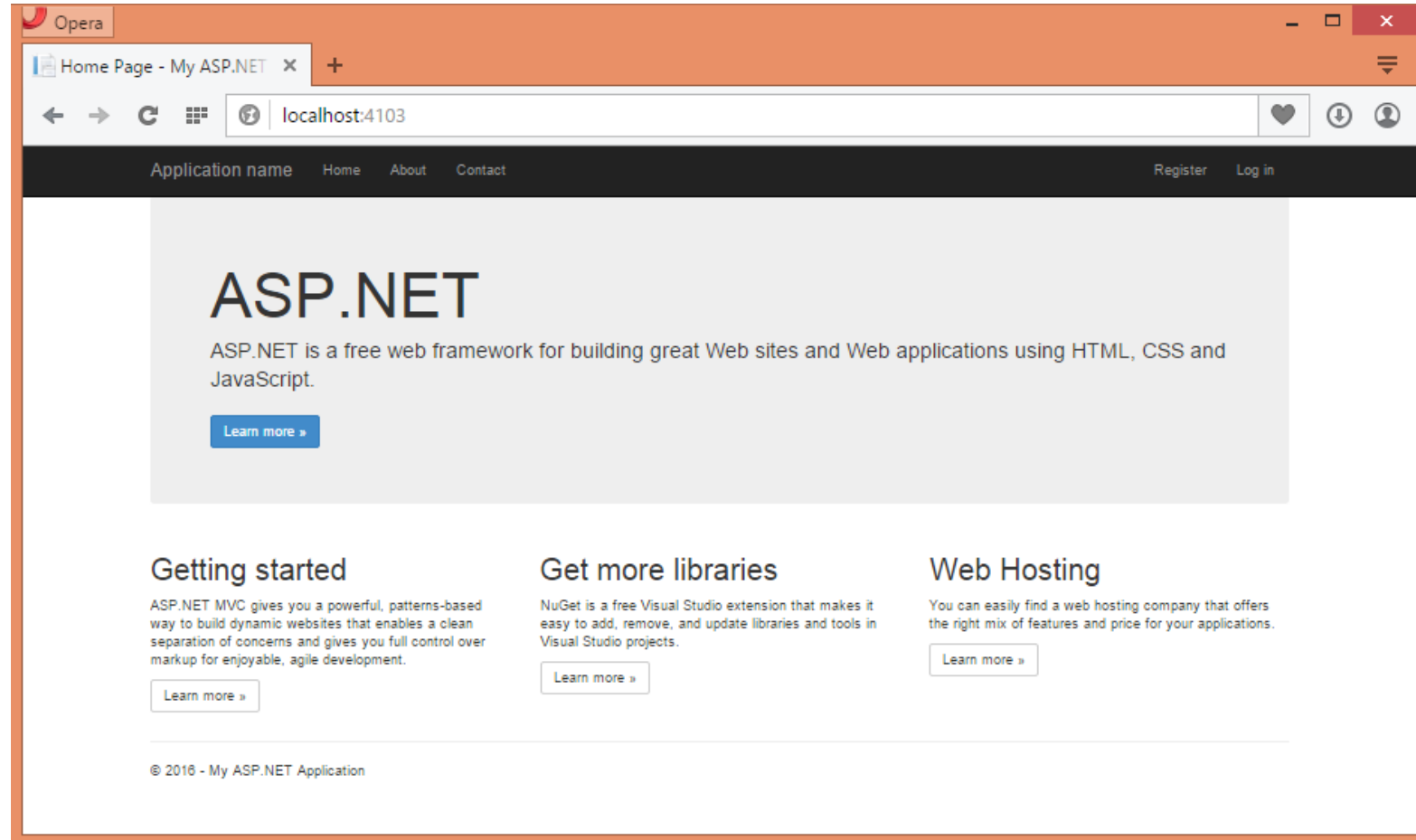
Khởi động Visual Studio 2013



TẠO ỨNG DỤNG ASP.NET MVC

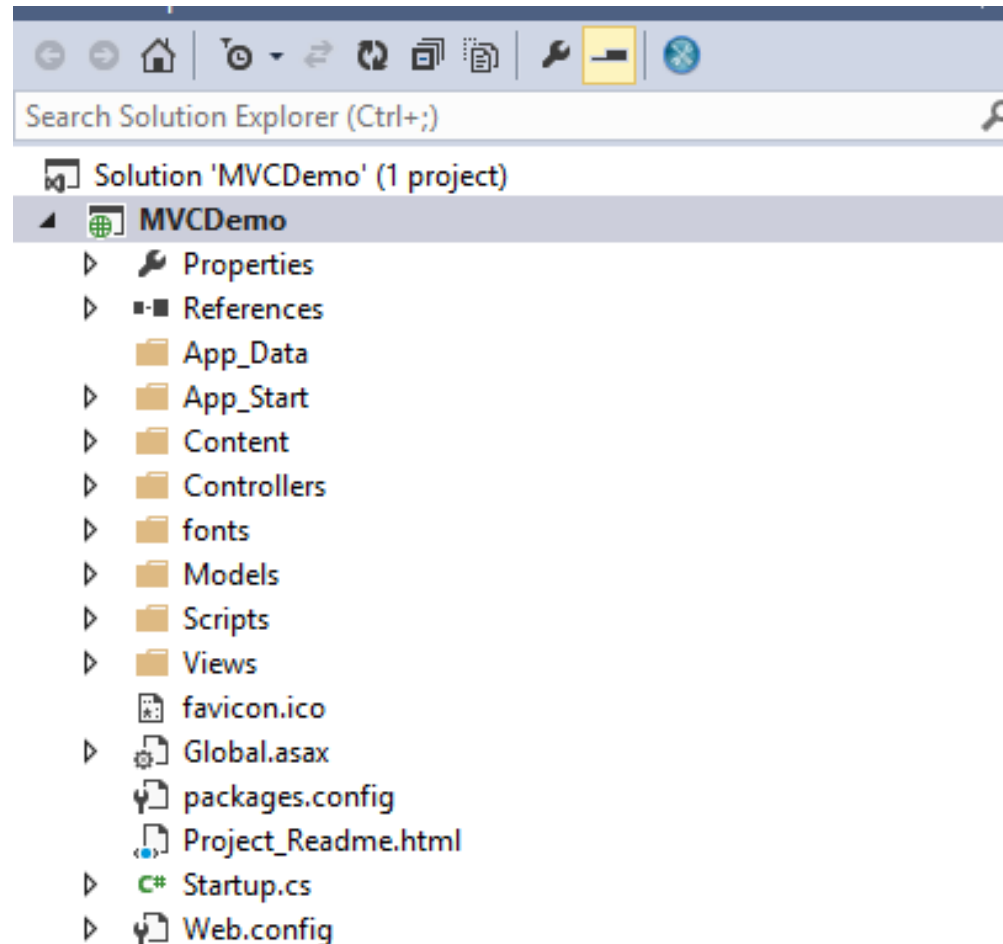


Chạy ứng dụng



TẠO ỨNG DỤNG ASP.NET MVC

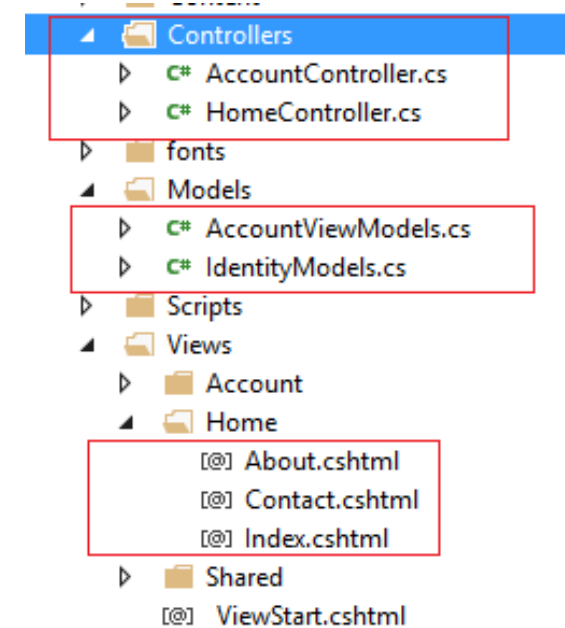
✓ Cấu trúc một project



TẠO ỨNG DỤNG ASP.NET MVC

✓ Cấu trúc một project

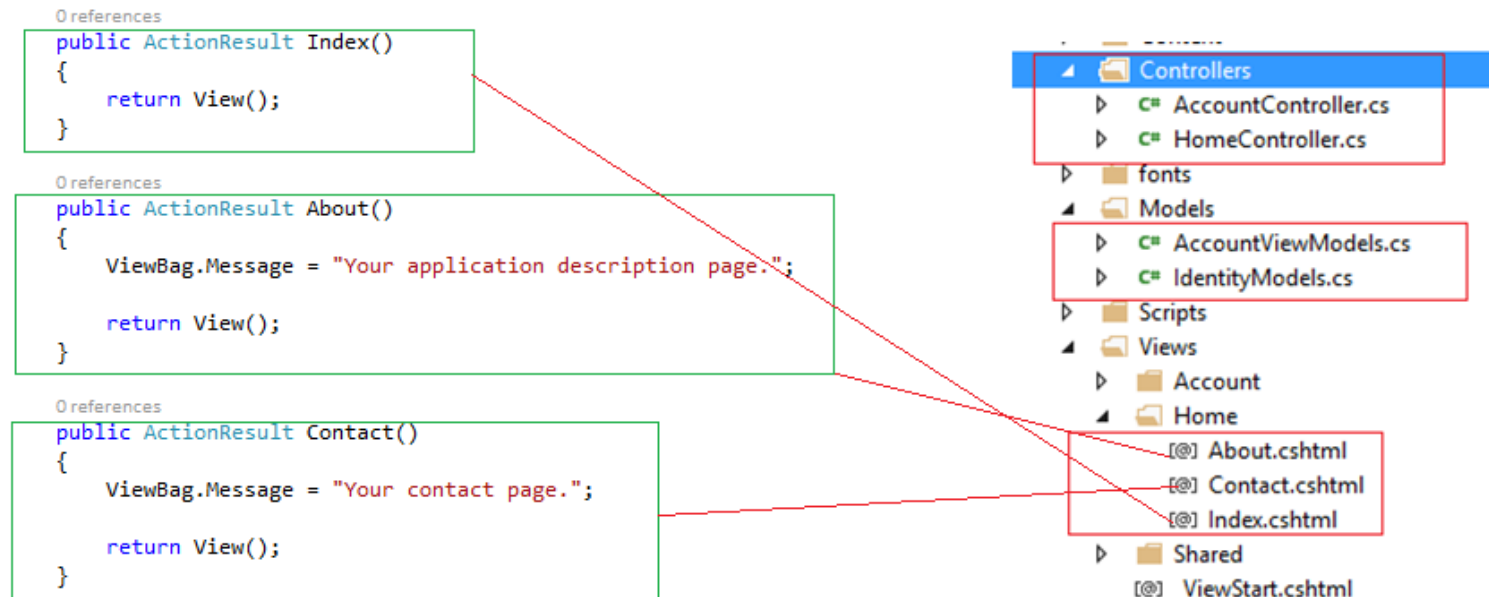
- **Các folders chính: Models, Views, Controllers** - là nơi sẽ lưu các file .cs, .cshtml tương ứng với các chức năng của mô hình MVC.
- **HomeController.cs** với View tương ứng là **Home**. Action Index của Home Controller sẽ được gọi khi chạy ứng dụng. Mã HomeController như sau:



TẠO ỨNG DỤNG ASP.NET MVC

✓ Cấu trúc một project

- Trong HomeController có sẵn 3 Action là Index, Contact, About. Mỗi Action này sẽ có 3 View tương ứng trong thư mục View



TẠO ỨNG DỤNG ASP.NET MVC

✓ Cấu trúc một project

- Content : nơi chứa các file css , ảnh
- Script : Chứa các file thư viện javascript như jQuery , AJAX,...
- App_Start : Chứa các lớp định nghĩa routed , bundled .

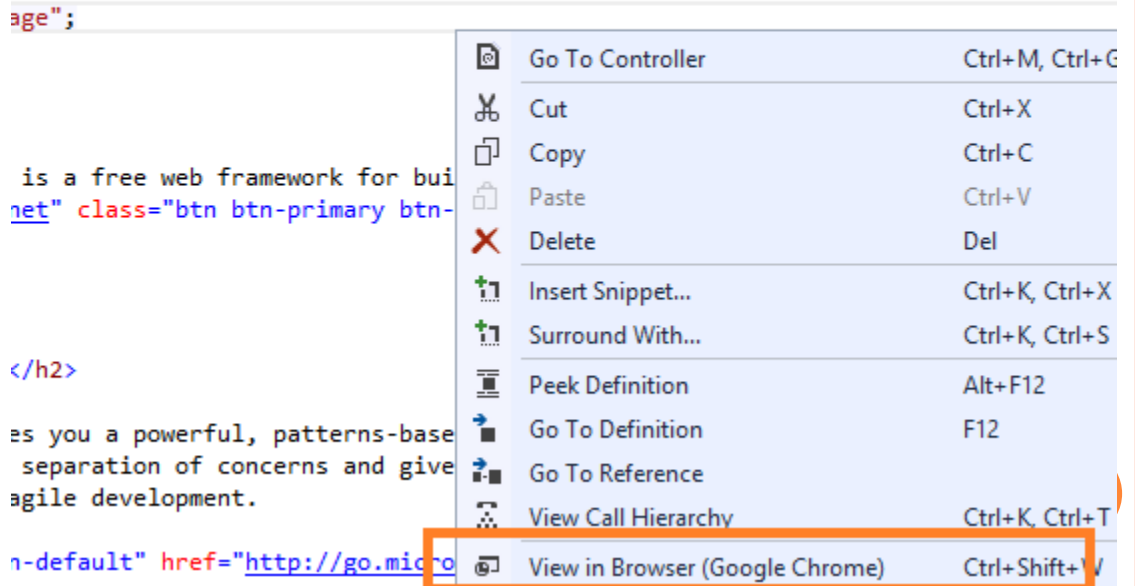
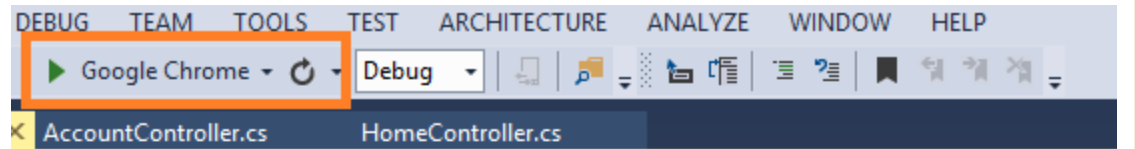


TẠO ỨNG DỤNG ASP.NET MVC

✓ Chạy project

- - Khi chạy ứng dụng thì View Index trong thư mục : View/Home/Index được gọi. Việc qui định này được khai báo trong thư File RouteConfig.cs trong thư mục App_Start.
- Cách 2 : Chuột phải lên View

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



Lập trình WEB

Controller

GIỚI THIỆU



Khái niệm MVC

- Controller chịu tương tác giữa models và views. Là nơi trao đổi dữ liệu giữa View và Model .
- Controller sẽ lấy dữ liệu và trả về View , Controller cũng lấy các yêu cầu trên view để xử lý , cũng như tương tác với CSDL.
- URL báo routing mechanism là controller class được thể hiện và action method được gọi. Sau đó controller's method quyết định view sử dụng, tiếp theo view đó lại renders ra HTML



