**TRƯỜNG ĐẠI HỌC MỞ THÀNH PHỐ HỒ CHÍ MINH**

**KHOA CÔNG NGHỆ THÔNG TIN**

**-o0o-**



**BÀI TẬP LỚN MÔN LẬP TRÌNH WEB**

**ĐỀ BÀI:**

Building Dynamic Data Applications:

Introducing ASP.NET Dynamic Data.

Building a Dynamic Data Application.

Working with Dynamic Data Templates.

Building ASP.NET MVC Applications:

Introducing the ASP.NET MVC Framework.

Creating Your First ASP.NET MVC Application.

Building an MVC Page.

Accepting Form Input.

Sinh viên thực hiện:

**1551010145 - Nguyễn Thị Bích Tuyền**

Giảng viên hướng dẫn

**ThS. Tô Oai Hùng**

**Mục lục:**

[1. Tổng quan về ASP.NET Dynamic Data: 3](#_Toc529632496)

[1.1. Giới thiệu về ASP.NET Dynamic Data: 3](#_Toc529632497)

[1.2. Cách tạo 1 chương trình Dynamic Data. 3](#_Toc529632498)

[1.3. Làm việc với mô hình Dynamic Data. 7](#_Toc529632499)

[Making Shared Template Changes 8](#_Toc529632500)

[Creating Type and Entity-Specific Templates 8](#_Toc529632501)

[2. Tổng quan về ASP.NET MVC: 10](#_Toc529632502)

[2.1. Giới thiệu về ASP.NET MVC Framework: 10](#_Toc529632503)

[2.1.1. Tổng quan về mô hình MVC: 10](#_Toc529632504)

[2.1.2. Tổng quan về lập trình ASP.NET MVC: 11](#_Toc529632505)

[2.2. Cách tạo 1 chương trình ASP.NET MVC: 12](#_Toc529632506)

[2.3. Chương trình ASP.NET MVC mặc định: 15](#_Toc529632507)

[2.3.1. Cấu trúc chương trình: 15](#_Toc529632508)

[2.3.2. Chạy chương trình: 15](#_Toc529632509)

[2.3.3. Giải thích chương trình: 17](#_Toc529632510)

[2.4. Accept Input: 19](#_Toc529632511)

[2.5. Demo chương trình ASP.NET MVC: 21](#_Toc529632512)

# Tổng quan về ASP.NET Dynamic Data:

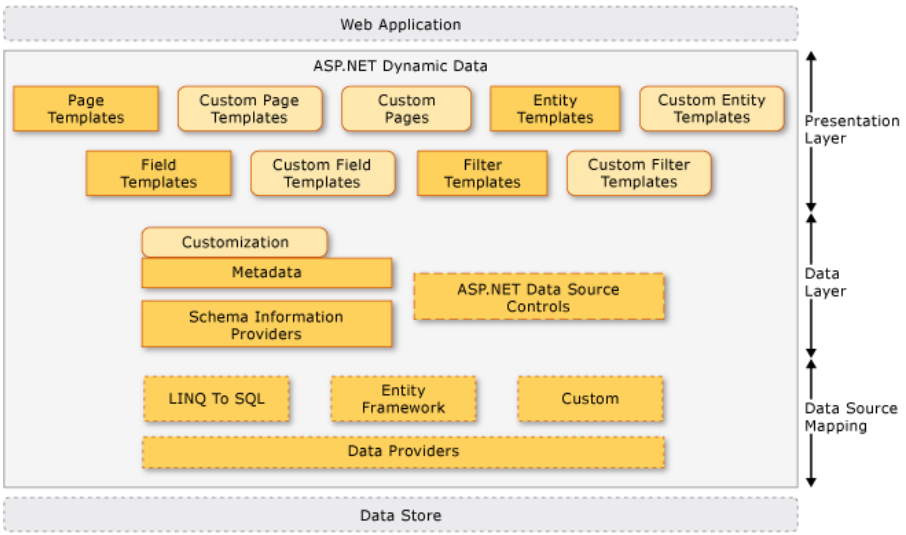
## Giới thiệu về ASP.NET Dynamic Data:

ASP.NET Dynamic Data cung cấp một Framework cho phép chúng ta nhanh chóng xây dựng một chức năng ứng dụng driver-data, dựa trên LINQ to SQL hay Entity Framework.

Dựa trên cấu trúc của CSDL mà Dynamic Data Framework (DDF) sẽ tạo nên các trang web cho phép người dùng xem/chèn/xóa/sửa dữ liệu.

Nhiều tính linh hoạt cho các DetailsView, FormView, GridView, ListView trong kiểm tra tính hợp lệ của dữ liệu, hoặc chỉnh sửa lại các mẫu để thay đổi cách hiển thị dữ liệu.

**Cấu trúc Dynamic Data**



**Presentation Layer**: chứa các phần tử được sử dụng để tạo giao diện người dùng, hiển thị và chỉnh sửa các thực thể dữ liệu.

**Data Layer:** chứa mô hình dữ liệu đại diện cho các thực thể cơ sở dữ liệu như các kiểu CLR.

**Data Source Mapping:** chứa các phần tử là một phần của CLR nhưng được sử dụng bởi dữ liệu động. Các phần tử này là các công nghệ như [LINQ to SQL](http://go.microsoft.com/fwlink/?LinkId=177401) và [ADO.NET Entity Framework](http://go.microsoft.com/fwlink/?LinkId=177400) được sử dụng để tạo ra các mô hình dữ liệu.

**Thành phần lớp Presentation**

Phần này cung cấp tổng quan về các yếu tố Dynamic Data được sử dụng để tạo giao diện người dùng, hiển thị và chỉnh sửa các thực thể dữ liệu.

**Page Templates (Mẫu trang)**

Các mẫu trang là các trang web hiển thị dữ liệu từ bất kỳ bảng nào trong cơ sở dữ liệu. Các mẫu trang sẽ hiển thị dữ liệu theo các chế độ sau: chế độ xem danh sách, chế độ xem chi tiết và chế độ xem chỉnh sửa. Mặc định, là chế độ xem danh sách.

Khi bạn tạo một dự án Web dữ liệu động, Visual Studio thêm một thư mục DynamicData \ PageTemplates vào dự án. Thư mục này chứa các mẫu trang mặc định. Bạn có thể sửa đổi các mẫu mặc định hoặc tạo mẫu mới để xác định cách hiển thị bảng.

**Entity Templates (Mẫu thực thể)**

Mẫu thực thể cho phép bạn tùy chỉnh giao diện người dùng cho toàn bộ thực thể dữ liệu, chẳng hạn như hàng hoặc bảng. Điều này mang lại sự linh hoạt hơn việc tùy chỉnh các trường dữ liệu riêng lẻ và hữu ích nếu bạn muốn tạo giao diện người dùng tùy chỉnh và áp dụng nó cho các mẫu trang mặc định hoặc tùy chỉnh.

Khi bạn tạo một dự án web dữ liệu động, Visual Studio thêm một thư mục DynamicData \ EntityTemplates vào dự án. Thư mục này chứa các mẫu thực thể mặc định. Có các mẫu để hiển thị, chỉnh sửa và chèn hoạt động. Các mẫu thực thể mặc định hiển thị dữ liệu trong hai cột, sử dụng nhãn cho tên trường và điều khiển thích hợp cho giá trị trường. Bạn có thể sửa đổi các mẫu thực thể mặc định để thay đổi giao diện và hành vi của các hàng dữ liệu cho toàn bộ trang web.

Các mẫu thực thể được bao gồm trong các mẫu trang Details.aspx, Insert.aspx và Update.aspx và bất kỳ trang tùy chỉnh nào. Các mẫu thực thể lần lượt sử dụng các mẫu trường để hiển thị dữ liệu thực tế.

**Field Templates (Mẫu trường)**

Các mẫu trường là các điều khiển dùng để hiển thị giao diện người dùng cho các trường dữ liệu riêng lẻ. Theo mặc định, dữ liệu động chọn mẫu trường dựa trên kiểu dữ liệu của trường đang được hiển thị.

Ví dụ: để hiển thị dữ liệu Boolean, Dữ liệu động sử dụng mẫu trường Boolean; để hiển thị dữ liệu văn bản, Dữ liệu động sử dụng mẫu trường văn bản...

Thường có một mẫu trường để hiển thị dữ liệu và một mẫu khác cho phép người dùng nhập hoặc chỉnh sửa giá trị trường.

Khi bạn tạo một dự án web dữ liệu động, Visual Studio thêm một thư mục DynamicData \ FieldTemplates vào dự án. Thư mục này chứa các mẫu trường mặc định. Bạn có thể sửa đổi các mẫu trường mặc định hoặc tạo mẫu mới để chỉ định cách các trường dữ liệu riêng lẻ được hiển thị.

**Filter Templates (Mẫu bộ lọc)**

Các mẫu bộ lọc là các điều khiển hiển thị giao diện người dùng để lọc dữ liệu, cho phép người dùng chọn các hàng trong bảng để hiển thị dựa trên giá trị cột.

Ví dụ: bạn có thể hiển thị tất cả các hàng chứa một danh mục nhất định. Theo mặc định, dữ liệu động cung cấp các mẫu bộ lọc Boolean, khóa ngoài và liệt kê.