GIỚI THIỆU

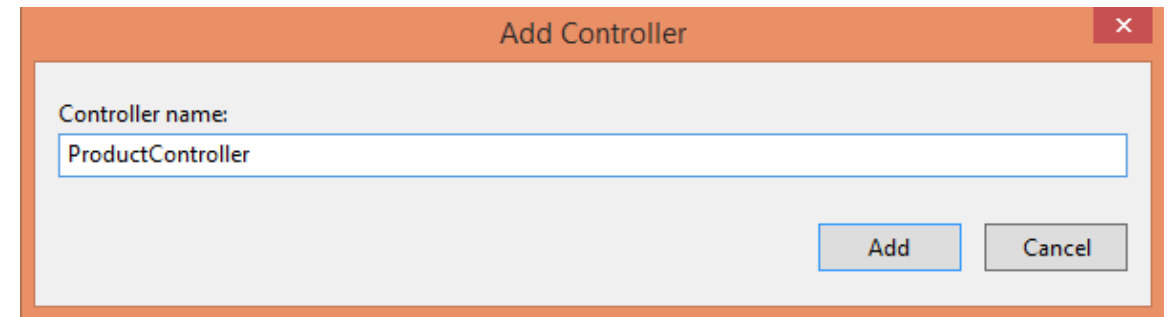
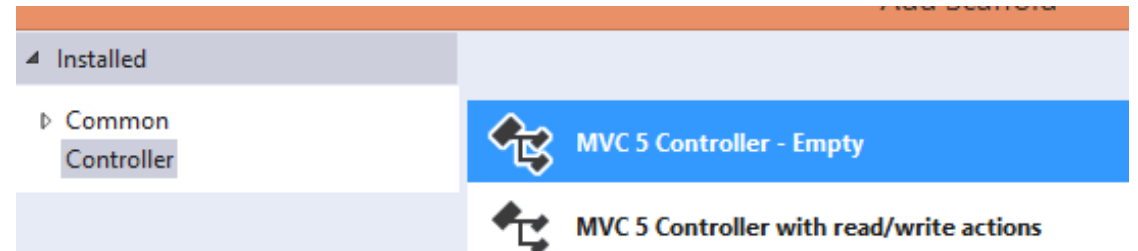
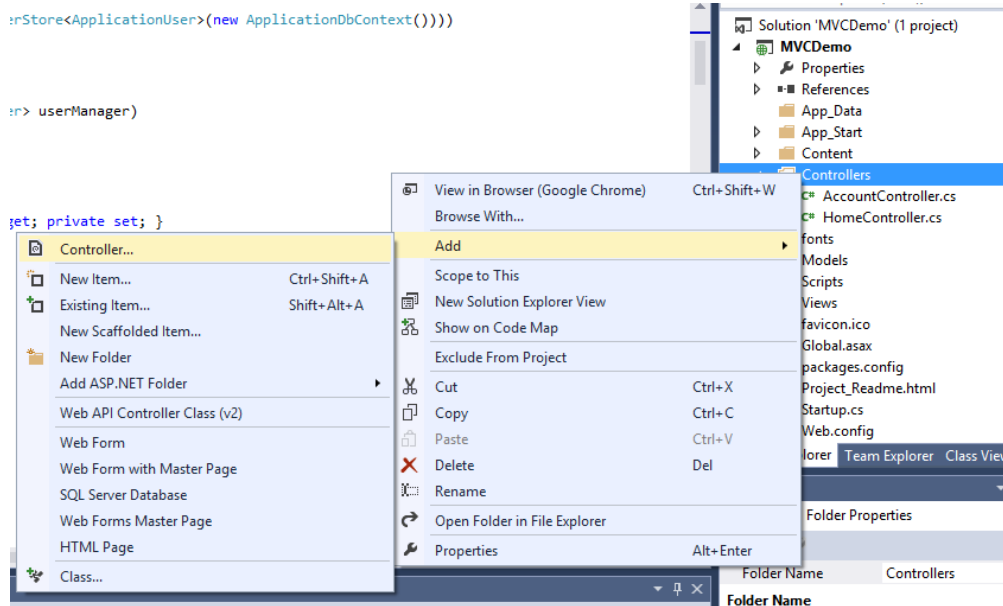


Tạo Controller

```
ifStore<ApplicationUser>(new ApplicationDbContext()))
```

```
ifStore<userManager>
```

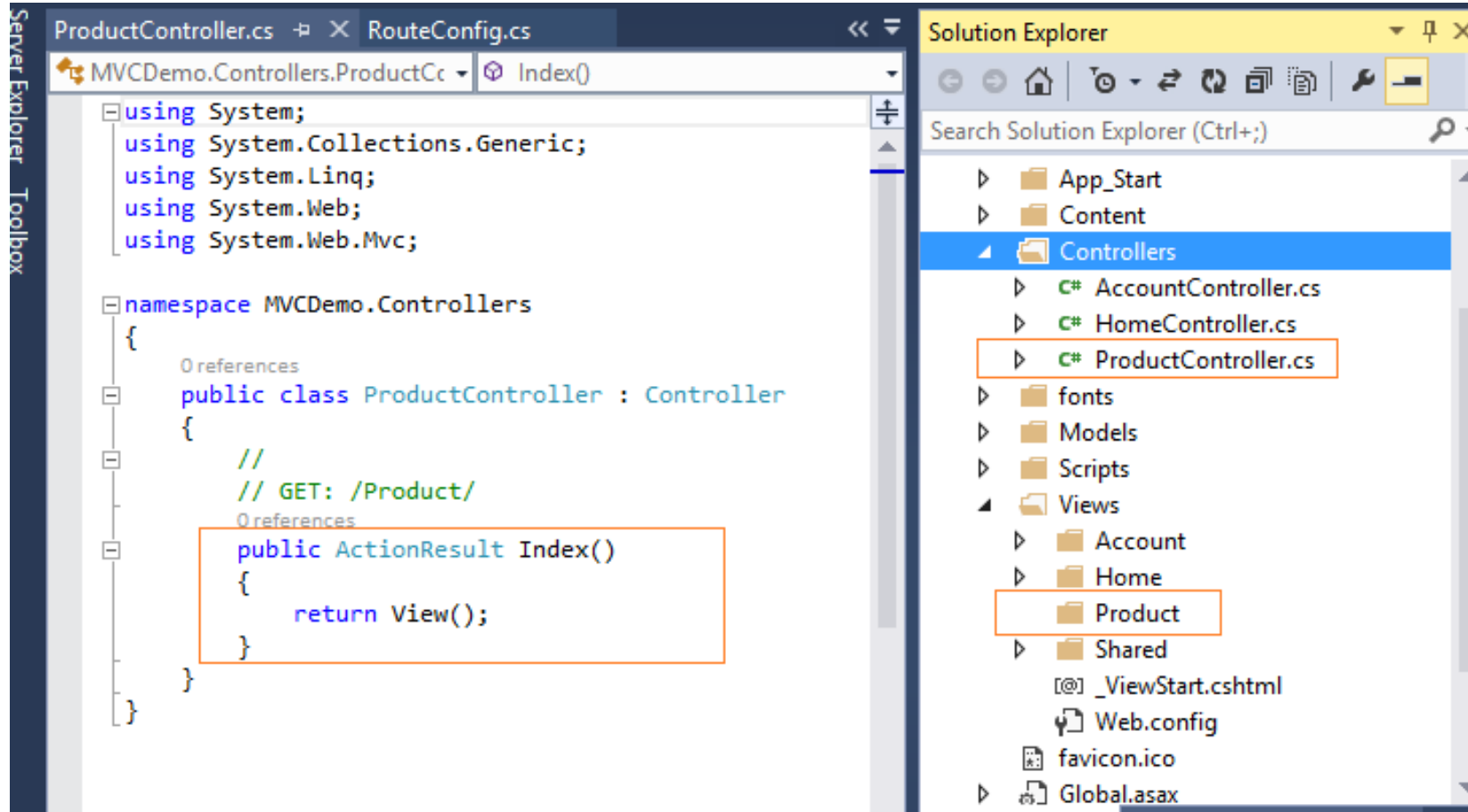
```
get; private set; }
```



GIỚI THIỆU



Tạo Controller



GIỚI THIỆU

✓ Phân tích, mô tả Controller

- Lớp **ProductController** kế thừa lại lớp **Controller** (Microsoft cung cấp sẵn trong thư viện Asp.net MVC).
- Tới **Action Index** là một **ActionResult**, **Action** này trả về một **View** (kiểu html)

```
namespace MVCDemo.Controllers
{
    0 references
    public class ProductController : Controller
    {
        //
        // GET: /Product/
        0 references
        public ActionResult Index()
        {
            return View();
        }
    }
}
```



GIỚI THIỆU

✓ Phân tích, mô tả Controller

- Trong Controller có thể có các phương thức trả về các kiểu dữ liệu thông thường như là string , int , class ...
- Action Index trả về một View (các thẻ html) .View này có tên là Index và phải đặt trong thư mục :
View/Product/Index.cshtml.

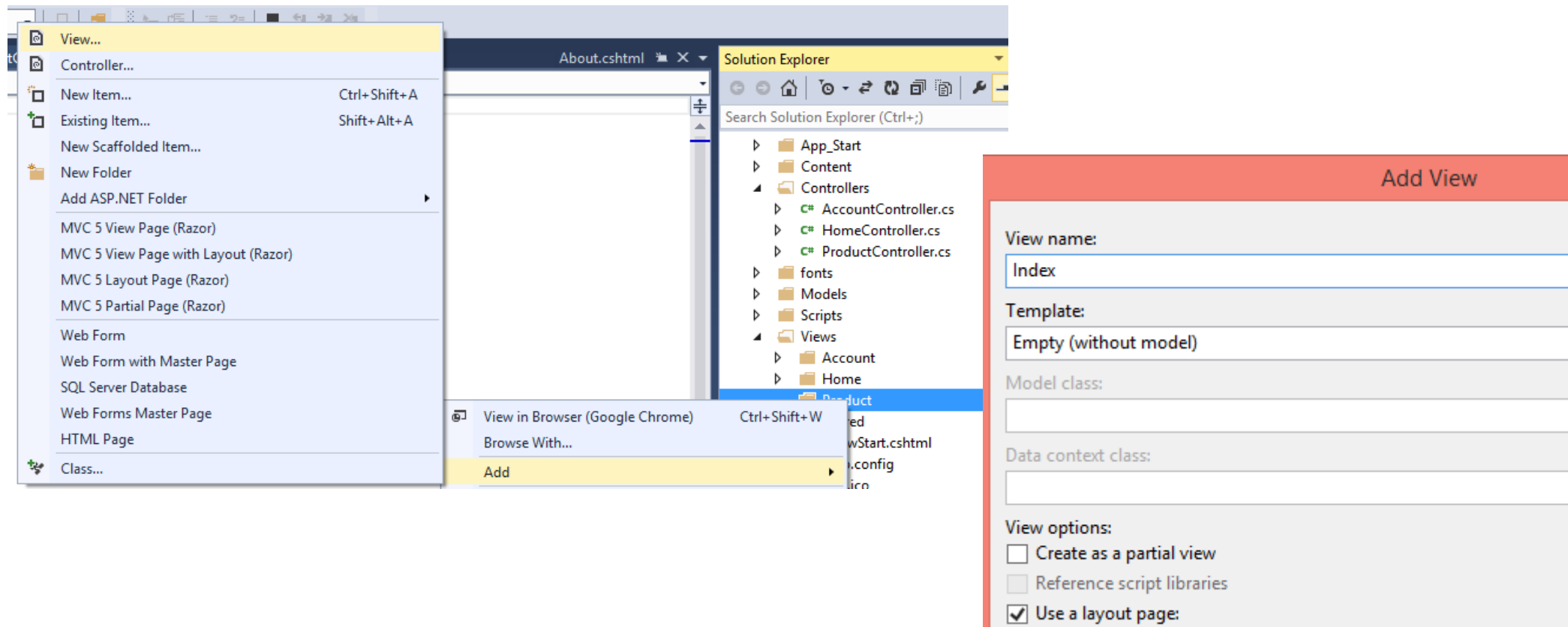
```
namespace MVCDemo.Controllers
{
    0 references
    public class ProductController : Controller
    {
        //
        // GET: /Product/
        0 references
        public ActionResult Index()
        {
            return View();
        }
    }
}
```



GIỚI THIỆU

✓ Phân tích, mô tả Controller

- Trong Tạo View tương ứng với Controller



The image shows a screenshot of the Visual Studio IDE. On the left, the 'View...' menu is open, displaying options for creating new items, scaffolding, and adding ASP.NET folders. The 'Add ASP.NET Folder' option is expanded, showing a list of MVC 5 View Page (Razor) templates. The 'Solution Explorer' on the right shows the project structure, including folders for App_Start, Content, Controllers, fonts, Models, Scripts, and Views. The 'Controllers' folder is expanded, showing AccountController.cs, HomeController.cs, and ProductController.cs. The 'Views' folder is also expanded, showing subfolders for Account and Home. The 'Add View' dialog is open on the right, with the 'View name' field set to 'Index' and the 'Template' set to 'Empty (without model)'. The 'Model class' field is empty. The 'Data context class' field is empty. The 'View options' section has three checkboxes: 'Create as a partial view' (unchecked), 'Reference script libraries' (unchecked), and 'Use a layout page' (checked).

View...

- Controller...
- New Item... Ctrl+Shift+A
- Existing Item... Shift+Alt+A
- New Scaffolded Item...
- New Folder
- Add ASP.NET Folder
 - MVC 5 View Page (Razor)
 - MVC 5 View Page with Layout (Razor)
 - MVC 5 Layout Page (Razor)
 - MVC 5 Partial Page (Razor)
 - Web Form
 - Web Form with Master Page
 - SQL Server Database
 - Web Forms Master Page
 - HTML Page
- Class...

Solution Explorer

- App_Start
- Content
- Controllers
 - AccountController.cs
 - HomeController.cs
 - ProductController.cs
- fonts
- Models
- Scripts
- Views
 - Account
 - Home

Add View

View name: Index

Template: Empty (without model)

Model class:

Data context class:

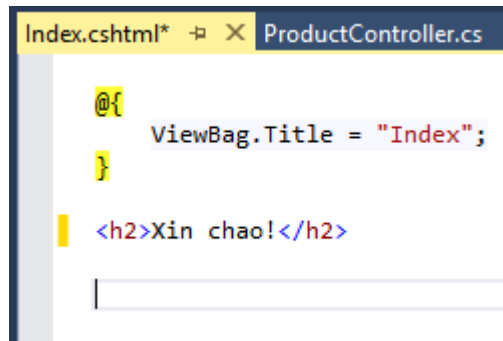
View options:

- ☐ Create as a partial view
- ☐ Reference script libraries
- ☒ Use a layout page:

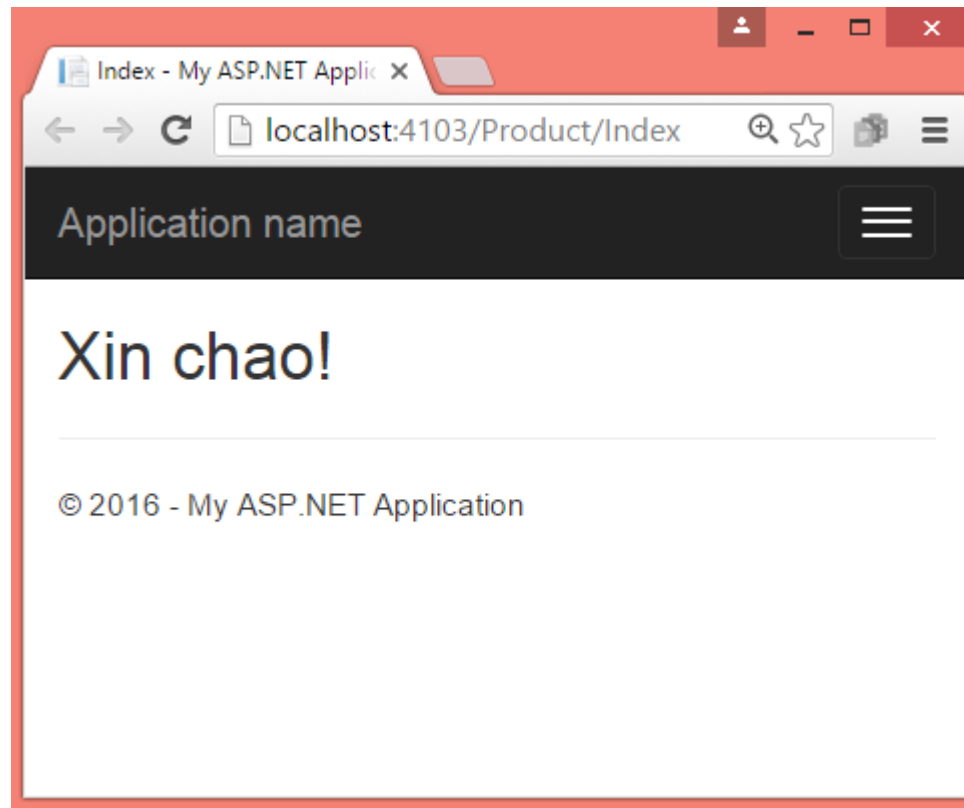
GIỚI THIỆU

✓ Phân tích, mô tả Controller (MVC_Main_vd2)

- Trong Tạo View tương ứng với Controller



```
@{  
    ViewBag.Title = "Index";  
}  
  
<h2>Xin chào!</h2>
```



GIỚI THIỆU



Nhận dữ liệu đầu vào trong controller

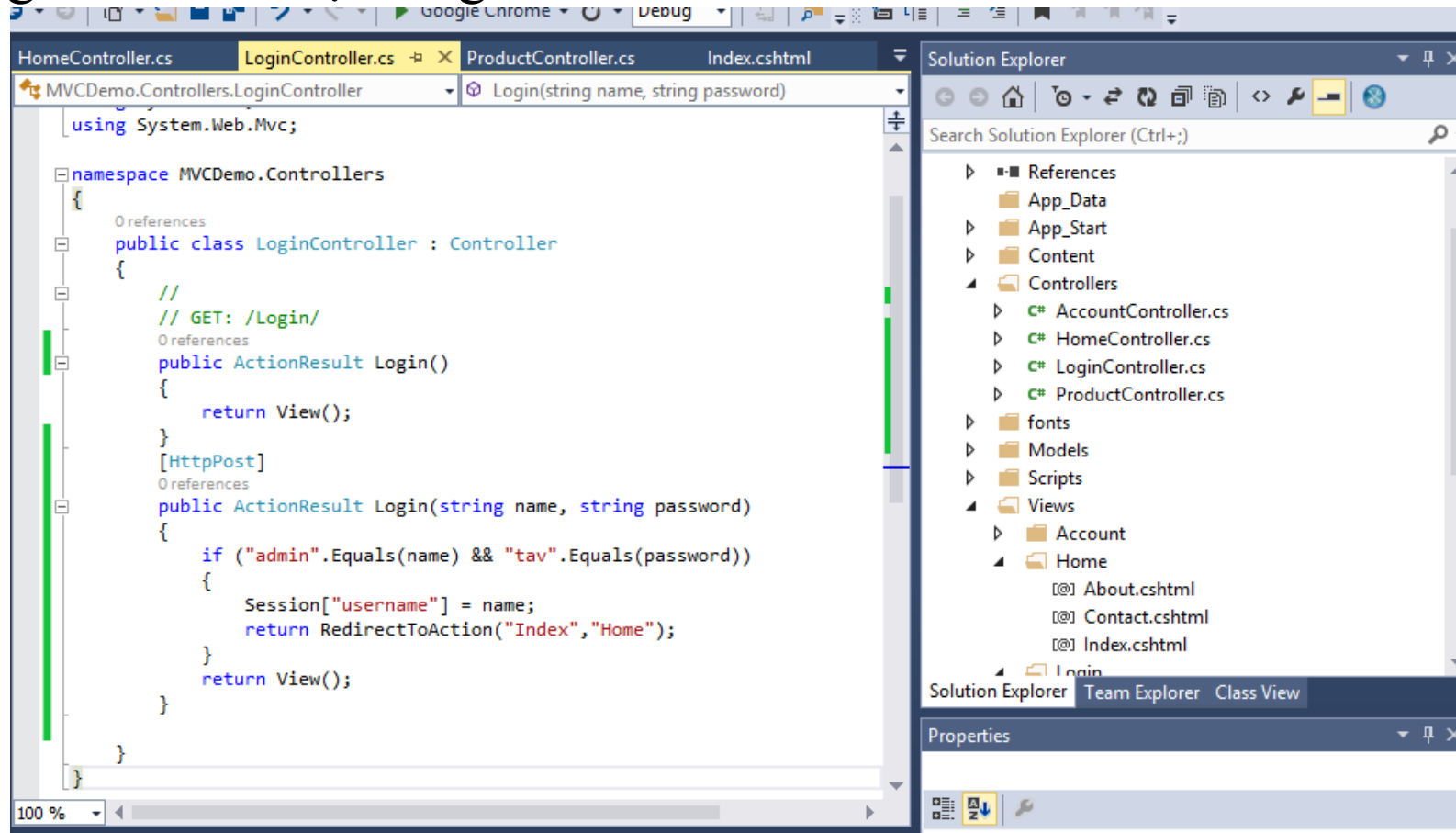
- Controller thường xuyên làm công việc nhận dữ liệu đầu vào như là chuỗi query, giá trị từ form đưa lên hoặc là những tham số được hệ thống route parse từ incoming url. Có 3 cách chính để truy xuất vào các dữ liệu loại này:
 - Có dữ liệu truyền vào như là tham số của action
 - Lấy dữ liệu từ những đối tượng context
 - Gọi tính năng model binding một cách tường minh



GIỚI THIỆU

✓ Lấy dữ liệu truyền vào như là tham số của action (vd3)

- Trong Controller tạo LoginController



The screenshot displays the Visual Studio IDE. The main editor window shows the `LoginController.cs` file, which is part of the `MVCDemo.Controllers` namespace. The code defines a `LoginController` class that inherits from `Controller`. It includes two actions: a GET action `Login()` that returns the `View()`, and a POST action `Login(string name, string password)` that checks if the username is "admin" and the password is "tav". If the credentials are correct, it sets the session username and redirects to the "Index" action of the "Home" controller. Otherwise, it returns the `View()`.

```
using System.Web.Mvc;

namespace MVCDemo.Controllers
{
    // GET: /Login/
    public ActionResult Login()
    {
        return View();
    }

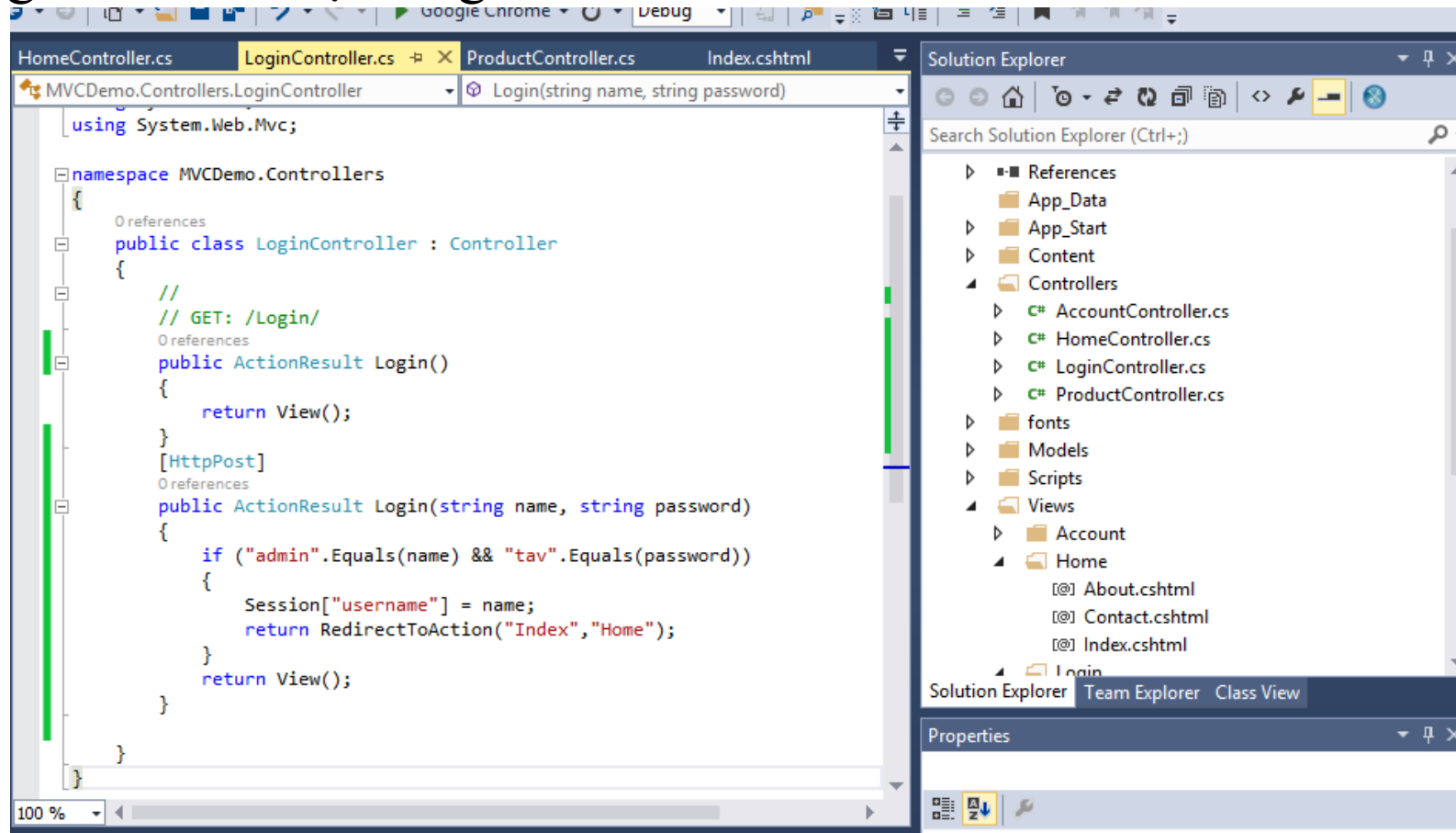
    [HttpPost]
    public ActionResult Login(string name, string password)
    {
        if ("admin".Equals(name) && "tav".Equals(password))
        {
            Session["username"] = name;
            return RedirectToAction("Index", "Home");
        }
        return View();
    }
}
```

The Solution Explorer on the right shows the project structure, including the `Controllers` folder which contains `AccountController.cs`, `HomeController.cs`, `LoginController.cs`, and `ProductController.cs`. The `Views` folder is also visible, containing subfolders for `Account` and `Home`, and files like `About.cshtml`, `Contact.cshtml`, and `Index.cshtml`.

GIỚI THIỆU

✓ Lấy dữ liệu truyền vào như là tham số của action (vd3)

- Trong Controller tạo LoginController



The screenshot displays the Visual Studio IDE. The main editor window shows the `LoginController.cs` file, which is part of the `MVCDemo.Controllers` namespace. The code defines a `LoginController` class that inherits from `Controller`. It includes two actions: a GET action `Login()` that returns the `View()`, and a POST action `Login(string name, string password)` that checks if the username is "admin" and the password is "tav". If the credentials are correct, it sets the session username and redirects to the "Index" action of the "Home" controller. Otherwise, it returns the `View()`.

```
using System.Web.Mvc;

namespace MVCDemo.Controllers
{
    // GET: /Login/
    public ActionResult Login()
    {
        return View();
    }

    [HttpPost]
    public ActionResult Login(string name, string password)
    {
        if ("admin".Equals(name) && "tav".Equals(password))
        {
            Session["username"] = name;
            return RedirectToAction("Index", "Home");
        }
        return View();
    }
}
```

The Solution Explorer on the right shows the project structure, including the `Controllers` folder which contains `AccountController.cs`, `HomeController.cs`, `LoginController.cs`, and `ProductController.cs`. The `Views` folder is also visible, containing subfolders for `Account` and `Home`, and files like `About.cshtml`, `Contact.cshtml`, and `Index.cshtml`.

GIỚI THIỆU



Lấy dữ liệu truyền vào như là tham số của action (vd3)

- Trong Views/Login tạo Login.cshtml

```
@{
    ViewBag.Title = "Login";
}

<h2>Login</h2>
@using(Html.BeginForm())
{
    <div>@Html.Label("Login name")</div>
    <div>@Html.TextBox("name")</div>
    <div>@Html.Label("Password")</div>
    <div>@Html.TextBox("password")</div>
    <input type="submit" value="Login">
}
}
```



GIỚI THIỆU



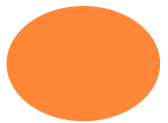
Lấy dữ liệu truyền vào như là tham số của action (vd3)

- Thay đổi code của HomeController

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Mvc;

namespace MVCDemo.Controllers
{
    public class HomeController : Controller
    {
        public ActionResult Index()
        {
            if(Session["username"]==null)
            {
                return RedirectToAction("Login","Login");
            }
            return View();
        }

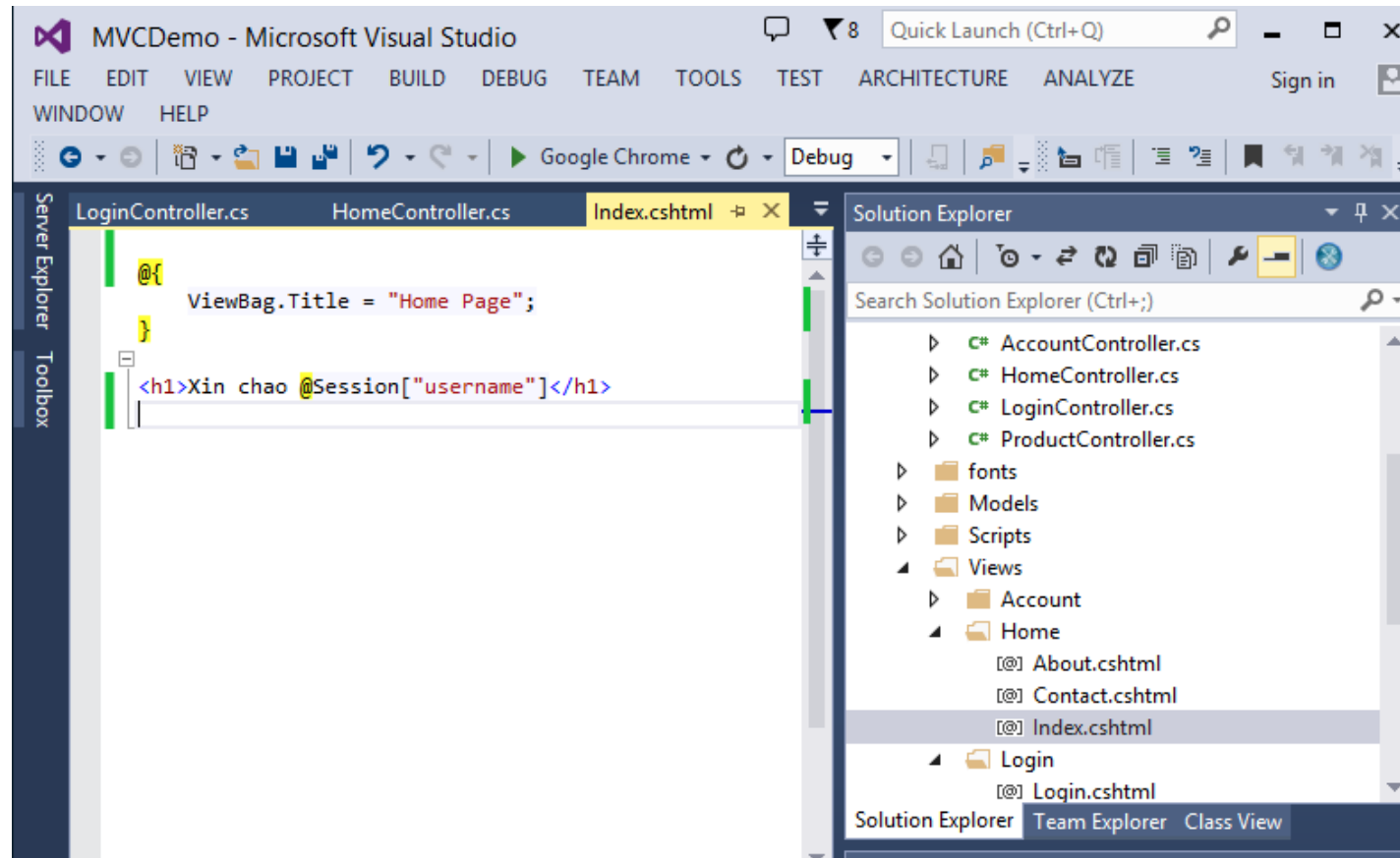
        public ActionResult About()
        {
            ViewBag.Message = "Your application description p...";
            return View();
        }
    }
}
```



GIỚI THIỆU

✓ Lấy dữ liệu truyền vào như là tham số của action (vd3)

- Thay đổi code của Views/Home/Index.cshtml



NHẬN DỮ LIỆU ĐẦU VÀO TRONG CONTROLLER

✓ Lấy dữ liệu từ những đối tượng context

- Những context object thông dụng

Thuộc tính	Kiểu	Mô tả
Request.QueryString	NameValueCollection	Những biến GET gửi cùng request
Request.Form	NameValueCollection	Những biến POST gửi cùng request
Request.Cookies	HttpCookieCollection	Những cookie do browser gửi cùng request này
Request.HttpMethod	string	Phương thức HTTP (như là GET hay POST) được sử dụng cùng request này
Request.Headers	NameValueCollection	Tập http header gửi cùng request này
Request.Url	Uri	Url được yêu cầu

NHẬN DỮ LIỆU ĐẦU VÀO TRONG CONTROLLER



Lấy dữ liệu từ những đối tượng context

■ Những context object thông dụng

Thuộc tính	Kiểu	Mô tả
Request.UserHostAddress	string	Địa chỉ IP của người yêu cầu làm request này
RouteData.Route	RouteBase	Route được chọn cho request này
RouteData.Values	RouteDataDictionary	Những tham số của route (Được chiết từ URL hoặc từ những giá trị mặc định)
HttpContext.Application	HttpApplicationStateBase	Nơi lưu trữ trạng thái ứng dụng
HttpContext.Cache	Cache	Nơi lưu trữ cache ứng dụng
HttpContext.Items	IDictionary	Nơi lưu trữ trạng thái cho request hiện tại
HttpContext.Session	HttpSessionStateBase	Lưu trữ trạng thái của phiên làm việc
User	IPrincipal	Thông tin chứng thực về người đăng nhập hệ thống

GIỚI THIỆU

✓ Lấy dữ liệu từ những đối tượng context (vd4)

- Sửa lại từ vd3, Controller tạo LoginController