Dữ liệu động cho phép bạn sử dụng đánh dấu trang để tạo giao diện người dùng cho lọc dữ liệu trong một cột. Khi bạn tạo một dự án Web dữ liệu động, Visual Studio thêm một thư mục DynamicData \ FilterTemplates vào dự án. Thư mục này chứa các mẫu bộ lọc mặc định.

Nếu mẫu bộ lọc không tồn tại cho loại cột đó, bạn có thể tạo một mẫu tùy chỉnh và hướng dẫn Dữ liệu động sử dụng nó khi tạo giao diện người dùng. Bạn cũng có thể tạo đánh dấu trang để cho phép dữ liệu động tự động tạo giao diện người dùng cho bất kỳ loại cột nào mà mẫu bộ lọc mặc định tồn tại.

## Yếu tố lớp dữ liệu

ASP.NET Dynamic hỗ trợ các mô hình dữ liệu [LINQ to SQL](http://go.microsoft.com/fwlink/?LinkId=177400) và [ADO.NET Entity Framework](http://go.microsoft.com/fwlink/?LinkId=177400) . Dữ liệu động sử dụng các loại này để truy vấn cơ sở dữ liệu và tạo, đọc, cập nhật và xóa dữ liệu (nghĩa là, để thực hiện các hoạt động CRUD). Mô hình dữ liệu cung cấp một cách dễ dàng để tích hợp xác thực dữ liệu tùy chỉnh và các quy tắc logic nghiệp vụ.

Vào thời gian chạy, Dữ liệu động sẽ tự động trích xuất thông tin siêu dữ liệu từ mô hình dữ liệu, chẳng hạn như thuộc tính trường dữ liệu. Từ thông tin này, nó sẽ hướng dẫn cách tạo giao diện người dùng để hiển thị và chỉnh sửa dữ liệu.

Khi bạn tạo một dự án mới trong Visual Studio, bạn có thể chọn kiểu mô hình được dự án sử dụng. Giàn giáo dữ liệu động chỉ có thể hỗ trợ một kiểu dữ liệu trong cùng một dự án.

## Tùy chọn cho Cách sử dụng dữ liệu động

Dữ liệu động hỗ trợ các tùy chọn thiết kế sau:

* Tạo một trang web bằng cách sử dụng giàn giáo. Bạn có thể tạo một ứng dụng Web cơ bản sử dụng ASP.NET Dynamic Data đòi hỏi ít hoặc không có mã hóa. Khi giàn giáo được kích hoạt, ASP.NET phân tích mô hình dữ liệu và tạo ra các trang Web dựa trên thông tin siêu dữ liệu động. Các trang web được tạo tự động này cung cấp khả năng CRUD cho mỗi bảng. Để biết thêm thông tin, hãy xem [Cách thực hiện: Thêm hành vi động vào kiểm soát ràng buộc dữ liệu](https://msdn.microsoft.com/en-us/library/cc488562.aspx) .
* Bật hành vi động cho các điều khiển ràng buộc dữ liệu riêng lẻ (chẳng hạn như các điều khiển **GridView** và **ListView** ) trong một trang Web ASP.NET hiện có. Điều này cho phép bạn sử dụng các tính năng Dữ liệu động như xác thực trường dữ liệu thời gian chạy, các trang được xác định trước để thực hiện các hoạt động CRUD và tùy chỉnh trường dữ liệu. Để biết thêm thông tin, hãy xem [Cách thực hiện: Thêm hành vi động vào kiểm soát ràng buộc dữ liệu](https://msdn.microsoft.com/en-us/library/cc488562.aspx) .
* Thêm giàn giáo dữ liệu động vào trang Web ASP.NET hiện có (hoặc mới). Cách tiếp cận này cho phép bạn kiểm soát nhiều hơn đối với tuỳ chỉnh hơn là cho phép hành vi động cho các điều khiển riêng lẻ. Để biết thêm thông tin, hãy xem [Hướng dẫn: Thêm giàn giáo dữ liệu động vào các trang web ASP.NET hiện có](https://msdn.microsoft.com/en-us/library/ee923692.aspx) .

### Tùy chỉnh dữ liệu động

Một tính năng Dữ liệu động quan trọng là tùy chỉnh. Tùy chọn tùy chỉnh có thể được nhóm thành các danh mục sau:

* Tùy chỉnh hành vi chung (lớp trình bày). Bạn có thể sửa đổi cách Dữ liệu động thực hiện các tác vụ không dựa vào lược đồ cơ sở dữ liệu cụ thể. Bạn thực hiện các sửa đổi này trong lớp trình bày bằng cách sửa đổi các mẫu mặc định hoặc bằng cách tạo các mẫu tùy chỉnh để hiển thị và tương tác với cơ sở dữ liệu. Sau đó, bạn có thể sử dụng các sửa đổi này trong bất kỳ dự án Dữ liệu động nào. Để biết thêm thông tin, xem [ASP.NET Dynamic Data Presentation Layer Customization](https://msdn.microsoft.com/en-us/library/ee211433.aspx) .
* Tùy chỉnh hành vi cụ thể theo lược đồ. Bạn có thể tùy chỉnh hành vi Dữ liệu động cho các bảng hoặc trường riêng lẻ dựa trên mô hình dữ liệu hoặc lược đồ cơ sở dữ liệu. Trong trường hợp này, tùy chỉnh cụ thể cho cá thể của mô hình dữ liệu. Ví dụ, bạn có thể tùy chỉnh giao diện người dùng cho một trường dữ liệu riêng lẻ hoặc bạn có thể tạo mẫu trang tùy chỉnh hoặc sử dụng mẫu thực thể để hiển thị một bảng riêng lẻ. Bạn có thể sửa đổi giao diện và hành vi của các trường dữ liệu riêng lẻ bằng cách sử dụng các thuộc tính [System.ComponentModel.DataAnnotations](https://msdn.microsoft.com/en-us/library/system.componentmodel.dataannotations.aspx) . Bạn cũng có thể tùy chỉnh hành vi của một ứng dụng, cho phép bạn tạo xác nhận tùy chỉnh và gắn kết các bảng hoặc các trường trong mô hình dữ liệu với các mẫu cụ thể. Để biết thêm thông tin, hãy xem [Tùy chỉnh lớp dữ liệu động ASP.NET](https://msdn.microsoft.com/en-us/library/ee225428.aspx) .

## Các kịch bản dữ liệu động

Bảng sau đây liệt kê các tình huống và các chủ đề Dữ liệu động mô tả cách thực hiện chúng.