```
public ActionResult Login()
{
    return View();
}

[HttpPost]
0 references
public ActionResult LoginAction()
{
    string name = Request.Form["name"];
    string password = Request.Form["password"];
    if ("admin".Equals(name) && "tav".Equals(password))
    {
        Session["username"] = name;
        return RedirectToAction("Index", "Home");
    }
    return View();
}
```



GIỚI THIỆU

✓ Lấy dữ liệu từ những đối tượng context (vd4)

- Sửa lại từ vd3, Views/Login/Login.cshtml

```
<form action="Login" method="post">  
    Username: <input type="text" name="username"/><br/>  
    Password: <input type="password" name="password" /><br />  
    <input type="submit" value="Dang nhap" /><br />  
</form>
```



GIỚI THIỆU

✓ Lấy dữ liệu truyền vào thông qua Data Model Binding(vd5)

- Trong folder Models, thêm class Account:

```
using System;
using System.Web;

namespace MVCDemo.Models
{
    1 reference
    public class Account
    {
        0 references
        public string Name { get; set; }
        1 reference
        public string Password { get; set; }
    }
}
```



GIỚI THIỆU



Lấy dữ liệu truyền vào thông qua Data Model Binding(vd5)

- Từ vd4, thay đổi LoginAction trong LoginController:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Mvc;
using MVCDemo.Models;

namespace MVCDemo.Controllers
{
    0 references
    public class LoginController : Controller
    {
        0 references
        public ActionResult Login()
        {
            return View();
        }

        [HttpPost]
        0 references
        public ActionResult LoginAction(Account acc)
        {
            ViewBag.Title = acc.Password + "!!";
            return View("Login");
        }
    }
}
```



GIỚI THIỆU

✓ Lấy dữ liệu từ những đối tượng context (vd5)

- Sửa lại từ vd4, Views/Login/Login.cshtml

```
@model MVCDemo.Models.Account

<h2>Login</h2>
@using (Html.BeginForm("LoginAction", "Login"))
{
    @Html.LabelFor(m => m.Name)
    @Html.TextBoxFor(m => m.Name)
    <br />
    @Html.LabelFor(m => m.Password)
    @Html.PasswordFor(m => m.Password)
    <input type="submit" value="Login">
}
```

- Chạy chương trình và xem sự thay đổi của title sau khi Login



CUNG CẤP DỮ LIỆU ĐẦU RA

✓ Giới thiệu

- Controller sau khi nhận dữ liệu thông qua Request, tiến hành xử lý, sẽ trả về dữ liệu thông qua một Response.
- Có thể trả về nội dung thông qua Response.Write.
- Có thể điều hướng trang thông qua Response.Redirect



CUNG CẤP DỮ LIỆU ĐẦU RA

✓ Giới thiệu

- Mô hình MVC sử dụng đối tượng một đối tượng dẫn xuất từ class ActionResult thay cho đối tượng Response.
- MVC Framework chứa một số built-in action result types được dẫn xuất từ ActionResult, và chúng có các phương thức helper thuận tiện trong class Controller.



CUNG CẤP DỮ LIỆU ĐẦU RA



Các ActionResult Types

Type	Helper Methods	Mô tả
ViewResult	View	Sinh ra chỉ định hoặc default view template
PartialViewResult	PartialView	Sinh ra chỉ định hoặc default partial view template
RedirectToRouteResult	RedirectToAction RedirectToActionPermanent RedirectToRoute RedirectToRoutePermanent	Điều hướng đến một Action method hay route chỉ định.
RedirectResult	Redirect RedirectPermanent	Điều hướng đến URL chỉ định.



CUNG CẤP DỮ LIỆU ĐẦU RA

✓ Các ActionResult Types

- Chỉ định view được render là “Contact” bằng sử dụng phương thức View

```
public ActionResult Index()
{
    return View("Contact");
}
```

- Chỉ định view được render là Index (trùng tên Action method, tên thực sự của view xác định bởi RouteData.Values["action"]).

```
public ActionResult Index()
{
    return View();
}
```



CUNG CẤP DỮ LIỆU ĐẦU RA

✓ Truyền Data từ Action Method đến View (vd6)

- Truyền đối tượng bằng cách truyền nó như một tham số tới View

```
public ActionResult Index()
{
    Account acc = new Account();
    acc.Name = "admin";
    acc.Password = "tav";
    return View(acc);
}
```

- Phía View tiếp nhận. Để truy cập đối tượng trong view sử dụng Razor Model keyword hoặc Razor model keyword

```
@using MVCDemo.Models
@{
    ViewBag.Title = "Home Page";
}
```

```
<h1> Xin chao @(((Account)Model).Name)</h1>
```

```
@using MVCDemo.Models
@model Account
@{
    ViewBag.Title = "Home Page";
}
```

```
<h1> Xin chao @Model.Name</h1>
```



CUNG CẤP DỮ LIỆU ĐẦU RA

✓ Truyền Data từ Action Method đến View (vd7)

- Truyền Data với View Bag: Có thể truyền data qua dynamic object và truy cập chúng trên view, bằng cách này ta có thể truyền đối tượng dữ liệu khác nhau:

```
public ActionResult Index()
{
    Account acc = new Account();
    acc.Name = "admin";
    acc.Password = "tav";

    ViewBag.Message = "Xin chao";
    ViewBag.Date = DateTime.Now;
    ViewBag.acc = acc;
    return View();
}
```

```
@using MVCDemo.Models
@model Account
@{
    ViewBag.Title = "Home Page";

    <h1> Xin chao @ViewBag.acc.Name</h1>
    <h1> Bay gio la @ViewBag.Date.DayOfWeek</h1>
}
```



CUNG CẤP DỮ LIỆU ĐẦU RA

✓ Điều hướng thông qua **Redirections** (vd7)

- Điều hướng đến Literal URL: Để điều hướng trình duyệt ta gọi phương thức Redirect với tham số là URL cần được dẫn tới

```
public ActionResult Index()  
{  
  
    return Redirect("/Home/Contact");  
}
```



CUNG CẤP DỮ LIỆU ĐẦU RA

✓ Điều hướng thông qua Redirections (vd9)

- Điều hướng Routing System URL: Sử dụng Routing System URL để sinh ra valid URLs dùng RedirectToRoute method, tại đó xác định controller, action, id được điều hướng

```
public ActionResult Index()
{
    TempData["Mes"] = "Xin chao!";
    TempData["Time"] = DateTime.Now;
    return RedirectToAction("About");
}
```

```
public ActionResult About()
{
    ViewBag.Message = TempData["Mes"];
    ViewBag.Date = TempData["Time"];
    return View();
}
```

```
@{
    ViewBag.Title = "About";
}
<h2>@TempData["Mes"]</h2>
<<h2>@TempData["Time"]</h2>
```



ROUTE



Giới thiệu

- Route (bộ định tuyến) trong MVC là các khai báo để ứng dụng MVC có thể biết được Action và Controller nào sẽ được gọi để xử lý yêu cầu từ phía người dùng.
- Khác với trong webform, việc gọi các file aspx đường dẫn mang tính vật lý, thì các việc dùng route bằng code và chúng ta có thể tùy biến các route để được các URL gọn gàng và mạch lạc.
- Route so khớp với các requests đến với một file trong file system và ánh xạ request đó tới controller action.



ROUTE

✓ Giới thiệu

- Route được định nghĩa trong file RouteConfig.cs trong thư mục App_Start

```
public class RouteConfig
{
    1 reference
    public static void RegisterRoutes(RouteCollection routes)
    {
        routes.IgnoreRoute("{resource}.axd/{*pathInfo}");

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



ROUTE



Giới thiệu

- Có thể định nghĩa nhiều Route, ở đây tạo sẵn một route có tên là Default .
- Home Controller, Action Index là giá trị mặc định khi ứng dụng chạy .
- Việc gọi các Action trong Controller bất kỳ tuân theo cấu trúc :

/TenController/Action/ThamSo

- Với những Action không có tham số thì chỉ cần gọi : **/TenController/Action/**
- Với Action có tên là Index thì chỉ cần gọi : **/TenController**



ROUTE



Thêm các route mới (vd10)

- Khi 1 yêu cầu từ phía client thì router sẽ tiến hành map từ trên xuống dưới, khi nào tìm được route có đủ các yêu cầu về tham số và tên thì dừng lại.

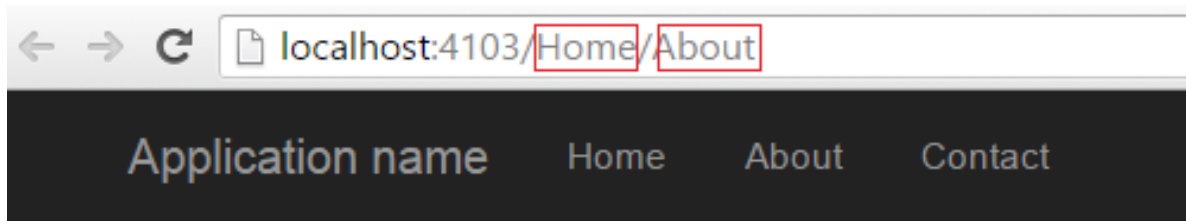
```
public static void RegisterRoutes(RouteCollection routes)
{
    routes.IgnoreRoute("{resource}.axd/{*pathInfo}");
    routes.MapRoute(
        name: "Login",
        url: "Admin/{controller}/{action}/{id}",
        defaults: new
        {
            controller = "Login",
            action = "Login",
            id = UrlParameter.Optional
        }
    );
    routes.MapRoute(
        name: "Default",
        url: "{controller}/{action}/{id}",
        defaults: new { controller = "Home",
            action = "Index", id = UrlParameter.Optional }
    );
}
```



ROUTE

✓ Giới thiệu

- Các thành phần tương ứng trong route hiển thị trên URL



Xin chào!



ROUTE



Giới thiệu

- Phương thức RegisterRoutes được gọi từ tệp Global.asax.cs

```
public class MvcApplication : System.Web.HttpApplication
{
    0 references
    protected void Application_Start()
    {
        AreaRegistration.RegisterAllAreas();
        FilterConfig.RegisterGlobalFilters(GlobalFilters.Filters);
        RouteConfig.RegisterRoutes(RouteTable.Routes);
        BundleConfig.RegisterBundles(BundleTable.Bundles);
    }
}
```



Lập trình WEB

View

GIỚI THIỆU

Khái niệm View

- View: có nhiệm vụ tiếp nhận dữ liệu từ controller và hiển thị nội dung sang các đoạn mã HTML.
- Trong mô hình MVC Controller thường cung cấp thông tin cho view bằng cách truyền dữ liệu. View chuyển dữ liệu này thành định dạng biểu diễn được cho người dùng.



GIỚI THIỆU

✓ Khái niệm View

- Mỗi controller có một thư mục trùng tên trong thư mục view /Views/ControllerName (Không chứa hậu tố controller).
- Theo mặc định, mỗi controller folder chứa một hoặc nhiều view cho từng action method có tên trùng với tên action method



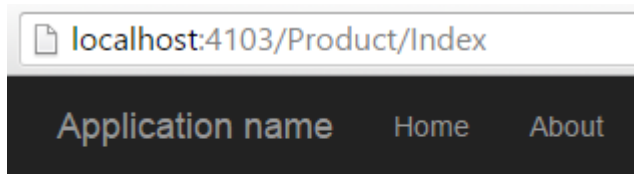
GIỚI THIỆU

✓ Thay đổi View mặc định

- Ta có thể định nghĩa lại các quy tắc xác định view cách chỉ định tên view muốn được render.

+ Cùng thư mục:

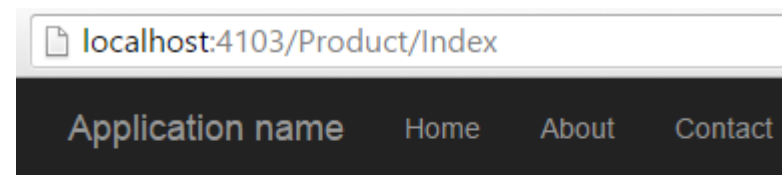
```
public ActionResult Index()
{
    return View("List");
}
```



List

+ Khác thư mục:

```
public ActionResult Index()
{
    return View("~/Views/Home/About.cshtml");
}
```



About.

Use this area to provide additional information.



GIỚI THIỆU



Tạo View

- Chuột phải lên Action Index hoặc chọn Add view trong thư mục views tương ứng

The screenshot shows the Visual Studio IDE with the MVC Demo project running. The code editor displays the `ProductController.cs` file, which includes the `Index()` action. The `Index()` action is highlighted, and a context menu is open, showing options for adding a new view. The menu includes options like `View...`, `Controller...`, `New Item...`, `Existing Item...`, `New Scaffolded Item...`, `New Folder`, `Add ASP.NET Folder`, `MVC 5 View Page (Razor)`, `MVC 5 View Page with Layout (Razor)`, `MVC 5 Layout Page (Razor)`, `MVC 5 Partial Page (Razor)`, `ASP.NET Module`, and `Web Form`. The `Add` option is highlighted, and a sub-menu is open, showing options like `View in Browser (Google Chrome)`, `Browse With...`, `Add`, `Scope to This`, `New Solution Explorer View`, `Show on Code Map`, `Exclude From Project`, `Cut`, `Copy`, `Paste`, `Delete`, `Rename`, `Open Folder in File Explorer`, and `Properties`.

GIỚI THIỆU



Tạo View

- View name : tên view tương ứng với tên Action ở đây là Action Index thì ta cần khai báo là Action Index.
- Trong folder Models tạo class Student như sau:

```
namespace MVCDemo.Models
{
    4 references
    public class Student
    {
        1reference
        public int ID { get; set; }
        1reference
        public string Name { get; set; }
        1reference
        public int Age { get; set; }

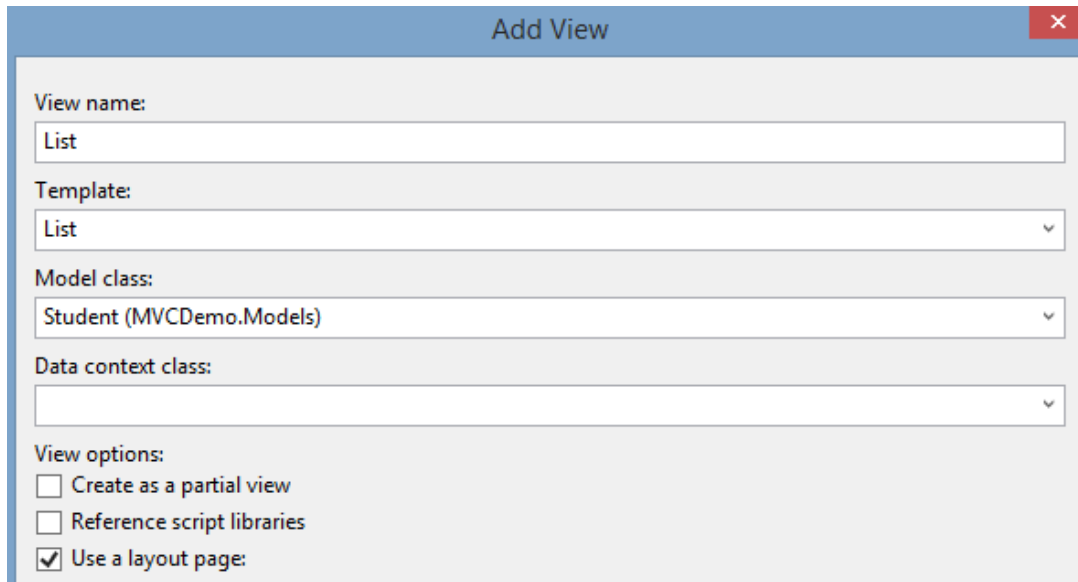
        0references
        public static IEnumerable<Student> GetList(int Count)
        {
            var st = new List<Student>();
            for (int i = 1; i <= Count; i++)
            {
                st.Add(new Student { ID = i, Name = "Student" + i.ToString(), Age=20 });
            }
            return st;
        }
    }
}
```



GIỚI THIỆU

✓ Tạo View (vd11)

- Tạo một View với các thông số sau:



View name: List

Template: List

Model class: Student (MVCDemo.Models)

Data context class:

View options:

- ☐ Create as a partial view
- ☐ Reference script libraries
- ☒ Use a layout page:

```
@foreach (var item in Model) {  
    <tr>  
        <td>  
            @Html.DisplayFor(modelItem => item.Name)  
        </td>  
        <td>  
            @Html.DisplayFor(modelItem => item.Age)  
        </td>  
        <td>  
            @Html.ActionLink("Edit", "Edit", new { id=item.ID }) |  
            @Html.ActionLink("Details", "Details", new { id=item.ID }) |  
            @Html.ActionLink("Delete", "Delete", new { id=item.ID })  
        </td>  
    </tr>  
}
```



GIỚI THIỆU

✓ Tạo View

- Trong StudentController thay đổi như sau:

```
namespace MVCDemo.Controllers
{
    0 references
    public class StudentController : Controller
    {
        //
        // GET: /Student/
        0 references
        public ActionResult Index()
        {
            return View("List", Student.GetList(5));
        }
    }
}
```



GIỚI THIỆU



Tạo View

■ Kết quả:

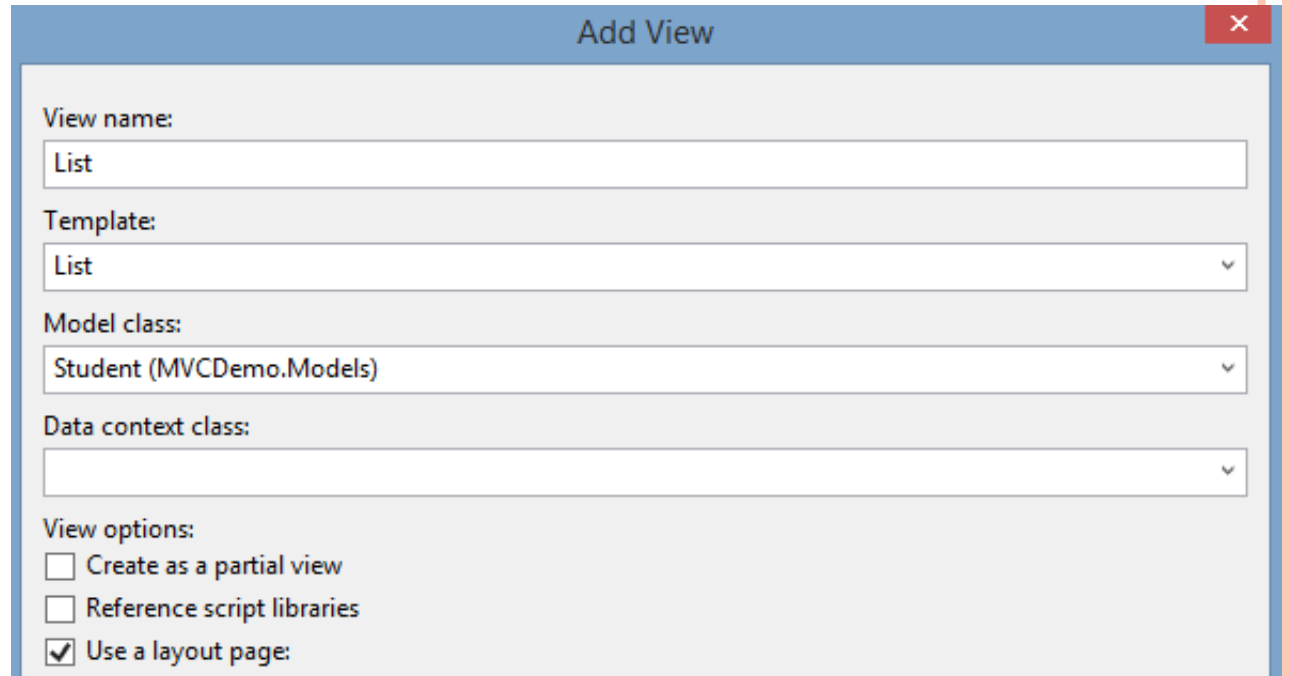
localhost:4103/Student		
Application name	Home	About Contact
List		
Create New		
Name	Age	
Student1	20	Edit Details Delete
Student2	20	Edit Details Delete
Student3	20	Edit Details Delete
Student4	20	Edit Details Delete
Student5	20	Edit Details Delete



GIỚI THIỆU

✓ Giải thích View (vd11)

- View name : tên view
- Template: là kiểu template được View nhận về, ở đây ta khai báo kiểu là list thì model trong View sẽ nhận về kiểu `IEnumerable<T>`
- Model class: ta chọn là Student, cụ thể ở đây View sẽ nhận về kiểu `IEnumerable< Student >`
- Layout là là 1 file để các View khác kế thừa (tương tự như masterpage trong Asp web form)



The screenshot shows the 'Add View' dialog box. The 'View name' field is set to 'List'. The 'Template' dropdown is set to 'List'. The 'Model class' dropdown is set to 'Student (MVCDemo.Models)'. The 'Data context class' field is empty. Under 'View options', the 'Use a layout page' checkbox is checked, while 'Create as a partial view' and 'Reference script libraries' are unchecked.

GIỚI THIỆU

✓ Giải thích View (vd11)

- Strongly Typed Models (model được ép kiểu mạnh) và từ khóa @model
- Có thể truyền dữ liệu đến view thông qua đối tượng ViewBag hoặc ViewData, tuy nhiên có một vấn đề khi sử dụng hai đối tượng này (tuy hai mà một) là ứng dụng sẽ không biết được các đối tượng có chứa những gì, và mã lệnh có đúng hay không cho đến khi ứng dụng chạy. Nếu lập trình sơ sót, khả năng sinh lỗi là rất cao.
- ASP.NET MVC cũng cung cấp khả năng để truyền các đối tượng có kiểu rõ ràng cho view.
- Bằng cách sử dụng dòng @model bên trong tập tin view template, có thể quy định kiểu của đối model mà view sẽ nhận được, lúc đó view sẽ hiểu model được nhận có kiểu gì.



GIỚI THIỆU



Giải thích View (vd11)

- `@model IEnumerable<MVCDemo.Models.Student>`
- Khai báo `@model` giúp cho view dễ dàng truy xuất đến danh sách tác vụ mà controller truyền đến thông qua thuộc tính Model của view.



GIỚI THIỆU



Template

- Xem các vd12 (create), vd13(delete), vd14(edit)

STT	Tên	Mô tả
1	Create	Tạo view với form được sinh ra tương ứng với model. Sinh ra các một label và input field cho mỗi thuộc tính của model type.
2	Delete	Tạo view với form cho việc xóa các thể hiện của model. Hiển thị label và current value mỗi thuộc tính của model.
3	Details	Tạo view hiển thị một label và giá trị cho mỗi thuộc tính của model type.
4	Edit	Tạo view với form cho việc sửa các thể hiện của model. Sinh một label và input field cho mỗi thuộc tính của model type.
5	Empty	Tạo empty view. Chỉ model type được chỉ định sử dụng cú pháp @model
6	List	Tạo view với danh sách dữ liệu.

TƯƠNG TÁC DỮ LIỆU TỪ CONTROLLER SANG VIEW

✓ Truyền bằng model (vd11)

- Trong các Action, khi trả về sẽ trả về View sẽ kèm theo dữ liệu
- Dữ liệu được trả về có thể là kiểu dữ liệu dạng (string, int,...), lớp đối tượng hay là Collection như array, List<T>,...

```
public ActionResult Index()
{
    return View("List", Student.GetList(5));
}
```

- Trong View khai báo biến Model cũng có kiểu List<T> như sau:

@model IEnumerable<T> ví dụ: @model IEnumerable<MVCDemo.Models.Student>

- Khai báo IEnumerable<T> tổng quát cho mọi Collection, trong đó có cả lớp List<T> hiện thực interface.



TƯƠNG TÁC DỮ LIỆU TỪ CONTROLLER SANG VIEW

✓ Truyền bằng ViewBag (vd16)

- Trong trường hợp Action trả về nhiều đối tượng Model khác thì ta dùng ViewBag
- ViewBag là đối tượng Dynamic, có thể gán cho chúng bất cứ dữ liệu nào, trong View có thể gọi trực tiếp chúng thông qua tên

```
public ActionResult Index()
{
    ViewBag.Mes = "Xin chào!";
    ViewBag.students=Student.GetList(10);
    return View("List");
}
```



TƯƠNG TÁC DỮ LIỆU TỪ CONTROLLER SANG VIEW

✓ Truyền bằng ViewModel (vd17)

- Có thể dùng ViewModel thay thế ViewBag, giả sử ta muốn truyền 2 list về View

```
namespace MVCDemo.Models
{
    1 reference
    public class Group
    {
        1 reference
        public IEnumerable<Student> Students { get; set; }
        1 reference
        public IEnumerable<Teacher> Teachers { get; set; }
    }
}

public ActionResult Index()
{
    var group = new Group {Students=Student.GetList(5), Teachers=Teacher.GetList(5)};
    return View("List", group);
}
```



RAZOR ENGINE

✓ Giới thiệu (vd18)

- Các phiên bản MVC3, MVC4, MVC5, ASP.net MVC có hỗ trợ một loại Engine mới đó là Razor View Engine, bản MVC trước đó và webform chỉ hỗ trợ aspx Engine.
- Mục đích của View Engine là sinh ra HTML bằng lập trình
- Những khai báo trong Razor” Các mã Razor, các thẻ html thuần, các HTML Helper. Thực chất Html Helper là các thư viện của Razor hỗ trợ lập trình MVC tạo ra các mã HTML.



RAZOR ENGINE

✓ Viết mã Razor

- Trong Razor có thể viết mã html thông thường như sau:

```
<div> xin chao! </div>
```

- Viết mã Razor có cấu trúc như mã C# được khai báo sau ký tự @
- Khối code được bao trong cặp @{...}

```
@ {  
    string say="Xin chao!";  
}
```

```
<h2>@say</h2>
```

- Chuỗi hằng ký tự nằm trong cặp "..."
- Các biến được khai báo bằng từ khóa var
- C# files có đuôi mở rộng .cshtml



RAZOR ENGINE

✓ Một số ví dụ về Razor

- Biến được khai báo bắt đầu từ khóa var (ASP.NET tự động xác định kiểu dữ liệu) hoặc kiểu dữ liệu.

```
@{
```

```
var hello = "Xin chào";  
var counter = 2;  
var day = DateTime.Today;
```

```
}
```



RAZOR ENGINE

✓ Loops và Arrays

■ Arrays

```
@{  
    string[] mylist = { "tav", "tavistu", "antv", "tva" };  
    int n = Array.IndexOf(mylist, "Kai") + 1;  
    int len = mylist.Length;  
    string x = mylist[3]; }  
<h3>Student</h3>  
@foreach (var person in mylist)  
{ <p>@person</p> }
```



RAZOR ENGINE

✓ Condition

▪ if

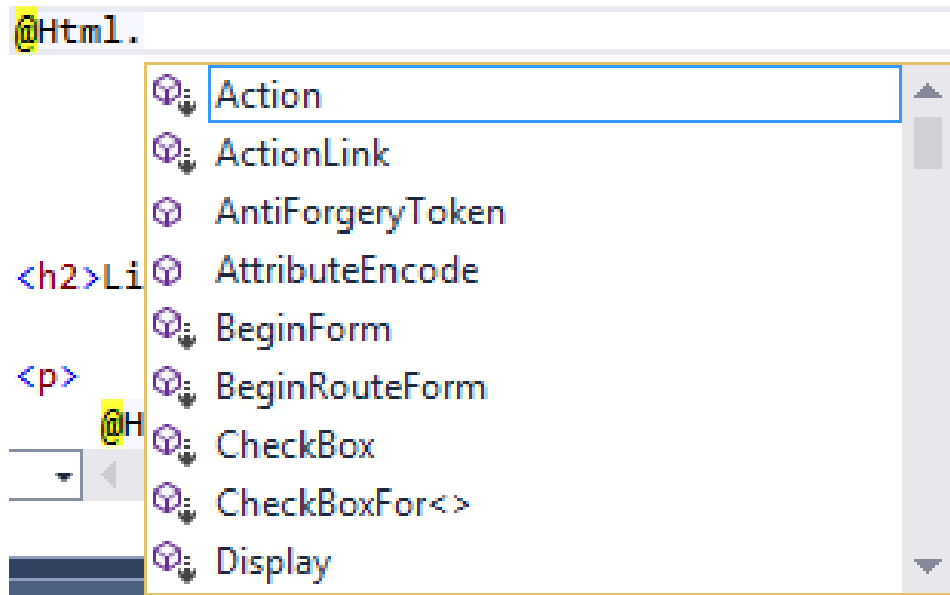
```
@{  
    var age = 20;  
}  
@if (age >= 50)  
{  
    <p>old</p>  
}  
else if (age > 10 && age < 30)  
{  
    <p>young</p>  
}  
else  
{  
    <p>kid</p>  
}
```



HTML HELPERS

✓ Giới thiệu

- HTML Helper bao gồm các phương thức giúp tạo các thuộc tính HTML trên view.



HTML HELPERS

✓ Giới thiệu

- HTML Helper Control dùng vào việc hiển thị dữ liệu được load từ CSDL rồi tự động sinh ra các mã HTML tương ứng
- Ví dụ control thông thường:
- `@Html.TextBox(“”)` - sinh ra thẻ input text tương ứng
- `@Html.DropDownList(“”)` – thẻ select
- Control hiển thị:
- `@Html.EditorFor(model => model.Name)` - Một textbox nhập nội dung name
- `@Html.LabelFor(model => model.Name)`- Một Label hiển thị nội dung
- `@Html.DisplayFor(model => model.Name)` - Một giá trị hiển thị trực tiếp



HTML HELPERS

✓ ActionLink (vd19)

- Gọi tới Action của Controller bất kỳ.

```
@Html.ActionLink("Thêm sinh viên", "Add", "Student");
```

```
@Html.ActionLink("Sửa sinh viên", "Edit", "Student", new { ID = 5 }, null);
```

- Gọi tới Action chứa html Attribute

```
@Html.ActionLink("Sửa sinh viên", "Edit", "Student", null, new { @style="color:red;" });
```



HTML HELPERS

✓ ActionLink (vd19)

- Gọi tới Action chứa parameter

```
@Html.ActionLink("Sửa sinh viên", "Edit", "Student", new { ID=6}, new { @style = "color:red;" });
```

- Gọi tới Action hiện tại

```
@Html.ActionLink("Details", "Details", new { id = item.UserName })
```

- ActionLink rendered event onclick

```
@Html.ActionLink("Sửa sinh viên", "Delete", null, new { onclick=" return confirm('Xóa dữ liệu?' );"});
```



HTML HELPERS

- ✓ **Đối tượng Helpers và Function trong Razor code**
 - Razor code hỗ trợ ngôn ngữ C#, vb.net ngay trong trang .cshtml hay .vbhtml
 - Razor code hỗ trợ khai báo namespace để tham chiếu tới các thư viện bên ngoài các trang .cshtml hay .vbhtml .
 - Razor cung cấp 2 đối tượng Helpers và Functions như là các hàm, phương thức xử lý mã html và C#



HTML HELPERS

✓ Đối tượng Function (vd20)

- Khai báo mã C# trong bộ từ khóa sau:

```
@functions{  
    // code C#  
}
```

- Razor Code không hỗ trợ đầy đủ các kiểu dữ liệu hay thư viện .net, tuy nhiên ta có thể code mã C# để xây dựng **đối tượng, phương thức** trực tiếp trong đó.

```
@functions{  
    string sayHello(string name)  
    {  
        return "Hello " + name;  
    }  
}  
@sayHello("TAV");
```



HTML HELPERS

✓ Đối tượng Function (vd21)

- Xây dựng đối tượng, phương thức trực tiếp trong đó.

```
@functions{
    public class Student
    {
        public int ID { get; set; }
        public string Name { get; set; }
        public int Age { get; set; }

        public static IEnumerable<Student> GetList(int Count)
        {
            var st = new List<Student>();
            for (int i = 1; i <= Count; i++)
            {
                st.Add(new Student { ID = i,
                    Name = "Student" + i.ToString(), Age = 20 });
            }
            return st;
        }
    }
}
```

```
<table>
    <tr><td>Name</td><td>Age</td></tr>
    @foreach (var item in Student.GetList(5))
    {
        <tr>
            <td>@item.Name</td>
            <td>@item.Age</td>
        </tr>
    }
</table>
```



HTML HELPERS

✓ Đối tượng Function (vd21)

- Tại ví dụ vd21, ta có thể khai báo Student trong Models với namespace MVCDemo.Models, vậy ta có thể sử dụng ngay lớp đó với

@using MVCDemo.Models;

```
@using MVCDemo.Models;

<table>
    <tr><td>Name</td><td>Age</td></tr>
    @foreach (var item in Student.GetList(5))
    {
        <tr>
            <td>@item.Name</td>
            <td>@item.Age</td>
        </tr>
    }
</table>
```



HTML HELPERS

✓ Đối tượng **Function**

- Có thể hiểu rằng chúng ta có thể tham chiếu tới cả những lớp bên ngoài ứng dụng (các lớp trong thư viện dll động)
- Như vậy các nhà phát triển có thể xây dựng kiến trúc phần mềm riêng cho mình dựa vào MVC .net



HTML HELPER XỬ LÝ NỘI DUNG - TEMPLATE

✓ **Html.Partial**

- `Html.Partial`, `Html.RenderPartial` dùng để gọi một `PartialView`.
- `PartialView` là một loại thẻ đặc biệt, chứa các thẻ `html` thuần hoặc các thẻ `html` chứa `model` hiển thị dữ liệu.
- Cấu trúc của `HTML.Partial`

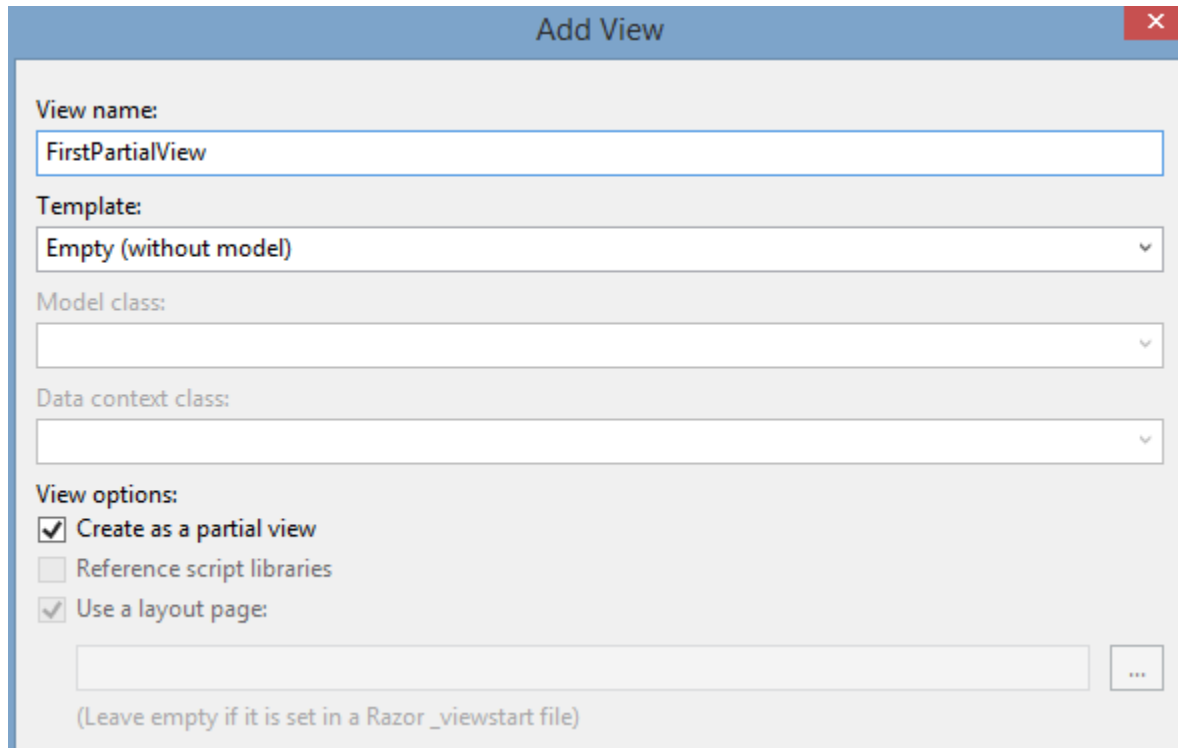
```
@Html.Partial("PartialView")  
@Html.Partial("PartialView", Model)
```



HTML HELPER XỬ LÝ NỘI DUNG - TEMPLATE

✓ Tạo Partial

- Trong Views/Login chọn
- Add/View



View name:
FirstPartialView

Template:
Empty (without model)

Model class:

Data context class:

View options:
☒ Create as a partial view
☐ Reference script libraries
☒ Use a layout page:

(Leave empty if it is set in a Razor _viewstart file)

HTML HELPER XỬ LÝ NỘI DUNG - TEMPLATE

✓ Tạo Partial

- Nhập nội dung cho FirstPartialView

```
<h1>Xin chào!</h1>
```

- Trong file Login.cshtml

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

```
@Html.Partial("FirstPartialView");
```



HTML HELPER XỬ LÝ NỘI DUNG - TEMPLATE

✓ Tạo Partial (vd23)

- Có thể truyền vào PartialView nội dung theo một Model nào đó theo cú pháp

```
@Html.Partial("PartialView", Model)
```

- Ví dụ: Trong Login.cshtml

```
@Html.Partial("FirstPartialView", Student.GetList(5));
```

- Trong FirstPartialView.cshtml

```
@model IEnumerable<MVCDemo.Models.Student>
<h1>Xin chào!</h1>

<table>
    <tr><td>Name</td><td>Age</td></tr>
    @foreach (var item in Model)
    {
        <tr> <td>@item.Name</td><td>@item.Age</td></tr>
    }
</table>
```



HTML HELPER XỬ LÝ NỘI DUNG - TEMPLATE

✓ Tạo Partial

- Có thể đặt PartialView cùng với các View khác hoặc để sang một folder khác

```
@Html.Partial("/Views/MyPartialsFirstPartialView", Student.GetList(5));
```



HTML HELPER XỬ LÝ NỘI DUNG - TEMPLATE



RenderPartial



HTML HELPER XỬ LÝ NỘI DUNG - TEMPLATE



Html.PartialView



HTML HELPER XỬ LÝ NỘI DUNG - TEMPLATE

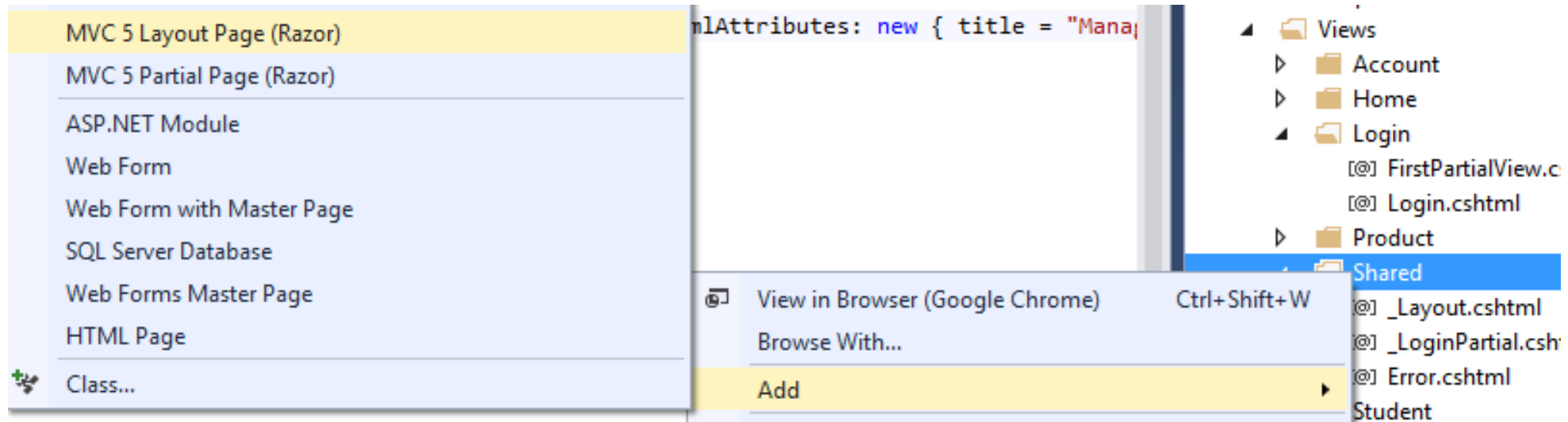
✓ **Html.Action và Html.RenderAction**



LAYOUT

✓ Layout trong MVC

- Layout là các vùng được ngăn ra trong các Template. Các vùng đó có thể là header, footer, left, right,...
- Layout trong MVC đặt trong Views/Shared
- Tạo layout riêng:



LAYOUT

✓ Layout trong MVC

- Trong Layout , các thành phần thẻ bắt buộc như html, body
- @RenderBody() là nơi đặt nội dung kế thừa của các trang khác
- Một View muốn sử dụng layout thì khai báo như sau:

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



LAYOUT

✓ Layout trong MVC

- Layout là một template hỗ trợ hiển thị cố định, các trang web khác kế thừa nội dung.
- Các đối tượng xử lý layout trong Razor: `RenderBody`, `RenderPage`, `RenderSection`, `LayoutPage`
- `RenderBody`: Nội dung trang kế thừa layout sẽ hiển thị đúng vào vị trí `RenderBody`
- `RenderPage` dùng để gọi nội dung từ một View khác
- `RenderSection` cần được khai báo trong các trang kế thừa



LAYOUT

✓ RenderSection (vd24)

- RenderSection cần được khai báo trong các trang kế thừa
- Tại _Layout.cshtml

```
<div class="container body-content">
    @RenderSection("hello",required:true);
    @RenderBody()
    <hr />
    <footer>
        <p>&copy; @DateTime.Now.Year - My ASP.
    </footer>
</div>
```

- Tại trang kế thừa Login.cshtml

```
@section hello{
    @foreach (var item in Student.GetList(5))
    {
        <tr> <td>@item.Name</td><td>@item.Age</td></tr>
    }
}
```



LAYOUT

✓ RenderPage(vd25)

- Tạo folder Common trong Views
- Tạo Empty Page với tên _Footer.cshtml

```
@using System;
<footer>
    <p>&copy; @DateTime.Now.Year - My ASP.NET Application</p>
    @{
        DateTime value = new DateTime(2016, 3, 20);
    }
    @value
</footer>
```

- Trong _Layout.cshtml

```
<div class="container body-content">
    @RenderSection("hello",required:true);
    @RenderBody()
    <hr />
    @RenderPage("~/Views/Common/_Footer.cshtml");
</div>
```

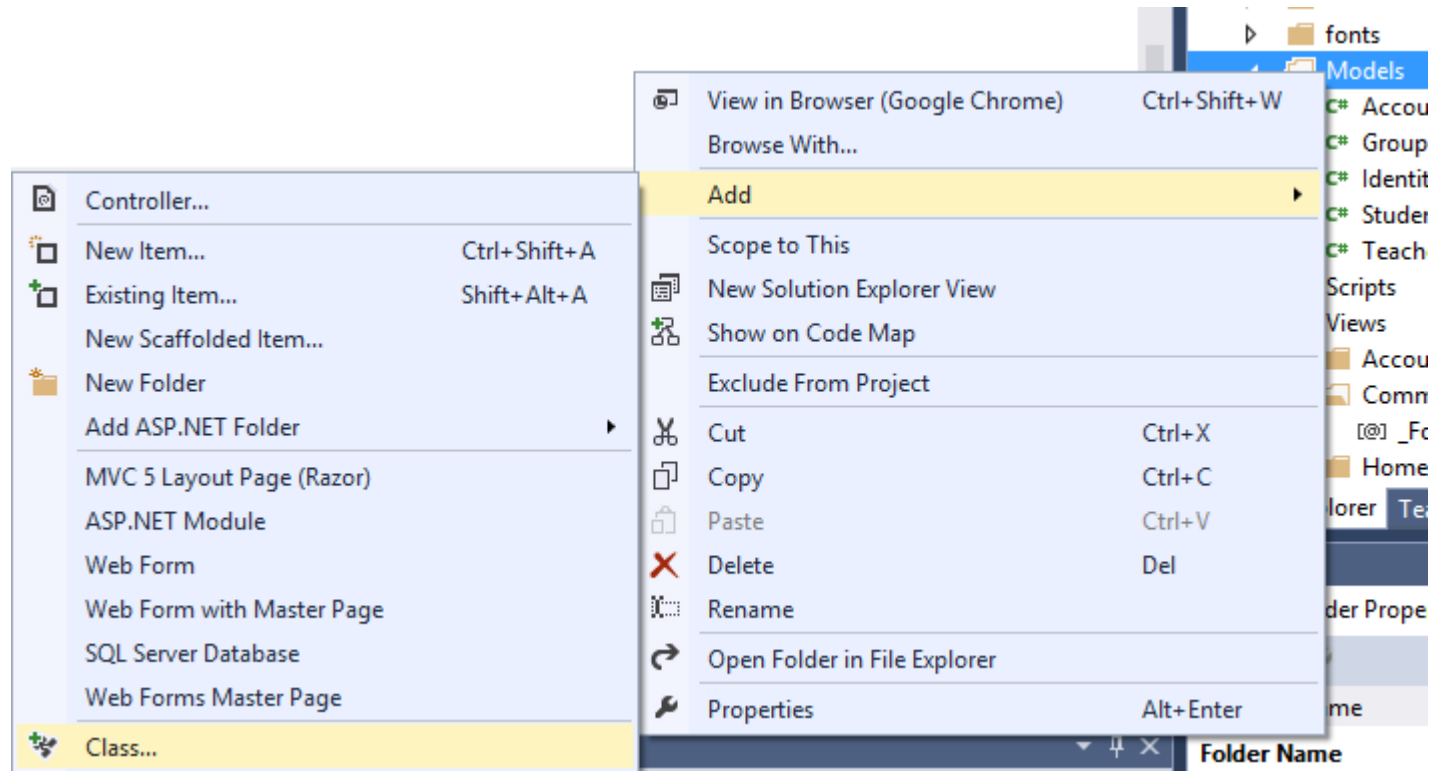


Lập trình WEB Model

GIỚI THIỆU

✓ Xây dựng Model

- Tạo mới Class Product.cs trong thư mục Model theo hình sau :



GIỚI THIỆU

✓ Xây dựng Model

- Tạo mới Class Product.cs trong thư mục Model theo hình sau :