|  |  |  |
| --- | --- | --- |
| **Kịch bản** | **Đề tài** | **Mô tả và ví dụ** |
| Bật hành vi động trong các điều khiển ràng buộc dữ liệu trong một trang Web hiện có | [Cách: Thêm hành vi động vào kiểm soát ràng buộc dữ liệu](https://msdn.microsoft.com/en-us/library/cc488562.aspx) . | Cách tiếp cận này rất hữu ích nếu bạn có một trang Web hiện có và bạn muốn thêm xác thực Dữ liệu động và các mẫu sử dụng ít hoặc không có mã hóa. |
| Tùy chỉnh giao diện và hành vi của một trường dữ liệu riêng lẻ | [Hướng dẫn: Tùy chỉnh diện mạo và hành vi của trường dữ liệu trong Mô hình dữ liệu](https://msdn.microsoft.com/en-us/library/ee712708.aspx) | Cách tiếp cận này cho phép bạn tùy chỉnh cách một trường cụ thể trong mô hình dữ liệu được hiển thị. Đây là tùy chỉnh theo lược đồ cụ thể mà bạn tạo mẫu trường tùy chỉnh.  Xem video hiển thị tính năng này: [Xem tùy chỉnh Giao diện và hành vi của trường dữ liệu](https://www.microsoft.com/downloads/details.aspx?FamilyID=DF5AEBB5-3252-4595-9326-5335DDE48504) . |
| Tùy chỉnh giao diện và hành vi cho loại trường dữ liệu | [Làm thế nào để: Tùy chỉnh dữ liệu động mặc định](https://msdn.microsoft.com/en-us/library/cc488533.aspx) | Cách tiếp cận này cho phép bạn tùy chỉnh giao diện người dùng của loại trường dữ liệu trong toàn bộ ứng dụng. Nó là một tùy chỉnh hành vi chung trong lớp trình bày, trong đó bạn tùy chỉnh các mẫu trường mặc định.  Xem video hiển thị tính năng này: [Xem tùy chỉnh mẫu trường mặc định](https://www.microsoft.com/downloads/details.aspx?FamilyID=A49B1992-7E19-4D7C-A7D0-8C2FC86866A7) . |
| Tùy chỉnh giao diện và hành vi cho loại trường dữ liệu không nguyên thủy | [Làm thế nào để: Tùy chỉnh diện mạo và hành vi của trường dữ liệu cho các kiểu dữ liệu không nội tại trong mô hình dữ liệu](https://msdn.microsoft.com/en-us/library/cc668199.aspx) | Cách tiếp cận này cho phép bạn tùy chỉnh giao diện người dùng cho một trường dữ liệu riêng lẻ. Đây là một tùy chỉnh theo lược đồ cụ thể mà bạn ánh xạ trường vào một mẫu trường cụ thể hơn mẫu mặc định cho kiểu dữ liệu đó. Ví dụ, bạn có thể đánh dấu một trường văn bản có chứa địa chỉ e-mail dưới dạng một loại e-mail. |
| Tùy chỉnh giao diện và hành vi của một bảng riêng lẻ bằng cách sử dụng mẫu trang tùy chỉnh | [Cách: Tùy chỉnh bố cục của một bảng riêng lẻ bằng cách sử dụng mẫu trang tùy chỉnh](https://msdn.microsoft.com/en-us/library/cc488537.aspx) | Cách tiếp cận này cho phép bạn tùy chỉnh một bảng riêng bằng cách sửa đổi một mẫu mặc định, (ví dụ như List.aspx) cho một bảng dữ liệu cụ thể. Đó là một tùy chỉnh theo lược đồ cụ thể. |
| Tùy chỉnh giao diện và hành vi của một bảng riêng lẻ bằng cách sử dụng các mẫu thực thể tùy chỉnh | [Hướng dẫn: Tùy chỉnh bố cục bảng bằng cách sử dụng mẫu thực thể](https://msdn.microsoft.com/en-us/library/dd723645.aspx) | Cách tiếp cận này cho phép bạn tùy chỉnh một bảng riêng lẻ. Đó là tùy chỉnh hành vi chung mà bạn tạo mẫu thực thể tùy chỉnh. Nó là một thay thế cho phương pháp tiếp cận mẫu trang tùy chỉnh và cung cấp cho bạn quyền kiểm soát tốt hơn đối với bố cục bảng.  Một dự án Visual Studio với mã nguồn có sẵn để đi cùng chủ đề này: [Mẫu dữ liệu động](http://go.microsoft.com/fwlink/?linkid=151901) . |
| Tùy chỉnh xác thực trường dữ liệu bằng cách sử dụng thuộc tính hoặc bằng cách tùy chỉnh phương thức xử lý thay đổi trường hoặc bằng cách tùy chỉnh trình xử lý sự kiện xác thực | [Cách: Tùy chỉnh xác thực trường dữ liệu trong mô hình dữ liệu](https://msdn.microsoft.com/en-us/library/cc488527.aspx) | Cách tiếp cận này cho phép bạn tùy chỉnh xác thực của một trường dữ liệu riêng lẻ. Đó là một tùy chỉnh theo lược đồ cụ thể được lấy ví dụ bằng cách ghi đè trình xử lý sự kiện **OnValidate** .  Chạy một ví dụ trực tuyến về tính năng này: [Chạy](http://go.microsoft.com/fwlink/?LinkId=120736&sref=DynamicData.CustomValidation&spage=CustomValidation.aspx) . |
| Tùy chỉnh xác thực trường dữ liệu bằng cách sử dụng thuộc tính xác thực tùy chỉnh | [Cách: Tùy chỉnh xác thực trường dữ liệu trong Mô hình dữ liệu bằng cách sử dụng thuộc tính tùy chỉnh](https://msdn.microsoft.com/en-us/library/cc668224.aspx) | Cách tiếp cận này cho phép bạn tùy chỉnh xác thực của một trường dữ liệu riêng lẻ. Đó là tùy chỉnh theo lược đồ cụ thể mà bạn áp dụng một thuộc tính xác thực tùy chỉnh cho trường dữ liệu trong mô hình dữ liệu. Bạn sử dụng phương pháp này khi các thuộc tính xác thực dữ liệu có sẵn không đáp ứng các yêu cầu của bạn.  Xem ví dụ về mã thời gian chạy của tính năng này: [Chạy](http://go.microsoft.com/fwlink/?LinkId=120736&sref=DynamicData.CustomAttributeValidation&spage=CustomAttributeValidation.aspx) . |
| Lọc các hàng trong bảng để hiển thị trong bảng con bằng cách sử dụng giá trị khóa ngoài từ bảng cha | [Hướng dẫn: Lọc hàng trong bảng có mối quan hệ cha mẹ và con](https://msdn.microsoft.com/en-us/library/dd985039.aspx) | Cách tiếp cận này cho phép bạn lọc các hàng trong bảng để hiển thị dựa trên giá trị khóa ngoài của bảng cha.  Một dự án Visual Studio với mã nguồn có sẵn để đi cùng chủ đề này: [Scaffolding Dynamic Data](http://go.microsoft.com/fwlink/?LinkId=162154) . . |

## Cách tạo 1 chương trình Dynamic Data.

Bước 1: Mở Visual Studio hoặc Visual Web Developer.

Bước 2: Trên thanh menu chọn “File”, chọn “Web site”.

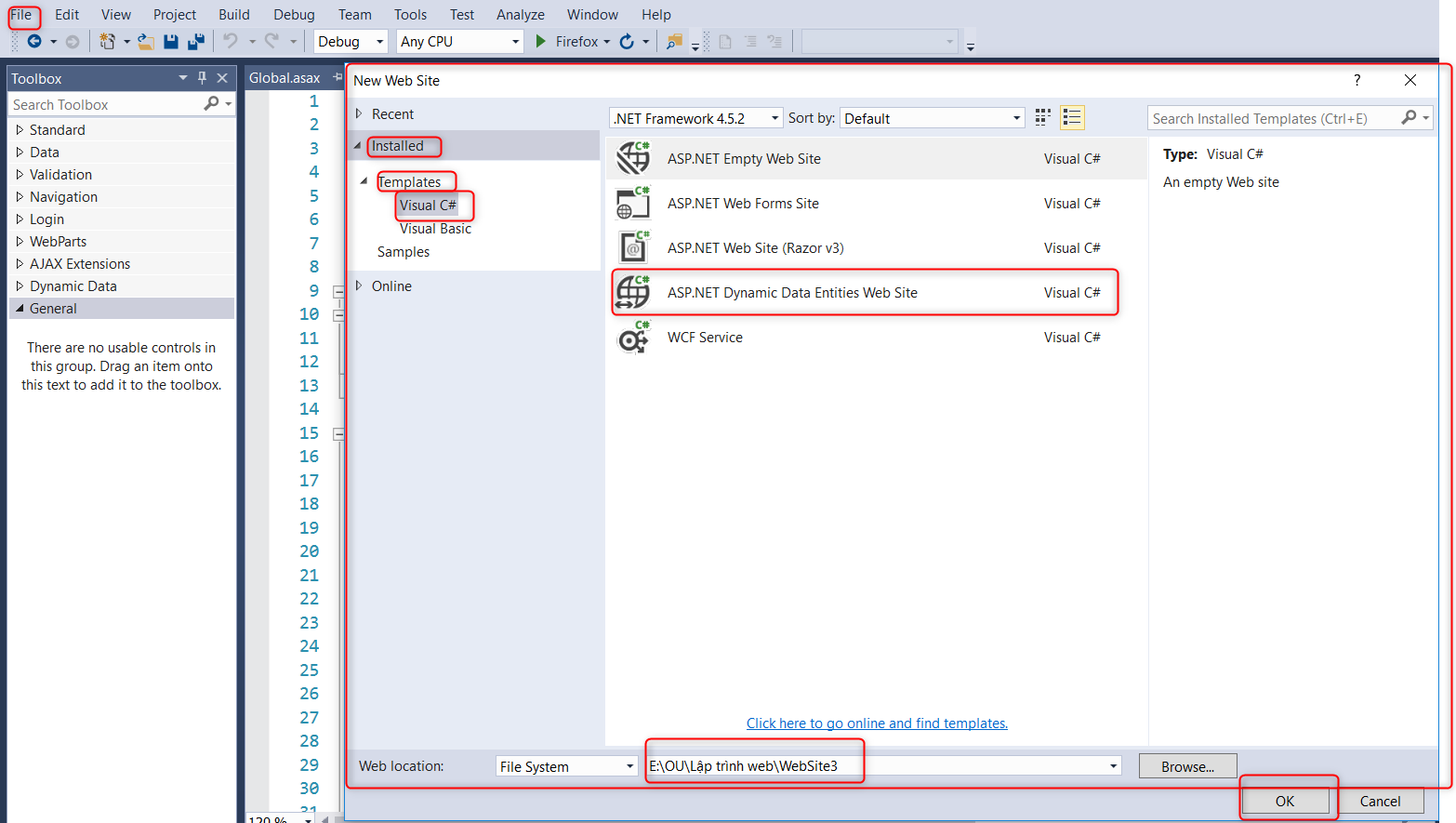
Cửa sổ “New Web Site” sẽ hiện ra.

Bước 3: Dưới mục “Installed/Templates”, chọn Visual C#.

Bước 4: Chọn ASP.NET Dynamic Data Entities Web Site.

Bước 5: Chọn địa chỉ để lưu project.

Bước 6: Click OK.



*Hình: Tạo project ASP.NET Dynamic Data*

Before we start building a sample application, we need something to build. In the spirit of keeping things simple so that we can focus this chapter solely on dynamic data, the application must also be simple.

We call it ZombiePedia, a website for cataloguing the appearances of various zombies. This way, during the zombie apocalypse, we can all hit ZombiePedia from our smart phones to log zombies we’ve seen and look up repeat zombies to find their vital statistics. After all, we need to know if we can touch a zombie with bare hands or if it’s a virus carrier, don’t we?