```
namespace MVCDemo.Models
{
    Oreferences
    public class Product
    {
        Oreferences
        public int ID { get; set; }
        Oreferences
        public string Name { get; set; }
    }
}
```

- Đây là Class dùng để lưu trữ thông tin cũng như xử lý dữ liệu .Nên ta sẽ tạo tự động 1 bộ dữ liệu lưu trên bộ nhớ . Lớp Product.cs sẽ ánh xạ tạo 1 bảng dưới CSDL.



GIỚI THIỆU

✓ Xây dựng Model

- Tạo mới Class Product.cs trong thư mục Model theo hình sau :
- Đây là Class dùng để lưu trữ thông tin cũng như xử lý dữ liệu .Nên ta sẽ tạo tự động 1 bộ dữ liệu lưu trên bộ nhớ . Lớp Product.cs sẽ ánh xạ tạo 1 bảng dưới CSDL.

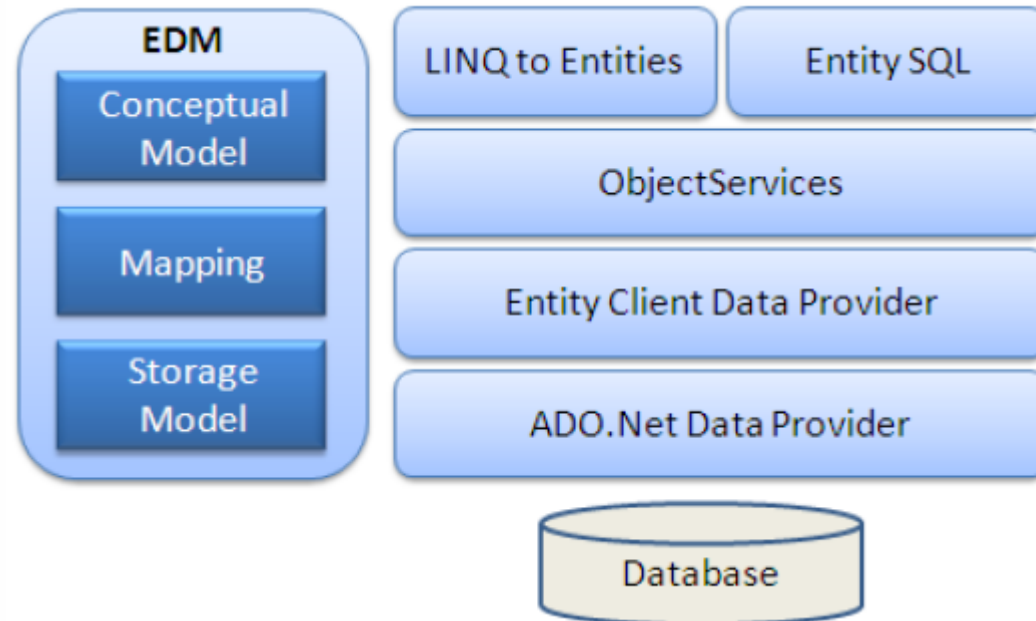
```
namespace MVCDemo.Models
{
    3 references
    public class Product
    {
        1 reference
        public int ID { get; set; }
        1 reference
        public string Name { get; set; }
        0 references
        public static IEnumerable<Product> GetList(int Count)
        {
            var Products = new List<Product>();
            for (int i = 1; i <= Count; i++)
            {
                Products.Add(new Product { ID = i, Name = "Product" + i.ToString() });
            }
            return Products;
        }
    }
}
```



ENTITYFRAMEWORK (EF)

✓ Giới thiệu

- Entity framework là Object/Relational Mapping (O/RM) framework



ENTITYFRAMEWORK (EF)

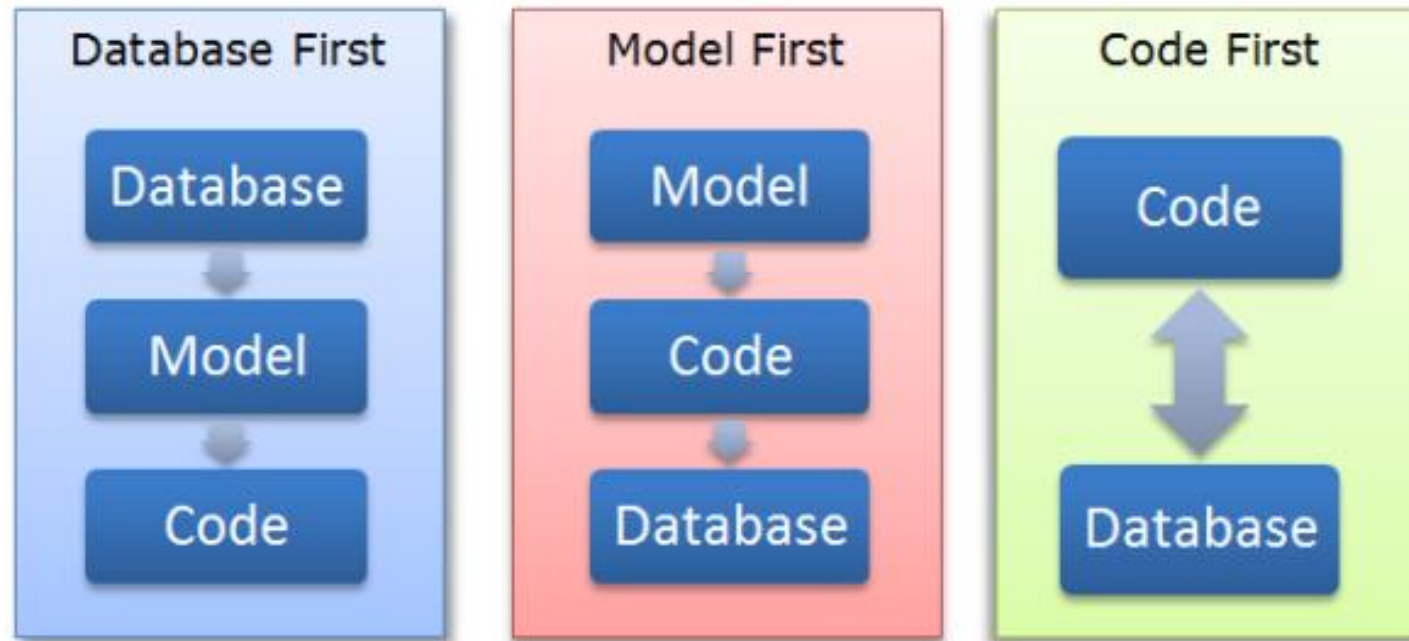
✓ Giới thiệu

- Khi sử dụng EF chúng ta có 3 lựa chọn đó là:
 - Database First (thiết kế database trước) , sau đó dùng VS tạo lớp entity .Kỹ thuật này giống với kỹ thuật tạo lớp Entity dùng Linq to Sql .
 - Model first : Dùng VS tạo lớp Entity , sau đó từ Entity sinh ra database
 - Code first : Tạo các lớp ứng dụng , và lớp Entity như các đối tượng C# (vb.net) thông thường. Khi ứng dụng chạy sẽ tự động tạo ra 1 CSDL tương ứng với các lớp .



ENTITYFRAMEWORK (EF)

✓ Giới thiệu



ENTITYFRAMEWORK (EF)

Code first

- Cài đặt EF vào Visual Studio 2013 :

<https://www.microsoft.com/en-us/download/details.aspx?id=40762>

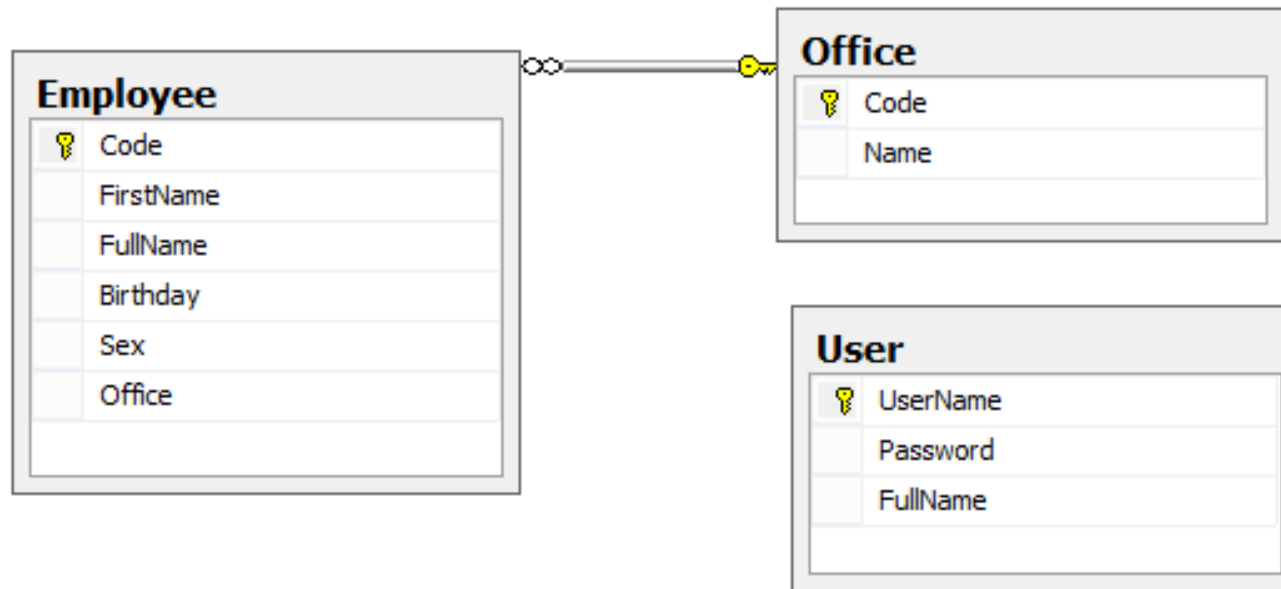
- EFTools6.1.3ForVS2013.msi



ENTITYFRAMEWORK (EF)

✓ Code first (vd26)

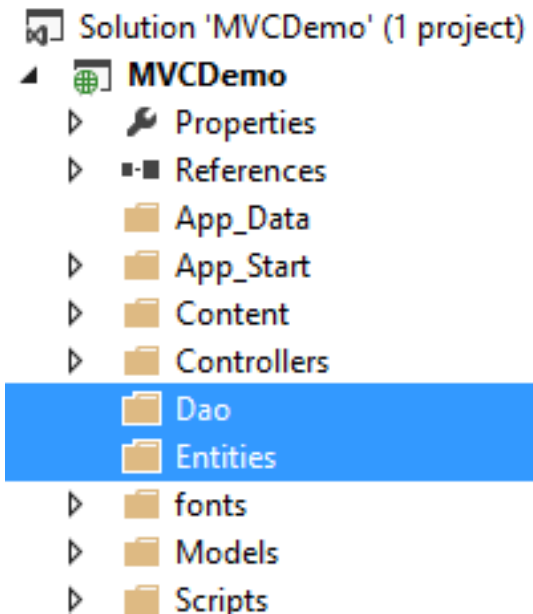
- Giả sử ta có một mô hình CSDL



ENTITYFRAMEWORK (EF)

✓ Code first

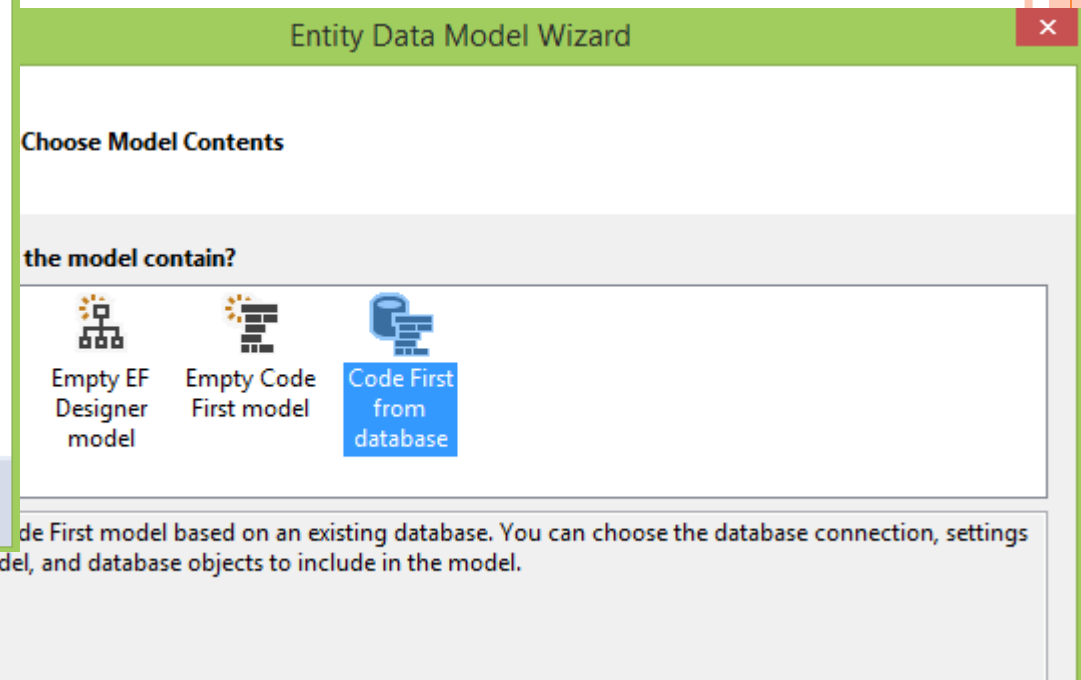
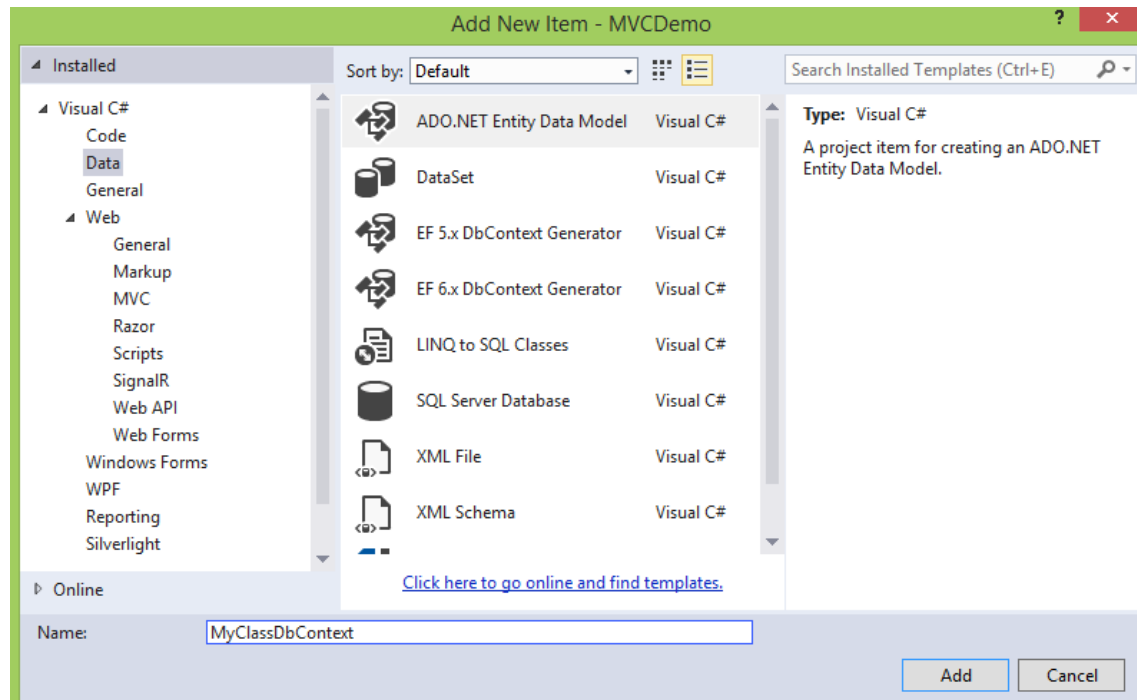
- Tạo thêm các folder Dao và Entites trong Project



ENTITYFRAMEWORK (EF)

✓ Code first

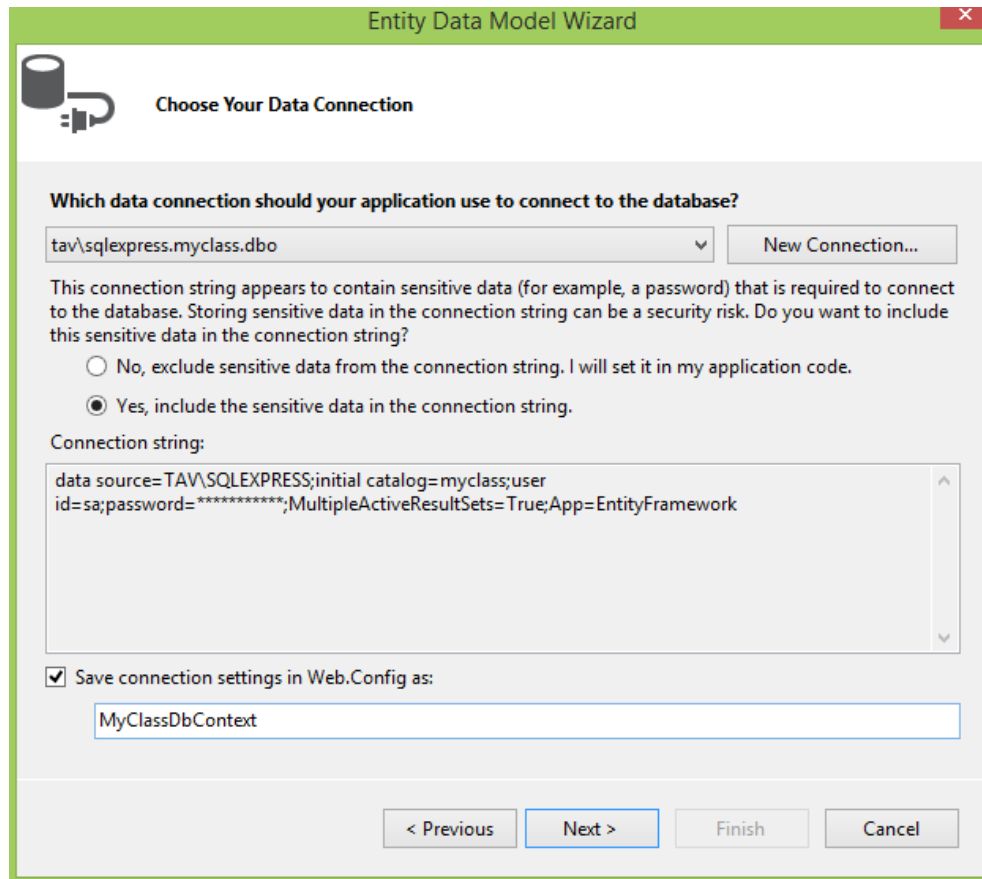
- Chuột phải vào Entites, Add->New Item..



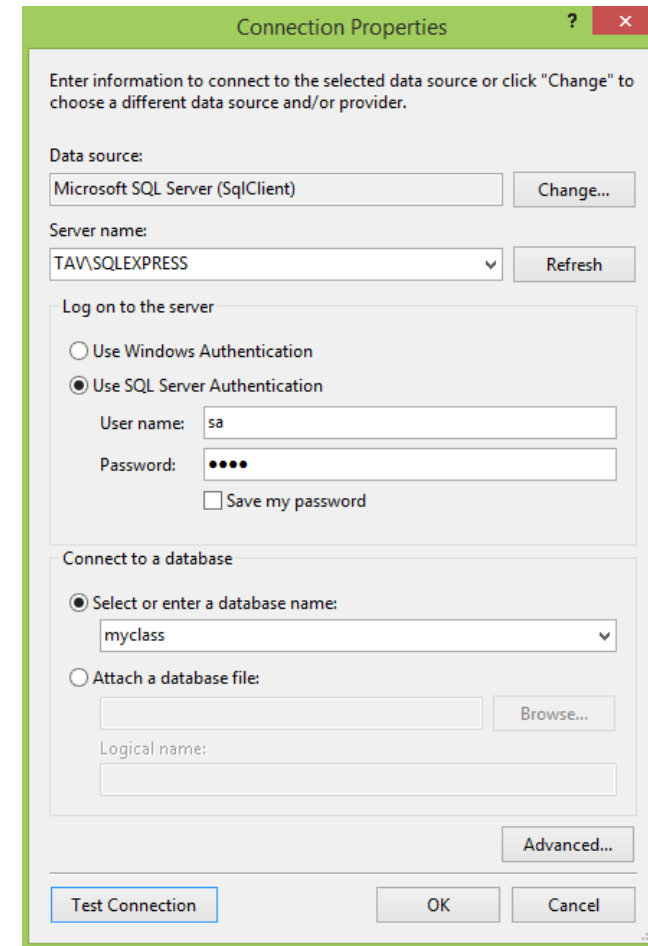
ENTITYFRAMEWORK (EF)

✓ Code first

- Chọn kết nối đến CSDL cần thiết



The Entity Data Model Wizard is shown with the title "Entity Data Model Wizard". The main heading is "Choose Your Data Connection". Below this, it asks "Which data connection should your application use to connect to the database?". A dropdown menu shows "tav\sqlexpress.myclass.dbo". To the right is a "New Connection..." button. Below the dropdown, a warning message states: "This connection string appears to contain sensitive data (for example, a password) that is required to connect to the database. Storing sensitive data in the connection string can be a security risk. Do you want to include this sensitive data in the connection string?". There are two radio buttons: "No, exclude sensitive data from the connection string. I will set it in my application code." and "Yes, include the sensitive data in the connection string." The second option is selected. Below this, the "Connection string:" is displayed in a text box with the value: "data source= TAV\SQLEXPRESS;initial catalog=myclass;user id=sa;password=*****;MultipleActiveResultSets=True;App=EntityFramework". At the bottom, there is a checkbox "Save connection settings in Web.Config as:" which is checked. Next to it is a text box containing "MyClassDbContext". At the very bottom are buttons: "< Previous", "Next >", "Finish", and "Cancel".

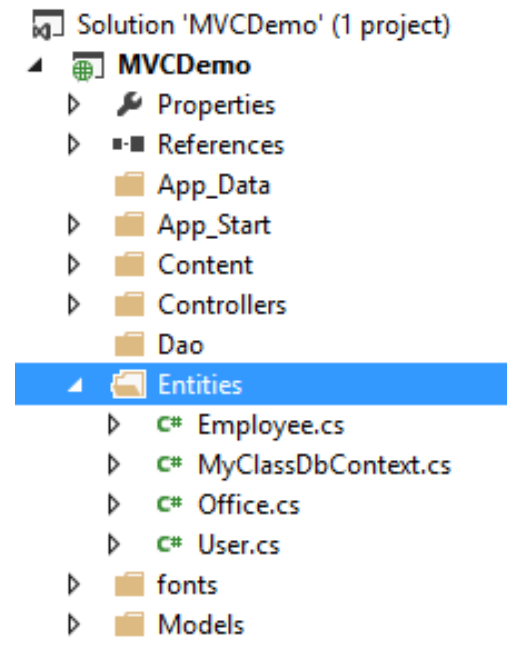
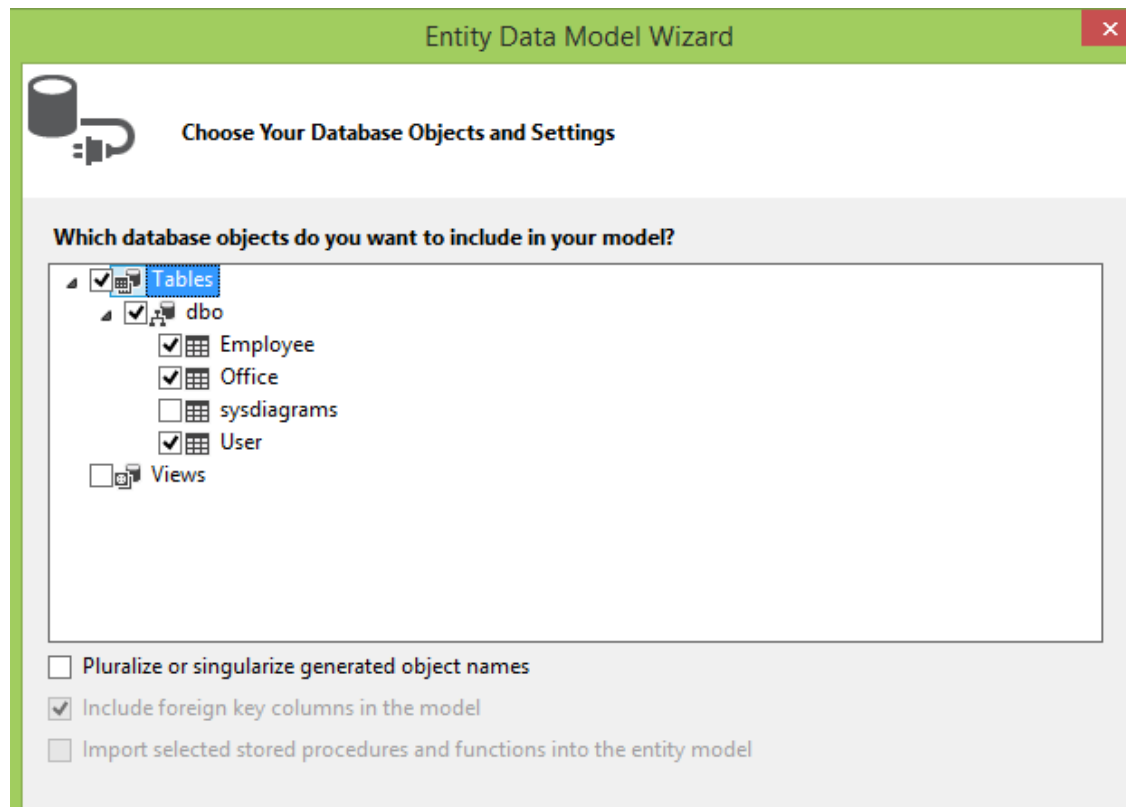


The Connection Properties dialog box is shown with the title "Connection Properties". It contains the following fields and options: "Data source:" with a dropdown menu showing "Microsoft SQL Server (SqlClient)" and a "Change..." button; "Server name:" with a dropdown menu showing "TAV\SQLEXPRESS" and a "Refresh" button; "Log on to the server" section with two radio buttons: "Use Windows Authentication" and "Use SQL Server Authentication" (which is selected). Below these are "User name:" (sa) and "Password:" (masked with dots) fields, and a "Save my password" checkbox. The "Connect to a database" section has two radio buttons: "Select or enter a database name:" (selected) and "Attach a database file:". Below the first radio button is a dropdown menu showing "myclass". Below the second radio button is a "Browse..." button. At the bottom, there is a "Logical name:" field. At the very bottom are buttons: "Test Connection", "OK", and "Cancel".

ENTITYFRAMEWORK (EF)

✓ Code first

- Chọn các bảng



ENTITYFRAMEWORK (EF)

✓ Code first

- Trong folder Dao tạo UserDao

```
public class UserDao
{
    MyClassDbContext db;
    O references
    public UserDao()
    {
        db = new MyClassDbContext();
    }
    O references
    public bool Login(string UserName, string Password)
    {
        var rs = db.User.Count(x => x.UserName == UserName && x.Password == Password);
        if (rs > 0)
            return true;
        else
            return false;
    }
}
```



ENTITYFRAMEWORK (EF)

✓ Code first (vd26)

- Sửa lại LoginController như sau

```
[HttpPost]
//references
public ActionResult LoginAction(Account acc)
{
    UserDao dao = new UserDao();
    bool check = dao.Login(acc.Name, acc.Password);
    if (check)
    {
        Session["UserName"] = acc.Name;
        return RedirectToAction("Index", "Home");
    }
    else
        return View("Login");
}
```



ENTITYFRAMEWORK (EF)

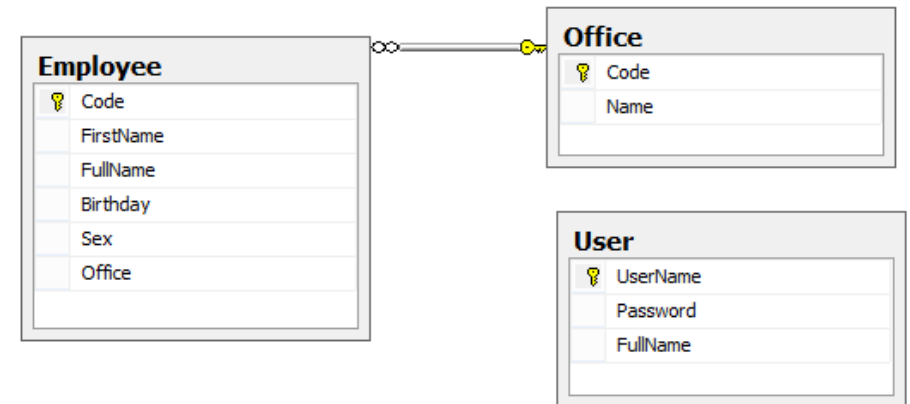
✓ DbContext

- Class trong EF được dẫn xuất từ class System.Data.Entity.DbContext
- Là cầu nối giữa entity classes và database

```
public partial class MyClassDbContext : DbContext
{
    1 reference
    public MyClassDbContext()
        : base("name=MyClassDbContext")
    {
    }

    0 references
    public virtual DbSet<Employee> Employee { get; set; }
    0 references
    public virtual DbSet<Office> Office { get; set; }
    1 reference
    public virtual DbSet<User> User { get; set; }

    0 references
    protected override void OnModelCreating(DbModelBuilder modelBuilder)
```



- Entities
 - ▶ C# Employee.cs
 - ▶ C# MyClassDbContext.cs
 - ▶ C# Office.cs
 - ▶ C# User.cs

ENTITYFRAMEWORK (EF)

✓ DbContext

- EntitySet: DbContext chứa tập các thực thể (DbSet<TEntity>) đối với tất cả các thực thể ánh xạ đến DB tables, ở đây là Employee, Office, User

```
public partial class MyClassDbContext : DbContext
{
    1 reference
    public MyClassDbContext()
        : base("name=MyClassDbContext")
    {
    }

    0 references
    public virtual DbSet<Employee> Employee { get; set; }
    0 references
    public virtual DbSet<Office> Office { get; set; }
    1 reference
    public virtual DbSet<User> User { get; set; }

    0 references
    protected override void OnModelCreating(DbModelBuilder modelBuilder)
```



ENTITYFRAMEWORK (EF)

✓ DbContext

- Querying: DbContext chuyển LINQ-to-Entities queries sang SQL query và gửi chúng đến database.
- Change Tracking: Theo dõi các thay đổi xuất hiện trong thực thể sau khi truy vấn từ database.
- Persisting Data: Thực hiện các thao tác Insert, Update and Delete trên database, dựa trên trạng thái thực thể.
- Caching: Lưu trữ các thực thể được truy xuất trong thời gian sống của context class.
- Manage Relationship: DbContext quản lý các quan hệ.
- Object Materialization: DbContext thực thể hóa table data thành entity objects



ENTITYFRAMEWORK (EF)

✓ DbSet

- Là thể hiện cho tập các thực thể ánh xạ tới CSDL
- Hỗ trợ các thao tác với dữ liệu:

+Insert

+Update

+Delete

+Select

```
0 references
public virtual DbSet<Employee> Employee { get; set; }
0 references
public virtual DbSet<Office> Office { get; set; }
1 reference
public virtual DbSet<User> User { get; set; }
```



TƯƠNG TÁC DỮ LIỆU VỚI LINQ

✓ Giới thiệu [vd27]

- LINQ to SQL là một phiên bản hiện thực hóa của O/RM (object relational mapping) có bên trong .NET Framework bản “Orcas” (nay là .NET 3.5), nó cho phép bạn mô hình hóa một cơ sở dữ liệu dùng các lớp .NET



TƯƠNG TÁC DỮ LIỆU VỚI LINQ

✓ Obtaining a Data Source [vd27]

- Trong Linq, sử dụng từ khóa from ở đầu tiên để chỉ ra nguồn dữ liệu

```
public IQueryable<Office> ListOffice()
{
    var res = (from s in db.Office select s);
    return res;
}
```



TƯƠNG TÁC DỮ LIỆU VỚI LINQ

✓ Filtering [vd27]

- Trong Linq, sử dụng từ khóa where, sau where là các biểu thức điều kiện, có thể dùng && (AND) và || (OR)

```
public IQueryable<Office> ListOffice()
{
    var res = (from s in db.Office
                where s.Code>2 && s.Name.Length>2
                select s);
    return res;
}
```

```
public IQueryable<Office> ListOffice()
{
    var res = (from s in db.Office
                where s.Code>2 || s.Name.Length>2
                select s);
    return res;
}
```



TƯƠNG TÁC DỮ LIỆU VỚI LINQ

✓ Ordering [vd27]

- Sử dụng từ khóa **orderby** dùng để sắp xếp dữ liệu theo những thứ tự mặc định cho từng loại dữ liệu. Ví dụ, dạng chuỗi kí tự sẽ là sắp xếp theo thứ tự từ A->Z với từ khóa **ascending** , theo chiều ngược lại với từ khóa **descending**

```
public IQueryable<Office> ListOffice()
{
    var res = (from s in db.Office
                where s.Code>2 || s.Name.Length>2
                orderby s.Name descending
                select s);
    return res;
}
```



ENTITYFRAMEWORK (EF)

✓ DbSet [vd27]

- Các phương thức cơ bản của DbSet:
- Add: Thêm một kiểu thực thể

```
public int InsertOffice(string Name)
{
    Office office = new Office();
    office.Name = Name;
    db.Office.Add(office);
    db.SaveChanges();
    return office.Code;
}
```



ENTITYFRAMEWORK (EF)

✓ DbSet [vd27]

- Các phương thức cơ bản của DbSet:
- Update dữ liệu:

```
public void UpdateOffice(Office officeTmp)
{
    Office office = db.Office.Find(officeTmp.Code);
    if(office!=null)
    {
        office.Name = officeTmp.Name;
        db.SaveChanges();
    }
}
```



ENTITYFRAMEWORK (EF)

✓ DbSet [vd27]

- Các phương thức cơ bản của DbSet:
- Delete dữ liệu:

```
public void DeleteOffice(Office officeTmp)
{
    Office office = db.Office.Find(officeTmp.Code);
    if (office != null)
    {
        db.Office.Remove(office);
        db.SaveChanges();
    }
}
```



HỎI ĐÁP