Without dynamic data, we would have to hand-code the page that displays the list of zombie sightings, the sighting detail and edit forms, the ability to create new sightings, and the pages that add, remove, and edit zombie types, and a bunch more plumbing. Doing this by hand, even with a rapid development framework like the MVC Framework, would be tedious and time-consuming. If our goal is to produce a functioning prototype that we can use as the launching point to start building our production application, we should use dynamic data.

To start, let’s create a new web application. Open up Visual Studio 2010 and create a new project. Under the Installed Templates panel in the Web category, click ASP.NET Dynamic Data Entities Web Application (shown in Figure 32.1). Call the application whatever you like, but ZombiePedia might be a good choice.

This particular template creates an ASP.NET Dynamic Data website that comes preequipped with code, scaffolding, and references to build a dynamic user interface on top of ADO.NET Entity Framework Entity Data Model (EDM).

Before we go any further, we need to add the entity data model to our project. You can mock this up quickly by creating two simple tables in a SQL database: ZombieType (ID, Name, Description) and ZombieSighting (ID, Name, Longitude, Latitude, and so on). Figure 32.2 shows the dialog box with which you are presented after right-clicking the project and choosing to add a new ADO.NET Entity Model and choosing Generate from Database.

Building a Dynamic Data Application

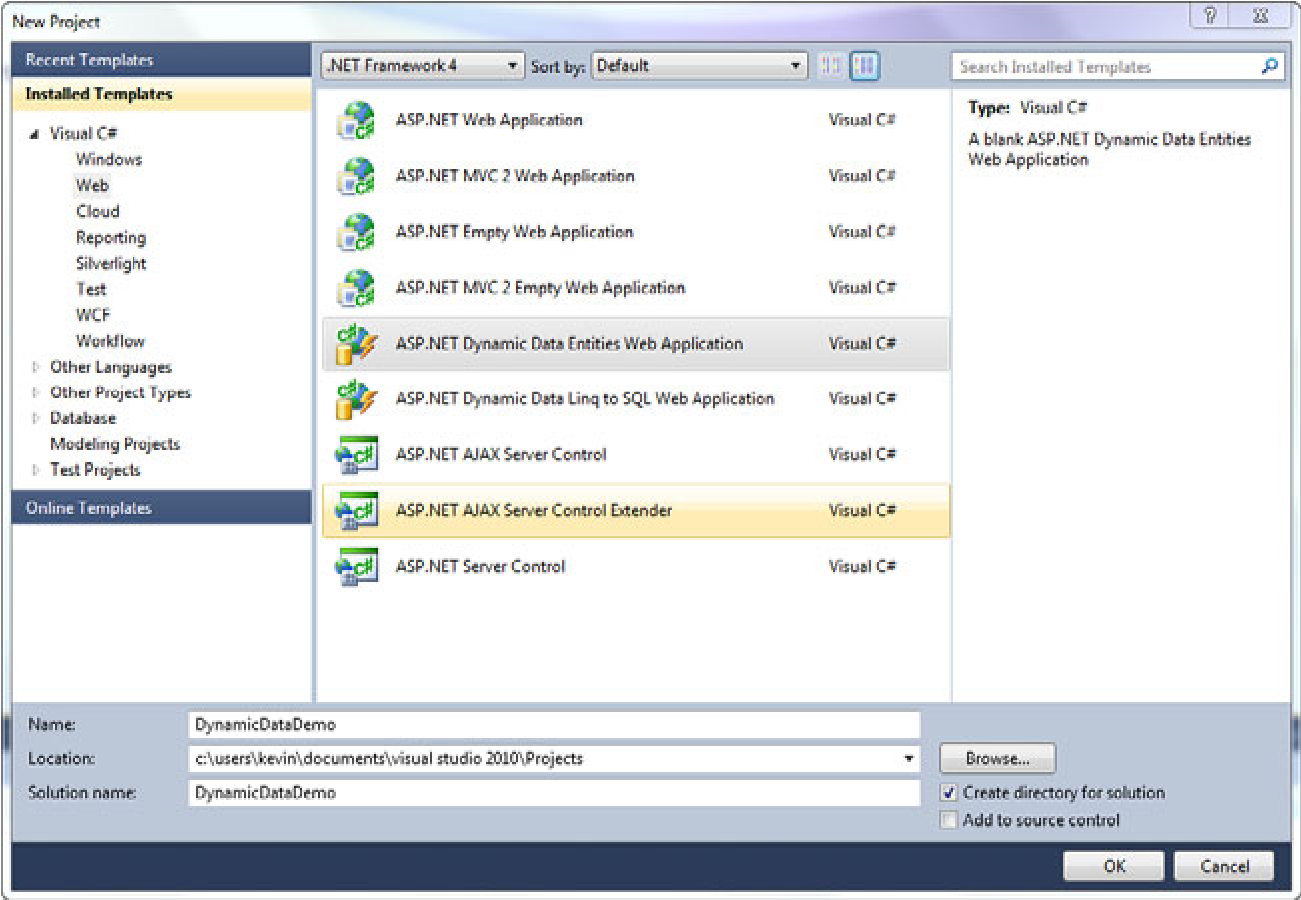


FIGURE 32.1 Creating a new ASP.NET Dynamic Data Entries Project.

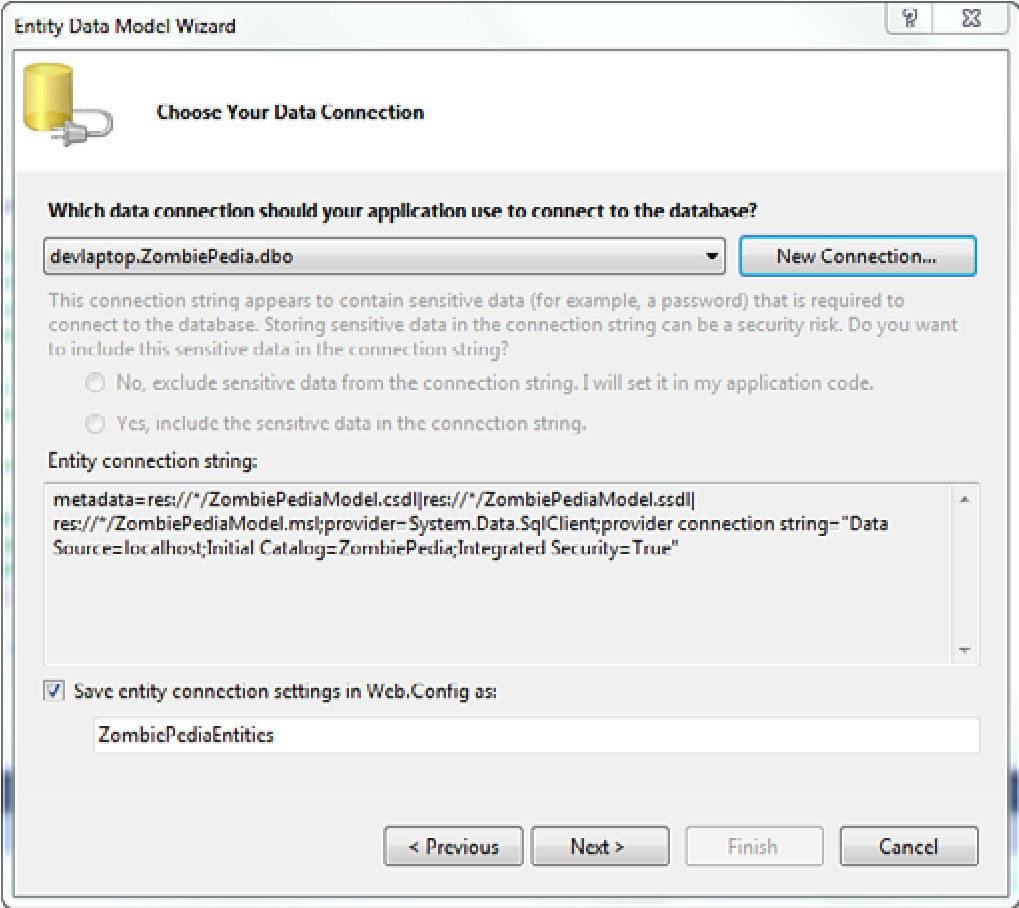


FIGURE 32.2 Adding an ADO.NET Entity Data Model by Generating from Database.

With a dynamic data website created and an entity data model added to the project, we’re almost ready to run the application—no code written yet! First, let’s take a look at the project that Visual Studio 2010 created for us.

The key folder of importance here is the DynamicData folder. A content folder serves as a root folder for all your content and media. The EntityTemplates folder contains user controls that apply to screens that deal with an individual entity. This folder comes with files for rendering an entity, editing an entity, and inserting a new entity. These files work against *any* entity type and are generic. This is a great starting point, but you can also customize these later and tailor the UI more specifically to your schema, as you see later in the chapter.

Also worth looking at is the FieldTemplates folder, which contains user controls for rendering read-only and edit-mode views of individual fields. There is a field here for virtually every type of column-level data you can think of including enumerated types, email addresses, dates, decimals, and even foreign-key controls such as ForeignKey and ManyToMany.

You can’t see it clearly in Figure 32.3, but PageTemplates folder contains default templates for all the different types of pages: Details, Edit, Insert, Delete, List, and ListDetails. It is no coincidence that a page template exists for each of the different types of reusable datadriven application patterns we mentioned at the beginning of the chapter.

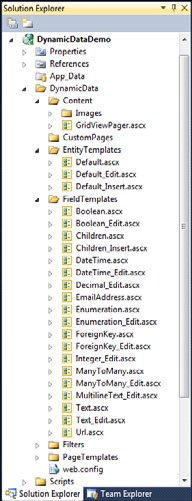


FIGURE 32.3 Default Project Created By ASP.NET Dynamic Data Template.

Before we can run the application as it stands, we need to tell the dynamic data framework about our entity model. To do that, open up the Global.asax.cs file.

A commented outline of code calls the DefaultModel.RegisterContext method. After uncommenting, it looks something like this:

Building a Dynamic Data Application

DefaultModel.RegisterContext( typeof(Entities),

new ContextConfiguration() { ScaffoldAllTables = true });

In our situation, the name of the entity model context created was called Entities, so that’s what we passed as the first parameter to the method. The ScaffoldAllTables parameter instructs dynamic data to enable all the tables in our entity model to be scaffolded. That is, each table will have its own fully functioning UI that enables for listing, detail viewing, editing, inserting, and deleting. We can choose not to scaffold all tables and manually control which tables are scaffolded if we want.

With this in place, and assuming that the entities connection string is sitting in our Web.config file properly, we can then run the application. When we run the application, the home page (shown in Figure 32.4) presents us with a list of tables. This list of tables, as mentioned previously, is either all the tables in the model or is manually defined by code. In our case we have two tables: ZombieTypes and ZombieSightings. Note that these table names were pluralized by the Entity Framework—SQL Server thinks these tables are called ZombieType and ZombieSighting.

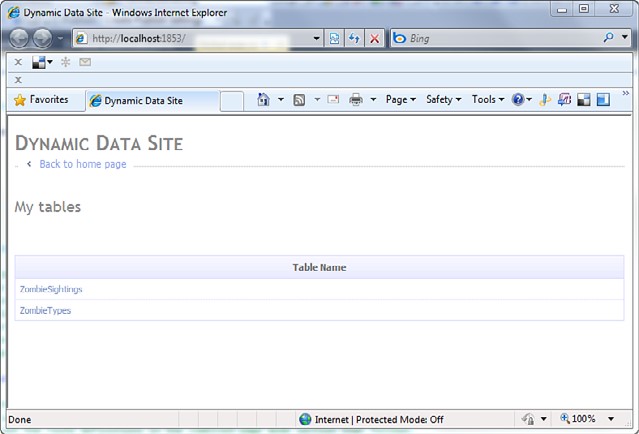


FIGURE 32.4 List of Scaffolded Tables in Dynamic Data Application.

Clicking the ZombieSightings link takes us to the list display page for the zombie sightings table. The URL for this is /ZombieSightings/List.aspx. Our application does not contain a ZombieSightings folder. The combination of the URL Routing engine (see Chapter 26, “Using the Login Controls,”) and dynamic data work together to merge the information in the URL with the dynamic data templates.

Figure 32.5 shows the list view. Not only do we have a pagination-enabled grid view showing the rows of data, but also we can filter on columns (even columns that are foreign keys!) and sort by clicking the column headers.

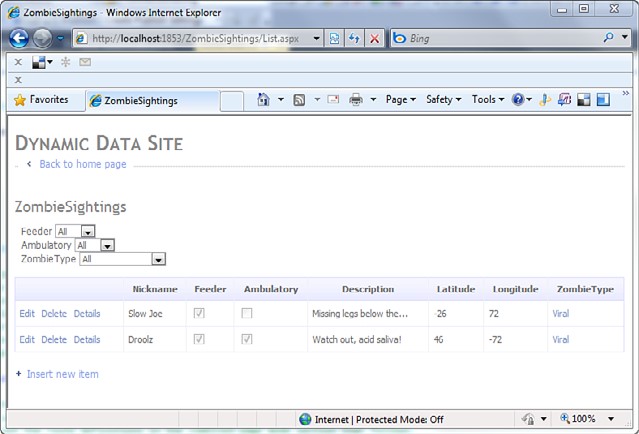


FIGURE 32.5 Displaying Data in an Entity List Scaffold view.

Automatically, the reverse of each foreign key relationship is made available on the corresponding entity page. So if you click the Viral zombie type in the screen, as shown in Figure 32.5, you will be taken to the details page for the Viral zombie type. On that page there is a View ZombieSightings link, enabling you to see the other end of the foreign key relationship. Any developer who has ever built a data-driven website knows how much code and manual labor this normally takes—and we haven’t written a single line of code!

## Làm việc với mô hình Dynamic Data.

Using the default templates is a great place to start for building a data-driven, dynamic web application. They provide all the functionality that you need to start. However, they should be used as just that—a starting point.

Typical web applications, especially those applications in service of a line of business, have specific functionality to that application. There is custom business logic, custom UI, subtle tweaks to the way controls work, and the overall user experience. All this combines to make a compelling user experience and an excellent web application. To do that with a dynamic data application, you need to know how to create your own templates and customize the shared templates.

Following are two types of template changes you can make:

. **Shared Templates**—Apply to all entities within an application. Making a change here can cause the change to appear in the UI everywhere in the application.

. **Custom Templates**—Apply only to a specific entity and override the shared templates when the dynamic data framework finds them.

Working with Dynamic Data Templates

### Making Shared Template Changes

If you make changes to the user controls in the FieldTemplates, PageTemplates,

EntityTemplates, or Filters directory, those changes apply to *all* entities for which your application provides scaffolding.

To see how this works in action, let’s modify the list template to pretty up the message that we get when no records are found. This message displays when there are no records and when the search criteria returns no results. To help clarify that, we change the text a little.

Open the List.aspx page in the PageTemplates folder, and scroll down to the GridView1 grid view control and locate the EmptyDataTemplate element. Replace it with the following:

<EmptyDataTemplate> There are currently no <%= table.DisplayName%> found.

</EmptyDataTemplate>

Now when we run the application, no matter which table we view, we see this new display text when we attempt to set a filter that returns 0 rows or if the table is empty.

Obviously in a production application, the level of customization is much higher. The great thing about these templates is that because they apply automatically to all tables, it becomes incredibly easy to make sweeping changes to the look, feel, and functionality of your application regardless of the underlying data model. These custom templates, because of their loosely coupled nature, can even be reused among multiple applications within the same organization to create a strong, branded feel and shared functionality.

### Creating Type and Entity-Specific Templates

Having the ability to make changes to every piece of your website with a single code change is powerful but also not necessarily the most prudent way to make changes. It brings up the old hitting-a-nail-with-a-sledgehammer analogy. After you make your subtle changes to the shared templates, you might want to make changes to the templates for a single entity.

To do this, you make use of the CustomPages folder. For this exercise, we modify the list template for the ZombieSightings table so that we are not using auto-generated columns and we have a bit more control over how this grid displays.

To start, create a new folder called ZombieSightings under the CustomPages folder. Make a copy of the List.aspx page from the PageTemplates folder, and copy it to the ZombieSightings folder.

Now open up the List.aspx page in your new ZombieSightings folder. At this point we have build errors because there is a type name conflict. Change the Inherits property at the top of the page to ZombieList; then in the List.aspx.cs file, change the class name to ZombieList. This now prevents the custom list template from having a name collision with the shared list template at compile time.

Modify the markup for List.aspx for our custom list so that it looks like the following code:

<%@ Page Language=”C#” MasterPageFile=”~/Site.master”

CodeBehind=”List.aspx.cs” Inherits=”DynamicDataDemo.ZombieList” %>

<%@ Register src=”~/DynamicData/Content/GridViewPager.ascx” tagname=”GridViewPager” tagprefix=”asp” %>

<asp:Content ID=”headContent” ContentPlaceHolderID=”head” Runat=”Server”>

</asp:Content>

<asp:Content ID=”Content1” ContentPlaceHolderID=”ContentPlaceHolder1”

➥Runat=”Server”>

<asp:DynamicDataManager ID=”DynamicDataManager1” runat=”server” ➥AutoLoadForeignKeys=”true”> <DataControls>

<asp:DataControlReference ControlID=”GridView1” />

</DataControls>

</asp:DynamicDataManager>

<h2 class=”DDSubHeader”>Zombie Sightings!</h2>

<asp:UpdatePanel ID=”UpdatePanel1” runat=”server”>

<ContentTemplate>

<div class=”DD”>

<asp:ValidationSummary

ID=”ValidationSummary1” runat=”server” EnableClientScript=”true”

HeaderText=”List of validation errors” CssClass=”DDValidator” />

<asp:DynamicValidator runat=”server” ID=”GridViewValidator” ControlToValidate=”GridView1”

Display=”None” CssClass=”DDValidator” />

<asp:QueryableFilterRepeater runat=”server” ID=”FilterRepeater”>

<ItemTemplate> <asp:Label runat=”server” Text=’<%# Eval(“DisplayName”) %>’

OnPreRender=”Label\_PreRender” />

<asp:DynamicFilter runat=”server” ID=”DynamicFilter”

OnFilterChanged=”DynamicFilter\_FilterChanged” /><br />

</ItemTemplate>

</asp:QueryableFilterRepeater>

<br />

</div>

<asp:GridView ID=”GridView1” runat=”server”

DataSourceID=”GridDataSource” EnablePersistedSelection=”true” AllowPaging=”True” AllowSorting=”True” CssClass=”DDGridView”

AutoGenerateColumns=”false”

RowStyle-CssClass=”td” HeaderStyle-CssClass=”th” CellPadding=”6”>

<Columns>

<asp:TemplateField>

Working with Dynamic Data Templates

<ItemTemplate>

<asp:DynamicHyperLink runat=”server” Action=”Edit”

➥Text=”Edit”

/>&nbsp;<asp:LinkButton runat=”server”

CommandName=”Delete” Text=”Delete”

OnClientClick=’

return confirm(“Are you sure you want to delete this item?”);’ />&nbsp;<asp:DynamicHyperLink runat=”server” ➥Text=”Details” />

</ItemTemplate>

</asp:TemplateField>

<asp:DynamicField DataField=”Nickname” />

<asp:DynamicField DataField=”ZombieType” HeaderText=”Zombie Type” ItemStyle-HorizontalAlign=”Center”/>

<asp:DynamicField DataField=”Description” />

<asp:DynamicField DataField=”Feeder” HeaderText=”Feeder?”

ItemStyle-HorizontalAlign=”Center” />

<asp:DynamicField DataField=”Ambulatory” HeaderText=”Ambulatory?” ItemStyle-HorizontalAlign=”Center”/>

</Columns>

<AlternatingRowStyle BackColor=”LightGoldenrodYellow” />

<PagerStyle CssClass=”DDFooter”/>

<PagerTemplate>

<asp:GridViewPager runat=”server” />

</PagerTemplate>

<EmptyDataTemplate> There are currently no <%= table.DisplayName%> found.

</EmptyDataTemplate>

</asp:GridView>

<asp:EntityDataSource ID=”GridDataSource” runat=”server” ➥EnableDelete=”true” />

<asp:QueryExtender TargetControlID=”GridDataSource”

ID=”GridQueryExtender” runat=”server”>

<asp:DynamicFilterExpression ControlID=”FilterRepeater” />

</asp:QueryExtender>

<br />

<div class=”DDBottomHyperLink”> <asp:DynamicHyperLink ID=”InsertHyperLink” runat=”server”

Action=”Insert”>

<img runat=”server” src=”~/DynamicData/Content/Images/plus.gif” alt=”Insert new item” />Insert new item</asp:DynamicHyperLink>

</div>

</ContentTemplate>

</asp:UpdatePanel>

</asp:Content>

After having made these changes, we can run the application and click the

ZombieSightings link from the home page. This presents a UI that looks like the one shown in Figure 32.6. Go back to the main page; then click on the ZombieTypes link to verify that it still conforms to the globally shared list template.

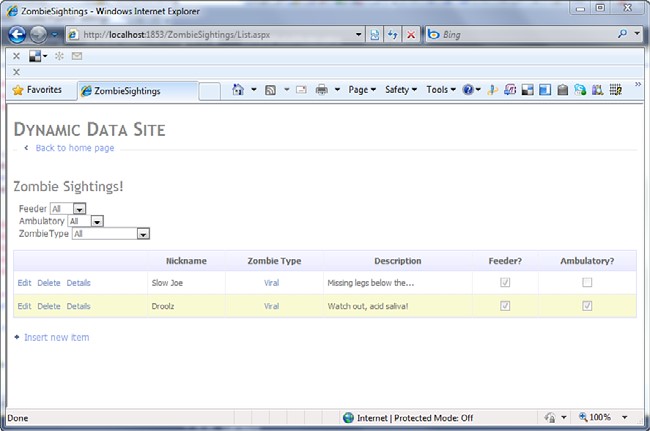


FIGURE 32.6 Custom Entity-Specific List template.

What we’ve just done is just the tip of the iceberg. If we stay within the structure of the dynamic data template system, we can create incredibly powerful, reusable templates that dramatically increase the productivity of developers and decrease the time gap between database schema creation and the corresponding web UI.

# Tổng quan về ASP.NET MVC:

## Giới thiệu về ASP.NET MVC Framework:

### Tổng quan về mô hình MVC:

*Hình: Mô hình MVC*

MVC là một design partern đã tồn tại rất lâu trong ngành công nghệ phần mềm. Một ứng dụng viết theo mô hình MVC sẽ bao gồm 3 thành phần tách biệt nhau đó là Model, View, Controller.

Giống như trong mô hình 3 lớp, mô hình MVC giúp tách biệt 3 tầng trong mô hình lập trình web, vì vậy giúp tối ưu ứng dụng, dễ dàng thêm mới và chỉnh sửa nội dung hoặc giao diện.

**Model**

Mô hình 3 lớp thì trong đó gồm có 2 tầng Data Access Layer và tầng Business Logic Layer. Hai tầng này là hai tầng tương đương với tầng model trong mô hình MVC.

**View**

Là tầng giao diện, hiển thị dữ liệu được truy xuất từ tầng model. Tầng này tương đương với tầng Presentation Layer trong mô hình 3 lớp.

**Controller**

Đây là tầng giúp kết nối giữa tầng model và tầng view trong mô hình MVC, có nghĩa là nếu phía client yêu cầu hiển thị dữ liệu thì controller gọi dữ liệu từ model và trả về cho view vì view tương tác trực tiếp với client.

Ví dụ:

User yêu cầu hiển thị thông tin cá nhân của user.

User gửi một yêu cầu tới controller.

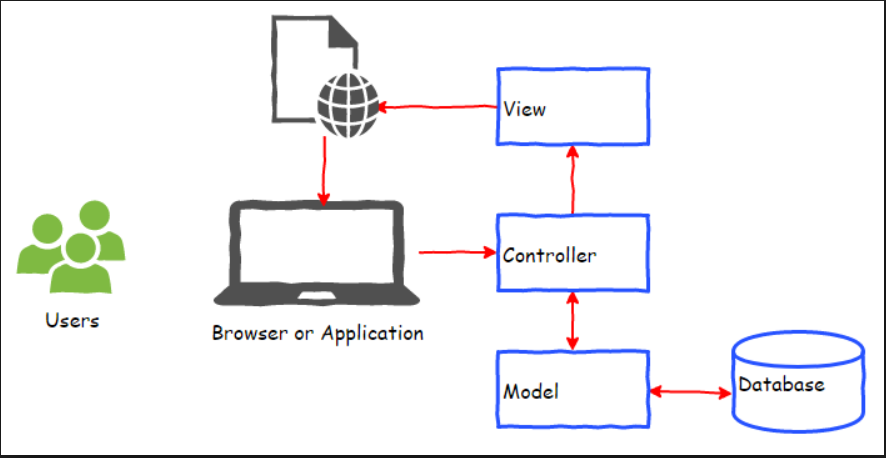
Controller nhận yêu cầu, xử lý yêu cầu, nếu yêu cầu cần truy xuất dữ liệu thì controller sẽ gửi yêu đó xuống tầng model để truy xuất dữ liệu.

Tầng model sẽ lấy dữ liệu từ database sau đó truyền dữ liệu qua tầng view thông qua tầng controller để tầng view hiển thị dữ liệu cho User.

User sẽ thấy thông tin hiển thị ở giao diện và cụ thể ở đây là tầng view.

### Tổng quan về lập trình ASP.NET MVC:

**Cơ chế hoạt động của ASP.Net MVC**



User gửi 1 yêu cầu tới server bằng cách truyền vào 1 URL trong browser.

Yêu cầu đó được gửi tới controller đầu tiên, controller sẽ xử lý yêu cầu, nếu yêu cầu cần truy xuất dữ liệu thì controller sẽ chuyển qua tầng model.

Tại tầng model, dữ liệu được truy xuất từ database và sau đó truyền qua view thông qua controller.

Controller sẽ giúp dữ liệu được chuyển từ model qua view.

View là tầng cuối cùng giao tiếp với User, mọi dữ liệu sẽ được hiển thị cho User thông qua tầng View.

**Những ưu điểm mà ASP.Net MVC mạng lại cho chúng ta:**

Do sử dụng mô hình MVC nên trong ASP.Net MVC đã tách biệt được các tầng trong mô hình lập trình web vì vậy giúp tối ưu ứng dụng và dễ dàng trong việc viết code, giao diện

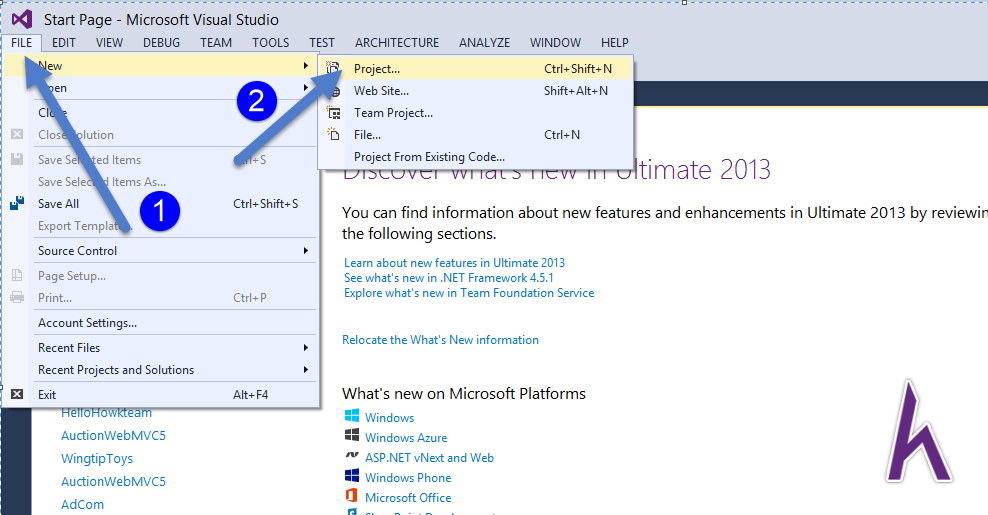
Giao diện trong ASP.Net MVCsử dụng công nghệ thiết kế web HTML, CSS nền việc thiết kế giao diện trở nên dễ dàng và giúp cho designer linh hoạt trong việc thiết kế

ASP.Net MVC không sử dụng view state vì vậy trang web không bị tăng kích thước do đó hiệu năng hoạt động không bị bị giảm

## Cách tạo 1 chương trình ASP.NET MVC:

Bước 1: Đầu tiên các bạn mở Visual Studio lên

Bước 2: Tại thanh menu bạn chọn FILE >New >Project hoặc là cách đơn giản hơn là bạn nhấn tổ hợp phím tắt **Ctrl + Shift + N**



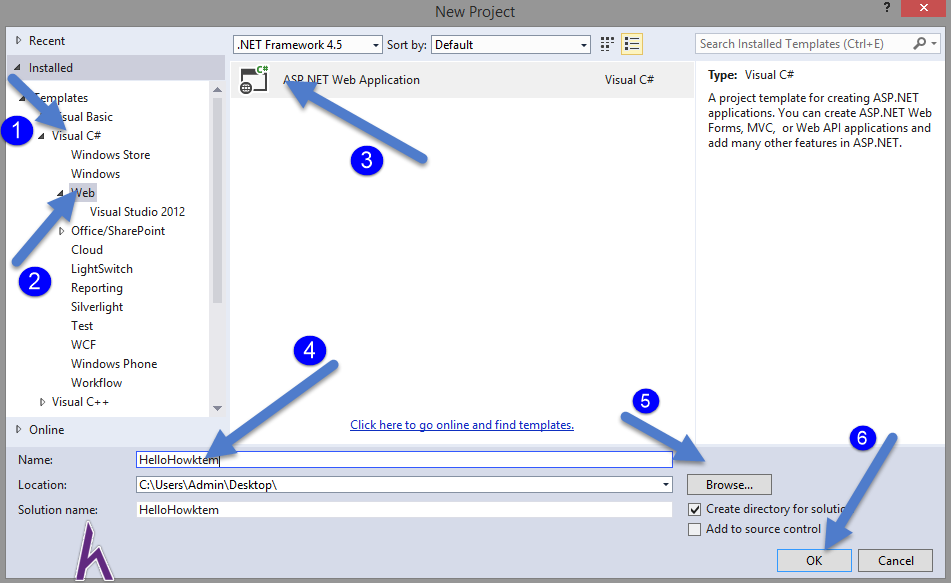
Bước 3: Sau khi bạn làm xong bước trên, chúng ta tiếp tục làm theo như hình:

Installed/ Templates 🡺 Visual C# 🡺 Web

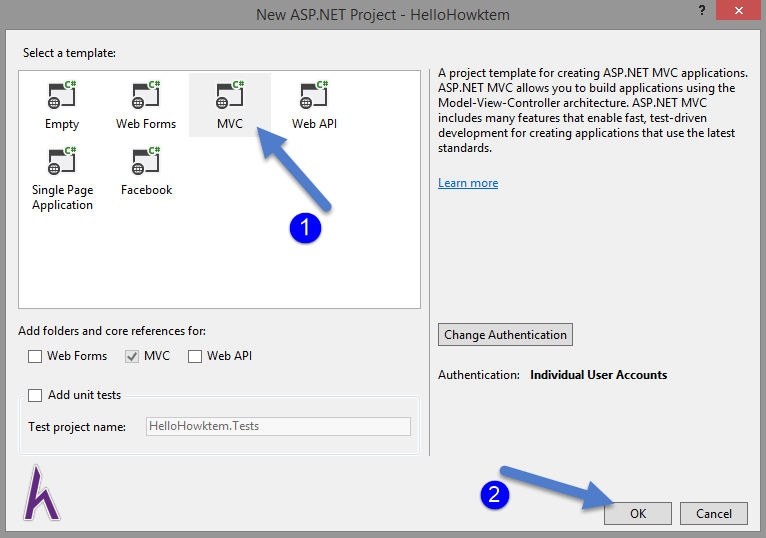
ASP.NET Web Application

Nhập tên chương trình, địa chỉ lưu trữ

OK

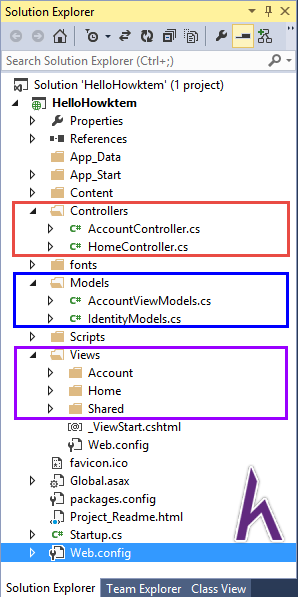


Bước 4: Sau khi bạn nhấn OK ở bước trên thì sẽ có 1 bảng hội thoại xuất hiện: Chọn MVC và nhấn OK.



## Chương trình ASP.NET MVC mặc định:

### Cấu trúc chương trình:

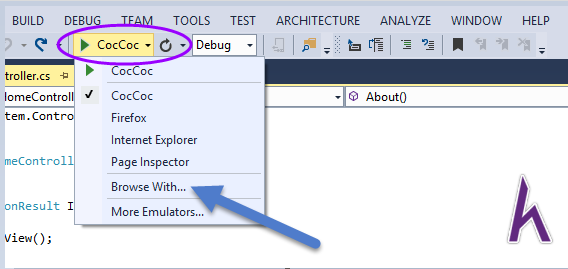


Như hình mình show ra cho các bạn ở phía trên, vùng màu đỏ chính là Controller, vùng màu xanh là Models và vùng màu tím chính là Views trong mô hình MVC.

Tương ứng với 1 controller thì sẽ có 1 view đi kèm theo, mặc định project khi tạo gồm có 2 controller đó là Account và Home vì vậy ta sẽ thấy có 2 folder đó là Account và Home trong phần View.

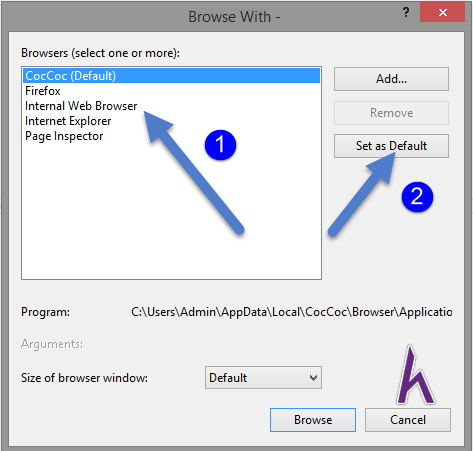
### Chạy chương trình:

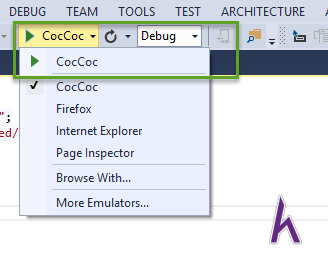
Trên thanh menu sẽ nhìn thấy nút mũi tên màu xanh, đây là nút giúp **Run project** lên, tuy nhiên ban đầu nếu thấy tại nút này không có hiển thị trình duyệt mặc định trong máy mình như cốc cốc, chrome, firefox … thì các ta chọn **Browse With..**



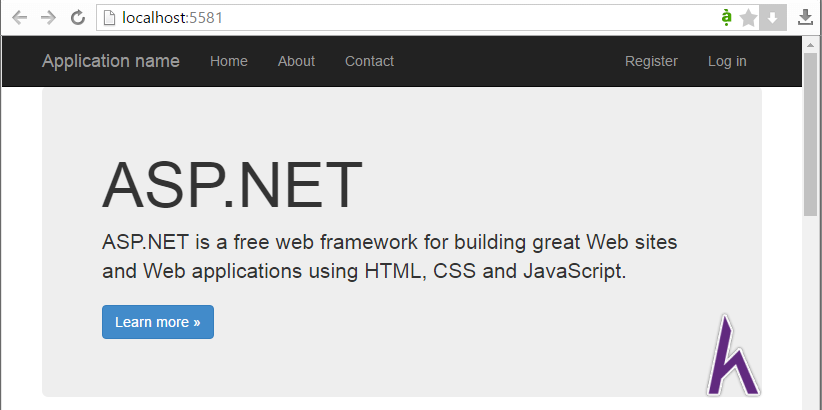
Sau khi chọn, 1 hộp thoại hiện lên, đầu tiên các ta chọn trình duyệt mà mình muốn sử dụng mặc định khi chạy project lên, sau đó chọn Set as Default.

Để chạy project bạn click vào nút **Run Project**





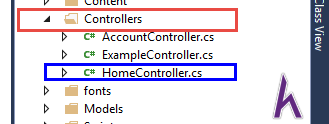
 Giao diện hiển thị khi chạy thành công như hình :



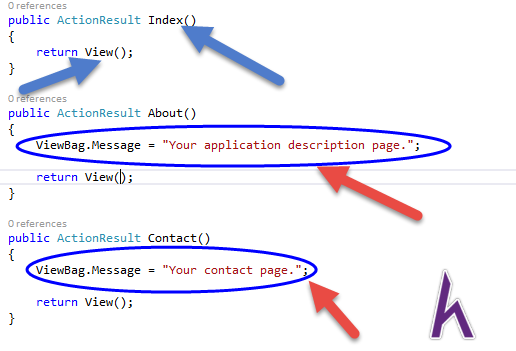
### Giải thích chương trình:

Để hiển thị được giao diện như trên ta sẽ tìm hiểu như sau:

Bước 1: Vào mục **Controller**.



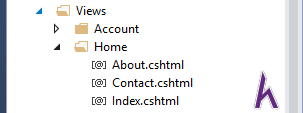
Bước 2: Sau đó chọn và mở **HomeController**:



Controller có cung cấp 1 phương thức có tên gọi là ActionResult. Phương thức này trả về rất nhiều kiểu dữ liệu khác nhau, và trong trường hợp này kiểu trả về là View().

Kiểu trả về View() dùng để hiển thị giao diện mà mình muốn xuất ra cho người dùng khi ActionResult được gọi lên.

 Bước 3: Mở thư mục **View:**

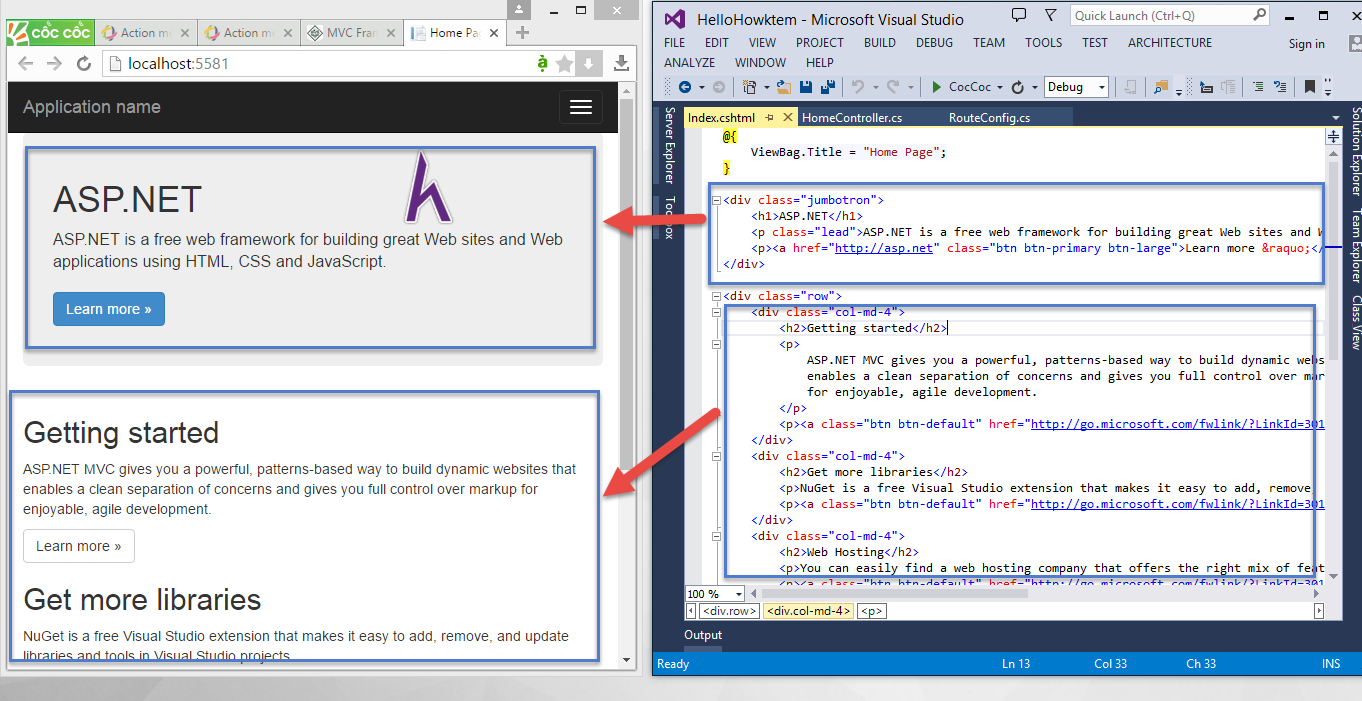


Trong HomeController bao gồm 3 ActionResult đó là **About**, **Contact**, **Index**. Mà cả 3 ActionResult này đều trả về kiểu dữ liệu là View() vì vậy trong folder home sẽ cũng có 3 View tương ứng cho 3 ActionResult.

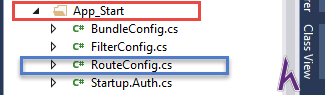
Cụ thể khi ActionResult Index được gọi lên thì Index.cshtml sẽ được hiển thị, và 2 cái còn lại tượng tự như vậy.

Bước 4: Click vào **Index.cshtml**:

Nội dung của file giống với nội dung được hiển thị khi chúng ta chạy project lên.



Cài đặt file mặc định khi chương trình chạy: **App\_Start 🡺 RouteConfig.**

****

## Accept Input:

Invariably, after seeing a little bit of what MVC is and how it works, developers immediately begin wondering how to invoke server-side events or how to manipulate data and accept input from users. First, you need to remember that in the MVC framework, there is no concept of server-side events. There are no server controls and no concept of postback and invoking server-side events.

So if you don’t have postbacks and server-side events, how do you provide interactivity for your users? To answer this, we have to dust off the corner of our brains that remembers what HTML actually is and unlearn the abstraction layer placed over HTTP by Web Forms.

To demonstrate both MVC’s simplicity and power, let’s create a simple form and write a controller method that responds to that form.

To start, open the Index.aspx page in the Demo folder and add the following block of code below the Index text:

<% using (Html.BeginForm(“DoSomething”, “Demo”, FormMethod.Post)) { %> Enter your favorite color:

<%= Html.TextBox(“favoriteColor”, “Blue”) %><br />

<input type=”submit” id=”submit” value=”Submit” />

<% } %>

Here we use another HTML extender, the BeginForm method to tell the MVC Framework that we plan to POST the contents of the user-submitted form to whatever URL is mapped to the Demo controller’s DoSomething method.

Because we post to the DoSomething method, we should probably create that method in the DemoController class:

[HttpPost] public ActionResult DoSomething(string favoriteColor) { ViewData[“color”] = favoriteColor; return View();

}

There is an incredibly powerful model binding subsystem that is part of ASP.NET MVC that can take form values and either turn them into method parameters as shown here or even convert them into model classes. For example, if all the form values in a form belong to a Customer class, you can actually pass an instance of the Customer class to the method to which your form posts, and the MVC model binder takes care of mapping everything accordingly. It is a huge timesaver and creates incredibly readable code.

Now let’s just create the view. Repeat the process we used last time by right-clicking the

Demo folder in the Views folder and add a new view called DoSomething. Put the following markup in the file:

<%= ViewData[“color”] %>

Now when you request the page at the /demo URL, you are prompted to enter your favorite color. When you do so and click the Submit button, your browser makes an HTTP

POST request to /demo/DoSomething, which invokes the code you just added to the DoSomething() method. This sets a variable in the ViewData dictionary. (A facility for providing information from the controller to the view; more can be found on this by Summary

checking out the resources at the end of the chapter.) This variable is then accessed from the DoSomething.aspx view, displaying your favorite color, as shown in Figure 33.6.

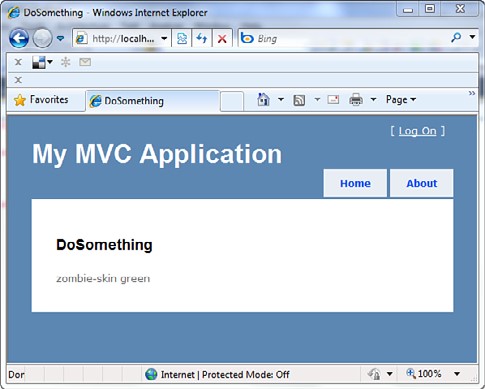


FIGURE 33.6 Results of posting a form to a Controller method.

## Demo chương trình ASP.NET MVC: