
Tài liệu ASP.NET MVC 6

Phóng thích

Microsoft

02 Tháng Ba, 2016

Machine Translated by Google

Nội dung

1 Tổng quan về ASP.NET MVC	3
2 Bắt đầu	5
2.1 Xây dựng ứng dụng MVC 6 đầu tiên của bạn	5
2.2 Xây dựng Web API đầu tiên của bạn với MVC 6	87
3 Hướng dẫn	103
3.1 Hướng dẫn về Cửa hàng	103
âm nhạc 3.2 Tạo Dịch vụ phụ trợ cho các ứng dụng di động gốc	103
4 kiểu máy	105
4.1 Mô hình ràng buộc	105
4.2 Xác thực mô hình	107
4.3 Định dạng	108
4.4 Bộ định dạng tùy chỉnh	108
5 Lượt	109
xem 5.1 Cú pháp Razor	109
5.2 Chế độ xem động so với chế độ xem được nhập mạnh	109
5.3 Trình trợ giúp HTML	109
5.4 Trình trợ giúp thẻ	110
5.5 Chế độ xem một phần	132
5.6 Đưa Dịch vụ vào Lượt xem	132
5.7 Xem các thành phần	138
5.8 Tạo một công cụ xem tùy chỉnh	145
5.9 Xây dựng các chế độ xem cụ thể trên thiết bị di động	146
6 Bộ điều khiển	147
6.1 Bộ điều khiển, Hành động và Kết quả Hành động	147
6.2 Định tuyến đến Hành động của Bộ điều khiển 6.3	149
Xử lý lỗi	149
6.4 Bộ lọc	149
6.5 Bộ điều khiển và tiêm phụ thuộc	149
6.6 Kiểm tra Logic của bộ điều khiển	154
6.7 Các linh vực 6.8 Làm việc với Mô hình Ứng dụng	154
7 Hiệu suất	157
Bộ nhớ đệm phản hồi	157

8 Bảo mật	161
8.1 Bộ lọc cấp phép 8.2	161
Thực thi SSL	161
8.3 Chống yêu cầu giả mạo	161
8.4 Xác định chính sách CORS	162
9 Di chuyển	165
9.1 Di chuyển từ ASP.NET MVC 5 sang MVC 6	165
9.2 Di chuyển cấu hình từ ASP.NET MVC 5 sang MVC 6	179
9.3 Di chuyển từ ASP.NET Web API 2 sang MVC 6	180
9.4 Di chuyển xác thực và nhận dạng từ ASP.NET MVC 5 sang MVC 6	188
10 đóng góp	193

Lưu ý: Tài liệu này đang được tiến hành. Các chủ đề được đánh dấu bằng một phần giữ chỗ chưa được viết. Bạn có thể theo dõi trạng thái của những chủ đề này thông qua trình [theo dõi vấn đề tài liệu công khai của chúng tôi](#). Tìm hiểu cách bạn có thể [đóng góp](#) trên GitHub. Giúp định hình phạm vi và trọng tâm của nội dung ASP.NET bằng cách tham gia [Khảo sát Tài liệu ASP.NET 5](#).

Tổng quan về ASP.NET MVC

Lưu ý: Chúng tôi hiện đang làm việc về chủ đề này.

Chúng tôi hoan nghênh ý kiến đóng góp của bạn để giúp định hình phạm vi và cách tiếp cận. Bạn có thể theo dõi trạng thái và cung cấp thông tin đầu vào về vấn đề này tại [GitHub](#).

Nếu bạn muốn xem lại các bản nháp và phác thảo ban đầu của chủ đề này, vui lòng để lại ghi chú với thông tin liên hệ của bạn trong số này.

Tìm hiểu thêm về cách bạn có thể [đóng góp](#) trên GitHub.

Bắt đầu

2.1 Xây dựng ứng dụng MVC 6 đầu tiên của bạn

2.1.1 Bắt đầu với ASP.NET MVC 6

Bởi Rick Anderson

Hướng dẫn này sẽ dạy bạn những kiến thức cơ bản về xây dựng ứng dụng web ASP.NET MVC 6 bằng Visual Studio 2015.

Phần

- Cài đặt Visual Studio và ASP.NET
- Tạo một ứng dụng web

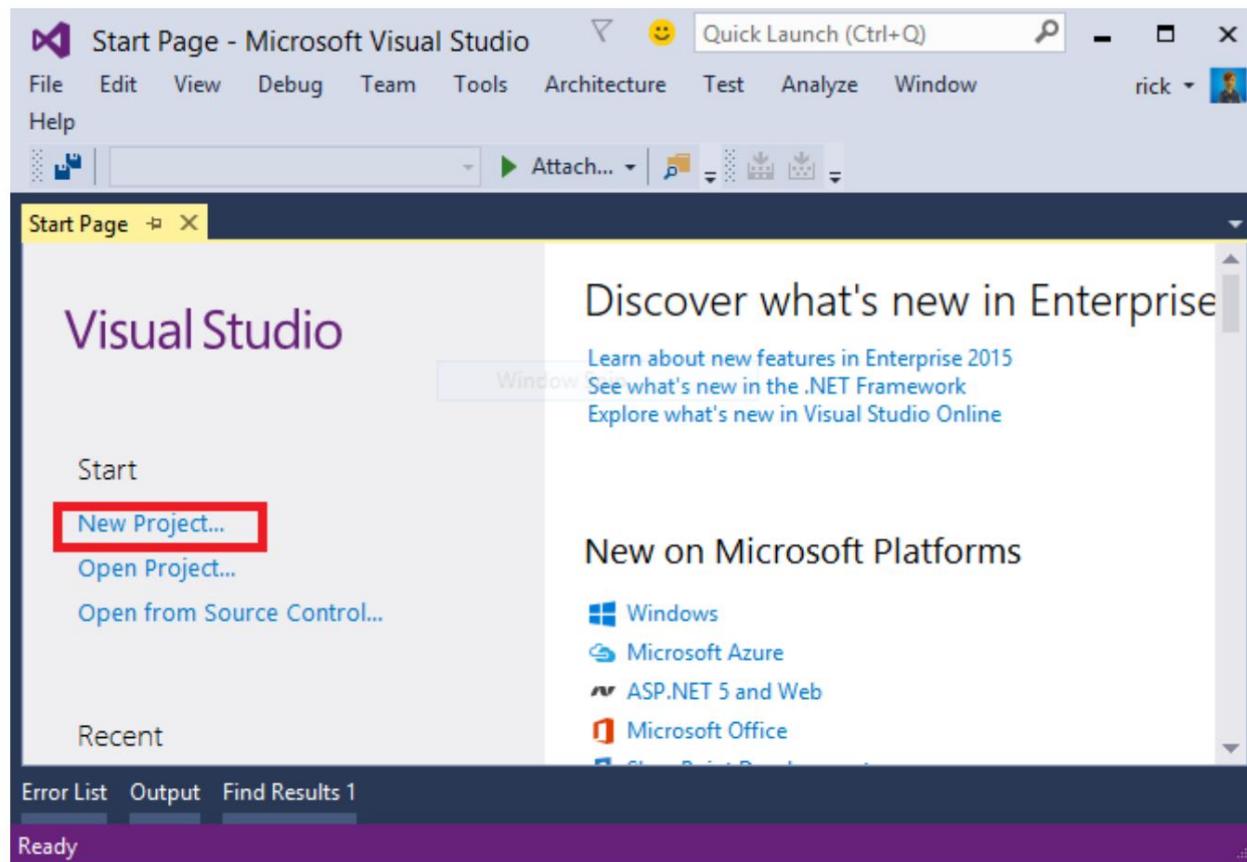
Cài đặt Visual Studio và ASP.NET

Visual Studio là một IDE (môi trường phát triển tích hợp) để xây dựng ứng dụng. Tương tự như sử dụng Microsoft Word để viết tài liệu, bạn sẽ sử dụng Visual Studio để tạo ứng dụng web.

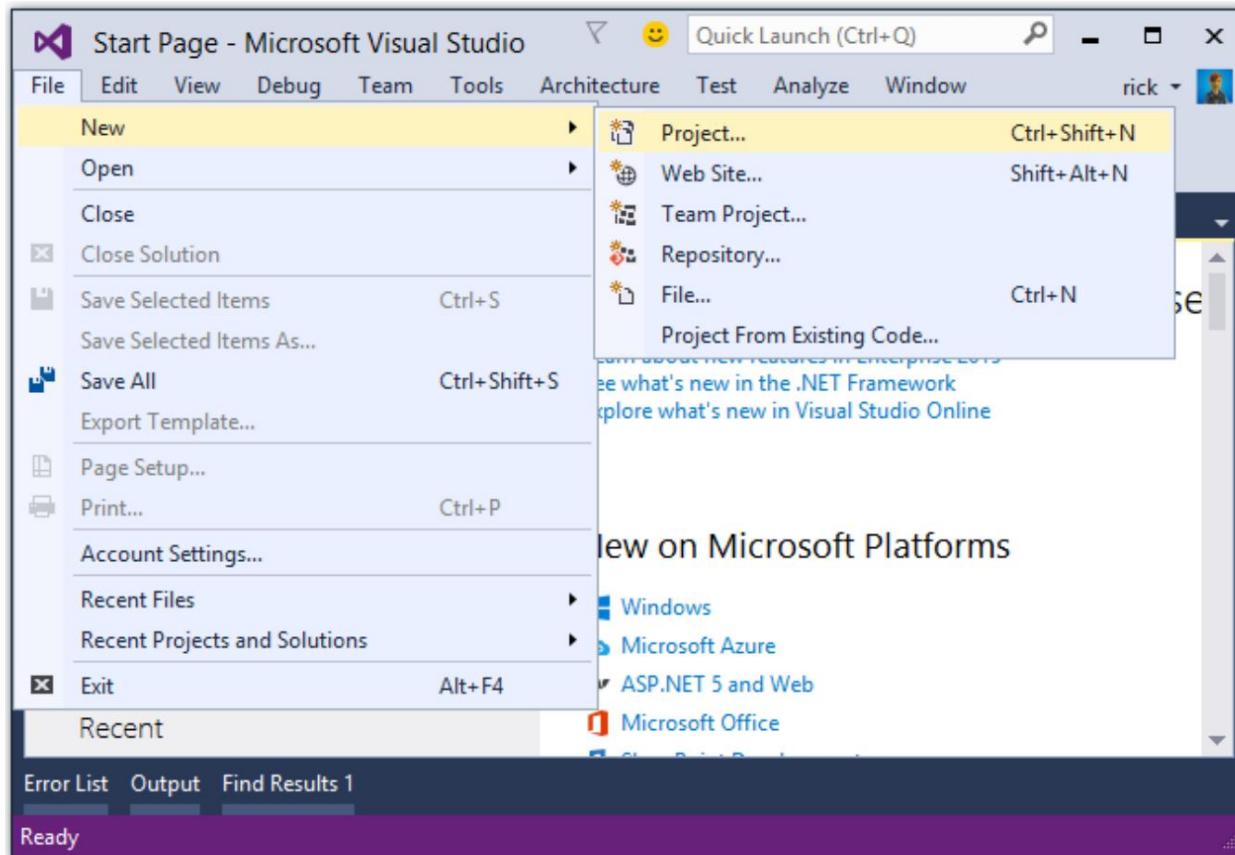
Cài đặt ASP.NET 5 và Visual Studio 2015.

Tạo một ứng dụng web

Từ trang Bắt đầu của Visual Studio , chạm vào Dự án mới.

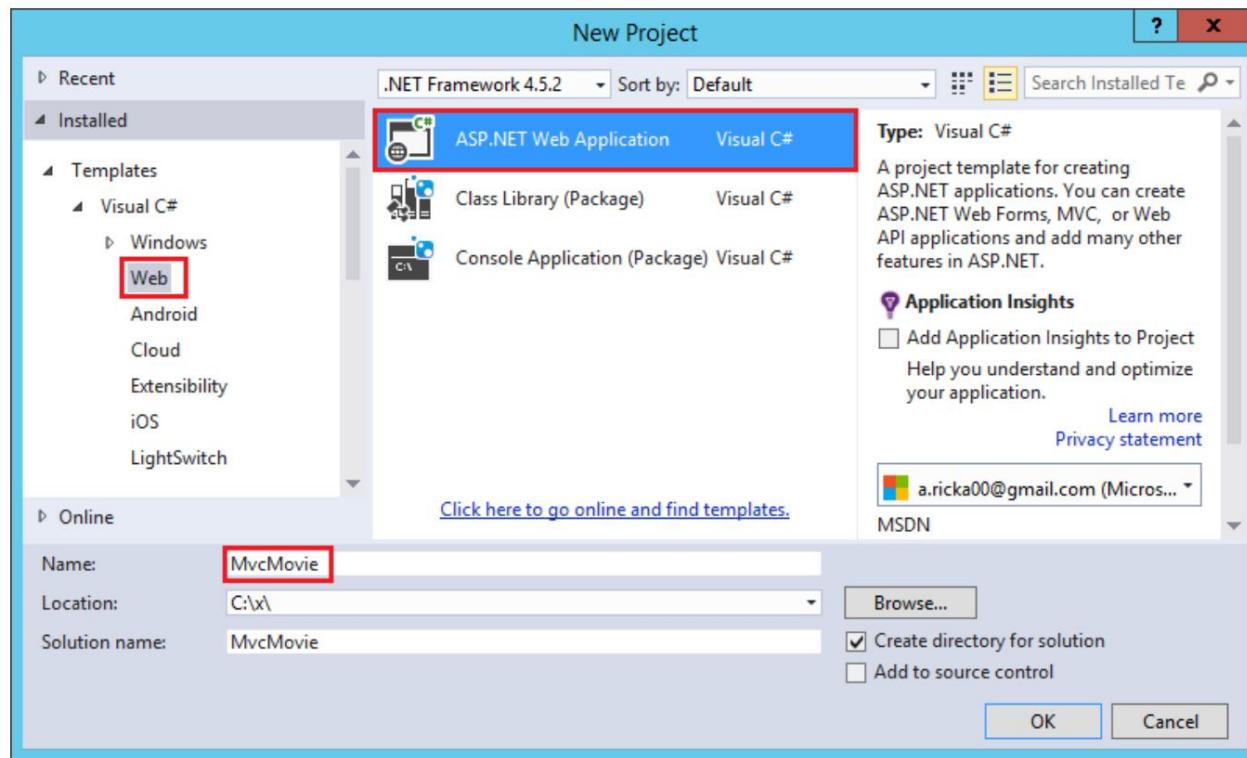


Ngoài ra, bạn có thể sử dụng các menu để tạo một dự án mới. Nhấn vào Tệp> Mới> Dự án.

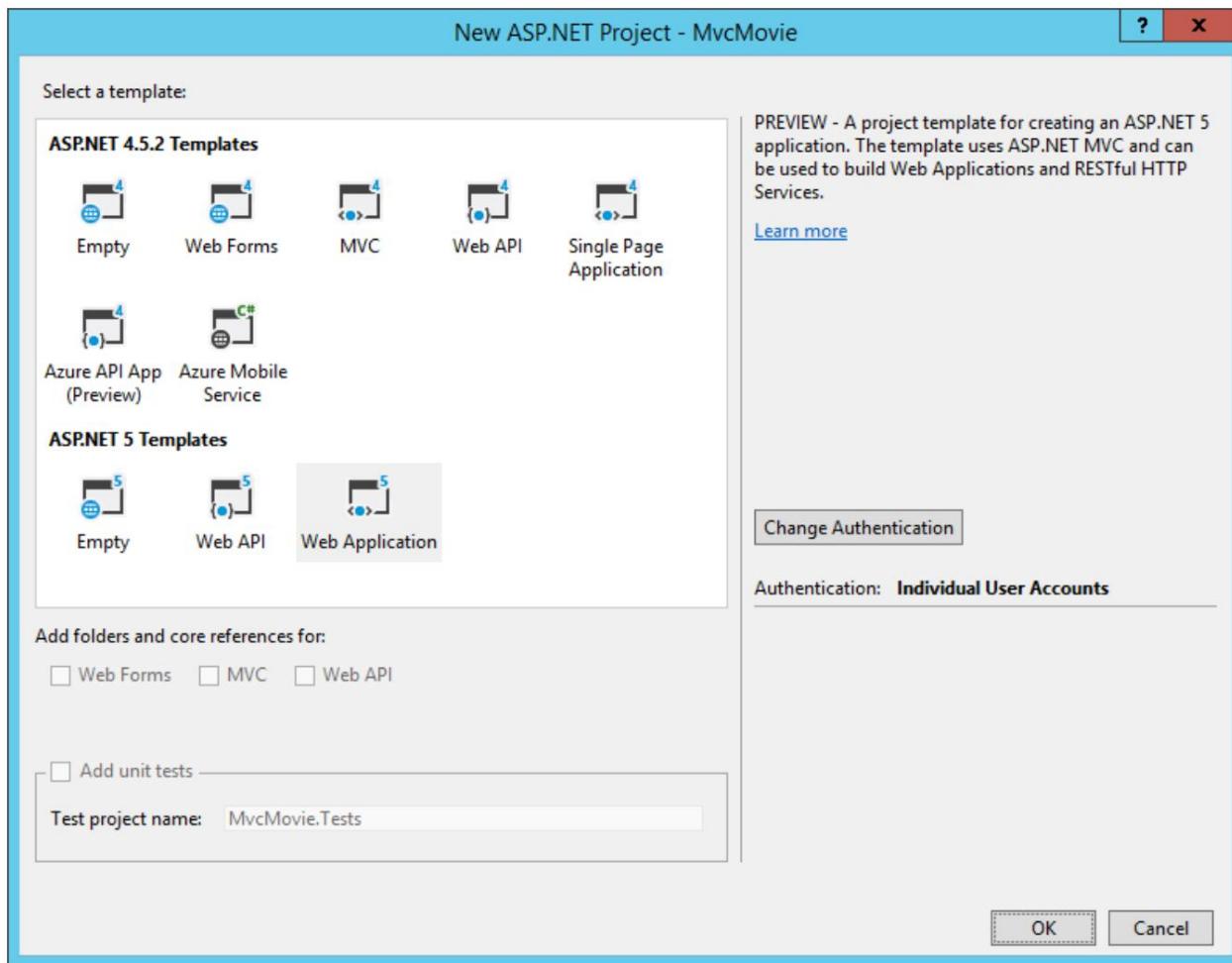


Hoàn thành hộp thoại Dự án mới :

- Trong ngăn bên trái, nhấp vào **Web**
- Trong ngăn trung tâm, nhấp vào **Ứng dụng web ASP.NET**
- Đặt tên cho dự án là **"MvcMovie"** (Điều quan trọng là đặt tên cho dự án là **"MvcMovie"** để khi bạn sao chép mã, không gian tên sẽ phù hợp.)
- Nhấn **OK**



Trong hộp thoại Dự án ASP.NET Mới - MvcMovie , chạm vào Ứng dụng web, sau đó chạm vào OK.



Visual Studio đã sử dụng mẫu mặc định cho dự án MVC mà bạn vừa tạo, vì vậy bạn có một ứng dụng hoạt động ngay bây giờ bằng cách nhập tên dự án và chọn một vài tùy chọn. Đây là một câu đơn giản "Xin chào Thế giới!" và đó là một nơi tốt để bắt đầu,

Nhấn F5 để chạy ứng dụng ở chế độ gỡ lỗi hoặc Ctrl-F5 ở chế độ không gỡ lỗi.

MvcMovie

Bring in libraries from NuGet, Bower, Packages | NuGet npm Bower Gulp and npm, and automate tasks using Grunt or Gulp. [Learn More](#)

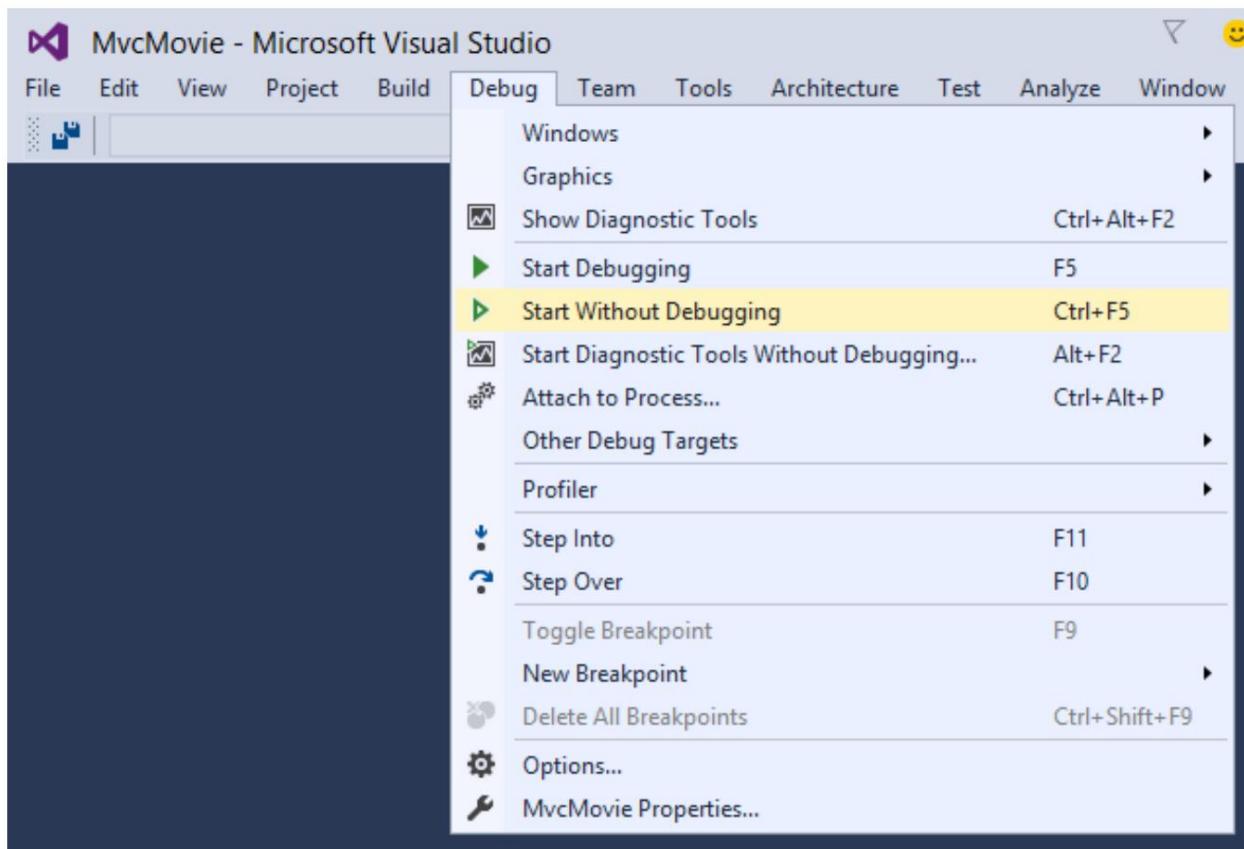
Application uses

- Sample pages using ASP.NET 5 (MVC 6)
- [Gulp](#) and [Bower](#) for managing client-side resources
- Theming using [Bootstrap](#)

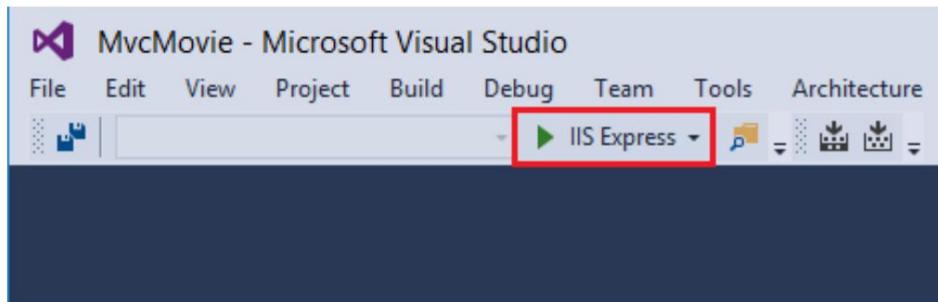
New concepts

- [Conceptual overview of ASP.NET 5](#)
- [Fundamentals in ASP.NET 5](#)
- [Client-Side Development using npm, Bower and Gulp](#)
- [Develop on different platforms](#)

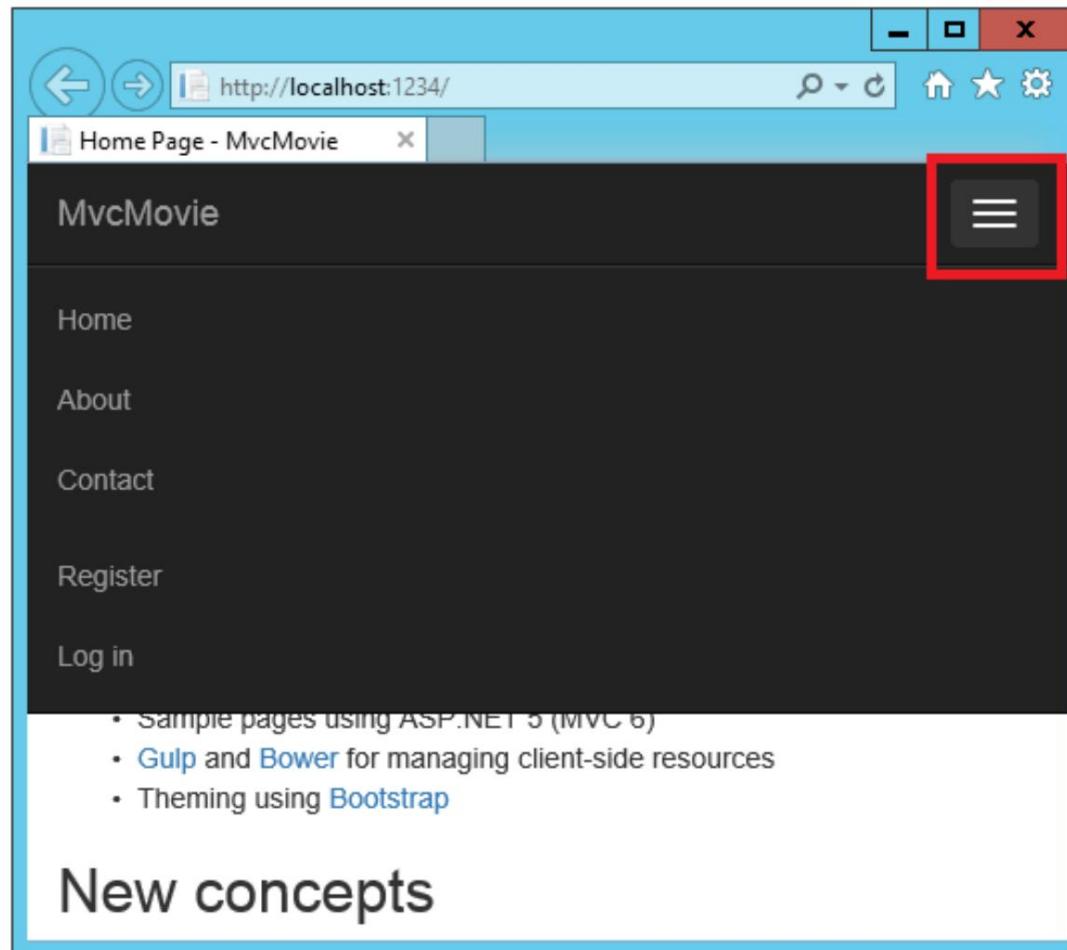
- Visual Studio khởi động IIS Express và chạy ứng dụng của bạn. Lưu ý rằng thanh địa chỉ hiển thị localhost: port # chứ không phải là một cái gì đó giống như example.com. Đó là bởi vì localhost luôn trỏ đến máy tính cục bộ của riêng bạn, trong trường hợp này đang chạy ứng dụng bạn vừa tạo. Khi Visual Studio tạo một dự án web, một cổng ngẫu nhiên được sử dụng cho máy chủ web. Trong hình trên, số cổng là 1234. Khi chạy ứng dụng, bạn sẽ thấy một số cổng khác.
- Khởi chạy ứng dụng với Ctl-F5 (chế độ không gõ lỗi) cho phép bạn thực hiện thay đổi mã, lưu tệp, làm mới trình duyệt và xem các thay đổi mã. Nhiều nhà phát triển thích sử dụng chế độ không gõ lỗi để nhanh chóng khởi chạy ứng dụng và xem các thay đổi.
- Bạn có thể khởi chạy ứng dụng ở chế độ gõ lỗi hoặc không gõ lỗi từ mục menu Gõ lỗi :



- Bạn có thể gỡ lỗi ứng dụng bằng cách nhấp vào nút IIS Express



Ngay ra khỏi hộp, mẫu mặc định cung cấp cho bạn các trang Trang chủ, Liên hệ, Giới thiệu, Đăng ký và Đăng nhập. Hình ảnh trình duyệt ở trên không hiển thị các liên kết của đề tài. Tùy thuộc vào kích thước trình duyệt của bạn, bạn có thể cần phải nhấp vào biểu tượng điều hướng để hiển thị chúng.



Trong phần tiếp theo của hướng dẫn này, chúng ta sẽ tìm hiểu về MVC và bắt đầu viết một số mã.

2.1.2 Thêm bộ điều khiển

Bởi Rick Anderson

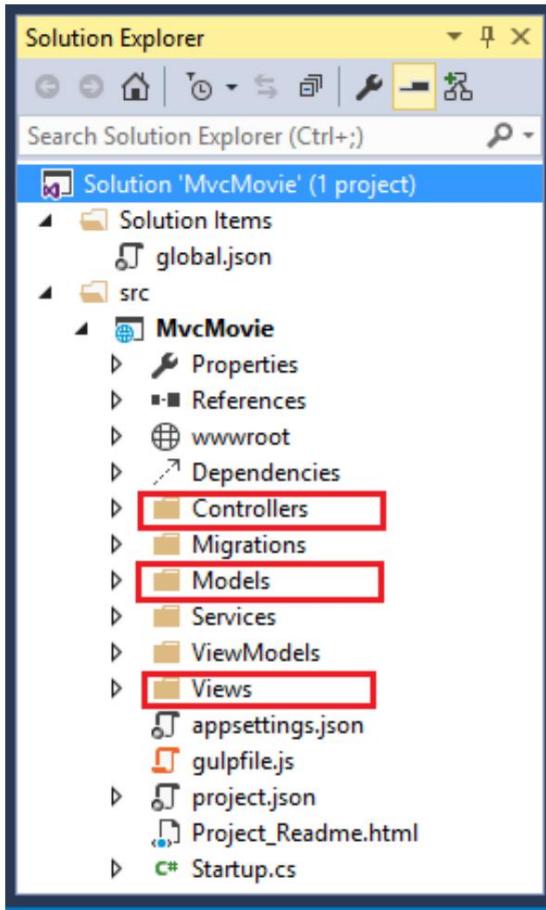
Mô hình kiến trúc Model-View-Controller (MVC) phân tách ứng dụng thành ba thành phần chính: Model, View và Controller. Mô hình MVC giúp bạn tạo các ứng dụng có thể kiểm tra được và dễ bảo trì và cập nhật hơn các ứng dụng nguyên khôi truyền thống. Các ứng dụng dựa trên MVC chứa:

- Mô hình: Các lớp đại diện cho dữ liệu của ứng dụng và sử dụng logic xác thực để thực thi các quy tắc kinh doanh cho dữ liệu đó. Thông thường, các đối tượng mô hình truy xuất và lưu trữ trạng thái mô hình trong cơ sở dữ liệu. Trong hướng dẫn này, Mô hình phim lấy dữ liệu phim từ cơ sở dữ liệu, cung cấp cho chế độ xem hoặc cập nhật. Dữ liệu cập nhật được ghi vào cơ sở dữ liệu SQL Server.
- Chế độ xem: Chế độ xem là các thành phần hiển thị giao diện người dùng (UI) của ứng dụng. Nói chung, giao diện người dùng này hiển thị dữ liệu mô hình.
- Bộ điều khiển: Các lớp xử lý các yêu cầu của trình duyệt, truy xuất dữ liệu mô hình, sau đó chỉ định các mẫu xem trả về phản hồi cho trình duyệt. Trong ứng dụng MVC, dạng xem chỉ hiển thị thông tin; bộ điều khiển xử lý và phản hồi thông tin đầu vào và tương tác của người dùng. Ví dụ: bộ điều khiển xử lý dữ liệu truyền đường và các giá trị chuỗi truy vấn, đồng thời chuyển các giá trị này tới mô hình. Mô hình có thể sử dụng các giá trị này để truy vấn cơ sở dữ liệu.

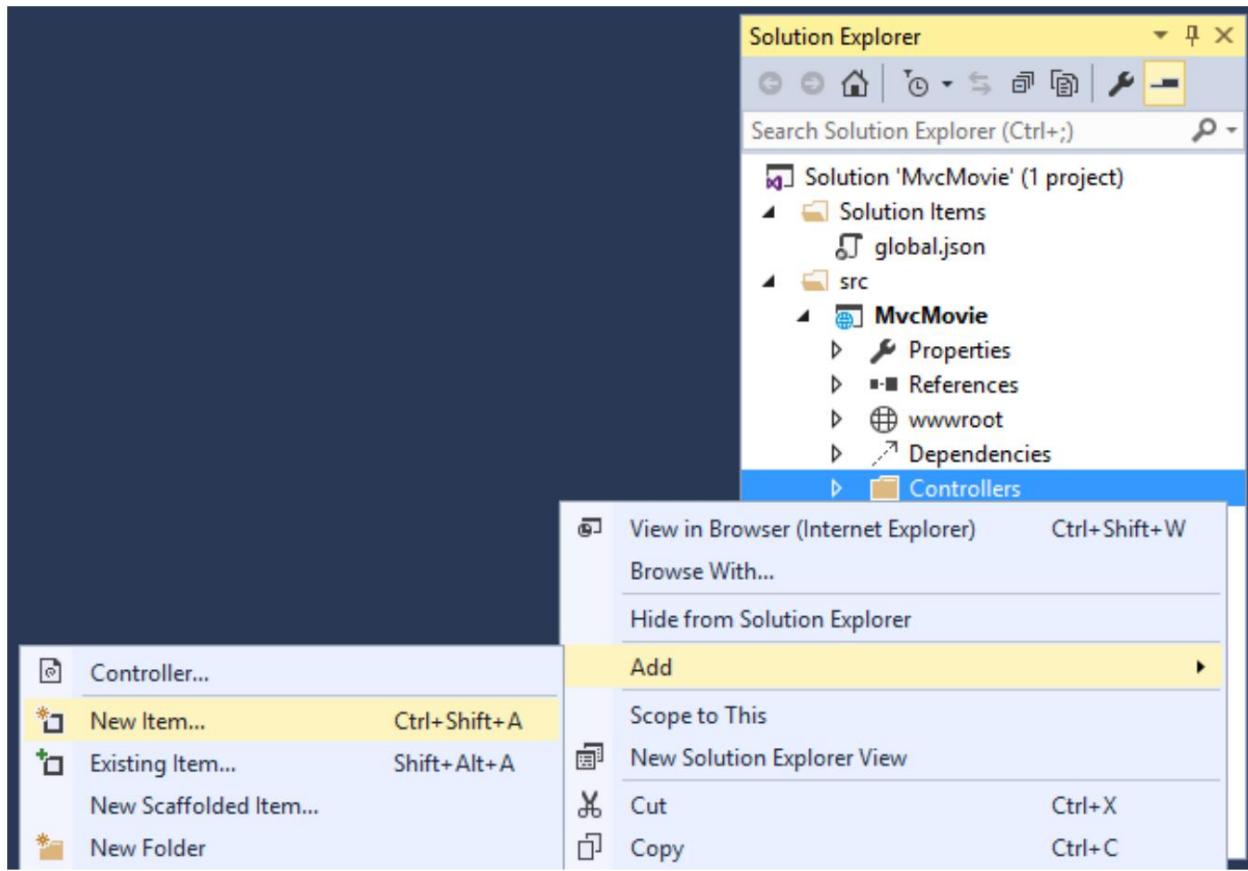
Mô hình MVC giúp bạn tạo các ứng dụng tách biệt các khía cạnh khác nhau của ứng dụng (logic đầu vào, nghiệp vụ

logic và logic giao diện người dùng), đồng thời cung cấp một khung nối lồng lêo giữa các phần tử này. Mẫu chỉ định vị trí của từng loại logic trong ứng dụng. Logic giao diện người dùng thuộc về chế độ xem. Logic đầu vào thuộc bộ điều khiển. Logic nghiệp vụ thuộc về mô hình. Sự tách biệt này giúp bạn quản lý sự phức tạp khi xây dựng một ứng dụng, vì nó cho phép bạn làm việc trên một khía cạnh của việc triển khai tại một thời điểm mà không ảnh hưởng đến mã của một ứng dụng khác. Ví dụ: bạn có thể làm việc trên mã chế độ xem mà không phụ thuộc vào mã logic nghiệp vụ.

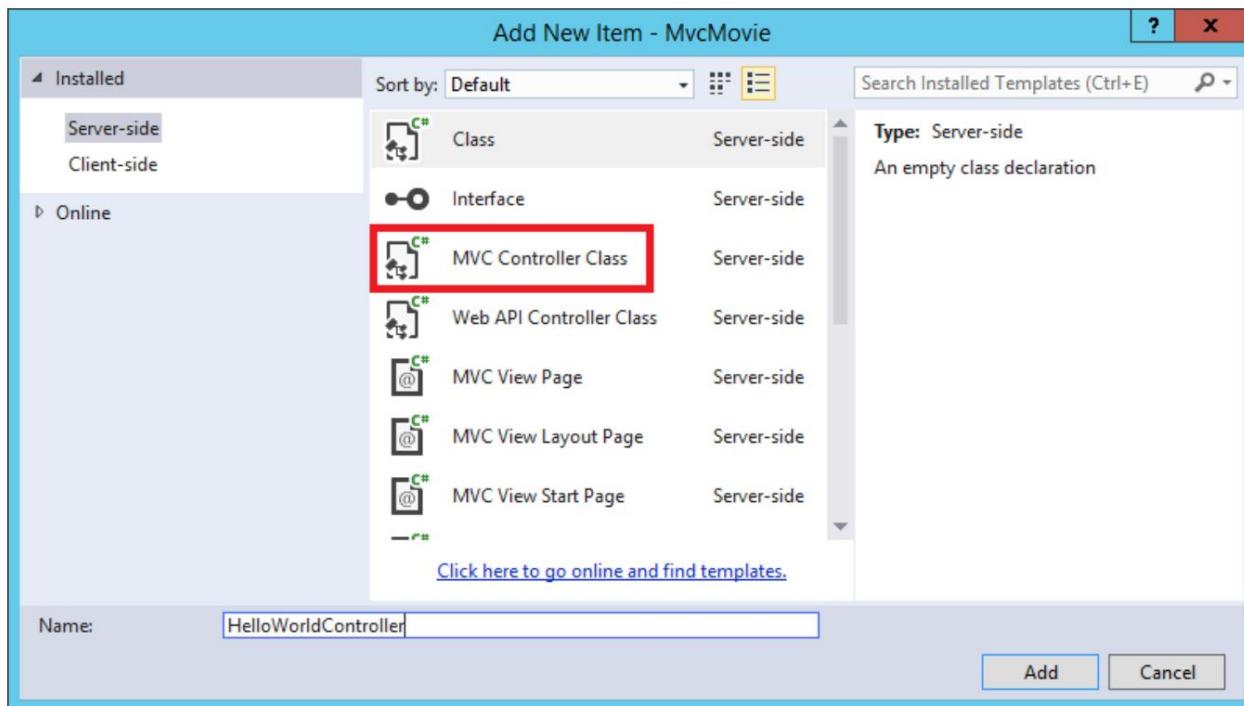
Chúng tôi sẽ trình bày tất cả các khái niệm này trong loạt bài hướng dẫn này và chỉ cho bạn cách sử dụng chúng để xây dựng một ứng dụng phim đơn giản. Hình ảnh sau đây cho thấy các thư mục Mô hình, Chế độ xem và Bộ điều khiển trong dự án MVC.



- Trong Giải pháp Explorer, bấm chuột phải vào Bộ điều khiển, sau đó Thêm> Mục mới.



- Trong hộp thoại Thêm mục mới - Phím
 - Nhấn Lớp điều khiển MVC
 - Nhập tên "HelloWorldController"
 - Nhấn vào Thêm



Thay thế nội dung của Controllers / HelloWorldController.cs bằng nội dung sau:

```

1 sử dụng Microsoft.Extensions.WebEncoders;
2
3 không gian tên MvcMovie.Controllers
4 {
5     public class HelloWorldController : Controller
6     {
7         //
8         // NHẬN: / HelloWorld /
9
10        chuỗi công khai Chỉ mục ()
11        {
12            return "Đây là hành động mặc định của tôi ...";
13        }
14
15        //
16        // NHẬN: / HelloWorld / Welcome /
17
18        chuỗi công khai Chào mừng ()
19        {
20            return "Đây là phương thức hành động Chào mừng ...";
21        }
22    }
23}

```

Mọi phương thức công khai trong bộ điều khiển đều có thể gọi được. Trong ví dụ trên, cả hai phương thức đều trả về một chuỗi. Lưu ý nhận xét trước mỗi phương pháp:

```

1 lớp công khai HelloWorldController : Bộ điều khiển
2 {
3     //
4     // NHẬN: / HelloWorld /
5

```

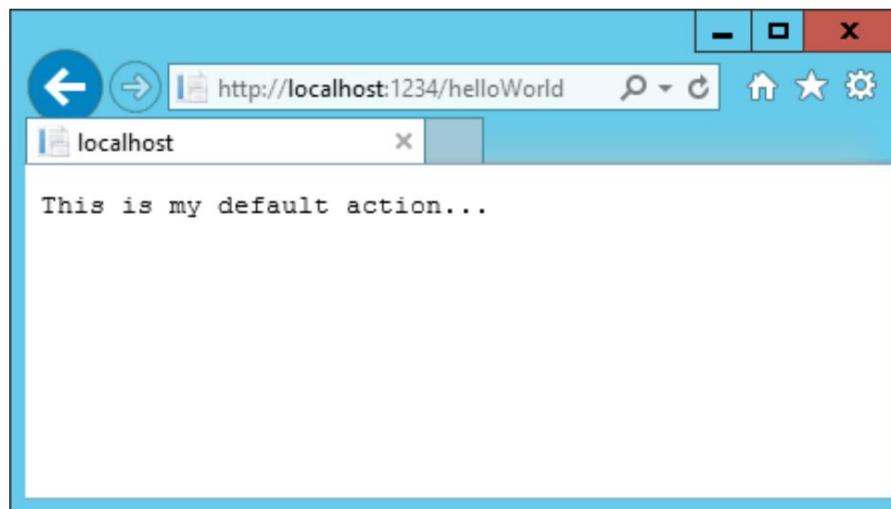
```

6     chuỗi công khai Chỉ mục ()
7     {
8         return "Đây là hành động mặc định của tôi ...";
9     }
10
11    //
12    // NHẬN: / HelloWorld / Welcome /
13
14    chuỗi công khai Chào mừng ()
15    {
16        return "Đây là phương thức hành động Chào mừng ...";
17    }
18 }
```

Nhận xét đầu tiên cho biết đây là **HTTP GET** phương thức được gọi bằng cách thêm "/ HelloWorld /" vào URL. Các nhận xét thứ hai chỉ định một **HTTP GET** phương thức được gọi bằng cách thêm "/ HelloWorld / Welcome /" vào URL. Ở phần sau của hướng dẫn, chúng tôi sẽ sử dụng công cụ giàn giáo để tạo các phương thức **HTTP POST**.

Hãy thử nghiệm các phương pháp này bằng trình duyệt.

Chạy ứng dụng ở chế độ không lỗi (nhấn **Ctrl + F5**) và thêm "HelloWorld" vào đường dẫn trong thanh địa chỉ. (Trong hình ảnh bên dưới, <http://localhost:1234>HelloWorld> được sử dụng, nhưng bạn sẽ phải thay thế 1234 bằng số cổng của ứng dụng.)
Phương thức Index trả về một chuỗi. Bạn đã yêu cầu hệ thống trả về một số HTML, và nó đã thực hiện!



MVC gọi các lớp bộ điều khiển (và các phương thức hành động bên trong chúng) tùy thuộc vào URL đến. Mặc định logic định tuyến URL được MVC sử dụng sử dụng định dạng như thế này để xác định mã nào sẽ gọi:

/ [Bộ điều khiển] / [Tên hành động] / [Thông số]

Bạn đặt định dạng cho định tuyến trong tệp Startup.cs.

```

1     app.UseMvc (các tuyến đường =>
2     {
3         các tuyến đường.MapRoute (
4             tên: "mặc định",
5             mẫu: "{controller = Home} / {action = Index} / {id?}");
6     });

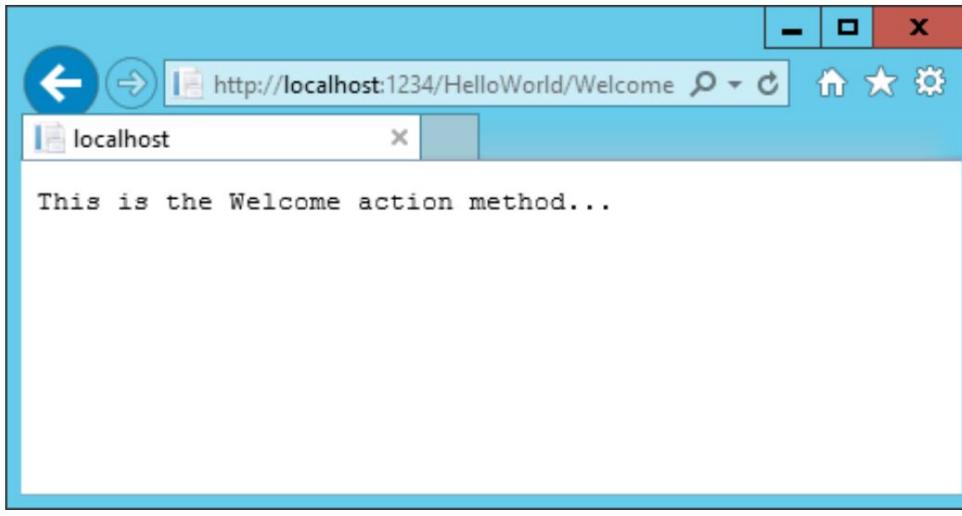
```

Khi bạn chạy ứng dụng và không cung cấp bất kỳ phân đoạn URL nào, ứng dụng đó sẽ được mặc định là bộ điều khiển "Trang chủ" và "Chỉ mục" phương thức được chỉ định trong dòng mẫu được đánh dấu ở trên.

Phân đoạn URL đầu tiên xác định lớp bộ điều khiển để chạy. Vì vậy, bản đồ localhost: xxxx / HelloWorld

vào lớp `HelloWorldController`. Phần thứ hai của phân đoạn URL xác định hành động trên lớp. Vì vậy, `localhost: xxxx / HelloWorld / Index` sẽ gây ra phương pháp `Index` của lớp `HelloWorldController` để chạy. Lưu ý rằng chúng tôi chỉ phải duyệt đến `localhost: xxxx / HelloWorld` và phương thức `Index` được gọi theo mặc định. Điều này là do Chỉ mục là phương thức mặc định sẽ được gọi trên một bộ điều khiển nếu tên phương thức không được chỉ định rõ ràng. Phần thứ ba của phân đoạn URL (Tham số) dành cho dữ liệu truyền đường. Chúng ta sẽ xem dữ liệu truyền đường ở phần sau trong hướng dẫn này.

Duyệt đến `http:// localhost: xxxx / HelloWorld / Welcome`, và trả về chuỗi "Đây là phương thức hành động Chào mừng ...".
 Phương thức Chào mừng chạy
 Định tuyến MVC mặc định là
`/ [Bộ điều khiển] / [Tên hành động] / [Thông số]`. Đối với URL này, bộ điều khiển là `HelloWorld` và Chào mừng là phương pháp hành động. Bạn chưa sử dụng phần [Thông số] của URL.



Hãy sửa đổi ví dụ một chút để bạn có thể chuyển một số thông tin tham số từ URL đến bộ điều khiển (ví dụ: `/ HelloWorld / Welcome? name = Scott & numtimes = 4`). Thay đổi phương thức `Welcome` thành hai tham số như hình dưới đây.
 Lưu ý rằng mã sử dụng tính năng tham số tùy chọn C# để chỉ ra rằng Tham số `numTimes` mặc định là 1 nếu không có giá trị nào được truyền cho tham số đó.

```
1 chuỗi công khai Chào mừng ( tên chuỗi, int numTimes = 1)
2 {
3     trả về HtmlEncoder.Default.HtmlEncode (
4         "Xin chào " + name + ", NumTimes là:" + numTimes);
5 }
```

Lưu ý: Đoạn mã trên sử dụng `HtmlEncoder.Default.HtmlEncode` để bảo vệ ứng dụng khỏi đầu vào độc hại (cụ thể là JavaScript).

Lưu ý: Trong Visual Studio 2015, khi bạn đang chạy mà không gõ lỗi (Ctrl + F5), bạn không cần tạo ứng dụng sau thay đổi mã. Chỉ cần lưu tệp, làm mới trình duyệt của bạn và bạn có thể thấy các thay đổi.

Chạy ứng dụng của bạn và duyệt đến:

`http:// localhost: xxxx / HelloWorld / Welcome? name = Rick & numtimes = 4`

(Thay `xxxx` bằng số cổng của bạn.) Bạn có thể thử các giá trị khác nhau cho tên và số lần trong URL. MVC hệ thống liên kết mô hình tự động ánh xạ các tham số được đặt tên từ chuỗi truy vấn trong thanh địa chỉ thành các tham số trong phương pháp của bạn. Xem [Mô hình ràng buộc để biết thêm thông tin](#).

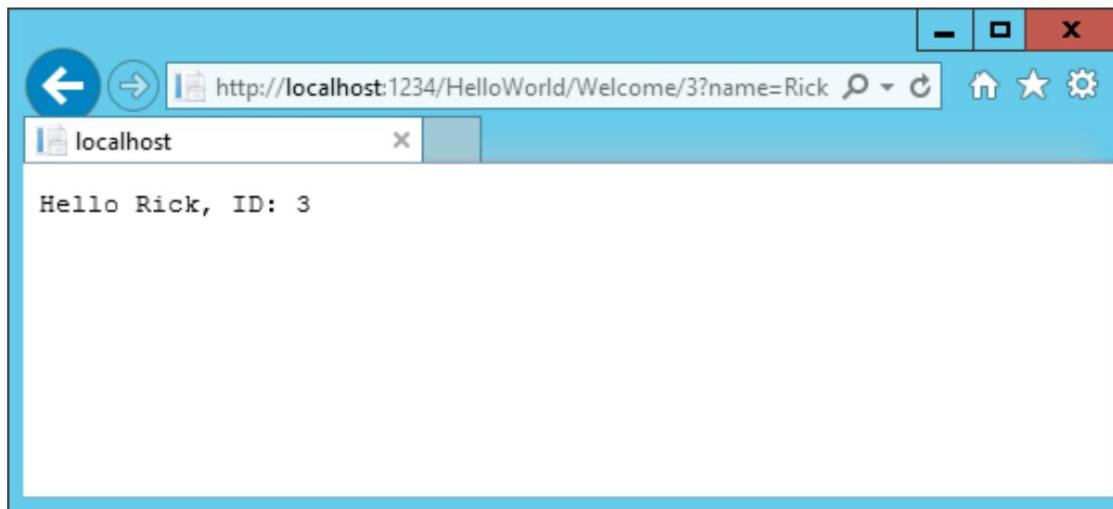


Trong ví dụ trên, phân đoạn URL (Tham số) không được sử dụng, tên và tham số numTimes được chuyển dưới dạng chuỗi truy vấn. Các ? (dấu chấm hỏi) trong URL ở trên là dấu phân tách và các chuỗi truy vấn sau. Nhân vật phân tách các chuỗi truy vấn.

Thay thế phương pháp Chào mừng bằng mã sau:

```
1 chuỗi công khai Chào mừng ( tên chuỗi, int ID = 1)
2 {
3     trả về HtmlEncoder.Default.HtmlEncode (
4         "Xin chào" + tên + ", ID: " + ID);
5 }
```

Chạy ứng dụng và nhập URL sau: http://localhost:xxx/HelloWorld/Welcome/3?Name=Rick



Lần này, phân đoạn URL thứ ba khớp với id thông số truyền đường. Phương thức Chào mừng có chứa một id tham số khớp với mẫu URL trong phương pháp MapRoute. Dấu vết? (trong id?) cho biết tham số id là không bắt buộc.

```
1 app.UseMvc (các tuyến đường =>
2 {
3     các tuyến đường.MapRoute (
4         tên: "mặc định",
5         mẫu: "{controller = Home} / {action = Index} / {id?}");
6 });
```

Trong các ví dụ này, bộ điều khiển đã thực hiện phần "VC" của MVC - nghĩa là chế độ xem và bộ điều khiển hoạt động. Bộ điều khiển đang trả về HTML trực tiếp. Nói chung, bạn không muốn bộ điều khiển trả về HTML trực tiếp, vì điều đó trở nên rất cồng kềnh để viết mã và duy trì. Thay vào đó, chúng tôi thường sử dụng tệp mẫu chế độ xem Razor riêng biệt để giúp tạo phản hồi HTML. Chúng tôi sẽ làm điều đó trong hướng dẫn tiếp theo.

2.1.3 Thêm chế độ xem

Bởi Rick Anderson

Trong phần này, bạn sẽ sửa đổi lớp HelloWorldController để sử dụng tệp mẫu dạng xem Razor để đóng gói gọn gàng quá trình tạo phản hồi HTML cho một ứng dụng khách.

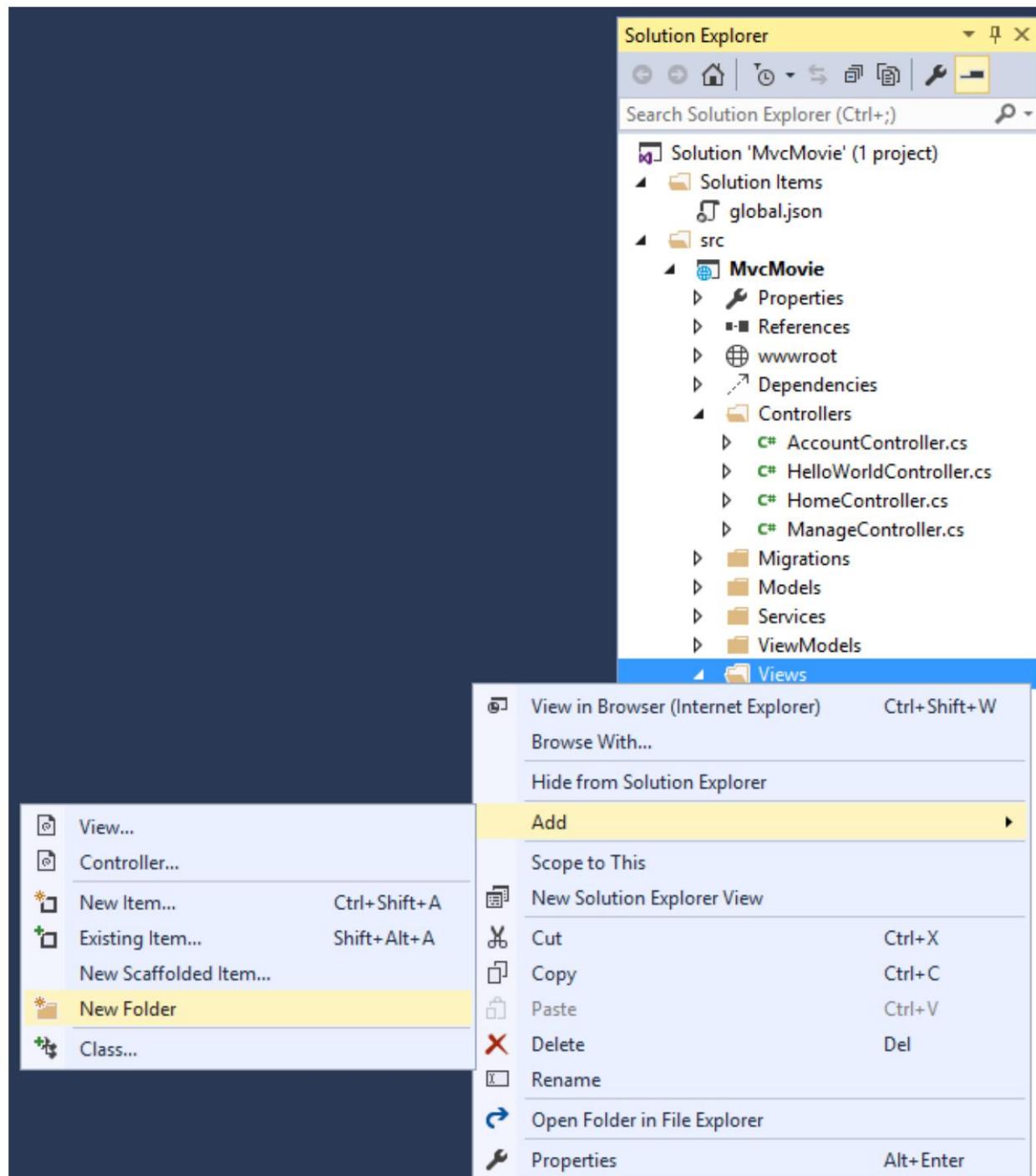
Bạn sẽ tạo tệp mẫu chế độ xem bằng công cụ chế độ xem Razor. Các mẫu chế độ xem dựa trên dao tạo có tệp .cshtml extension và cung cấp một cách đơn giản để tạo đầu ra HTML bằng C#. Razor giảm thiểu số lượng ký tự và tổ hợp phím cần thiết khi viết mẫu chế độ xem và cho phép quy trình mã hóa nhanh chóng, linh hoạt.

Hiện tại, phương thức Index trả về một chuỗi có thông báo được mã hóa cứng trong lớp bộ điều khiển. Thay đổi phương thức Chỉ mục để trả về một đối tượng Dạng xem, như được hiển thị trong đoạn mã sau:

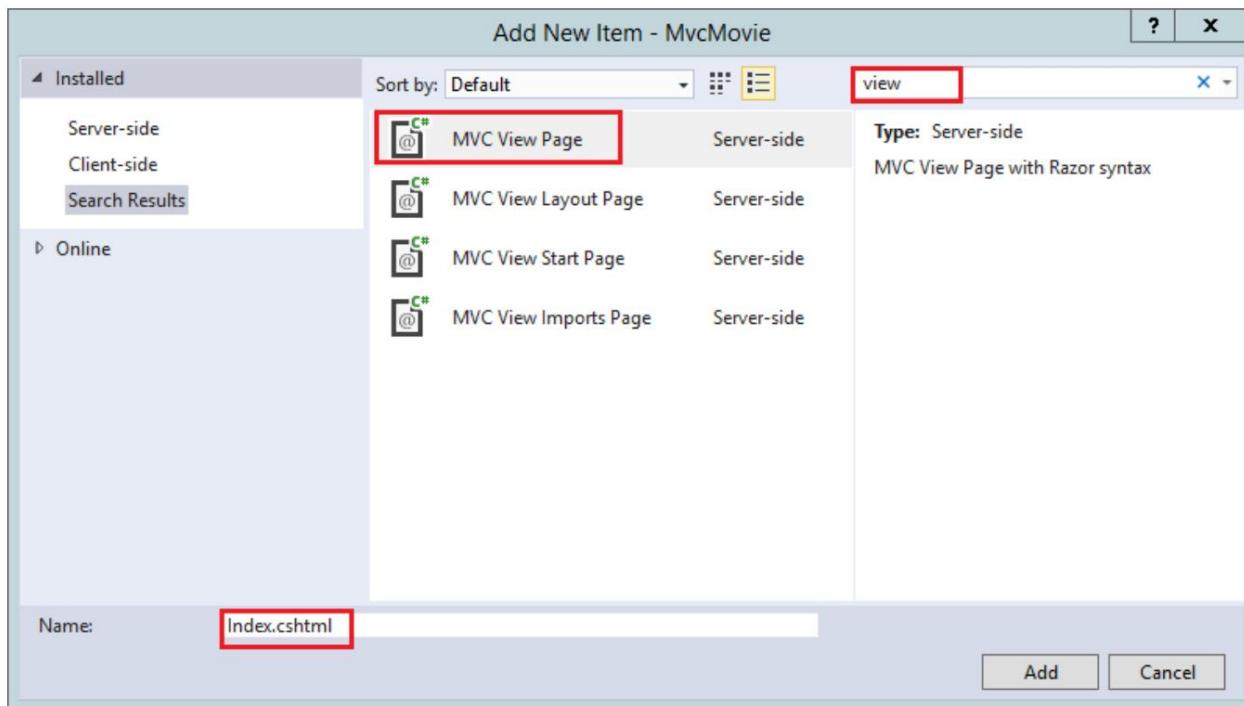
```
public IActionResult Index () {  
    return View ();  
}
```

Phương pháp Chỉ mục ở trên sử dụng mẫu chế độ xem để tạo phản hồi HTML cho trình duyệt. Phương thức bộ điều khiển (còn được gọi là [phương thức hành động](#)), chẳng hạn như phương thức Index ở trên, thường trả về một IActionResult (hoặc một lớp dẫn xuất từ ActionResult), không phải các kiểu nguyên thủy như chuỗi.

- Nhấp chuột phải vào thư mục Lượt xem, sau đó Thêm> Thư mục Mới và đặt tên thư mục là HelloWorld.



- Nhấp chuột phải vào thư mục Views / HelloWorld, sau đó Thêm> Mục mới.
- Trong hộp thoại Thêm mục mới - Phím
 - Trong hộp tìm kiếm ở phía trên bên phải, nhập chế độ xem
 - Nhấn vào MVC View Page
 - Trong hộp Tên , giữ Index.cshtml mặc định
 - Nhấn vào Thêm



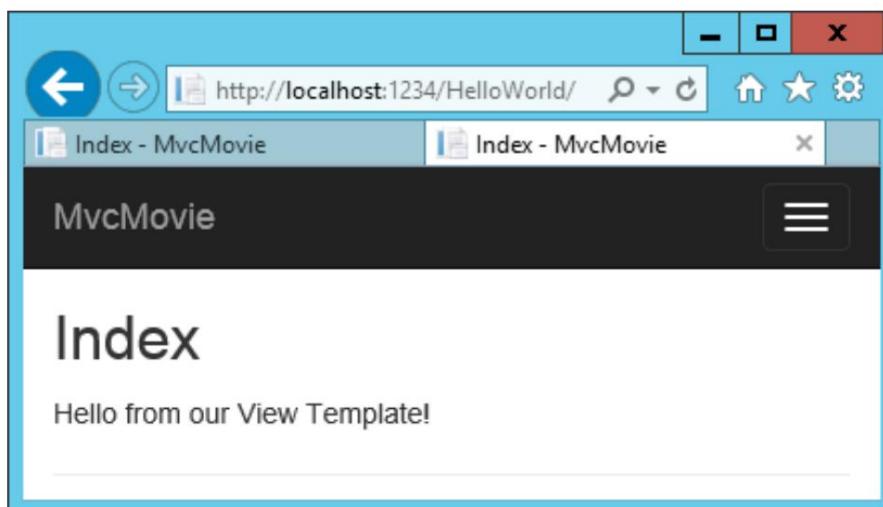
Thay thế nội dung của tệp chế độ xem Views / HelloWorld / Index.cshtml Razor bằng nội dung sau:

```
@ {
    ViewData["Title"] = "Chì mục";
}

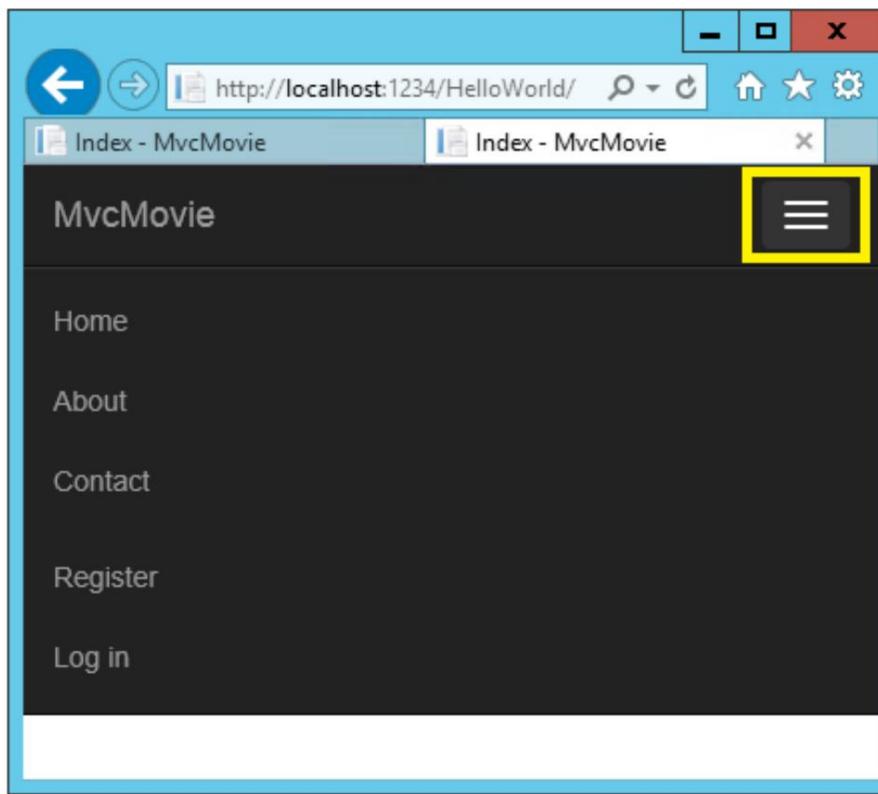
<h2> Chì mục </h2>

<p> Xin chào bạn từ Mẫu Ché độ xem của chúng tôi! </p>
```

Điều hướng đến



Nếu cửa sổ trình duyệt của bạn nhỏ (ví dụ: trên thiết bị di động), bạn có thể cần phải chuyển đổi (nhấn) Bootstrap nút điều hướng ở phía trên bên phải để xem các liên kết Trang chủ, Giới thiệu, Liên hệ, Đăng ký và Đăng nhập.



Thay đổi chế độ xem và trang bố cục

Nhấn vào các liên kết menu (MvcMovie, Trang chủ, Giới thiệu). Mỗi trang hiển thị cùng một bố cục menu. Bố cục menu là được triển khai trong tệp Views / Shared / _Layout.cshtml. Mở tệp Views / Shared / _Layout.cshtml.

Các mẫu bố cục cho phép bạn chỉ định bố cục vùng chứa HTML của trang web của bạn ở một nơi và sau đó áp dụng nó trên nhiều trang trong trang web của bạn. Tìm dòng @RenderBody(). RenderBody là một trình giữ chỗ nơi tất cả các trang cụ thể của chế độ xem mà bạn tạo hiển thị, được "bao bọc" trong trang bố cục. Ví dụ: nếu bạn chọn liên kết Giới thiệu, Chế độ xem Views / Home / About.cshtml được hiển thị bên trong phương thức RenderBody.

Thay đổi nội dung của phần tử tiêu đề. Thay đổi văn bản liên kết trong mẫu bố cục thành "Phim MVC" và bộ điều khiển từ Trang chủ đến Phim như được đánh dấu bên dưới:

```

1 <!DOCTYPE html>
2 <html>
3     <head>
4         <meta charset = "utf-8" />
5         <meta name = "viewport" content = "width = device-width, initial-scale = 1.0" />
6         <title> @ ViewData["Title"] - Ứng dụng phim </title>
7
8         <tên môi trường = "Phát triển">
9             <link rel = "stylesheet" href = "~ / lib / bootstrap / dist / css / bootstrap.css" />
10            <link rel = "stylesheet" href = "~ / css / site.css" />
11        <môi trường>
12            <tên môi trường = "Staging, Production">
13                <link rel = "stylesheet" href = "https://ajax.aspnetcdn.com/ajax/bootstrap/3.3.5/css/bootstrap.css" />

```

```

14         asp-fallback-href = "~ / lib / bootstrap / dist / css / bootstrap.min.css"
15         asp-fallback-test-class = "sr-only" asp-fallback-test-property = "vị trí" asp-fallba
16             <link rel = "stylesheet" href = "~ / css / site.min.css" asp-append-version = "true" />
17             <môî trührung>
18     </head>
19     <body>
20         <div class = "navbar navbar-inverse navbar-fixed-top">
21             <div class = "container">
22                 <div class = "navbar-header">
23                     <button type = "button" class = "navbar-toggle" data-toggle = "thu gọn" data-target =
24                         <span class = "sr-only"> Chuyển đổi điều hướng
25                         <span class = "icon-bar">
26                             <span class = "icon-bar">
27                             <span class = "icon-bar">
28                         </button>
29                         <a asp-controller="Movies" asp-action="Index" class="navbar-brand"> Phim Mvc </a>
30                     </div>
31                     <div class = "navbar-sập sụp đồ">
32                         <ul class = "nav navbar-nav">
33                             <li> <a asp-controller="Home" asp-action="Index"> Trang chủ </a> </li>
34                             <li> <a asp-controller="Home" asp-action="About"> Giới thiệu </a> </li>
35                             <li> <a asp-controller="Home" asp-action="Contact"> Liên hệ </a> </li>
36                         </ul>
37                         @await Html.PartialAsync ("_ LoginPartial")
38                     </div>
39                 </div>
40             <div class = "container body-content">
41                 @RenderBody ()
42                 <giờ />
43                 <footer>
44                     <p> & sao chép; 2015 - MvcMovie </p>
45                 </footer>
46             </div>
47
48             < tên môî trührung = "Phát triển">
49                 <script src = "~ / lib / jquery / dist / jquery.js"> </script>
50                 <script src = "~ / lib / bootstrap / dist / js / bootstrap.js"> </script>
51                 <script src = "~ / js / site.js" asp-append-version = "true"> </script>
52             <môî trührung>
53             < tên môî trührung = "Staging, Production">
54                 <script src = "https://ajax.aspnetcdn.com/ajax/jquery/jquery-2.1.4.min.js"
55                     asp-fallback-src = "~ / lib / jquery / dist / jquery.min.js"
56                     asp-fallback-test = "window.jQuery">
57                 </script>
58                 <script src = "https://ajax.aspnetcdn.com/ajax/bootstrap/3.3.5/bootstrap.min.js"
59                     asp-fallback-src = "~ / lib / bootstrap / dist / js / bootstrap.min.js"
60                     asp-fallback-test = "window.jQuery && window.jQuery.fn && window.jQuery.fn.modal">
61                 </script>
62                 <script src = "~ / js / site.min.js" asp-append-version = "true"> </script>
63             <môî trührung>
64
65             @RenderSection ("script", bắt buộc: false)
66         </body>
67     </html>

```

Lưu ý: Chúng tôi chưa triển khai bộ điều khiển Phim, vì vậy nếu bạn nhấp vào liên kết đó, bạn sẽ gặp lỗi.

Lưu các thay đổi của bạn và nhấn vào liên kết Giới thiệu . Chú ý cách mỗi trang hiển thị liên kết PhimMvc . Chúng tôi có thể thực hiện thay đổi một lần trong mẫu bố cục và có tất cả các trang trên trang web phản ánh văn bản liên kết mới và tiêu đề mới.

Kiểm tra tệp Views / _ViewStart.cshtml:

```
@ {
    Bố cục = "_Layout";
}
```

Tệp Views / _ViewStart.cshtml đưa tệp Views / Shared / _Layout.cshtml vào mỗi chế độ xem. Bạn có thể sử dụng thuộc tính Bố cục để đặt một dạng xem bố cục khác hoặc đặt nó thành null để không có tệp bố cục nào được sử dụng.

Bây giờ, hãy thay đổi tiêu đề của dạng xem Chỉ mục.

Mở Views / HelloWorld / Index.cshtml. Có hai nơi để thực hiện thay đổi:

- Văn bản xuất hiện trong tiêu đề của trình duyệt
- Tiêu đề phụ (phần tử <h2>).

Bạn sẽ làm cho chúng hơi khác một chút để bạn có thể thấy đoạn mã nào thay đổi phần nào của ứng dụng.

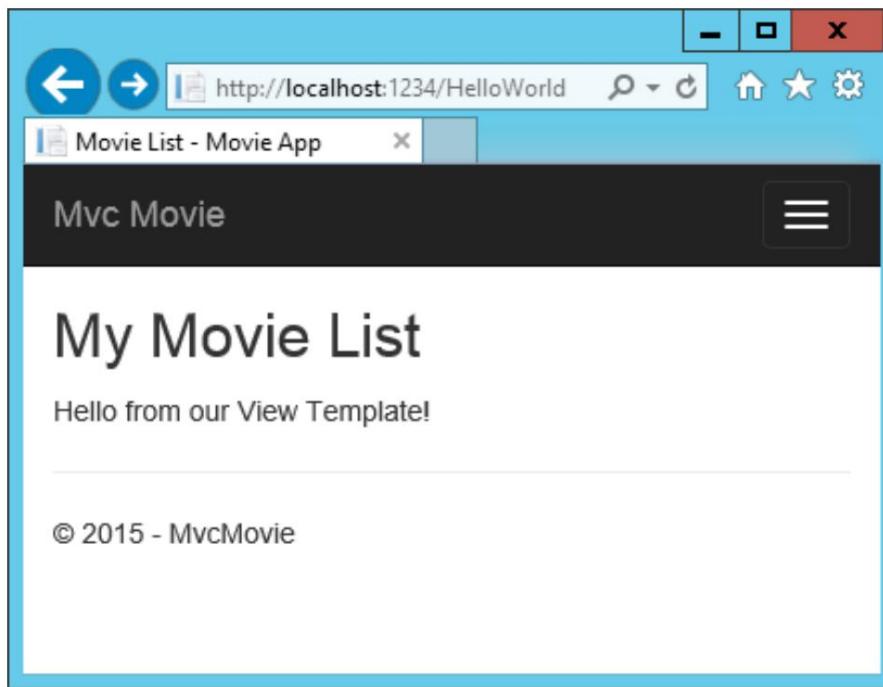
```
1 @ {
2     ViewData["Title"] = "Danh sách phim";
3 }
4
5 <h2> Danh sách phim của tôi </h2>
6
7 <p> Xin chào bạn từ Mẫu Chế độ xem của chúng tôi! </p>
```

Dòng số 2 trong đoạn mã trên đặt thuộc tính Title của ViewDataDictionary vào "Danh sách phim". Thuộc tính Title được sử dụng trong phần tử HTML <title> trong trang bố cục:

```
<title> @ViewData["Title"] - Ứng dụng phim </title>
```

Lưu thay đổi của bạn và làm mới trang. Lưu ý rằng tiêu đề trình duyệt, tiêu đề chính và tiêu đề phụ đã thay đổi. (Nếu bạn không thấy các thay đổi trong trình duyệt, bạn có thể đang xem nội dung được lưu trong bộ nhớ cache. Nhấn Ctrl + F5 trong trình duyệt của bạn để buộc tải phản hồi từ máy chủ.) Tiêu đề trình duyệt được tạo bằng ViewData ["Title"] chúng tôi đặt trong mẫu xem Index.cshtml và bổ sung "- Ứng dụng Phim" được thêm vào tệp bố cục.

Cũng lưu ý cách nội dung trong mẫu chế độ xem Index.cshtml được hợp nhất với mẫu chế độ xem Views / Shared / _Layout.cshtml và một phản hồi HTML duy nhất đã được gửi đến trình duyệt. Các mẫu bố cục giúp bạn thực sự dễ dàng thực hiện các thay đổi áp dụng trên tất cả các trang trong ứng dụng của mình.



Tuy nhiên, một chút "dữ liệu" của chúng tôi (trong trường hợp này là thông báo "Xin chào từ Mẫu xem của chúng tôi!") Được mã hóa cứng. Ứng dụng MVC có "V" (lượt xem) và bạn đã có "C" (bộ điều khiển), nhưng chưa có "M" (kiểu máy). Trong thời gian ngắn, chúng ta sẽ hướng dẫn cách tạo cơ sở dữ liệu và truy xuất dữ liệu mô hình từ nó.

Truyền dữ liệu từ Bộ điều khiển đến Chế độ xem

Tuy nhiên, trước khi chúng ta đi đến cơ sở dữ liệu và nói về các mô hình, trước tiên chúng ta hãy nói về việc chuyển thông tin từ bộ điều khiển sang dạng xem. Các lớp bộ điều khiển được gọi để đáp lại một yêu cầu URL đến. Lớp bộ điều khiển là nơi bạn viết mã để xử lý các yêu cầu trình duyệt đến, truy xuất dữ liệu từ cơ sở dữ liệu và cuối cùng quyết định loại phản hồi nào sẽ gửi lại cho trình duyệt. Sau đó, các mẫu xem có thể được sử dụng từ bộ điều khiển để tạo và định dạng phản hồi HTML cho trình duyệt.

Bộ điều khiển chịu trách nhiệm cung cấp bất kỳ dữ liệu hoặc đối tượng nào được yêu cầu để mẫu chế độ xem hiển thị phản hồi cho trình duyệt. Phương pháp hay nhất: Mẫu dạng xem không bao giờ được thực hiện logic nghiệp vụ hoặc tương tác trực tiếp với cơ sở dữ liệu. Thay vào đó, mẫu chế độ xem chỉ nên hoạt động với dữ liệu được bộ điều khiển cung cấp cho nó.

Duy trì "sự tách biệt các mối quan tâm" này giúp giữ cho mã của bạn sạch sẽ, có thể kiểm tra và dễ bảo trì hơn.

Hiện tại, phương thức Welcome trong lớp HelloWorldController nhận một tên và một tham số numTimes, sau đó xuất các giá trị trực tiếp đến trình duyệt. Thay vì để bộ điều khiển hiển thị phản hồi này dưới dạng chuỗi, hãy thay đổi bộ điều khiển để sử dụng mẫu chế độ xem thay thế. Mẫu chế độ xem sẽ tạo phản hồi động, có nghĩa là bạn cần chuyển các bit dữ liệu thích hợp từ bộ điều khiển đến chế độ xem để tạo phản hồi. Bạn có thể thực hiện việc này bằng cách yêu cầu bộ điều khiển đặt dữ liệu động (tham số) mà mẫu chế độ xem cần vào từ diễn biến ViewData mà mẫu chế độ xem sau đó có thể truy cập.

Quay lại tệp HelloWorldController.cs và thay đổi phương pháp Chào mừng để thêm giá trị Thông báo và NumTimes vào từ diễn biến ViewData. Từ diễn biến ViewData là một đối tượng động, có nghĩa là bạn có thể đưa bất cứ thứ gì bạn muốn vào nó; đối tượng ViewData không có thuộc tính xác định nào cho đến khi bạn đặt thứ gì đó vào bên trong nó. Hệ thống ràng buộc mô hình MVC tự động ánh xạ các tham số được đặt tên (tên và numTimes) từ chuỗi truy vấn trong thanh địa chỉ đến các tham số trong phương thức của bạn. Tệp HelloWorldController.cs hoàn chỉnh trông giống như sau:

```
sử dụng Microsoft.AspNet.Mvc;

không gian tên MvcMovie.Controllers
```

```
{
    public class HelloWorldController : Controller {

        public IActionResult Index () {
            return View ();
        }

        public IActionResult Chào mừng (string name, int numTimes = 1) {
            ViewData ["Message"] = "Xin chào" + tên;
            ViewData ["NumTimes"] = numTimes;

            return View ();
        }
    }
}
```

Đối tượng từ điển ViewData chứa dữ liệu sẽ được chuyển đến dạng xem. Tiếp theo, bạn cần một mẫu dạng xem Chào mừng.

- Nhấp chuột phải vào thư mục Views / HelloWorld, sau đó Thêm> Mục mới.
- Trong hộp thoại Thêm mục mới - Phím
 - Trong hộp tìm kiếm ở phía trên bên phải, nhập chế độ xem
 - Nhấn vào MVC View Page
 - Trong hộp Tên , nhập Welcome.cshtml
 - Nhấn vào Thêm

Bạn sẽ tạo một vòng lặp trong mẫu xem Welcome.cshtml hiển thị NumTimes "Xin chào". Thay thế nội dung của Views / HelloWorld / Welcome.cshtml bằng nội dung sau:

```
@ {
    ViewData ["Title"] = "Giới thiệu";
}

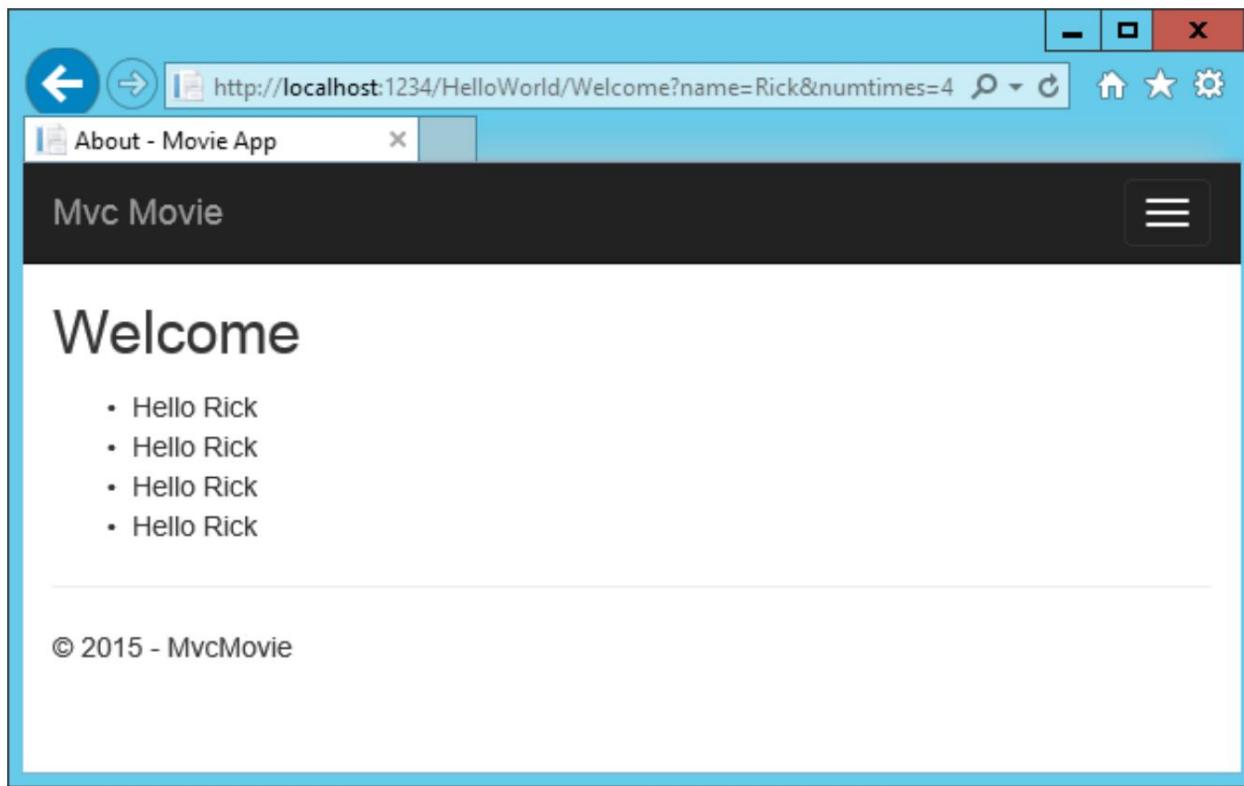
<h2> Chào mừng </h2>

<ul>
    @for (int i = 0; i <(int) ViewData ["NumTimes"]; i++) {
        <li> @ViewData ["Tin nhắn"] </li>
    }
}</ul>
```

Lưu các thay đổi của bạn và duyệt đến URL sau:

```
http://localhost:xxxx / HelloWorld / Welcome? name = Rick & numtimes = 4
```

Dữ liệu được lấy từ URL và chuyển đến bộ điều khiển bằng cách sử dụng **chất kết dính mô hình**. Bộ điều khiển gói dữ liệu vào một từ điển ViewData và chuyển đối tượng đó đến chế độ xem. Sau đó, chế độ xem hiển thị dữ liệu dưới dạng HTML cho trình duyệt.



Trong ví dụ trên, chúng tôi đã sử dụng từ điển ViewData để chuyển dữ liệu từ bộ điều khiển sang một chế độ xem. Phần sau của hướng dẫn này, chúng ta sẽ sử dụng mô hình khung nhìn để truyền dữ liệu từ bộ điều khiển sang một khung nhìn. Cách tiếp cận mô hình xem để chuyển dữ liệu thường được ưa thích hơn nhiều so với cách tiếp cận từ điển ViewData. Xem các chế độ xem được nhập mạnh động V để biết thêm thông tin.

Chà, đó là một loại chữ "M" cho mô hình, nhưng không phải là loại cơ sở dữ liệu. Hãy lấy những gì chúng ta đã học và tạo ra một cơ sở dữ liệu về phim.

2.1.4 Thêm một mô hình

Bởi Rick Anderson

Trong phần này, bạn sẽ thêm một số lớp để quản lý phim trong cơ sở dữ liệu. Các lớp này sẽ là phần "Mô hình" của ứng dụng MVC.

Bạn sẽ sử dụng công nghệ truy cập dữ liệu .NET Framework được gọi là Entity Framework để định nghĩa và làm việc với các lớp mô hình này. Khung thực thể (thường được gọi là EF) hỗ trợ một mô hình phát triển được gọi là Code First.

Code First cho phép bạn tạo các đối tượng mô hình bằng cách viết các lớp đơn giản. (Chúng còn được gọi là các lớp POCO, từ "các đối tượng CLR thuần túy".) Sau đó, bạn có thể tạo cơ sở dữ liệu ngay lập tức từ các lớp của mình, điều này cho phép quy trình phát triển rất sạch sẽ và nhanh chóng. Nếu bạn được yêu cầu tạo cơ sở dữ liệu trước, bạn vẫn có thể làm theo hướng dẫn này để tìm hiểu về cách phát triển ứng dụng MVC và EF.

Thêm lớp mô hình

Trong giải pháp Explorer, nhấp chuột phải vào thư mục Mô hình> Thêm > Lớp.

1 đang sử dụng Hệ thống;

2

Phim cấp 3 công

```

4 {
5     public int ID { get; bộ; }
6     chuỗi công khai Tiêu đề { get; bộ; }
7     public DateTime ReleaseDate { get; bộ; }
8     chuỗi công khai Thể loại { get; bộ; }
9     giá thậm phán công khai { get; bộ; }
10
}

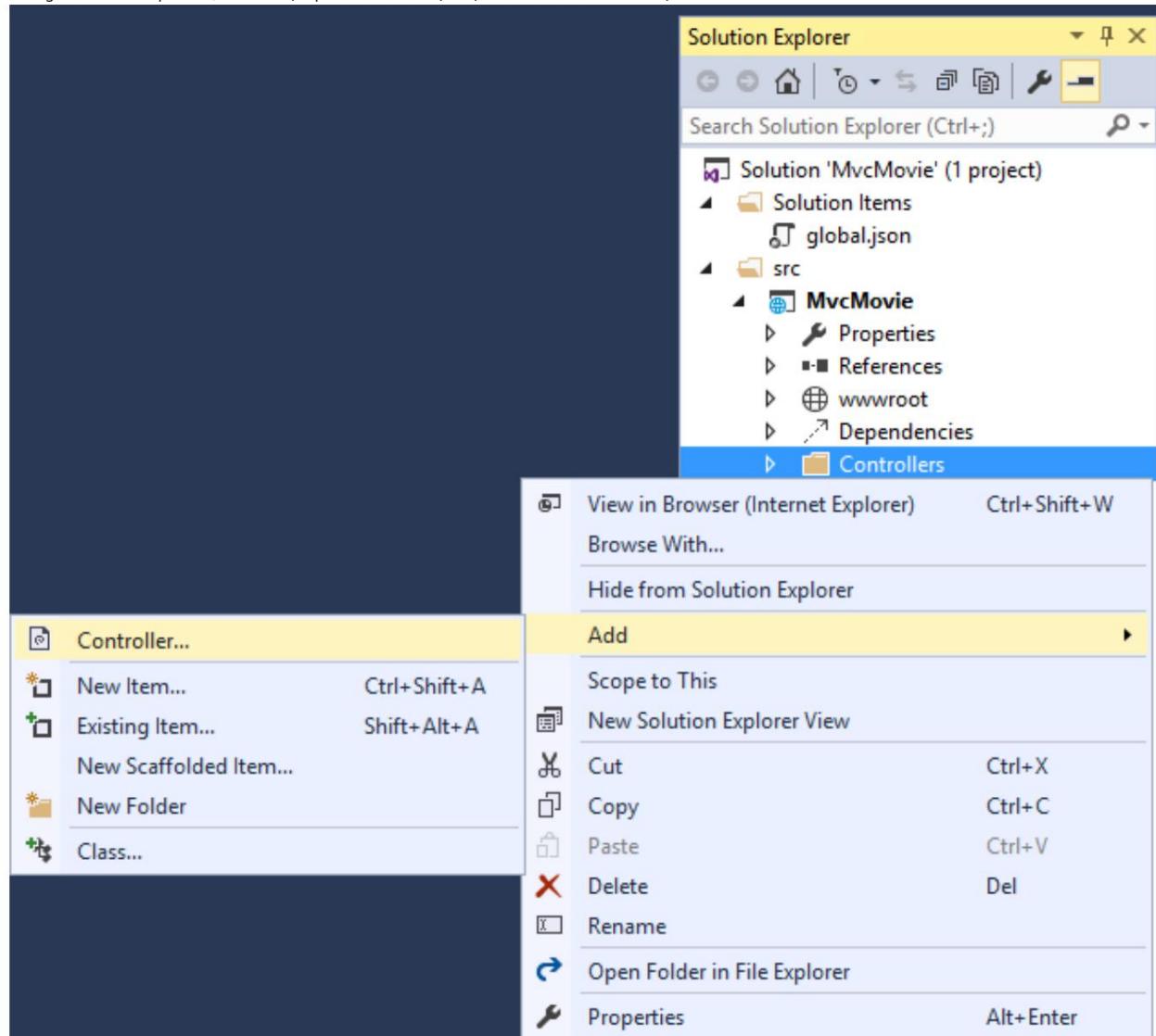
```

Ngoài các thuộc tính bạn muốn tạo mô hình phim, trường ID được DB yêu cầu cho khóa chính.

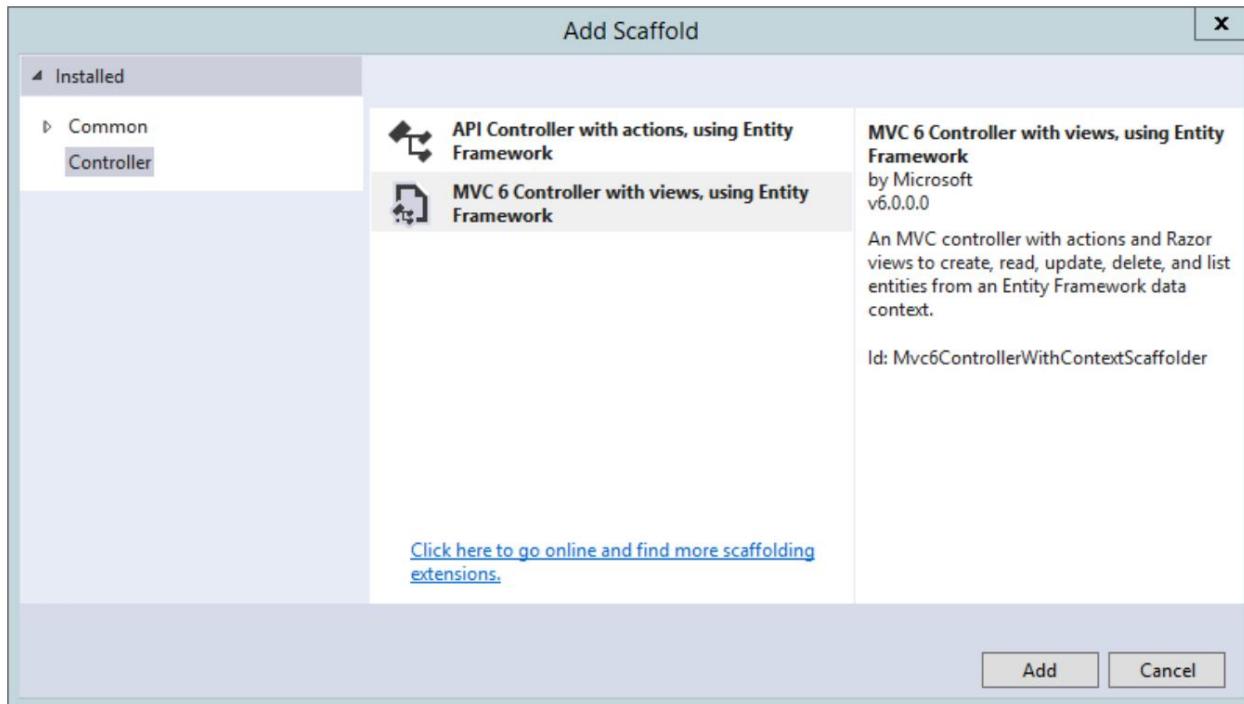
Xây dựng dự án. Nếu bạn không xây dựng ứng dụng, bạn sẽ gặp lỗi trong phần tiếp theo. Cuối cùng, chúng tôi đã thêm một Mô hình vào ứng dụng MVC của chúng tôi.

Giàn giáo một bộ điều khiển

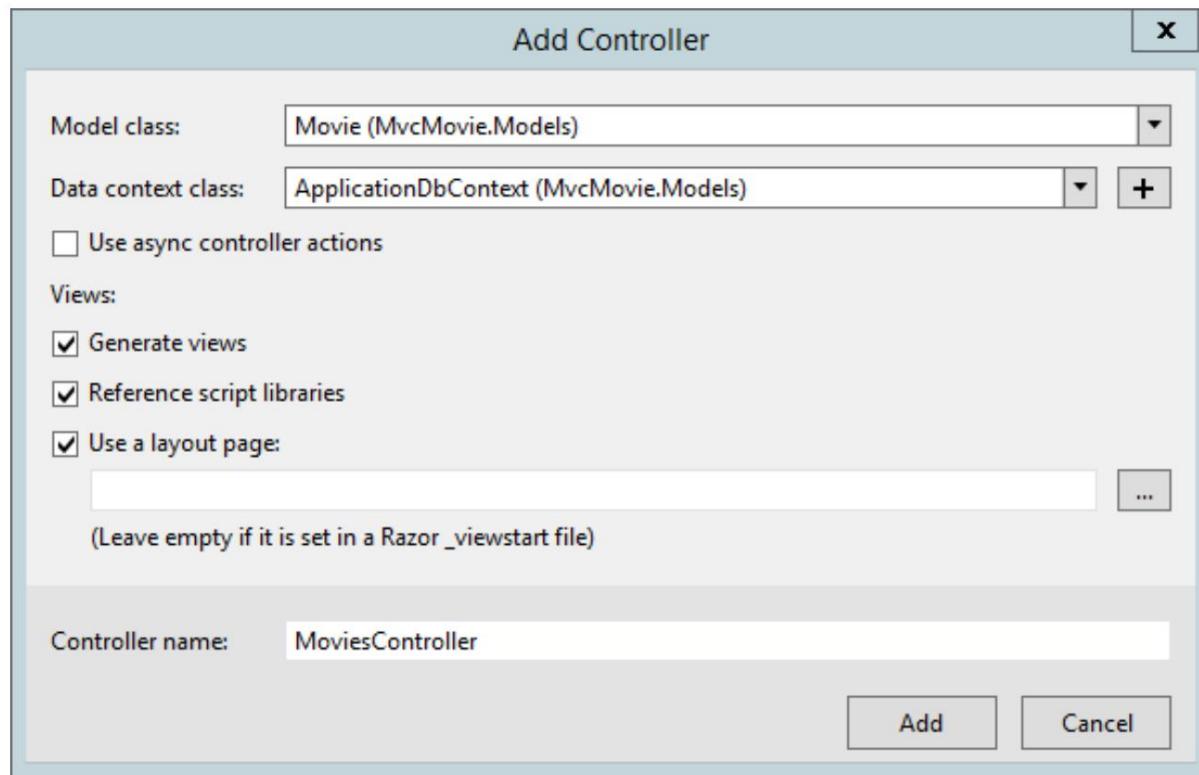
Trong Solution Explorer, bấm chuột phải vào thư mục Bộ điều khiển > Thêm> Bộ điều khiển.



Trong hộp thoại Thêm Scaffold , chạm vào Bộ điều khiển MVC 6 với các dạng xem, sử dụng Khung thực thể> Thêm.



- Hoàn thành hộp thoại Thêm bộ điều khiển
 - Lớp mô hình: Phim (MvcMovie.Models)
 - Lớp ngũ cành dữ liệu: ApplicationDbContext (MvcMovie.Models)
 - Tên bộ điều khiển: Giữ MoviesController mặc định
 - Lượt xem :: Giữ mặc định của từng tùy chọn được chọn
 - Tên bộ điều khiển: Giữ MoviesController mặc định
 - Nhấn vào Thêm



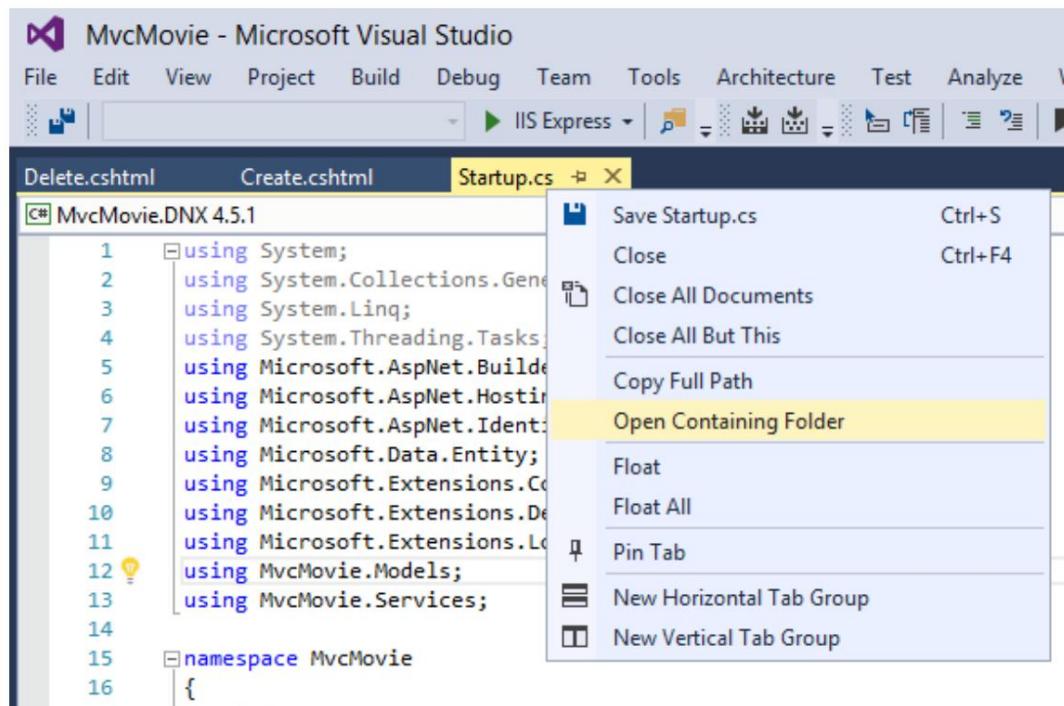
Công cụ giàn giáo Visual Studio tạo ra như sau:

- Bộ điều khiển phim (MoviesController.cs)
- Tạo, xóa, chi tiết, chỉnh sửa và lập chỉ mục các tệp xem Razor
- Các lớp di cư
 - Lớp CreateIdentitySchema tạo cơ sở dữ liệu thành viên ASP.NET Identity những cái bàn. Cơ sở dữ liệu Identity lưu trữ thông tin đăng nhập của người dùng cần thiết để xác thực. Chúng tôi sẽ không đề cập đến authentication trong hướng dẫn này, vì vậy bạn có thể theo dõi Tài nguyên bổ sung ở cuối hướng dẫn này.
 - Lớp ApplicationDbContextModelSnapshot tạo các thực thể EF được sử dụng để truy cập cơ sở dữ liệu Identity. Chúng ta sẽ nói thêm về các thực thể EF ở phần sau trong hướng dẫn.

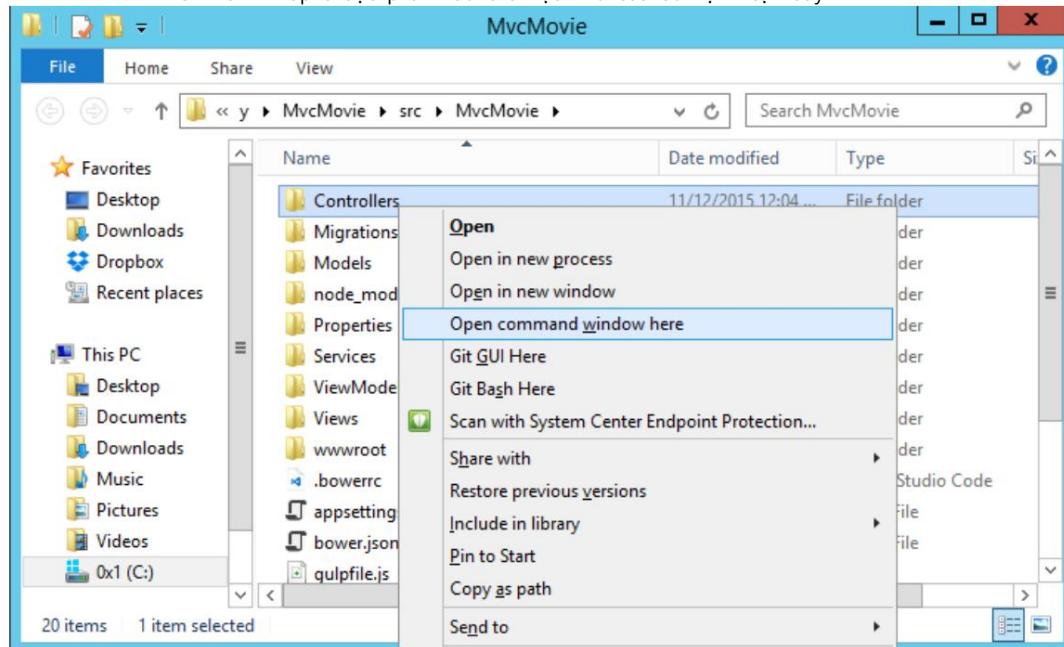
Visual Studio tự động tạo các phương thức hành động và khung nhìn CRUD (tạo, đọc, cập nhật và xóa) cho bạn (việc tạo tự động các phương thức và khung nhìn hành động CRUD được gọi là giàn giáo). Bạn sẽ sớm có một ứng dụng web đầy đủ chức năng cho phép bạn tạo, liệt kê, chỉnh sửa và xóa các mục phim.

Sử dụng di chuyển dữ liệu để tạo cơ sở dữ liệu

- Mở dấu nhắc lệnh trong thư mục dự án (MvcMovie / src / MvcMovie). Làm theo các hướng dẫn sau để biết cách mở nhanh một thư mục trong thư mục dự án.
 - Mở một tệp trong thư mục gốc của dự án (ví dụ: sử dụng Startup.cs.)
 - Nhập chuột phải vào Startup.cs > Mở Thư mục Chứa.



- Shift + nhấp chuột phải vào thư mục> Mở cửa sổ lệnh tại đây



- Chạy cd .. để chuyển trở lại thư mục dự án

- Chạy các lệnh sau trong dấu nhắc lệnh:

```

khôi phục dnu
dnvm sử dụng 1.0.0-rc1-update1 -p dnx ef di
chuyển thêm Cập nhật cơ sở dữ liệu dnx ef ban
dầu

```

```
C:\y\MvcMovie\src\MvcMovie>dnu restore
Microsoft .NET Development Utility Clr-x86-1.0.0-rc1-16147

Restoring packages for C:\y\MvcMovie\src\MvcMovie\project.json
Writing lock file C:\y\MvcMovie\src\MvcMovie\project.lock.json
Restore complete, 8152ms elapsed

NuGet Config files used:
  C:\Users\riande\AppData\Roaming\NuGet\nuget.config

C:\y\MvcMovie\src\MvcMovie>dnvm use 1.0.0-rc1-final -p
Adding C:\Users\riande\.dnx\runtimes\dnx-clr-win-x86.1.0.0-rc1-final\bin to process PATH
Adding C:\Users\riande\.dnx\runtimes\dnx-clr-win-x86.1.0.0-rc1-final\bin to user PATH

C:\y\MvcMovie\src\MvcMovie>dnx ef migrations add Initial
An operation was scaffolded that may result in the loss of data. Please review the migration
Done. To undo this action, use 'ef migrations remove'

C:\y\MvcMovie\src\MvcMovie>dnx ef database update
Applying migration '000000000000_CreatedIdentitySchema'.
Applying migration '20151112220059_Initial'.
Done.

C:\y\MvcMovie\src\MvcMovie>
```

- dnu restore Lệnh này xem xét các phần phụ thuộc trong tệp project.json và tải chúng xuống. Để biết thêm thông tin xem [Làm việc với DNX Projects](#) và [Tổng quan về DNX](#).
- dnvm use <version> dnvm là .NET Version Manager, là một tập hợp các tiện ích dòng lệnh được sử dụng để cập nhật và cấu hình .NET Runtime. Trong trường hợp này, chúng tôi yêu cầu dnvm thêm thời gian chạy ASP.NET 5 1.0.0-rc1 vào biến môi trường PATH của trình bao hiện tại.
- dnx DNX là viết tắt của .NET Execution Environment.

- dnx ef Lệnh ef được chỉ định trong tệp project.json:

```
1 "lệnh": {
2   "web": "Microsoft.AspNet.Server.Kestrel", "ef":
3     "EntityFramework.Commands"
4 },
```

- dnx ef di chuyển thêm Ban đầu Tạo một lớp có tên Ban đầu

```
public partial class Ban đầu : Di chuyển
```

Tham số “Ban đầu” là tùy ý, nhưng là thông lệ cho lần di chuyển cơ sở dữ liệu đầu tiên (ban đầu). Bạn có thể bỏ qua một cách an toàn cảnh báo có thể dẫn đến mất dữ liệu, nó đang loại bỏ các ràng buộc khóa ngoại chứ không phải bất kỳ dữ liệu nào. Cảnh báo là kết quả của quá trình di chuyển tạo ban đầu cho mô hình Identity không được cập nhật. Điều này sẽ được khắc phục trong phiên bản tiếp theo.

- Cập nhật cơ sở dữ liệu dnx ef Cập nhật cơ sở dữ liệu, tức là, áp dụng quá trình di chuyển.

Kiểm tra ứng dụng

- Chạy ứng dụng và nhấn vào liên kết Phim Mvc

- Nhấn vào liên kết Tao mới và tạo phim

Genre
Comedy

Price
1.99

ReleaseDate
11-18-15

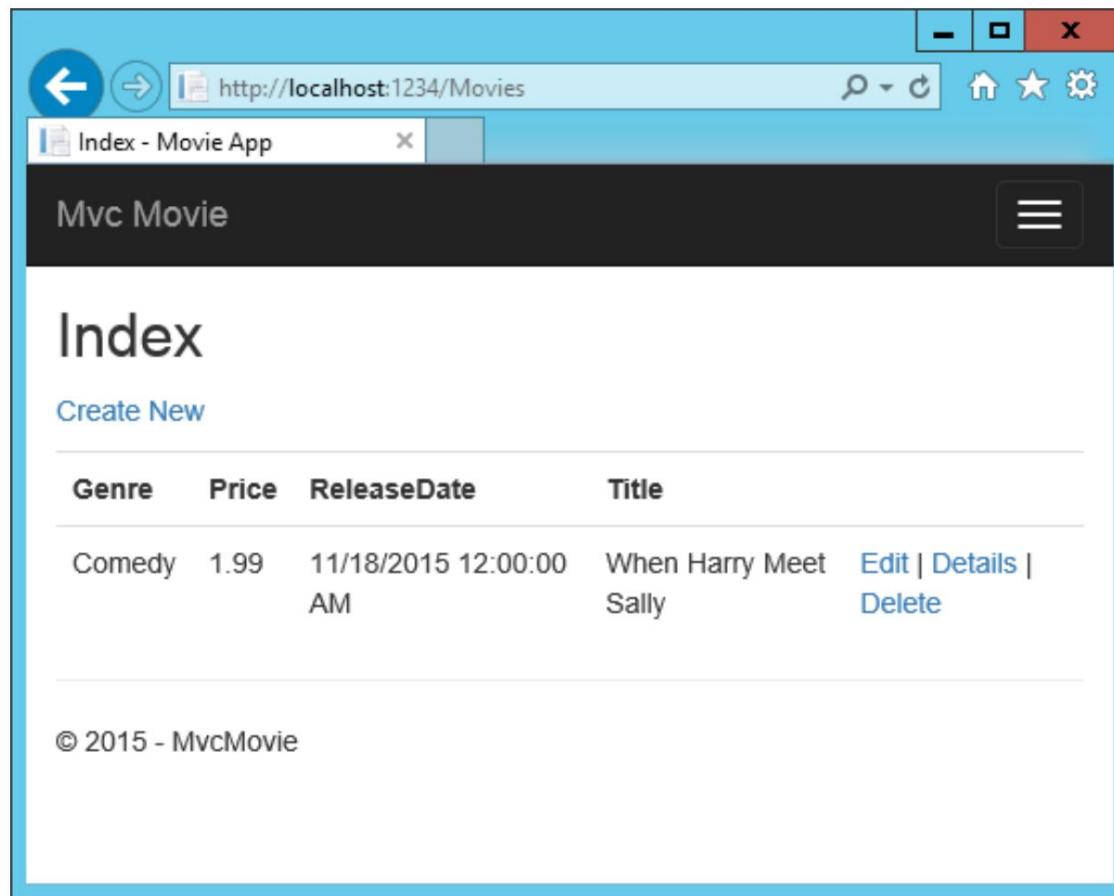
Title
When Harry Meet Sally

Create

Back to List

Lưu ý: Bạn có thể không nhập được dấu thập phân hoặc dấu phẩy vào trường Giá. Để hỗ trợ xác thực jQuery đối với các ngôn ngữ không phải tiếng Anh sử dụng dấu phẩy (",") cho dấu thập phân và các định dạng ngày không phải tiếng Anh Mỹ, bạn phải thực hiện các bước để toàn cầu hóa ứng dụng của mình. Xem Tài nguyên bổ sung để biết thêm thông tin. Hiện tại, chỉ cần nhập các số nguyên như 10.

Nhấn vào Tao khiền biểu mẫu được đăng lên máy chủ, nơi thông tin phim được lưu trong cơ sở dữ liệu. Sau đó, bạn được chuyển hướng đến URL / Phim, nơi bạn có thể xem phim mới được tạo trong danh sách.



Tạo thêm một vài mục phim. Hãy thử các liên kết Chỉnh sửa, Chi tiết và Xóa , tất cả đều có chức năng.

Kiểm tra mã đã tạo

Mở tệp Controllers / MoviesController.cs và kiểm tra phương thức Index được tạo. Một phần của bộ điều khiển phim với phương thức Chỉ mục được hiển thị bên dưới:

```

lớp công cộng PhimController : Controller {

    riêng ApplicationDbContext _context;

    public MoviesController (ApplicationDbContext context) {

        _context = ngũ cảnh;
    }

    public IActionResult Index () { return View
        (_context.Movie.ToList ());
    }
}

```

Hàm tạo sử dụng Dependency Injection để đưa ngũ cảnh cơ sở dữ liệu vào bộ điều khiển. Bối cảnh cơ sở dữ liệu được sử dụng trong mỗi CRUD các phương thức trong bộ điều khiển.

Yêu cầu tới bộ điều khiển Phim trả về tất cả các mục nhập trong bảng Phim và sau đó chuyển dữ liệu sang dạng xem Chỉ mục.

Các mô hình được đánh máy mạnh và từ khóa @model

Trước đó trong hướng dẫn này, bạn đã thấy cách bộ điều khiển có thể truyền dữ liệu hoặc đối tượng vào mẫu chế độ xem bằng cách sử dụng tinh ViewData. Từ điển ViewData là một đối tượng động cung cấp một cách giới hạn cuối thuận tiện để chuyển thông tin đến một dạng xem.

MVC cũng cung cấp khả năng chuyển các đối tượng được đánh mạnh vào một mẫu chế độ xem. Cách tiếp cận được đánh máy mạnh mẽ này cho phép kiểm tra thời gian biên dịch mã của bạn tốt hơn và IntelliSense phong phú hơn trong trình chỉnh sửa Visual Studio. Chủ nghĩa mech giàn giáo trong Visual Studio đã sử dụng cách tiếp cận này (nghĩa là truyền một mô hình được đánh máy mạnh) với lớp MoviesController và các mẫu xem khi nó tạo các phương thức và chế độ xem.

Kiểm tra phương thức Chi tiết được tạo trong tệp Controllers / MoviesController.cs. Phương pháp Chi tiết được hiển thị bên dưới.

```
// GET: Movies / Details / 5 public
IActionResult Details (int? Id) {

    if (id == null) {

        trả về HttpNotFound ();
    }

    Movie movie = _context.Movie.Single (m => m.ID == id); if (movie == null) {

        trả về HttpNotFound ();
    }

    return View (phim);
}
```

Thông số id thường được chuyển dưới dạng dữ liệu truyền đường, ví dụ: http://localhost: 1234 / movies / details / 1 sets:

- Bộ điều khiển đến bộ điều khiển phim (đoạn URL đầu tiên)
- Hành động đến chi tiết (phân đoạn URL thứ hai)
- Id thành 1 (đoạn URL cuối cùng)

Bạn cũng có thể chuyển vào id với một chuỗi truy vấn như sau:

http://localhost: 1234 / phim / chi tiết? id = 1

Nếu tìm thấy Phim, một phiên bản của mô hình Phim sẽ được chuyển đến chế độ xem Chi tiết:

```
return View (phim);
```

Kiểm tra nội dung của tệp Views / Movies / Details.cshtml:

```
@model MvcMovie.Models.Movie

@ {
    ViewData ["Title"] = "Chi tiết";
}

<h2> Chi tiết </h2>

<div>
    <h4> Phim </h4> <hr />

```

```

<dl class = "dl-ngang">
    <dt>
        @ Html.DisplayNameFor (model => model.Genre) </dt>

    <dd>
        @ Html.DisplayFor (model => model.Genre) </dd>

    <dt>
        @ Html.DisplayNameFor (model => model.Price) </dt>

    <dd>
        @ Html.DisplayFor (model => model.Price) </dd>
    <dt>

        @ Html.DisplayNameFor (model => model.ReleaseDate) </dt>
    <dd>

        @ Html.DisplayFor (model => model.ReleaseDate) </dd>
    <dt>

        @ Html.DisplayNameFor (model => model.Title) </dt>
    <dd>

        @ Html.DisplayFor (model => model.Title) </dd>
    </dl> </div>

<p>
    <a asp-action="Edit" asp-route-id="@Model.ID"> Chính sửa </a> | <a asp-
    action="Index"> Quay lại danh sách </a>
</p>

```

Bằng cách bao gồm một câu lệnh @model ở đầu tệp mẫu dạng xem, bạn có thể chỉ định loại đối tượng mà dạng xem mong đợi. Khi bạn tạo bộ điều khiển phim, Visual Studio tự động bao gồm câu lệnh @model sau ở đầu tệp Details.cshtml:

```
@model MvcMovie.Models.Movie
```

Chỉ thị @model này cho phép bạn truy cập phim mà bộ điều khiển đã chuyển đến chế độ xem bằng cách sử dụng đối tượng Model được nhập mạnh. Ví dụ: trong mẫu Details.cshtml, mã chuyển từng trường phim đến Trình trợ giúp DisplayNameFor và DisplayFor HTML với đối tượng Model được nhập mạnh. Các phương pháp Tạo và Chính sửa và xem mẫu cũng truyền một đối tượng Mô hình phim.

Kiểm tra mẫu xem Index.cshtml và phương pháp Chỉ mục trong bộ điều khiển Phim. Lưu ý cách mà tạo một đối tượng Danh sách khi nó gọi phương thức View helper trong phương thức hành động Index. Sau đó, mã sẽ chuyển danh sách Phim này từ phương thức hành động Chỉ mục đến chế độ xem:

```
public IActionResult Index () {

    return View (_context.Movie.ToList ());
}
```

Khi bạn tạo bộ điều khiển phim, Visual Studio tự động bao gồm câu lệnh @model sau ở đầu tệp Index.cshtml:

```
@model IEnumerable <MvcMovie.Models.Movie>
```

Chỉ thị @model cho phép bạn truy cập danh sách phim mà bộ điều khiển đã chuyển đến chế độ xem bằng cách sử dụng đối tượng Model được nhập mạnh. Ví dụ: trong mẫu Index.cshtml, mã lặp qua các bộ phim với

câu lệnh foreach trên đối tượng Model được gõ mạnh:

```

1 @model IEnumerable <MvcMovie.Models.Movie>
2
3 @{
4     ViewData["Title"] = "Chi mục";
5 }
6
7 <h2> Chi mục </h2>
...
9 <p>
10    <a asp-action="Create"> Tạo mới </a>
11 </p>
12 <table class = "table">
13     <tr>
14         <th>
15             @ Html.DisplayNameFor (model => model.Genre)
16         </th>
17         <th>
18             @ Html.DisplayNameFor (model => model.Price)
19         </th>
20         <th>
21             @ Html.DisplayNameFor (model => model.ReleaseDate)
22         </th>
23         <th>
24             @ Html.DisplayNameFor (model => model.Title)
25         </th>
26         <th> </th>
27     </tr>
28
29 @foreach (var item trong Model) {
30     <tr>
31         <td>
32             @ Html.DisplayFor (modelItem => item.Genre)
33         </td>
34         <td>
35             @ Html.DisplayFor (modelItem => item.Price)
36         </td>
37         <td>
38             @ Html.DisplayFor (modelItem => item.ReleaseDate)
39         </td>
40         <td>
41             @ Html.DisplayFor (modelItem => item.Title)
42         </td>
43         <td>
44             <a asp-action="Edit" asp-route-id="@item.ID"> Chính sửa </a> |
45             <a asp-action="Details" asp-route-id="@item.ID"> Chi tiết </a> |
46             <a asp-action="Delete" asp-route-id="@item.ID"> Xóa </a>
47         </td>
48     </tr>
49 }
50 </table>
```

Vì đối tượng Model được gõ mạnh (dưới dạng đối tượng `IEnumerable <Movie>`), mỗi mục trong vòng lặp được nhập như Phím. Trong số các lợi ích khác, điều này có nghĩa là bạn nhận được kiểm tra thời gian biên dịch của mã và `IntelliSense` đầy đủ hỗ trợ trong trình soạn thảo mã:

```

@foreach (var item in Model) {
    <tr>
        <td>
            @Html.DisplayFor(modelItem => item.Genre)
        </td>
        <td>
            @Html.DisplayFor(modelItem => item.Price)
        </td>
        <td>
            @Html.DisplayFor(modelItem => item.ReleaseDate)
        </td>
        <td>
            @Html.DisplayFor(modelItem => item.Title)
        </td>
        <td>
            <a asp-action="Edit" asp-route-id="@item.ID">Edit</a> | 
            <a asp-action="Details" asp-route-id="@item.ID">Details</a> | 
            <a asp-action="Delete" asp-route-id="@item.ID">Delete</a>
        </td>
    </tr>
}
</table>

```

The screenshot shows a code editor with an Intellisense dropdown open over the line of code `<a asp-action="Delete" asp-route-id="@item.ID">Delete`. The dropdown lists several properties for the variable `item.ID`:

- Equals
- Genre
- GetHashCode
- GetType
- ID**
- Price
- ReleaseDate
- Title
- ToString

Lưu ý: Phiên bản RC1 của công cụ giàn giáo tạo Trình trợ giúp HTML để hiển thị các trường (@ Html.DisplayNameFor (model => model.Genre)). Phiên bản tiếp theo sẽ sử dụng Trình trợ giúp thẻ để hiển thị các trường.

Bây giờ bạn có một cơ sở dữ liệu và các trang để hiển thị, chỉnh sửa, cập nhật và xóa dữ liệu. Trong hướng dẫn tiếp theo, chúng ta sẽ làm việc với cơ sở dữ liệu.

Các nguồn bổ sung

- Trình trợ giúp thẻ
- Tạo ứng dụng ASP.NET MVC an toàn và triển khai lên Azure
- Làm việc với các dự án DNX
- Tổng quan về DNX

2.1.5 Làm việc với SQL Server LocalDB

Bởi Rick Anderson

Lớp ApplicationDbContext xử lý nhiệm vụ kết nối với cơ sở dữ liệu và ánh xạ Movie objects tới các bản ghi cơ sở dữ liệu. Bởi cách cơ sở dữ liệu được đăng ký bằng cách `tiêm phụ thuộc` vùng chứa trong phương thức ConfigureServices trong tệp Startup.cs:

```

1 public void ConfigureServices (dịch vụ IServiceCollection)
2 {
3     // Thêm các dịch vụ khung.
4     services.AddEntityFramework ()
5         .AddSqlServer ()
6         .AddDbContext <ApplicationDbContext> (tùy chọn =>
7             options.UseSqlServer (Cấu hình ["Đã liệu: DefaultConnection: ConnectionString"]));

```

Cấu hình ASP.NET 5 hệ thống đọc Đã liệu: DefaultConnection: ConnectionString. Đối với địa phương phát triển, nó lấy chuỗi kết nối từ tệp appsettings.json:

```

1 {
2     "Đã liệu": {
3         "DefaultConnection": {
4             "Chuỗi kết nối": "Máy chủ = (localdb) \\\ mssqllocaldb; Cơ sở dữ liệu = aspnet5-MvcMovie-53e157ca-bf3b-46
5         }
6     },

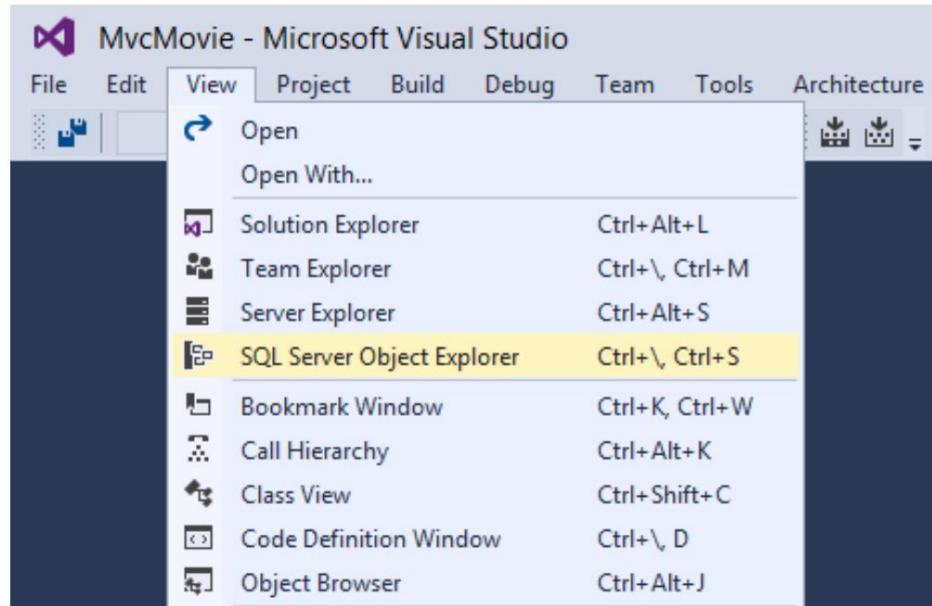
```

Khi triển khai ứng dụng tới máy chủ thử nghiệm hoặc sản xuất, bạn có thể sử dụng một biến môi trường hoặc một cách tiếp cận khác để đặt chuỗi kết nối thành SQL Server thực. Xem [cấu hình](#).

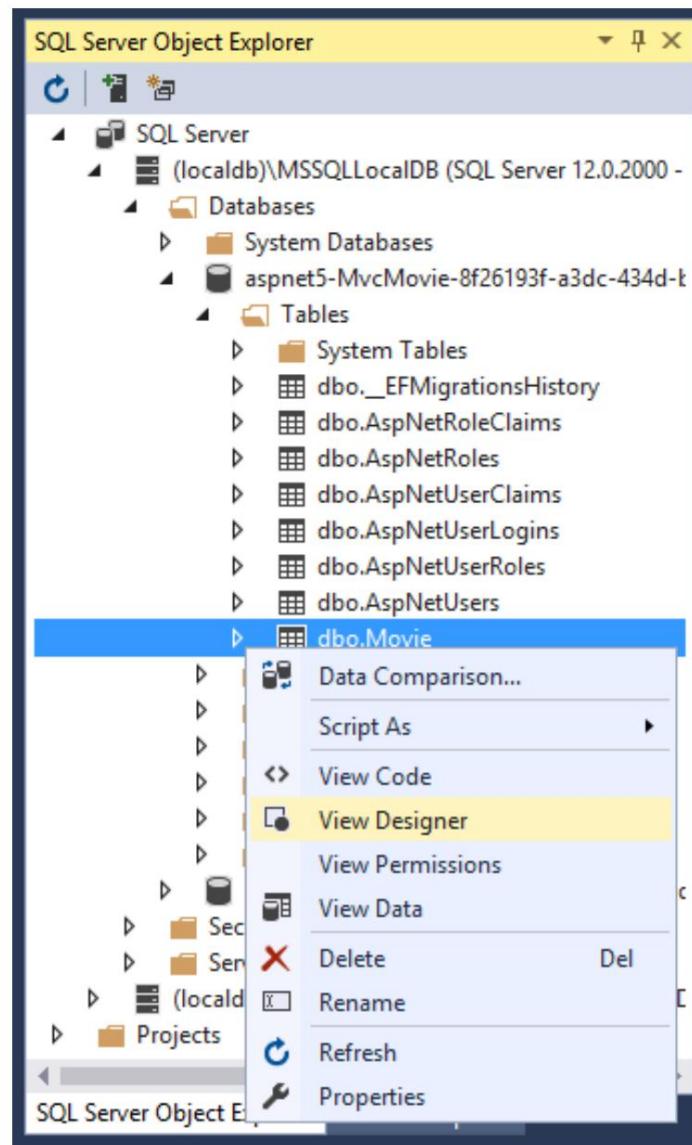
SQL Server Express LocalDB

LocalDB là một phiên bản nhẹ của SQL Server Express Database Engine được nhắm mục tiêu cho việc phát triển chương trình. LocalDB khởi động theo yêu cầu và chạy ở chế độ người dùng nên không có cấu hình phức tạp. Theo mặc định, LocalDB cơ sở dữ liệu tạo các tệp “*.mdf” trong thư mục C: / Users / <user>.

- Từ menu View , mở SQL Server Object Explorer (SSOX).



- Nhấp chuột phải vào bảng Movie > View Designer



dbo.Movie [Design] ➔ X

Update | Script File: **dbo.Movie.sql**

	Name	Data Type	Allow Nulls
PK	ID	int	<input type="checkbox"/>
	Genre	nvarchar(MAX)	<input checked="" type="checkbox"/>
	Price	decimal(18,2)	<input type="checkbox"/>
	ReleaseDate	datetime2(7)	<input type="checkbox"/>
	Title	nvarchar(MAX)	<input checked="" type="checkbox"/>
			<input type="checkbox"/>

Keys (1)
PK_Movie (Primary Key, Clustered: 1)

Check Constraints (0)

Indexes (0)

Foreign Keys (0)

Triggers (0)

Design **T-SQL**

```

1 CREATE TABLE [dbo].[Movie] (
2     [ID]           INT            IDENTITY (1, 1) NOT NULL,
3     [Genre]         NVARCHAR (MAX) NULL,
4     [Price]         DECIMAL (18, 2) NOT NULL,
5     [ReleaseDate]  DATETIME2 (7)  NOT NULL,
6     [Title]         NVARCHAR (MAX) NULL,
7     CONSTRAINT [PK_Movie] PRIMARY KEY CLUSTERED ([ID] ASC)
8 );
9

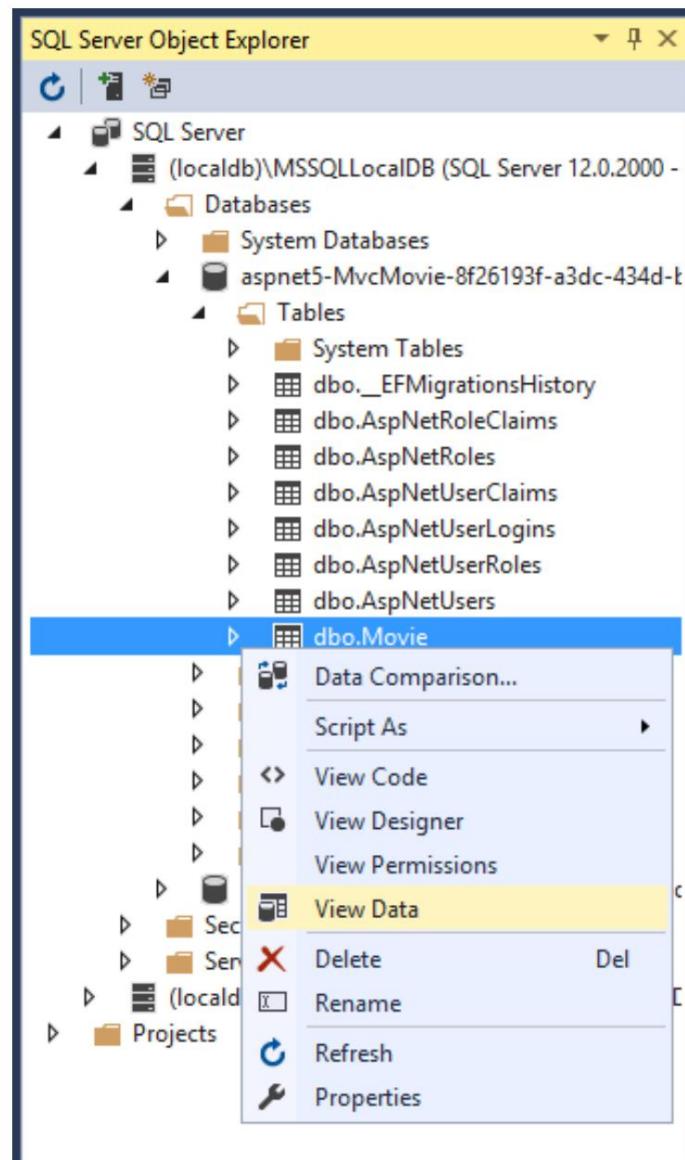
```

100 %

Connection Ready | (localdb)\MSSQLLocalDB | REDMOND\riande | aspnet5-MvcMovie-8f261...

Lưu ý biểu tượng chìa khóa bên cạnh ID. Theo mặc định, EF sẽ đặt thuộc tính có tên ID làm khóa chính.

- Nhấp chuột phải vào bảng Phim > Xem dữ liệu



The screenshot shows the Microsoft Visual Studio interface. The title bar says "MvcMovie - Microsoft V...". The menu bar includes File, Edit, View, Project, Build, Debug, Team, SQL, Tools, Architecture, Test, Analyze, Window, and Help. A user profile "rick" is shown in the top right. The toolbar includes icons for Save, Undo, Redo, Cut, Copy, Paste, Find, Replace, and others. The main area displays the "dbo.Movie [Data]" table. The table has columns: ID, Genre, Price, ReleaseDate, and Title. The data shows three rows of movies: "When Harry Met Sally", "Ghost Busters IV", and "Ghost Busters 7". The bottom status bar shows "3 Rows | Cell is Read Only" and "Ln 1 Col 1".

Hạt giống cơ sở dữ liệu

Chúng tôi sẽ tận dụng lợi thế của Dependency Injection (DI) để bắt đầu cơ sở dữ liệu. Bạn thêm các phụ thuộc phía máy chủ vào các dự án ASP.NET 5 trong tệp project.json. Mở project.json và thêm gói Microsoft DI. IntelliSense giúp chúng tôi thêm gói.

The screenshot shows the Microsoft Visual Studio code editor with the file "project.json" open. The schema is defined as "Schema: http://json.schemastore.org/project". The JSON code includes a "dependencies" section where "Microsoft.Extensions.DependencyInjection" is being typed. IntelliSense shows suggestions for "Microsoft.Extensions.DependencyInjection.Abstractions", "Microsoft.Extensions.DependencyInjection", and "Microsoft.Extensions.DependencyInjection.DiagnosticAdapter".

```

{
  "userSecretsId": "aspnet5-TestNuget-da05122b",
  "version": "1.0.0-*",
  "compilationOptions": {
    "emitEntryPoint": true
  },
  "dependencies": {
    "Microsoft.Extensions.DependencyInjection": ...
  }
}

```

Gói DI được đánh dấu bên dưới:

```
{
  "userSecretsId": "aspnet5-MvcMovie-53e157ca-bf3b-46b7-bb3f-82ac58612f5e", "phiên bản": "1.0.0-*",
  "compilationOptions": {
}
```

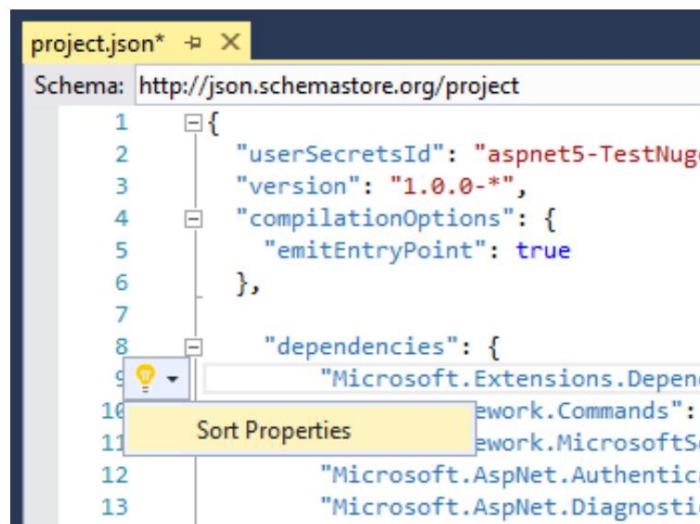
```

    "releaseEntryPoint": true },


  "dependencies": {
    "Microsoft.Extensions.DependencyInjection": "1.0.0-rc1-final",
    "EntityFramework.MicrosoftSqlServer": "7.0.0-rc1-final", "Microsoft.AspNet.Authentication.Cookies":
    " 1.0.0-rc1-final ",
  }
}

```

Tùy chọn: Nhấn vào biểu tượng bóng đèn tia và chọn Sắp xếp thuộc tính.



Tạo một lớp mới có tên SeedData trong thư mục Models. Thay thế mã đã tạo bằng mã sau:

```

sử dụng Microsoft.Extensions.DependencyInjection; sử dụng Hệ thống; sử
dụng System.Linq;

không gian tên MvcMovie.Models {

  public static class SeedData {

    public static void Initialize (IServiceProvider serviceProvider) {

      var context = serviceProvider.GetService <ApplicationDbContext> ();

      if (context.Database == null) {

        ném ngoại lệ mới ("DB là null");
      }

      if (context.Movie.Any ()) {

        trả về; // DB đã được seed
      }

      context.Movie.AddRange ( Phim mới
        {
          Title = "Khi Harry gặp Sally", ReleaseDate =
          DateTime.Parse ("1989-1-11"), Thể loại = "Hài lồng mạn",
        }
      );
    }
  }
}

```

```

        Giá = 7,99 triệu
    },

    phim mới
    {
        Title = "Ghostbusters",
        ReleaseDate = DateTime.Parse ("1984-3-13"),
        Genre = "Hài",
        Giá = 8,99 triệu
    },

    phim mới
    {
        Title = "Ghostbusters 2",
        ReleaseDate = DateTime.Parse ("1986-2-23"),
        Genre = "Hài",
        Giá = 9,99 triệu
    },

    phim mới
    {
        Title = "Rio Bravo",
        ReleaseDate = DateTime.Parse ("1959-4-15"),
        Thể loại = "Phương Tây",
        Giá = 3,99 triệu
    }
};

context.SaveChanges ();
}
}
}

```

Phương thức GetService đến từ gói DI mà chúng tôi vừa thêm vào. Chú ý nếu có bất kỳ bộ phim nào trong DB, khởi tạo hạt giống trả về.

```

1   if (context.Movie.Any ())
2   {
3       trả về; // DB đã được seed
4   }

```

Thêm trình khởi tạo hạt giống vào cuối phương pháp Định cấu hình trong tệp Startup.cs:

```

1   app.UseMvc (các tuyến đường =>
2   {
3       các tuyến đường.MapRoute (
4           tên: "mặc định",
5           mẫu: "{controller = Home} / {action = Index} / {id?}");
6   });
7
8   SeedData.Initialize (app.ApplicationServices);
9 }

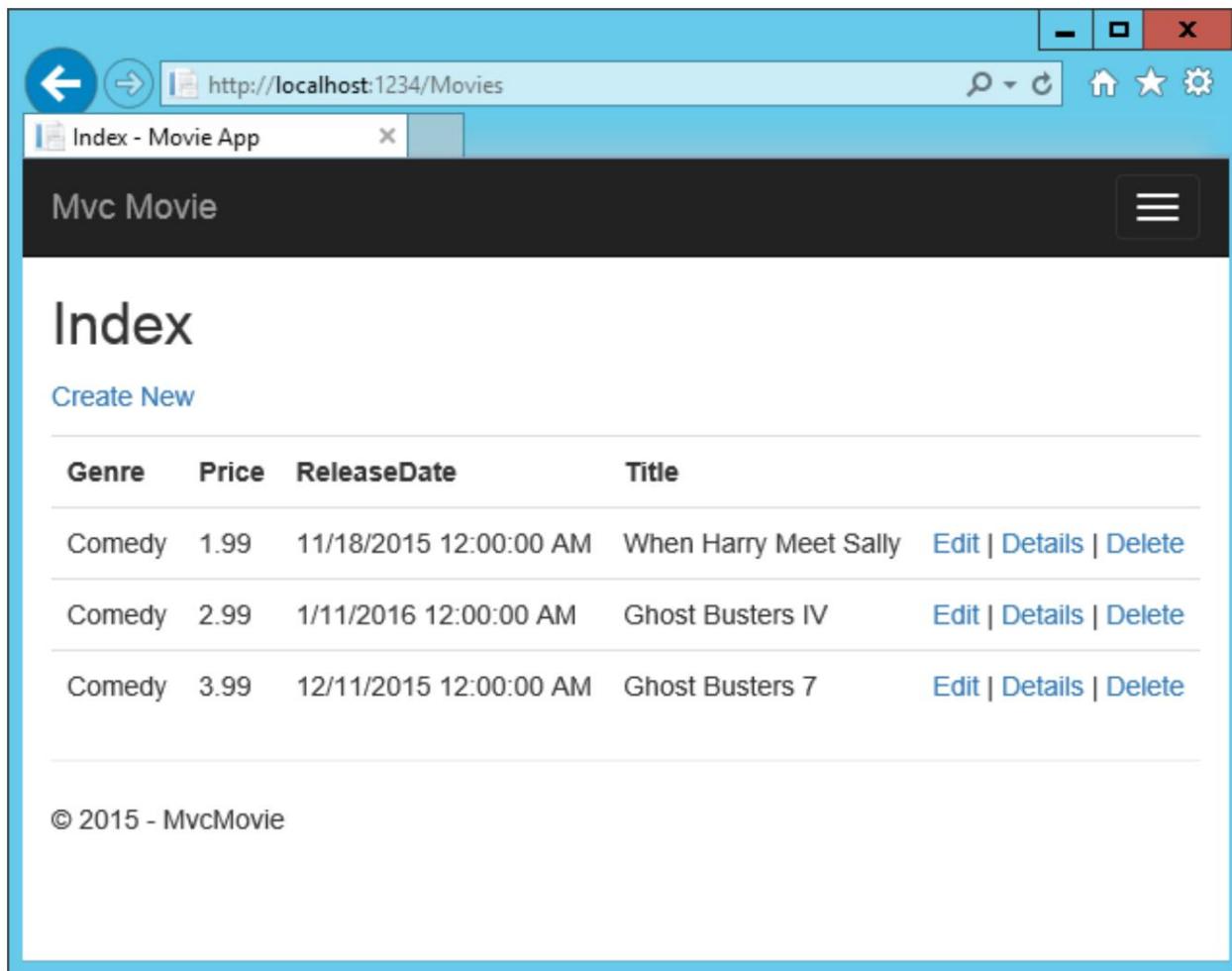
```

Kiểm tra ứng dụng

- Xóa tất cả các bản ghi trong DB. Bạn có thể thực hiện việc này bằng cách kết nối xóa trong trình duyệt hoặc từ SSMS.
- Buộc khởi chạy ứng dụng để phương thức hạt giống chạy. Bạn có thể làm điều này bằng cách đặt điểm ngắt trên dòng đầu tiên của phương pháp Khởi tạo dữ liệu SeedData và khởi chạy trình gỡ lỗi (Nhấn F5 hoặc nhấp vào nút IIS Express).

```
MvcMovie - Microsoft Visual... Quick Launch (Ctrl+Q) File Edit View Project Build Debug Team Tools Architecture rick Test Analyze Window Help IIS Express Diagnostic Tools SQL Server Object Explorer Solution Explorer SeedData.cs + X MvcMovie.DNX 4.5.1 MvcMovie.Models.SeedData Initialize(IServiceProvider serviceProvider) 1 using Microsoft.Framework.DependencyInjection; 2 using System; 3 using System.Linq; 4 5 namespace MvcMovie.Models 6 { 7     public static class SeedData 8     { 9         public static void Initialize(IServiceProvider serviceProvider) 10        { 11            var context = serviceProvider.GetService<ApplicationDbContext>(); 12 13            if (context.Database == null) 14            { 15                throw new Exception("DB is null"); 16            } 17        } 18    } 19 } 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100 % Error List Output Find Results 1 Ready Ln 11 Col 35 Ch 35 INS
```

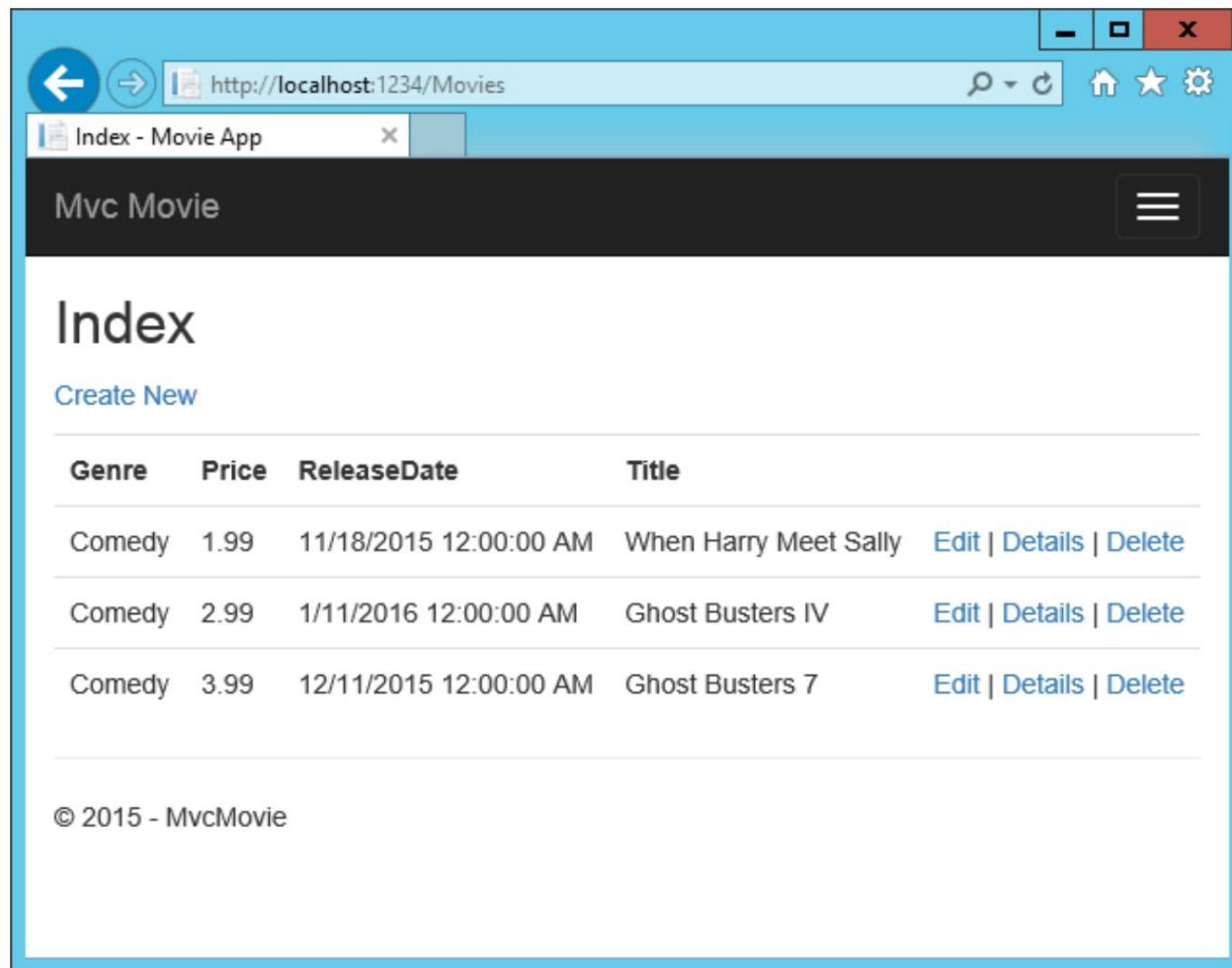
Ứng dụng hiển thị dữ liệu được giao.



2.1.6 Các phương pháp và khung nhìn của bộ điều khiển

Bởi Rick Anderson

Chúng tôi có một khởi đầu tốt với ứng dụng phim, nhưng phần trình bày không lý tưởng. Chúng tôi không muốn xem thời gian vào ngày phát hành và ReleaseDate phải là hai từ.

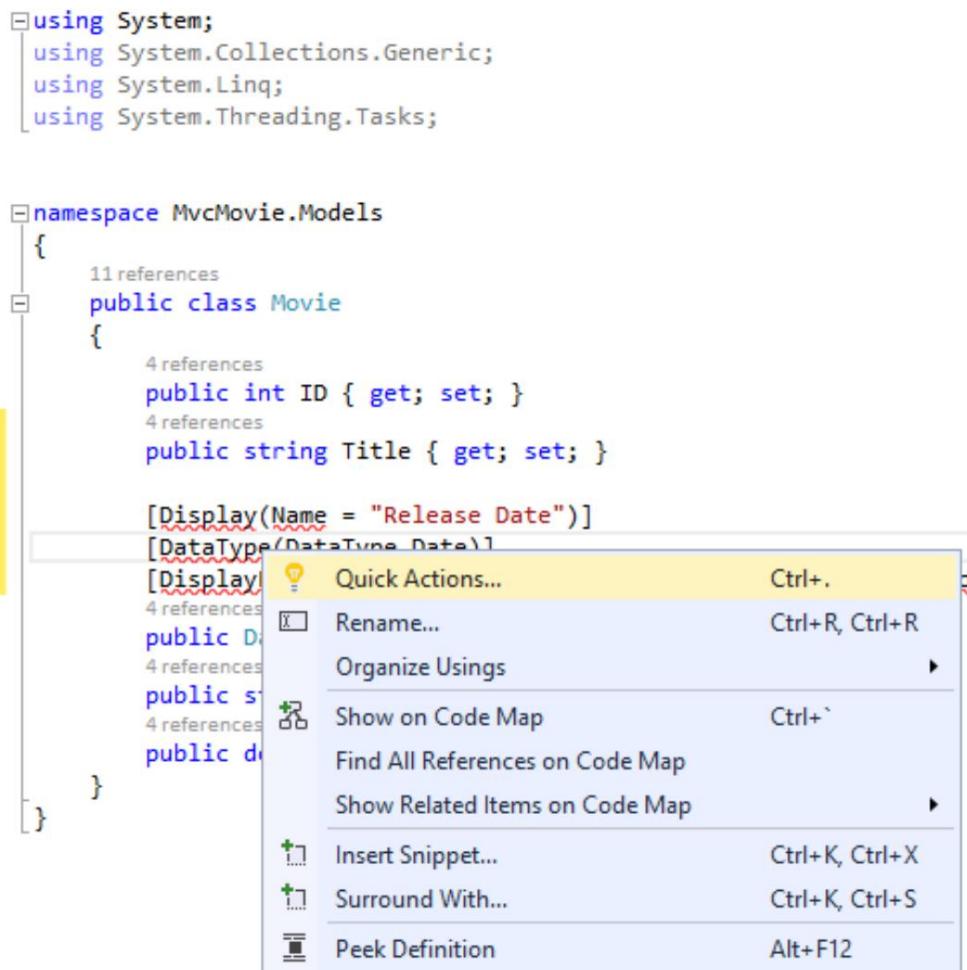


Mở tệp Models / Movie.cs và thêm các dòng được đánh dấu được hiển thị bên dưới:

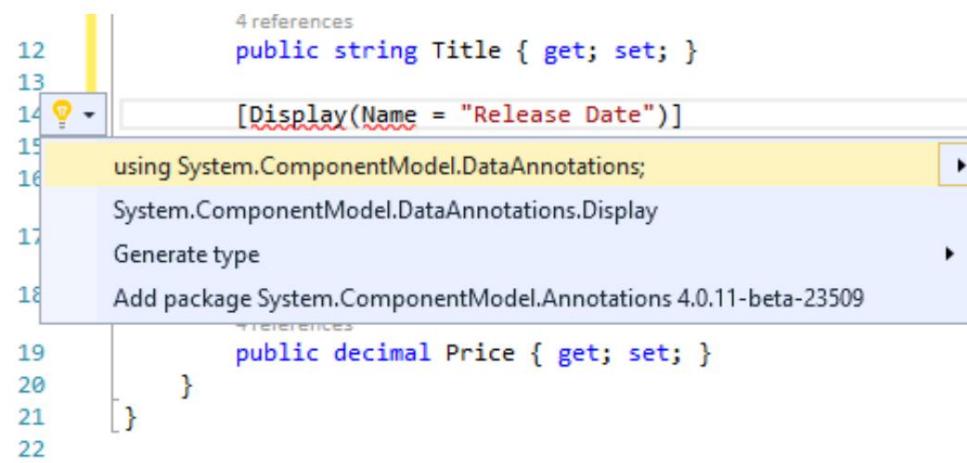
```

1 phím cấp công cộng
2 {
3     public int ID { get; bộ; }
4     chuỗi công khai Tiêu đề { get; bộ; }
5
6     [Hiển thị (Tên = "Ngày phát hành")]
7     [DataType (DataType.Date)]
8     public DateTime ReleaseDate { get; bộ; }
9     chuỗi công khai Thể loại { get; bộ; }
10    giá thập phân công khai { get; bộ; }
11 }
```

- Nhấp chuột phải vào dòng chữ ngoặc màu đỏ > Thao tác nhanh.

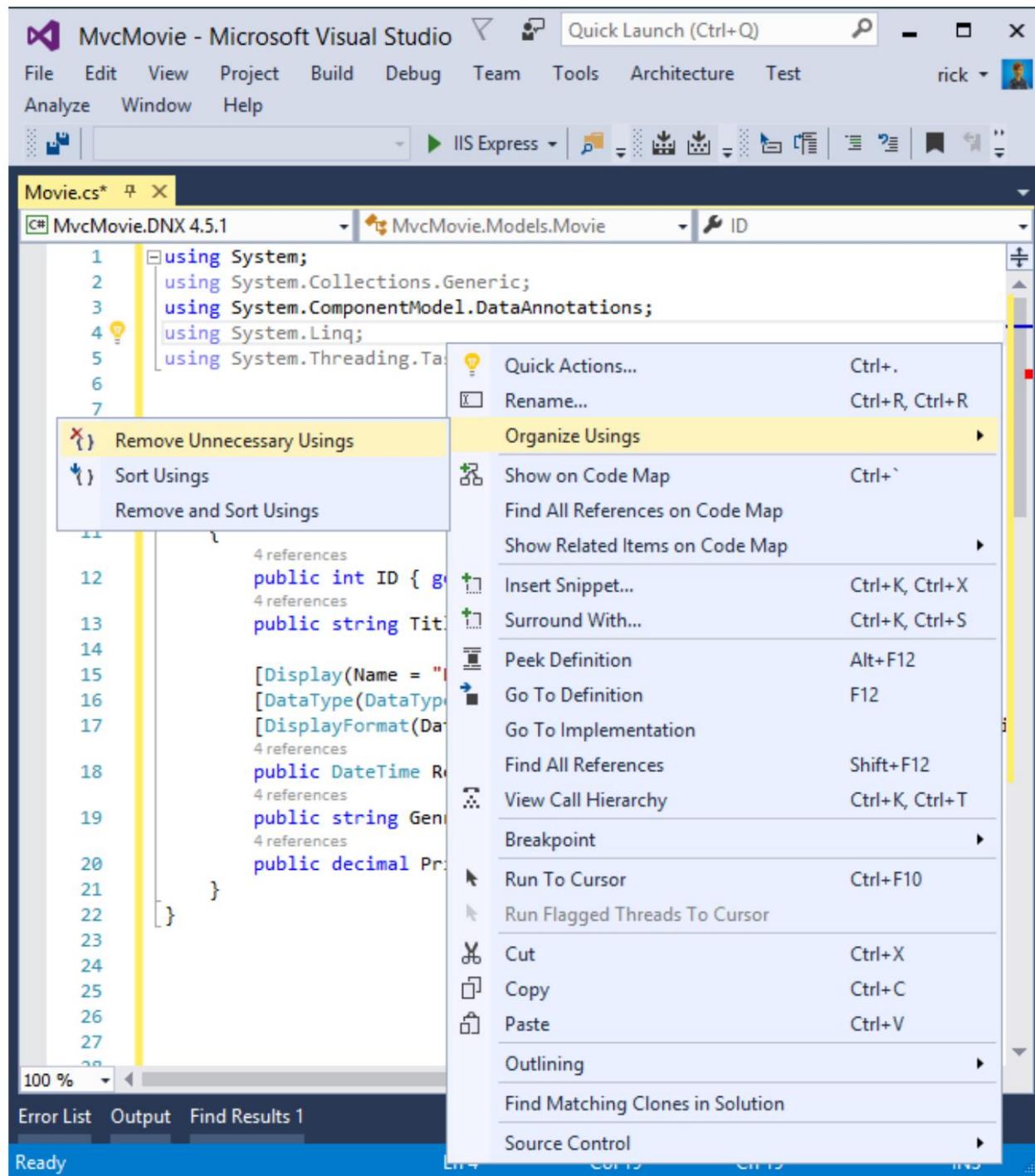


- Nhấn bằng cách sử dụng System.ComponentModel.DataAnnotations;



Visual studio bổ sung bằng cách sử dụng System.ComponentModel.DataAnnotations ;.

Hãy loại bỏ các câu lệnh using không cần thiết. Chúng hiển thị theo mặc định với phông chữ màu xám nhạt. Nhấp chuột phải vào bất kỳ đâu trong tệp Movie.cs > Sắp xếp Sử dụng> Loại bỏ Sử dụng Không cần thiết.



Hoàn thành được hiển thị bên dưới:

```

1 đang sử dụng Hệ thống;
2 bằng cách sử dụng System.ComponentModel.DataAnnotations;
3
4 không gian tên MvcMovie.Models
{
    phím cấp công
    {
        public int ID { get; bộ; }
    }
}

```

```

9  chuỗi công khai Tiêu đề { get; bộ; }

10
11     [Hiển thị (Tên = "Ngày phát hành")]
12     [DataType (DataType.Date)]
13     public DateTime ReleaseDate { get; bộ; }
14     chuỗi công khai Thể loại { get; bộ; }
15     giá thập phân công khai { get; bộ; }
16
17 }

```

Chúng tôi sẽ đề cập đến DataAnnotations trong hướng dẫn tiếp theo. Màn hình thuộc tính chỉ định những gì sẽ hiển thị cho tên của một trường (trong trường hợp này là “Ngày phát hành” thay vì “Ngày phát hành”). Loại dữ liệu thuộc tính chỉ định loại dữ liệu, trong này trường hợp đó là một ngày, vì vậy thông tin thời gian được lưu trữ trong trường không được hiển thị.

Duyệt đến bộ điều khiển Phim và giữ con trỏ chuột qua liên kết Chính sửa để xem URL đích.

Genre	Price	Release Date	Title	
Romantic Comedy	7.99	1/11/1989	When Harry Met Sally	Edit Details Delete
Comedy	8.99	3/13/1984	Ghostbusters	Edit Details Delete
Comedy	9.99	2/23/1986	Ghostbusters 2	Edit Details Delete
Western	3.99	4/15/1959	Rio Bravo	Edit Details Delete

© 2015 - MvcMovie

<http://localhost:1234/Movies/Edit/7>

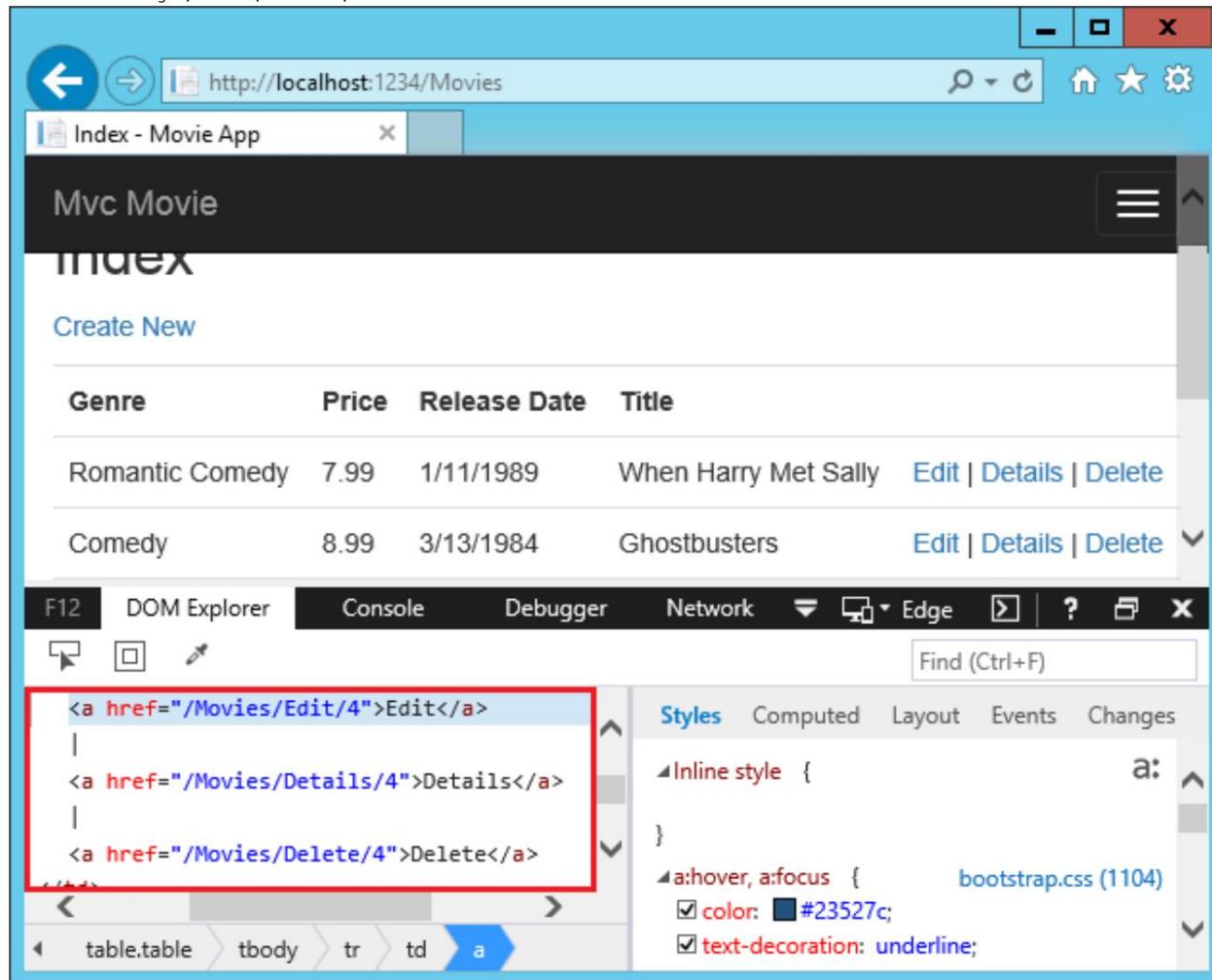
Các liên kết Chính sửa, Chi tiết và Xóa được tạo bởi Trình trợ giúp thẻ neo MVC 6 trong Lượt xem / Phim / Index.cshtml tập tin.

```

1 <td>
2     <a asp-action="Edit" asp-route-id="@item.ID"> Chính sửa </a> |
3     <a asp-action="Details" asp-route-id="@item.ID"> Chi tiết </a> |
4     <a asp-action="Delete" asp-route-id="@item.ID"> Xóa </a>
5 </td>

```

Trình trợ giúp thẻ cho phép mã phía máy chủ tham gia vào việc tạo và hiển thị các phần tử HTML trong tệp Razor. bên trong mã ở trên, Trình trợ giúp thẻ neo tự động tạo giá trị thuộc tính href trong HTML từ hành động của trình điều khiển phương thức và id tuyến đường. Bạn sử dụng View Source từ trình duyệt yêu thích của mình hoặc sử dụng các công cụ F12 để kiểm tra đánh dấu. Các công cụ F12 được hiển thị bên dưới.



Nhớ lại định dạng cho bộ định tuyến trong tệp Startup.cs.

```

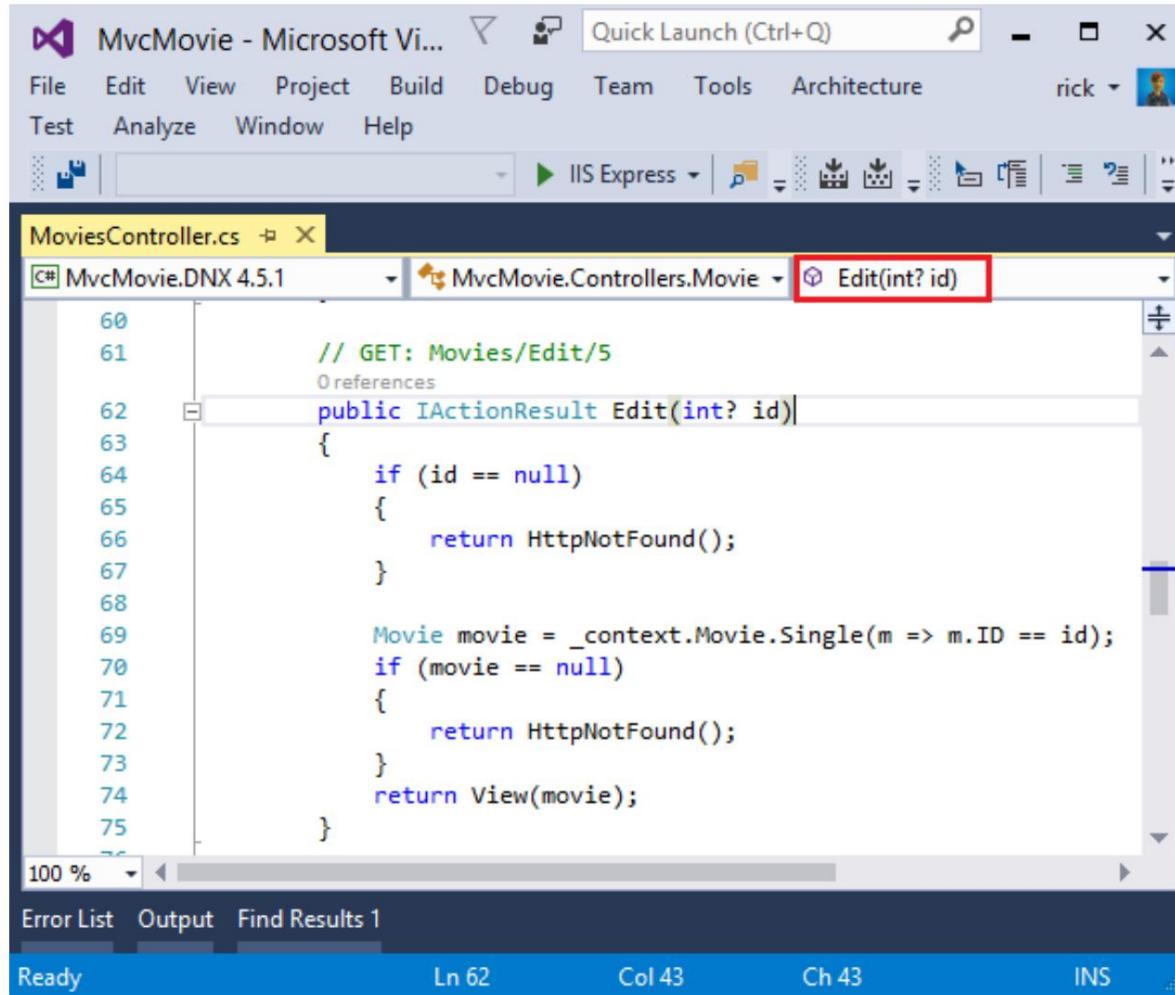
1 app.UseMvc (các tuyến đường =>
2 {
3     các tuyến đường.MapRoute (
4         tên: "mặc định",
5         mẫu: "{controller = Home} / {action = Index} / {id?}");
6 });

```

ASP.NET dịch http://localhost:1234/Movies/Edit/4 thành một yêu cầu đối với phương thức hành động Chính sửa của bộ điều khiển Phim với ID tham số là 4. (Các phương thức bộ điều khiển còn được gọi là **phương thức hành động**.)

Trình trợ giúp thẻ là một trong những tính năng mới phổ biến nhất trong ASP.NET 5. Xem Tài nguyên bổ sung để biết thêm thông tin.

Mở bộ điều khiển Phim và kiểm tra hai phương pháp hành động Chính sửa:



The screenshot shows the Microsoft Visual Studio IDE interface. The title bar says "MvcMovie - Microsoft Vi...". The menu bar includes File, Edit, View, Project, Build, Debug, Team, Tools, Architecture, Test, Analyze, Window, Help. A user profile "rick" is shown in the top right. The toolbar has various icons for file operations like Open, Save, and Print. The main window displays the code for "MoviesController.cs" under the "MvcMovie.DNX 4.5.1" project. The "MvcMovie.Controllers.Movie" namespace is selected. The "Edit(int? id)" method is highlighted with a red box. The code implements a GET request for the /Edit/{id} route, checking if the ID is null and returning a 404 error if true. It then finds a movie by ID and returns its view if found, or a 404 error if not. The status bar at the bottom shows "Ready", "Ln 62", "Col 43", "Ch 43", and "INS".

```

60
61     // GET: Movies/Edit/5
62     public IActionResult Edit(int? id)
63     {
64         if (id == null)
65         {
66             return NotFound();
67         }
68
69         Movie movie = _context.Movie.Single(m => m.ID == id);
70         if (movie == null)
71         {
72             return NotFound();
73         }
74         return View(movie);
75     }

```

1 chỉnh sửa IActionResult công khai (int? Id)

```

2 {
3     if (id == null)
4     {
5         trả về HttpNotFound ();
6     }
7
8     Movie movie = _context.Movie.Single (m => m.ID == id);
9     if (phim == null)
10    {
11        trả về HttpNotFound ();
12    }
13    return View (phim);
14 }
15
16 // BÀI ĐÁNG: Phim / Chính sửa / 5
17 [HttpPost]
18 [ValidateAntiForgeryToken]
19 public IActionResult Edit (Movie movie)
20 {
21     if (ModelState.IsValid)
22     {
23         _context.Update (phim);
24         _context.SaveChanges ();

```

```

25     return RedirectToAction ("Chỉ mục");
26 }
27 return View (phim);
28 }
```

Lưu ý: Mã công cụ giàn giáo được tạo ở trên có một lỗ hổng bảo mật nghiêm trọng do đăng tải quá mức. Hãy chắc chắn bạn hiểu cách bảo vệ khỏi việc đăng quá mức trước khi bạn xuất bản ứng dụng của mình. Lỗ hổng bảo mật này cần được sửa trong bản phát hành tiếp theo.

Thay thế phương thức hành động HTTP POST Edit bằng phương thức sau:

```

1 // BÀI ĐĂNG: Phim / Chính sửa / 6
2 [HttpPost]
3 [ValidateAntiForgeryToken]
4 chỉnh sửa IActionResult công khai (
5     [Ràng buộc ("ID, Tiêu đề, Ngày phát hành, Thể loại, Giá")] Phim điện ảnh)
6 {
7     if (ModelState.IsValid)
8     {
9         _context.Update (phim);
10        _context.SaveChanges ();
11        return RedirectToAction ("Chỉ mục");
12    }
13    return View (phim);
14 }
```

Thuộc tính [Bind] là một cách để bảo vệ chống lại việc đăng quá mức. Bạn chỉ nên đưa các thuộc tính vào [Bind] thuộc tính mà bạn muốn thay đổi. Áp dụng thuộc tính [Bind] cho từng phương thức hành động [HttpPost]. Nhìn thấy Bảo vệ bộ điều khiển của bạn khỏi việc đăng tải quá mức để biết thêm thông tin.

Lưu ý rằng phương thức hành động Chính sửa thứ hai được đặt trước thuộc tính [HttpPost].

```

1 // BÀI ĐĂNG: Phim / Chính sửa / 6
2 [HttpPost]
3 [ValidateAntiForgeryToken]
4 chỉnh sửa IActionResult công khai (
5     [Ràng buộc ("ID, Tiêu đề, Ngày phát hành, Thể loại, Giá")] Phim điện ảnh)
6 {
7     if (ModelState.IsValid)
8     {
9         _context.Update (phim);
10        _context.SaveChanges ();
11        return RedirectToAction ("Chỉ mục");
12    }
13    return View (phim);
14 }
```

Thuộc tính [HttpPost] chỉ định rằng phương thức Chính sửa này chỉ có thể được gọi cho các yêu cầu POST. Bạn có thể áp dụng thuộc tính [HttpGet] cho phương thức chỉnh sửa đầu tiên, nhưng điều đó không cần thiết vì [HttpGet] là mặc định.

Thuộc tính [ValidateAntiForgeryToken] được sử dụng để ngăn chặn việc giả mạo một yêu cầu và được ghép nối với một mã thông báo chống giả mạo được tạo trong tệp chép đè xem chỉnh sửa (Lượt xem / Phim / Edit.cshtml). Tệp dạng xem chỉnh sửa tạo mã thông báo chống giả mạo trong Trình trợ giúp thẻ biểu mẫu.

```
<form asp-action = "Chỉnh sửa">
```

Trình trợ giúp thẻ biểu mẫu tạo mã thông báo chống giả mạo cần phải khớp với [ValidateAntiForgeryToken] đã tạo mã thông báo chống giả mạo trong phương pháp Chính sửa của bộ điều khiển Phim. Để biết thêm thông tin, hãy xem [Chống yêu cầu giả mạo](#).

Phương thức `HttpGet Edit` lấy tham số ID phim, tra cứu phim bằng phương thức Entity Framework Single và trả phim đã chọn về dạng xem Chỉnh sửa. Nếu không tìm thấy phim, `NotFound` sẽ được trả về.

```

1 // NHÂN: Phim / Chính sửa / 5
2 chỉnh sửa IActionResult công khai (int? Id)
3 {
4     if (id == null)
5     {
6         trả về NotFound ();
7     }
8
9     Movie movie = _context.Movie.Single (m => m.ID == id); if (phim == null)
10
11     {
12         trả về NotFound ();
13
14     } return View (phim);
15 }
```

Khi hệ thống giàn giáo tạo ché độ xem Chỉnh sửa, nó đã kiểm tra lớp Phim và tạo mã để hiển thị các phần tử `<label>` và `<input>` cho từng thuộc tính của lớp. Ví dụ sau cho thấy ché độ xem Chỉnh sửa được tạo bởi hệ thống giàn giáo studio trực quan:

```

@model MvcMovie.Models.Movie

{
    ViewData ["Title"] = "Chỉnh sửa";
}



## Chỉnh sửa </h2> <form asp-action = "Chinh sửa"> <div class = "form-ngang"> <h4> Phim </h4> <hr /> <div asp-validation- Summary = "ValidationSummary.ModelOnly" class = "text-risk" /> <input type = "hidden" asp-for = "ID" /> <div class = "form-group"> <label asp-for = "Thể loại" class = "control-label col-md-2" /> <div class = "col-md-10"> <input asp-for = "Genre" class = "form-control" /> <span asp- validation-for = "Genre" class = "text-risk" /> </div> </div> <div class = "form-group"> <label asp-for = "Price" class = "control-label col-md-2" /> <div class = "col-md-10"> <input asp-for = "Price" class = "form-control" /> <span asp- validation-for = "Price" class = "text-risk" /> </div> </div> @ * Ngày phát hành và Đã xóa tiêu đề cho ngắn gọn. * @ <div class = "form-group"> <div class = "col-md-offset-2 col-md-10"> <input type = "submit" value = "Lưu" class = "btn btn-default" /> </div> </div> </div>


```

```
</form>

<div>
    <a asp-action="Index"> Quay lại danh sách </a> </div>

@section Scripts { <script
    src = "~ / lib / jquery / dist / jquery.min.js"> </script> <script src = "~ / lib /
    jquery-validation / jquery.validate.min.js"> </script> <script src = "~ / lib / jquery-validation-unobtrusive /
    jquery.validate.unobtrusive.min.js"> </script>
}
```

Lưu ý cách mẫu chế độ xem có câu lệnh @model MvcMovie.Models.Movie ở đầu tệp - điều này chỉ định rằng chế độ xem yêu cầu mô hình cho mẫu chế độ xem thuộc loại Phim.

Mã mở rộng sử dụng một số phương pháp Trình trợ giúp thẻ để hợp lý hóa đánh dấu HTML. Trình trợ giúp thẻ `nhấn` hiển thị tên của trường ("Tiêu đề", "Ngày phát hành", "Thể loại" hoặc "Giá"). Trình trợ giúp thẻ `đầu vào` vào hiển thị một phần tử HTML `<input>`. Trình trợ giúp Thẻ xác thực hiển thị bất kỳ thông báo xác thực nào được liên kết với thuộc tính đó.

Chạy ứng dụng và điều hướng đến URL / Phim. Nhập vào liên kết Cảnh sửa . Trong trình duyệt, hãy xem nguồn của trang. HTML được tạo cho phần tử `<form>` được hiển thị bên dưới.

```
<form action = "/ Movies / Edit / 7" method = "post">
    <div class = "form-ngang">
        <h4> Phim </h4> <hr />
        <> <div class = "text-risk" /> <input type = "hidden" data-val =
        = "true" data-val-Required = "Trường ID là bắt buộc. " id = "ID" n <div class = "form-group">
            <label class = "control-label col-md-2" for = "Thể loại" /> <div class = "col-
            md-10">
                <input class = "form-control" type = "text" id = "Genre" name = "Genre" value = "Western" /> <span class = "text-risk field-
                validation-valid" data-valmsg-for = "Thể loại" data-valmsg </div> </div> <div class = "form-group">
                    <label class = "control-label col-md-2" for = "Price" /> <div class = "col-md-10">
                        <input class = "form-control" type = "text" data-val = "true" data-val-number = "Trường P <span class ="
                            text-risk field-validation-
                            valid" data-valmsg-for = "Giá" data-valmsg </div> </div> <! - Đã xóa đánh dấu cho ngắn gọn -> <div class = "form-group">
                            <div class = "col-md-offset-2 col-md-10">
                                <input type = "submit" value = "Save" class = "btn btn-default" /> </div> </div> <input
                                name = "__RequestVerificationToken" type = "hidden" value = "CfDJ8Inyxgp63fRFqUePGvuI5jGzsloJu1" />
    </form>
```

Các phần tử `<input>` nằm trong phần tử `<form>` HTML có thuộc tính `action` được đặt để đăng lên URL / Movies / Edit / id. Dữ liệu biểu mẫu sẽ được đăng lên máy chủ khi nhập vào nút Lưu. Dòng cuối cùng trước phần tử đóng `</form>` hiển thị mã thông báo XSRF ẩn được tạo bởi Trình trợ giúp thẻ `biểu mẫu`.

Xử lý Yêu cầu ĐĂNG

Danh sách sau đây hiển thị phiên bản [HttpPost] của phương pháp hành động Chính sửa.

```
// ĐĂNG: Phim / Chính sửa / 6
[HttpPost]
[ValidateAntiForgeryToken] chỉnh
sửa IActionResult công khai ()
{
    if (ModelState.IsValid) {

        _context.Update (phim);
        _context.SaveChanges ();
        return RedirectToAction ("Chỉ mục");

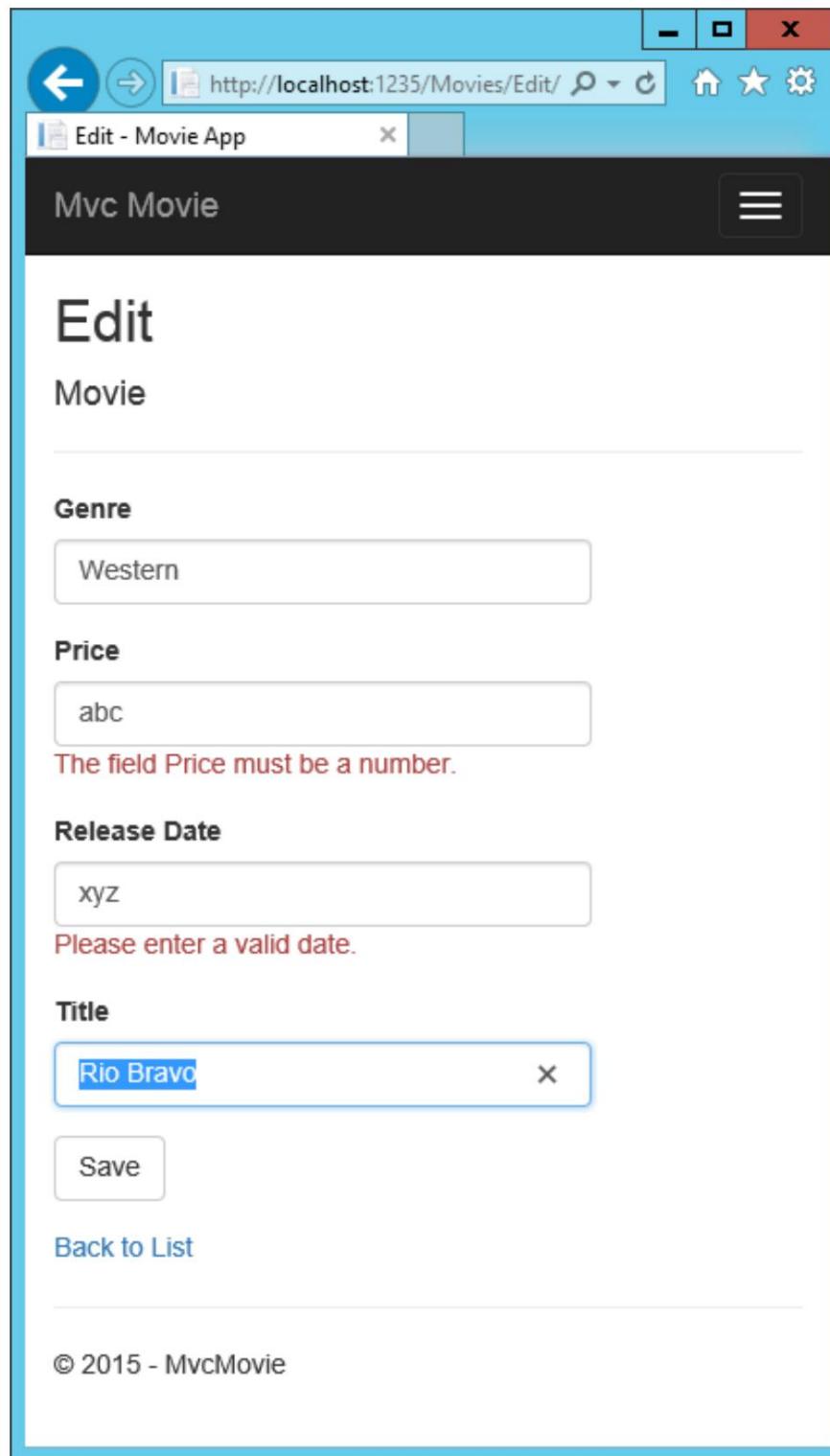
    } return View (phim);
}
```

Thuộc tính [ValidateAntiForgeryToken] xác thực mã thông báo XSRF ẩn được tạo bởi trình tạo mã thông báo chống giả mạo trong Trình trợ giúp thẻ biểu mẫu.

Chất kết dính mô hình ASP.NET MVC nhận các giá trị biểu mẫu đã đăng và tạo một đối tượng Phim được truyền dưới dạng tham số phim. Phương thức ModelState.IsValid xác minh rằng dữ liệu được gửi trong biểu mẫu có thể được sử dụng để sửa đổi (chỉnh sửa hoặc cập nhật) đối tượng Movie. Nếu dữ liệu hợp lệ, dữ liệu phim được lưu vào bộ sưu tập Phim của cơ sở dữ liệu (phiên bản ApplicationDbContext). Dữ liệu phim mới được lưu vào cơ sở dữ liệu bằng cách gọi phương thức SaveChanges của ApplicationDbContext. Sau khi lưu dữ liệu, mã chuyển hướng người dùng đến phương thức hành động Index của lớp MoviesController, phương thức này hiển thị bộ sưu tập phim, bao gồm cả những thay đổi vừa được thực hiện.

Ngay sau khi xác thực phía máy khách xác định các giá trị của trường không hợp lệ, một thông báo lỗi sẽ được hiển thị. Nếu bạn tắt JavaScript, bạn sẽ không có xác thực phía máy khách nhưng máy chủ sẽ phát hiện các giá trị đã đăng không hợp lệ và các giá trị biểu mẫu sẽ được hiển thị lại với các thông báo lỗi. Sau đó trong hướng dẫn, chúng tôi sẽ kiểm tra xác thực chi tiết hơn.

Trình trợ giúp thẻ xác thực trong mẫu chế độ xem Views / Book / Edit.cshtml sẽ quan tâm đến việc hiển thị các thông báo lỗi thích hợp.



Tất cả các phương thức `HttpGet` đều tuân theo một mẫu tương tự. Họ lấy một đối tượng phim (hoặc danh sách các đối tượng, trong trường hợp là Chi mục), và chuyển mô hình đến dạng xem. Phương thức Tạo chuyển một đối tượng phim trống đến dạng xem Tạo. Tất cả các phương thức tạo, chỉnh sửa, xóa hoặc sửa đổi dữ liệu đều làm như vậy trong trạng thái quá tải `[HttpPost]` của phương thức. Sửa đổi dữ liệu trong phương thức HTTP GET là một rủi ro bảo mật, như được mô tả trong bài đăng blog [ASP.NET MVC Mẹo # 46 - Không sử dụng Xóa liên kết vì chúng tạo Lỗ bảo mật](#). Việc sửa đổi dữ liệu trong phương thức HTTP GET cũng vi phạm các phương pháp hay nhất của HTTP và REST kiến trúc mẫu, chỉ định rằng các yêu cầu GET không được thay đổi trạng thái.

ứng dụng của bạn. Nói cách khác, thực hiện một hoạt động GET phải là một hoạt động an toàn không có tác dụng phụ và không sửa đổi dữ liệu vẫn tồn tại của bạn.

Các nguồn bổ sung

- Toàn cầu hóa và bản địa hóa
- Giới thiệu về Trình trợ giúp thẻ
- Trình trợ giúp thẻ tác giả
- Chống yêu cầu giả mạo
- Bảo vệ bộ điều khiển của bạn khỏi việc đăng tải quá mức
- Trình trợ giúp thẻ biểu mẫu
- Trình trợ giúp thẻ nhãn
- Trình trợ giúp thẻ đầu vào
- Trình trợ giúp thẻ xác thực
- Trình trợ giúp thẻ neo
- Chọn Trình trợ giúp thẻ

2.1.7 Thêm tìm kiếm

Bởi Rick Anderson

Thêm phương pháp tìm kiếm và chế độ xem tìm kiếm

Trong phần này, bạn sẽ thêm khả năng tìm kiếm vào phương pháp hành động Index cho phép bạn tìm kiếm phim theo thẻ loại hoặc tên.

Cập nhật chỉ mục

Bắt đầu bằng cách cập nhật phương pháp hành động Chỉ mục để cho phép tìm kiếm. Đây là mã:

```

1   public IActionResult Index (chuỗi tìm kiếm)
2   {
3       var phim = from m trong _context.Movie
4           chọn m;
5
6       if (! String.IsNullOrEmpty (searchString))
7       {
8           phim = phim.Where (s => s.Title.Contains (searchString));
9       }
10
11       return View (phim);
12   }

```

Dòng đầu tiên của phương pháp hành động Chỉ mục tạo ra một LINQ truy vấn để chọn phim:

```
var phim = from m trong _context.Movie
```

Truy vấn chỉ được xác định tại thời điểm này, nó chưa được chạy trên cơ sở dữ liệu.

Nếu tham số searchString chứa một chuỗi, truy vấn phim sẽ được sửa đổi để lọc giá trị của tìm kiếm chuỗi, sử dụng mã sau:

```
if (! String.IsNullOrEmpty (searchString)) {

    phim = phim.Where (s => s.Title.Contains (searchString));
}
```

Đoạn mã `s => s.Title` ở trên là một **Biểu thức Lambda**. Lambdas được sử dụng trong **LINQ** dựa trên phương pháp các truy vấn dưới dạng đối số cho các phương thức toán tử truy vấn tiêu chuẩn, chẳng hạn như `Where` phương pháp được sử dụng trong đoạn mã trên. Các truy vấn LINQ không được thực thi khi chúng được định nghĩa hoặc khi chúng được sửa đổi bằng cách gọi một phương thức như `Where` hoặc `OrderBy`. Thay vào đó, việc thực thi truy vấn bị hoãn lại, có nghĩa là việc đánh giá một biểu thức bị trì hoãn cho đến khi giá trị nhận ra của nó thực sự được lặp lại hoặc phương thức `ToList` được gọi. Trong mẫu Tìm kiếm, truy vấn được thực thi trong dạng xem `Index.cshtml`. Để biết thêm thông tin về thực thi truy vấn hoãn lại, hãy xem [Thực thi truy vấn](#). Lưu ý: `Chứa` phương thức được chạy trên cơ sở dữ liệu, không phải mã `c #` ở trên. Trên cơ sở dữ liệu, `Chứa` ánh xạ tới SQL `LIKE`, không phân biệt chữ hoa chữ thường.

Bây giờ bạn có thể cập nhật dạng xem Chỉ mục để hiển thị biểu mẫu cho người dùng.

Điều hướng đến / Phim / Chỉ mục. Nối một chuỗi truy vấn chẳng hạn như? `searchString = ghost` vào URL. Các phim đã lọc được hiển thị.

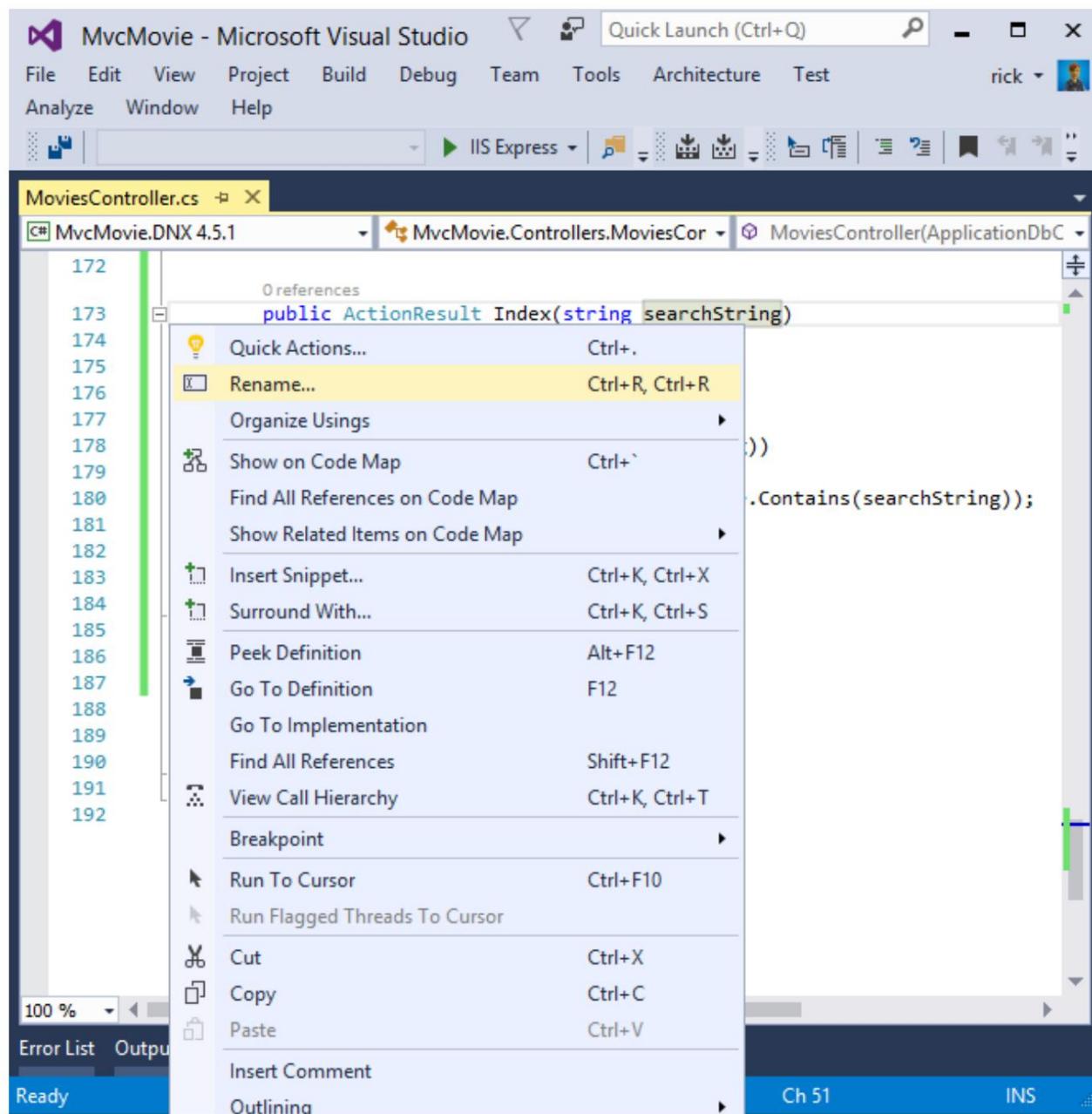
Genre	Price	Release Date	Title	
Comedy	8.99	3/13/1984	Ghostbusters	Edit Details Delete
Comedy	9.99	2/23/1986	Ghostbusters 22	Edit Details Delete

© 2015 - MvcMovie

Nếu bạn thay đổi ký tự của phương thức Chỉ mục để có tham số có tên `id`, thì tham số `id` sẽ khớp với trình giữ chỗ `{id}` tùy chọn cho các tuyến mặc định được đặt trong `Startup.cs`.

```
mẫu: "{controller = Home} / {action = Index} / {id?}";
```

Bạn có thể nhanh chóng đổi tên tham số `searchString` thành `id` bằng lệnh `rename`. Nhấp chuột phải vào chuỗi tìm kiếm > **Đổi tên**.



Các mục tiêu đổi tên được đánh dấu.

```

public ActionResult Index(string searchString)
{
    var movies = from m in _context.Movie
                 select m;

    if (!String.IsNullOrEmpty(searchString))
    {
        movies = movies.Where(s => s.Title.Contains(searchString));
    }

    return View(movies);
}

```

Thay đổi tham số thành id và tất cả các lần xuất hiện của chuỗi tìm kiếm đều thay đổi thành id.

```

public ActionResult Index(string id)
{
    var movies = from m in _context.Movie
                 select m;

    if (!String.IsNullOrEmpty(id))
    {
        movies = movies.Where(s => s.Title.Contains(id));
    }

    return View(movies);
}

```

Phương pháp chỉ mục trước đó:

```

1 chỉ số IActionResult công khai (chuỗi tìm kiếm)
2 {
3     var phim = from m trong _context.Movie
4                 chọn m;
5
6     if (! String.IsNullOrEmpty (searchString))
7     {
8         phim = phim.Where (s => s.Title.Contains (searchString));
9     }
10
11     return View (phim);
12 }

```

Phương pháp chỉ mục được cập nhật:

```

1 chỉ số IActionResult công khai (chuỗi id)
2 {
3     var phim = from m trong _context.Movie
4                 chọn m;
5
6     if (! String.IsNullOrEmpty (id))
7     {
8         phim = phim.Where (s => s.Title.Contains (id));
9     }
10
11     return View (phim);
12 }

```

Giờ đây, bạn có thể chuyển tiêu đề tìm kiếm dưới dạng dữ liệu truyền đường (phân đoạn URL) thay vì dưới dạng giá trị chuỗi truy vấn.

Genre	Price	Release Date	Title	
Comedy	8.99	3/13/1984	Ghostbusters	Edit Details Delete
Comedy	9.99	2/23/1986	Ghostbusters 22	Edit Details Delete

© 2015 - MvcMovie

Tuy nhiên, bạn không thể mong đợi người dùng sửa đổi URL mỗi khi họ muốn tìm kiếm phim. Vì vậy, bây giờ bạn sẽ thêm giao diện người dùng để giúp họ lọc phim. Nếu bạn đã thay đổi chữ ký của phương thức Chỉ mục để kiểm tra cách truyền tham số ID giới hạn theo tuyén, hãy thay đổi lại nó để nhận tham số có tên là searchString:

```
public IActionResult Index (string searchString) {

    var phim = from m trong _context.Movie
               chọn m;

    if (! String.IsNullOrEmpty (searchString)) {

        phim = phim.Where (s => s.Title.Contains (searchString));
    }

    return View (phim);
}
```

Mở tệp Views / Movies / Index.cshtml và thêm đánh dấu <form> được đánh dấu bên dưới:

```
1 @model IEnumerable <MvcMovie.Models.Movie>
2
3 @{
4     ViewData ["Title"] = "Chỉ mục";
5 }
6
7 <h2> Chỉ mục </h2>
..
```

```

9 <p>
10    <a asp-action="Create"> Tạo mới </a>
11 </p>
12
13 <form asp-controller = "Phim" asp-action = "Index">
14    <p>
15        Tiêu đề: <input type = "text" name = "SearchString">
16        <input type = "submit" value = "Filter" />
17    </p>
18 </form>
19
20 <table class = "table">
21    <tr>

```

Thẻ HTML <form> được tính phí siêu cao bởi Trình trợ giúp thẻ biểu mẫu, vì vậy khi bạn gửi biểu mẫu, chuỗi bộ lọc là được đăng lên hành động Index của bộ điều khiển phim. Lưu các thay đổi của bạn và sau đó kiểm tra bộ lọc.

Genre	Price	Release Date	Title	
Comedy	8.99	3/13/1984	Ghostbusters	Edit Details Delete
Comedy	9.99	2/23/1986	Ghostbusters 2	Edit Details Delete

Không có quá tài [HttpPost] của phương thức Chỉ mục. Bạn không cần nó, bởi vì phương pháp này không thay đổi trạng thái của ứng dụng, chỉ lọc dữ liệu.

Bạn có thể thêm phương pháp lập chỉ mục [HttpPost] sau đây.

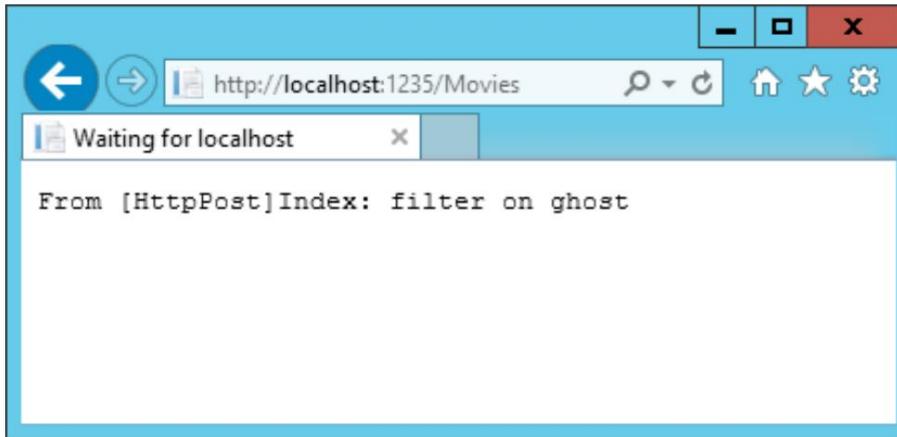
```

[HttpPost]
Chỉ mục chuỗi công khai (FormCollection fc, string searchString)
{

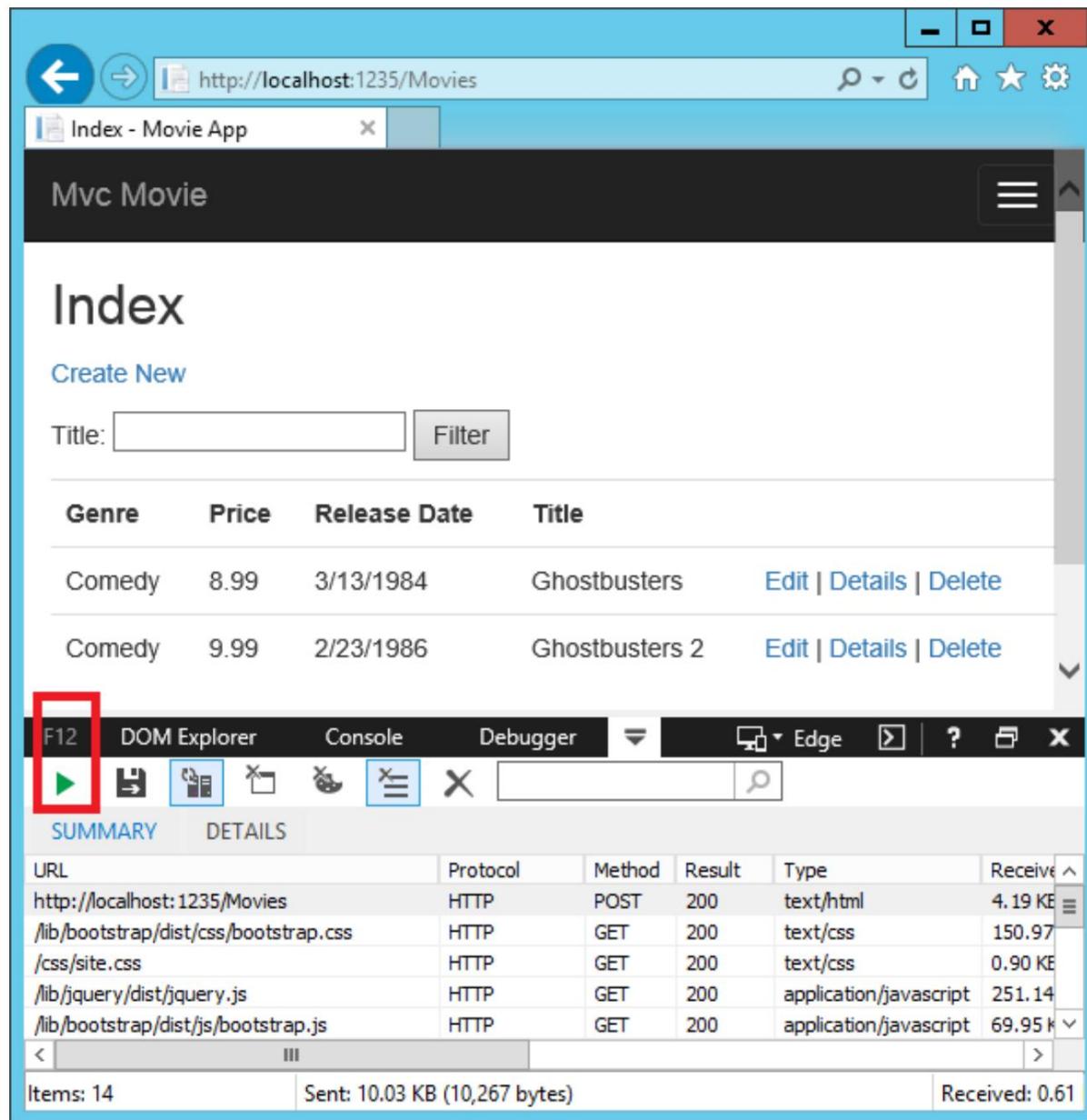
```

```
    return "Từ [HttpPost] Chỉ mục: bật bộ lọc" + chuỗi tìm kiếm;
}
```

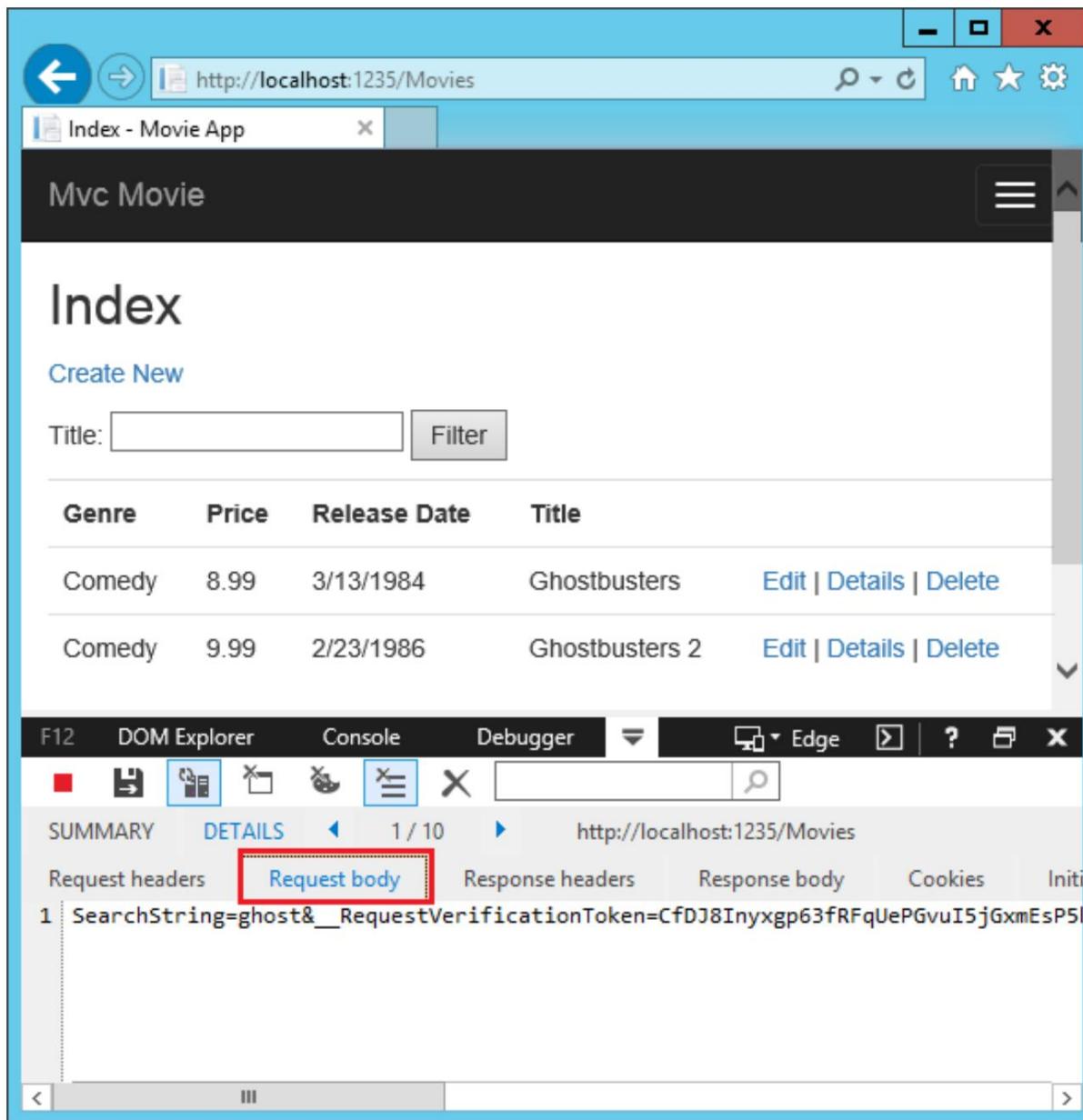
Nếu bạn đã làm như vậy, trình gọi hành động sẽ khớp với phương thức Chỉ mục [HttpPost] và phương thức Chỉ mục [HttpPost] sẽ chạy như thể hiện trong hình ảnh bên dưới.



Tuy nhiên, ngay cả khi bạn thêm phiên bản [HttpPost] này của phương thức Chỉ mục, vẫn có một giới hạn trong cách thực hiện tất cả điều này. Hãy tưởng tượng rằng bạn muốn đánh dấu một tìm kiếm cụ thể hoặc bạn muốn gửi một liên kết cho bạn bè mà họ có thể nhấp vào để xem cùng một danh sách phim đã lọc. Lưu ý rằng URL cho yêu cầu HTTP POST giống với URL cho yêu cầu GET (localhost: xxxxx / Movies / Index) - không có thông tin tìm kiếm trong chính URL. Ngay bây giờ, thông tin chuỗi tìm kiếm được gửi đến máy chủ dưới dạng giá trị trường biểu mẫu. Bạn có thể xác minh điều đó bằng công cụ F12 Developer hoặc công cụ Fiddler tuyệt vời. Khởi động công cụ F12 và nhấn vào biểu tượng Bật thu thập lưu lượng mạng .

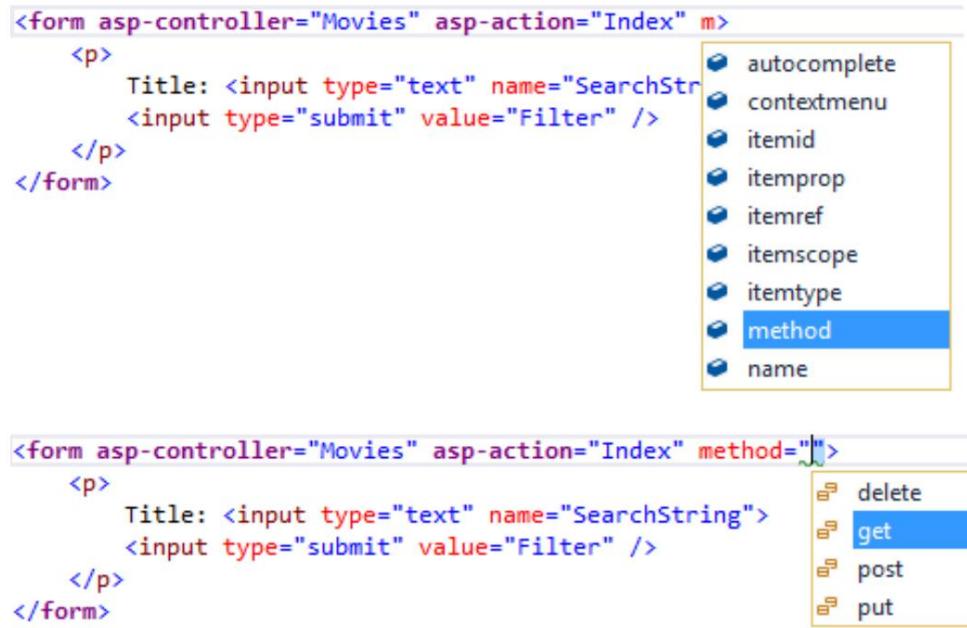


Nhấn đúp vào dòng `http://localhost:1235/Movies` HTTP POST 200 , sau đó nhấn vào Nội dung yêu cầu.



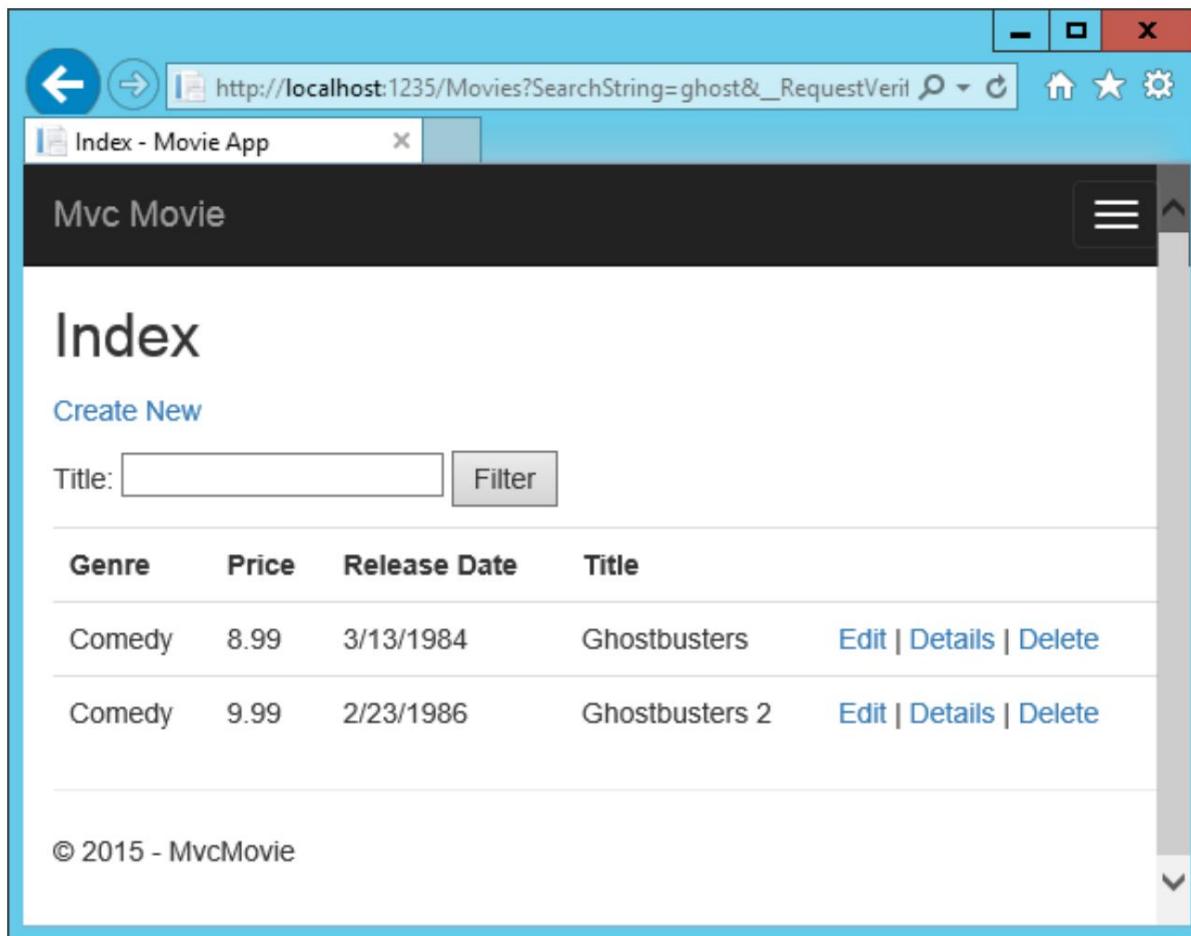
Bạn có thể thấy thông số tìm kiếm và mã thông báo CSRF trong nội dung yêu cầu. Lưu ý, như đã đề cập trong hướng dẫn trước, Trình trộn giúp thẻ biểu mẫu tạo mã thông báo chống giả mạo CSRF. Chúng tôi không sửa đổi dữ liệu, vì vậy chúng tôi không cần xác thực mã thông báo trong phương thức bộ điều khiển.

Vì thông số tìm kiếm nằm trong phần nội dung yêu cầu chứ không phải URL nên bạn không thể nắm bắt thông tin tìm kiếm đó để đánh dấu trang hoặc chia sẻ với người khác. Chúng tôi sẽ khắc phục điều này bằng cách chỉ định yêu cầu phải là HTTP GET. Lưu ý cách intelliSense giúp chúng tôi cập nhật đánh dấu.



Lưu ý phông chữ đặc biệt trong thẻ `<form>`. Phông chữ đặc biệt đó cho biết thẻ được hỗ trợ bởi Trình trợ giúp thẻ.

Bây giờ khi bạn gửi một tìm kiếm, URL sẽ chứa chuỗi truy vấn tìm kiếm. Tìm kiếm cũng sẽ chuyển đến phương thức hành động `HttpGet Index`, ngay cả khi bạn có phương thức `HttpPost Index`.



Mã thông báo XSRF và bất kỳ phần tử biểu mẫu đã đăng nào khác cũng sẽ được thêm vào URL.

Thêm tìm kiếm theo thể loại

Thay thế phương pháp Chỉ mục bằng mã sau:

```
chGenre

public IActionResult Index (string movieGenre, string searchString) {

    var GenreQry = from m in _context.Movie orderby
                    m.Genre select m.Genre;

    var GenreList = new Danh sách <string> ();
    GenreList.AddRange (GenreQry.Distinct()); ViewData
    ["movieGenre"] = new SelectList (GenreList);

    var phim = from m trong _context.Movie
               chọn m;

    if (! String.IsNullOrEmpty (searchString)) {

        phim = phim.Where (s => s.Title.Contains (searchString));
    }
}
```

```

if (! string.IsNullOrEmpty (movieGenre)) {

    phim = phim.Where (x => x.Genre == phimGenre);
}

return View (phim);
}

```

Phiên bản này của phương thức Index nhận tham số movieGenre. Một vài dòng mã đầu tiên tạo một đối tượng Danh sách để chứa các thể loại phim từ cơ sở dữ liệu.

Đoạn mã sau là một truy vấn LINQ lấy tất cả các thể loại từ cơ sở dữ liệu.

```
var GenreQry = from m in _context.Movie orderby m.Genre
```

Mã sử dụng `AddRange` phương pháp của `danh sách` chung bộ sưu tập để thêm tất cả các thể loại riêng biệt vào danh sách. (Không có công cụ sửa đổi Phân biệt, các thể loại trùng lặp sẽ được thêm vào - ví dụ: hài kịch sẽ được thêm hai lần vào mẫu của chúng tôi). Sau đó, mã lưu trữ danh sách các thể loại trong từ điển ViewData. Lưu trữ dữ liệu danh mục (thể loại phim chẳng hạn) dưới dạng đối tượng SelectList trong từ điển ViewData, sau đó truy cập dữ liệu danh mục trong hộp danh sách thả xuống là một cách tiếp cận điển hình cho các ứng dụng MVC.

Đoạn mã sau đây cho biết cách kiểm tra tham số movieGenre. Nếu nó không trống, mã hạn chế thêm truy vấn phim để giới hạn các phim đã chọn ở thể loại được chỉ định.

```

if (! string.IsNullOrEmpty (movieGenre)) {

    phim = phim.Where (x => x.Genre == phimGenre);
}

```

Như đã nêu trước đây, truy vấn không được chạy trên cơ sở dữ liệu cho đến khi danh sách phim được lặp lại (điều này xảy ra trong Chế độ xem, sau khi phương thức hành động Chỉ mục trả về).

Thêm tìm kiếm theo thể loại vào chế độ xem Chỉ mục

Thêm trình trợ giúp `Html.DropDownList` vào tệp Views / Movies / Index.cshtml. Đánh dấu đã hoàn thành được hiển thị là thấp:

```

1 <form asp-controller = "Phim" asp-action = "Index" method = "get">
2     <p>
3         Thể loại: @ Html.DropDownList ("movieGenre", "All")
4         Tiêu đề: <input type = "text" name = "SearchString"> <input type
5             = "submit" value = "Filter" /> </p> 7 </form>
6

```

Lưu ý: Phiên bản tiếp theo của hướng dẫn này sẽ thay thế trình trợ giúp `Html.DropDownList` bằng Trình trợ giúp thể chọn.

Kiểm tra ứng dụng bằng cách tìm kiếm theo thể loại, theo tên phim và cả hai.

2.1.8 Thêm một trường mới

Bởi Rick Anderson

Trong phần này, bạn sẽ sử dụng Khung thực thể Code First Migrations để di chuyển một số thay đổi đối với các lớp mô hình để thay đổi được áp dụng cho cơ sở dữ liệu.

Theo mặc định, khi bạn sử dụng Entity Framework Code First để tự động tạo cơ sở dữ liệu, như bạn đã làm trước đó trong hướng dẫn, Code First thêm một bảng vào cơ sở dữ liệu để giúp theo dõi xem lược đồ của cơ sở dữ liệu có đồng bộ với các lớp mô hình mà nó được tạo ra. Nếu chúng không đồng bộ, khung thực thể sẽ gây ra lỗi. Điều này làm cho nó dễ dàng hơn để theo dõi các vấn đề tại thời điểm phát triển mà bạn có thể chỉ tìm thấy (bởi các lỗi khó hiểu) tại thời điểm chạy.

Thêm Thuộc tính Xếp hạng vào Mô hình Phim

Mở tệp Models / Movie.cs và thêm thuộc tính Xếp hạng:

```
1 phim cấp công cộng
2 {
3     public int ID { get; bộ; }
4     chuỗi công khai Tiêu đề { get; bộ; }
5
6     [Hiển thị (Tên = "Ngày phát hành")]
7     [DataType (DataType.Date)]
8     public DateTime ReleaseDate { get; bộ; }
9     chuỗi công khai Thể loại { get; bộ; }
10    giá thập phân công khai { get; bộ; }
11    chuỗi công khai Đánh giá { get; bộ; }
12 }
```

Xây dựng ứng dụng (Ctrl + Shift + B).

Vì bạn đã thêm một trường mới vào lớp Phim, bạn cũng cần cập nhật danh sách trang liên kết nền mới này tài sản sẽ được bao gồm. Cập nhật thuộc tính [Bind] cho các phương pháp hành động Tạo và Chính sửa để bao gồm Xếp hạng tài sản:

```
[Ràng buộc ("ID, Tiêu đề, Ngày phát hành, Thể loại, Giá, Xếp hạng")]
```

Bạn cũng cần cập nhật các mẫu chế độ xem để hiển thị, tạo và chỉnh sửa thuộc tính Xếp hạng mới trong trình duyệt lượt xem.

Chỉnh sửa tệp /Views/Movies/Index.cshtml và thêm trường Xếp hạng:

```
1 <table class = "table">
2     <tr>
3         <th>
4             @ Html.DisplayNameFor (model => model.Genre)
5         </th>
6         <th>
7             @ Html.DisplayNameFor (model => model.Price)
8         </th>
9         <th>
10            @ Html.DisplayNameFor (model => model.ReleaseDate)
11        </th>
12        <th>
13            @ Html.DisplayNameFor (model => model.Title)
14        </th>
15        <th>
16            @ Html.DisplayNameFor (model => model.Rating)
17        </th>
18        <th> </th>
19    </tr>
20
21 @foreach (var item trong Model) {
22     <tr>
23         <td>
24             @ Html.DisplayFor (modelItem => item.Genre)
```

```

25     </td>
26     <td>
27         @ Html.DisplayFor (modelItem => item.Price)
28     </td>
29     <td>
30         @ Html.DisplayFor (modelItem => item.ReleaseDate)
31     </td>
32     <td>
33         @ Html.DisplayFor (modelItem => item.Title)
34     </td>
35     <td>
36         @ Html.DisplayFor (modelItem => item.Rating)
37     </td>
38     <td>

```

Cập nhật /Views/Movies/Create.cshtml với trường Xếp hạng. Bạn có thể sao chép / dán "nhóm biểu mẫu" trước đó và cho phép intelliSense giúp bạn cập nhật các trường. IntelliSense hoạt động với Trình trợ giúp thẻ.

```

</div>
<div class="form-group">
    <label asp-for="Title" class="col-md-2 control-label"></label>
    <div class="col-md-10">
        <input asp-for="Title" class="form-control" />
        <span asp-validation-for="Title" class="text-danger" />
    </div>
</div>
<div class="form-group">
    <label asp-for="Rating" class="col-md-2 control-label"></label>
    <div class="col-<input asp-for="Rating" type="text" class="form-control" />
        <span asp-validation-for="Rating" class="text-danger" />
    </div>
</div>
<div class="form-group">
    <div class="col-<input type="button" value="Tạo" class="btn btn-default" />
</div>
</div>
</form>

```

Những thay đổi được đánh dấu bên dưới:

```

1 <form asp-action = "Tạo">
2     <div class = "form-ngang">
3         <h4> Phim </h4>
4         <giờ />
5         <div asp-validation-summary = "ValidationSummary.ModelOnly" class = "text-risk"> </div>
6         <div class = "form-group">
7             <label asp-for = "Genre" class = "col-md-2 control-label"> </label>
8             <div class = "col-md-10">
9                 <input asp-for = "Thể loại" class = "form-control" />
10                <span asp-validation-for = "Thể loại" class = "text-risk" />
11            </div>
12        </div>
13        @ * Đã xóa đánh dấu cho ngắn gọn. * @
14        <div class = "form-group">

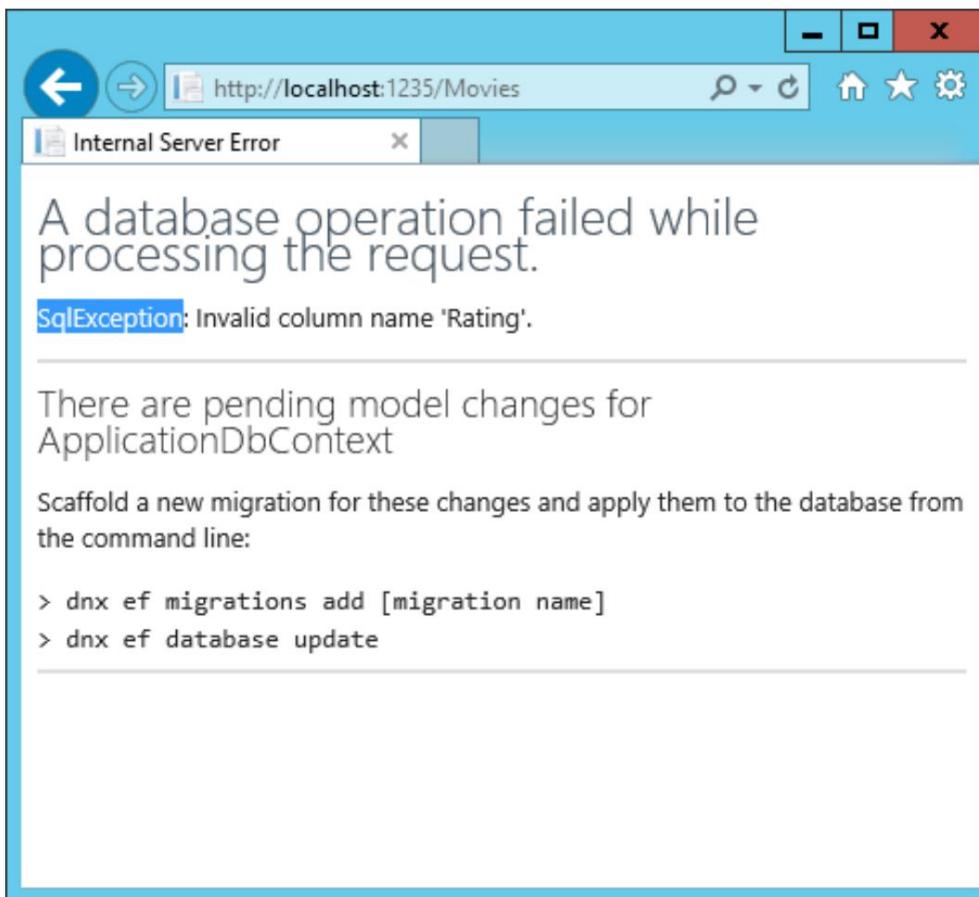
```

```

15 <label asp-for = "Rating" class = "col-md-2 control-label"> </label>
16 <div class = "col-md-10">
17     <input asp-for = "Xếp hạng" class = "form-control" />
18     <span asp-validation-for = "Xếp hạng" class = "text-risk" />
19 </div>
20 </div>
21 <div class = "form-group">
22     <div class = "col-md-offset-2 col-md-10">
23         <input type = "submit" value = "Tạo" class = "btn btn-default" />
24     </div>
25 </div>
26 </div>
27 </form>

```

Ứng dụng sẽ không hoạt động cho đến khi chúng tôi cập nhật DB để bao gồm trường mới. Nếu bạn chạy nó ngay bây giờ, bạn sẽ nhận được những thứ sau SQLException:



Bạn gặp lỗi này vì lớp mô hình Phim được cập nhật trong ứng dụng hiện khác với giản đồ của bảng Movie của cơ sở dữ liệu hiện có. (Không có cột Xếp hạng trong bảng cơ sở dữ liệu.)

Có một số cách tiếp cận để giải quyết lỗi:

- Để Khung thực thể tự động thả và tạo lại cơ sở dữ liệu dựa trên lược đồ lớp mô hình mới.

Cách tiếp cận này rất thuận tiện sớm trong chu kỳ phát triển khi bạn đang thực hiện phát triển tích cực trên một cơ sở dữ liệu thử nghiệm; nó cho phép bạn nhanh chóng phát triển mô hình và lược đồ cơ sở dữ liệu cùng nhau. Tuy nhiên, nhược điểm là rằng bạn mất dữ liệu hiện có trong cơ sở dữ liệu - vì vậy bạn không muốn sử dụng cách tiếp cận này trên cơ sở dữ liệu sản xuất! Sử dụng trình khởi tạo để tự động tạo cơ sở dữ liệu với dữ liệu thử nghiệm thường là một cách hiệu quả để phát triển đăng ký.

2. Sửa đổi rõ ràng lược đồ của cơ sở dữ liệu hiện có để nó phù hợp với các lớp mô hình. Ưu điểm của phương pháp này là bạn giữ được dữ liệu của mình. Bạn có thể thực hiện thay đổi này theo cách thủ công hoặc bằng cách tạo tập lệnh thay đổi cơ sở dữ liệu.

3. Sử dụng Code First Migrations để cập nhật lược đồ cơ sở dữ liệu.

Đối với hướng dẫn này, chúng tôi sẽ sử dụng Code First Migrations.

Cập nhật lớp SeedData để nó cung cấp giá trị cho cột mới. Một thay đổi mẫu được hiển thị bên dưới, nhưng bạn sẽ muốn thực hiện thay đổi này cho mỗi Phim mới.

```

1     phim mới
2     {
3         Title = "Ghostbusters ", ReleaseDate
4         = DateTime.Parse ("1984-3-13"), Thể loại = "Hài", Xếp hạng = "G",
5         Giá = 8,99 triệu
6
7     },
..
```

Xây dựng giải pháp sau đó mở dấu nhắc lệnh. Nhập các lệnh sau:

```
dnx ef migrations thêm Đánh giá cập nhật cơ
sở dữ liệu dnx ef
```

Lệnh bổ sung di chuyển yêu cầu khung di chuyển kiểm tra mô hình Movie hiện tại với lược đồ Movie DB hiện tại và tạo mã cần thiết để di chuyển DB sang mô hình mới. Tên "Xếp hạng" là tùy ý và được sử dụng để đặt tên cho tệp di chuyển. Sẽ rất hữu ích nếu sử dụng một tên có ý nghĩa cho bước di chuyển.

Nếu bạn xóa tất cả các bản ghi trong DB, quá trình khởi tạo sẽ khởi tạo DB và bao gồm trường Xếp hạng. Bạn có thể thực hiện việc này bằng cách liên kết xóa trong trình duyệt hoặc từ SSOX.

Chạy ứng dụng và xác minh rằng bạn có thể tạo / chỉnh sửa / hiển thị phim bằng trường Xếp hạng. Bạn cũng nên thêm trường Xếp hạng vào các mẫu chế độ xem Chính sửa, Chi tiết và Xóa.

2.1.9 Thêm xác thực

Bởi Rick Anderson

Trong phần này, bạn sẽ thêm logic xác thực vào mô hình Phim và bạn sẽ đảm bảo rằng các quy tắc xác thực được thực thi bắt cứ khi nào người dùng cố gắng tạo hoặc chỉnh sửa phim bằng ứng dụng.

Giữ mọi thứ khô

Một trong những nguyên lý thiết kế cốt lõi của ASP.NET MVC là DRY ("Đừng lặp lại chính mình"). ASP.NET MVC khuyến khích bạn chỉ định chức năng hoặc hành vi một lần và sau đó nó được phản ánh ở mọi nơi trong một ứng dụng. Điều này làm giảm số lượng mã bạn cần viết và làm cho mã bạn viết ít bị lỗi hơn, dễ kiểm tra hơn và dễ bảo trì hơn.

Hỗ trợ xác thực được cung cấp bởi ASP.NET MVC và Entity Framework Code First là một ví dụ tuyệt vời về nguyên tắc DRY đang hoạt động. Bạn có thể chỉ định một cách khai báo các quy tắc xác thực ở một nơi (trong lớp mô hình) và các quy tắc này được thực thi ở mọi nơi trong ứng dụng.

Hãy xem cách bạn có thể tận dụng hỗ trợ xác thực này trong ứng dụng phim.

Thêm quy tắc xác thực vào mô hình phim

Bạn sẽ bắt đầu bằng cách thêm một số logic xác thực vào lớp Phim.

Mở tệp Movie.cs. Lưu ý rằng không gian tên System.ComponentModel.DataAnnotations không kết hợp với Microsoft.AspNet.Mvc. DataAnnotations cung cấp một tập hợp các thuộc tính xác thực được tích hợp sẵn mà bạn có thể áp dụng khai báo cho bất kỳ lớp hoặc tài sản nào. (Nó cũng chứa các thuộc tính định dạng như DataType giúp định dạng và không cung cấp bất kỳ xác thực nào.)

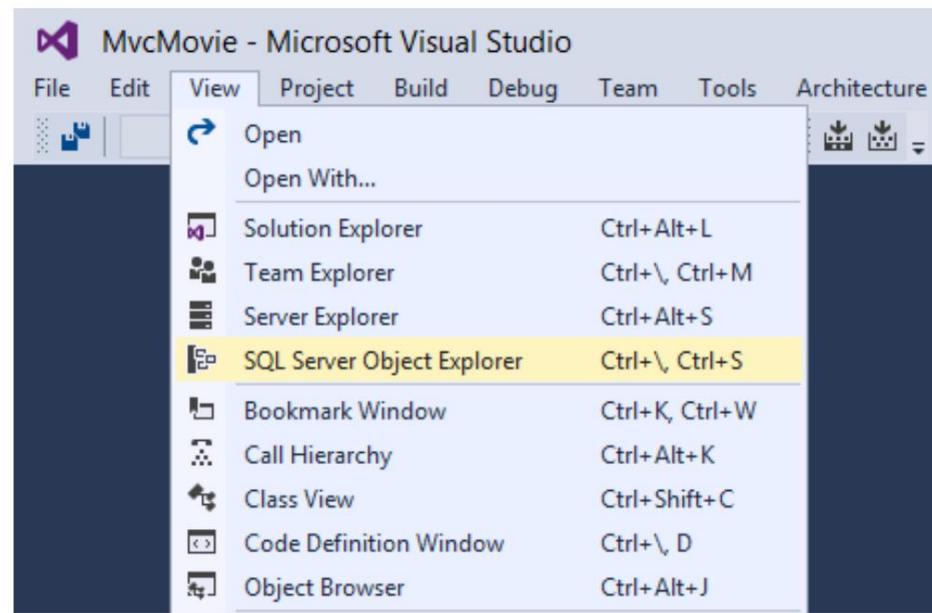
Bây giờ, hãy cập nhật lớp Phim để tận dụng lợi thế của Yêu cầu, Độ dài chuỗi tích hợp sẵn, Các thuộc tính Xác thực phạm vi và Biểu thức chính quy.

```

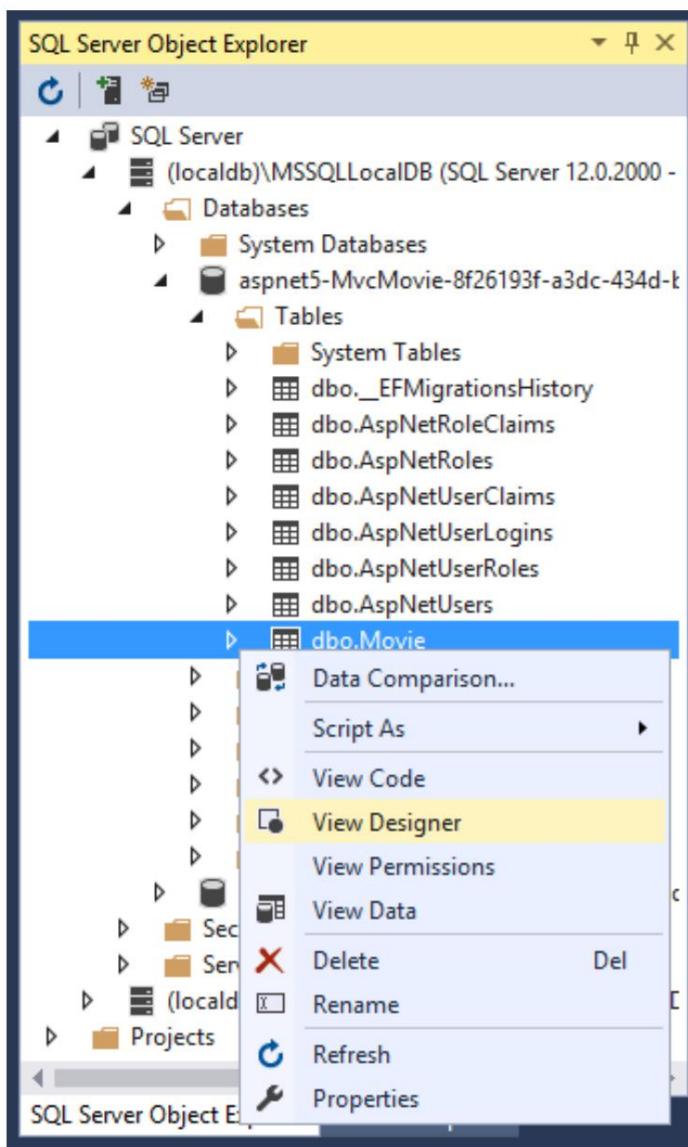
1 phim cấp công cộng
2 {
3     public int ID {get; bộ; }
4
5     [StringLength (60, MinimumLength = 3)]
6     chuỗi công khai Tiêu đề {get; bộ; }
7
8     [Hiển thị (Tên = "Ngày phát hành")]
9     [DataType (DataType.Date)]
10    public DateTime ReleaseDate {get; bộ; }
11
12    [Biểu thức chính quy (@ "^[AZ] + [a-zA-Z ' ' - '\ s] * $")]
13    [Yêu cầu]
14    [StringLength (30)]
15    chuỗi công khai Thể loại {get; bộ; }
16
17    [Phạm vi (1, 100)]
18    [DataType (DataType.Currency)]
19    giá thập phân công khai {get; bộ; }
20
21    [Biểu thức chính quy (@ "^[AZ] + [a-zA-Z ' ' - '\ s] * $")]
22    [StringLength (5)]
23    chuỗi công khai Đánh giá {get; bộ; }
24 }
```

Thuộc tính StringLength đặt độ dài tối đa của chuỗi và nó đặt giới hạn này trên cơ sở dữ liệu, do đó lược đồ cơ sở dữ liệu sẽ thay đổi.

- Từ menu View , mở SQL Server Object Explorer (SSOX).



- Nhấp chuột phải vào bảng Movie > View Designer



Hình ảnh sau đây cho thấy thiết kế bảng và T-SQL có thể tạo bảng.

dbo.Movie [Design]

Update | Script File: **dbo.Movie.sql**

	Name	Data Type	Allow Nulls
PK	ID	int	<input type="checkbox"/>
	Genre	nvarchar(MAX)	<input checked="" type="checkbox"/>
	Price	decimal(18,2)	<input type="checkbox"/>
	ReleaseDate	datetime2(7)	<input type="checkbox"/>
	Title	nvarchar(MAX)	<input checked="" type="checkbox"/>
	Rating	nvarchar(MAX)	<input checked="" type="checkbox"/>

Keys (1)
PK_Movie (Primary Key, Clustered)

Check Constraints (0)

Indexes (0)

Foreign Keys (0)

Triggers (0)

Design **T-SQL**

```

1  CREATE TABLE [dbo].[Movie] (
2      [ID]          INT            IDENTITY (1, 1) NOT NULL,
3      [Genre]        NVARCHAR (MAX) NULL,
4      [Price]        DECIMAL (18, 2) NOT NULL,
5      [ReleaseDate] DATETIME2 (7) NOT NULL,
6      [Title]        NVARCHAR (MAX) NULL,
7      [Rating]       NVARCHAR (MAX) NULL,
8      CONSTRAINT [PK_Movie] PRIMARY KEY CLUSTERED ([ID] ASC)
9  );
10
11

```

100 %

Connection Ready | (localdb)\MSSQLLocalDB | REDMOND\riande | aspnet5-MvcMovie-53e15...

Trong hình trên, bạn có thể thấy tất cả các trường chuỗi được đặt thành NVARCHAR (MAX).

Xây dựng dự án, mở cửa sổ lệnh và nhập các lệnh sau:

```
dnx ef migrations add DataAnnotations dnx ef
database update
```

Kiểm tra giản đồ Phim:

dbo.Movie [Design]

Update | Script File: **dbo.Movie.sql**

	Name	Data Type	Allow Nulls	Default
PK	ID	int	<input type="checkbox"/>	
	Genre	nvarchar(30)	<input type="checkbox"/>	
	Price	decimal(18,2)	<input type="checkbox"/>	
	ReleaseDate	datetime2(7)	<input type="checkbox"/>	
	Title	nvarchar(MAX)	<input checked="" type="checkbox"/>	
	Rating	nvarchar(MAX)	<input checked="" type="checkbox"/>	

Các trường chuỗi hiển thị giới hạn độ dài mới và Thể loại không còn giá trị nữa.

Các thuộc tính xác thực chỉ định hành vi mà bạn muốn thực thi trên các thuộc tính mô hình mà chúng được áp dụng. Các thuộc tính Required và MinimumLength chỉ ra rằng một thuộc tính phải có một giá trị; nhưng không có gì ngăn cản người dùng nhập khoảng trắng để đáp ứng xác nhận này. Thuộc tính RegularExpression được sử dụng để giới hạn những ký tự có thể được nhập vào. Trong đoạn mã trên, Thể loại và Xếp hạng chỉ được sử dụng các chữ cái (không được phép sử dụng khoảng trắng, số và ký tự đặc biệt). Thuộc tính Range ràng buộc giá trị trong một phạm vi được chỉ định. Thuộc tính StringLength cho phép bạn đặt độ dài tối đa của thuộc tính chuỗi và tùy chọn độ dài tối thiểu của nó.

Các kiểu giá trị (chẳng hạn như thập phân, int, float, DateTime) vốn là bắt buộc và không cần thuộc tính [Bắt buộc].

Code First đảm bảo rằng các quy tắc xác thực mà bạn chỉ định trên một lớp mô hình được thực thi trước khi ứng dụng lưu các thay đổi trong cơ sở dữ liệu. Ví dụ: đoạn mã dưới đây sẽ ném ra một ngoại lệ DbUpdateException khi phương thức SaveChanges được gọi, vì một số giá trị thuộc tính Movie bắt buộc bị thiếu.

```
Phim cap 3 = new Movie(); movie.Title =
    "Cuốn theo chiều gió"; _context.Movie.Add (phim);
_context.SaveChanges ();
// <= Sẽ ném ngoại lệ xác thực phía máy chủ
```

Đoạn mã trên ném ra ngoại lệ sau:

```
Thao tác cơ sở dữ liệu không thành công khi xử lý yêu cầu.
DbUpdateException: Đã xảy ra lỗi khi cập nhật các mục nhập.
Xem ngoại lệ bên trong để biết chi tiết.
SQLException: Không thể chèn giá trị NULL vào cột 'Thể loại', bảng 'aspnet5-MvcMovie-.dbo.Movie'; Scolumn không cho phép giá trị rỗng. INSERT không thành công.
```

Việc ASP.NET tự động thực thi các quy tắc xác thực giúp ứng dụng của bạn mạnh mẽ hơn. Nó cũng đảm bảo rằng bạn không thể quên xác nhận một cái gì đó và vô tình để dữ liệu xấu vào cơ sở dữ liệu.

Giao diện người dùng lỗi xác thực trong MVC

Chạy ứng dụng và điều hướng đến bộ điều khiển Phim.

Nhấn vào liên kết Tạo mới để thêm phim mới. Điền vào biểu mẫu với một số giá trị không hợp lệ. Ngay sau khi xác thực phía máy khách jQuery phát hiện ra lỗi, nó sẽ hiển thị thông báo lỗi.

The screenshot shows a browser window displaying an ASP.NET MVC 6 application titled "Mvc Movie". The page is titled "Create" and has a sub-section "Movie". It contains five input fields: "Genre", "Price", "Release Date", "Title", and "Rating". Each field has a validation error message displayed below it.

Genre
The Genre field is required.

Price
The field Price must be a number.

Release Date
Please enter a valid date.

Title
The field Title must be a string with a minimum length of 3 and a maximum length of 60.

Rating
The field Rating must match the regular expression "^[A-Z]+[a-zA-Z'-'\s]*\$".

[Create](#)

[Back to List](#)

Lưu ý: Bạn có thể không nhập được dấu thập phân hoặc dấu phẩy vào trường Giá. Để hỗ trợ xác thực jQuery đối với các ngôn ngữ không phải tiếng Anh sử dụng dấu phẩy (",") cho dấu thập phân và các định dạng ngày không phải tiếng Anh Mỹ, bạn phải thực hiện các bước để toàn cầu hóa ứng dụng của mình. Xem [Tài nguyên bổ sung](#) để biết thêm thông tin. Hiện tại, chỉ cần nhập các số nguyên như 10.

Lưu ý cách biểu mẫu đã tự động hiển thị thông báo lỗi xác thực thích hợp trong mỗi trường chứa giá trị không hợp lệ. Các lỗi được thực thi cả phía máy khách (sử dụng JavaScript và jQuery) và phía máy chủ (trong trường hợp người dùng đã tắt JavaScript).

Một lợi ích đáng kể là bạn không cần phải thay đổi một dòng mã nào trong lớp MoviesController hoặc trong chế độ xem Create.cshtml để kích hoạt giao diện người dùng xác thực này. Bộ điều khiển và các khung nhìn bạn đã tạo trước đó trong hướng dẫn này tự động chọn các quy tắc xác thực mà bạn đã chỉ định bằng cách sử dụng các thuộc tính xác thực trên các thuộc tính của lớp Mô hình phim. Kiểm tra xác thực bằng phương pháp hành động Chính sửa và xác thực tương tự cũng được áp dụng.

Dữ liệu biểu mẫu không được gửi đến máy chủ cho đến khi không có lỗi xác thực phía máy khách. Bạn có thể xác minh điều này bằng cách đặt một điểm ngắt trong phương thức HTTP Post, bằng cách sử dụng công cụ Fiddler hoặc các công cụ dành cho nhà phát triển F12.

Cách xác thực xảy ra trong Chế độ xem Tạo và Phương pháp Hành động Tạo

Bạn có thể tự hỏi làm thế nào mà giao diện người dùng xác thực được tạo ra mà không có bất kỳ cập nhật nào đối với mã trong bộ điều khiển hoặc chế độ xem. Danh sách tiếp theo hiển thị hai phương pháp Tạo.

```
// NHÂN: Phim / Tạo
public IActionResult Create () {

    return View ();
}

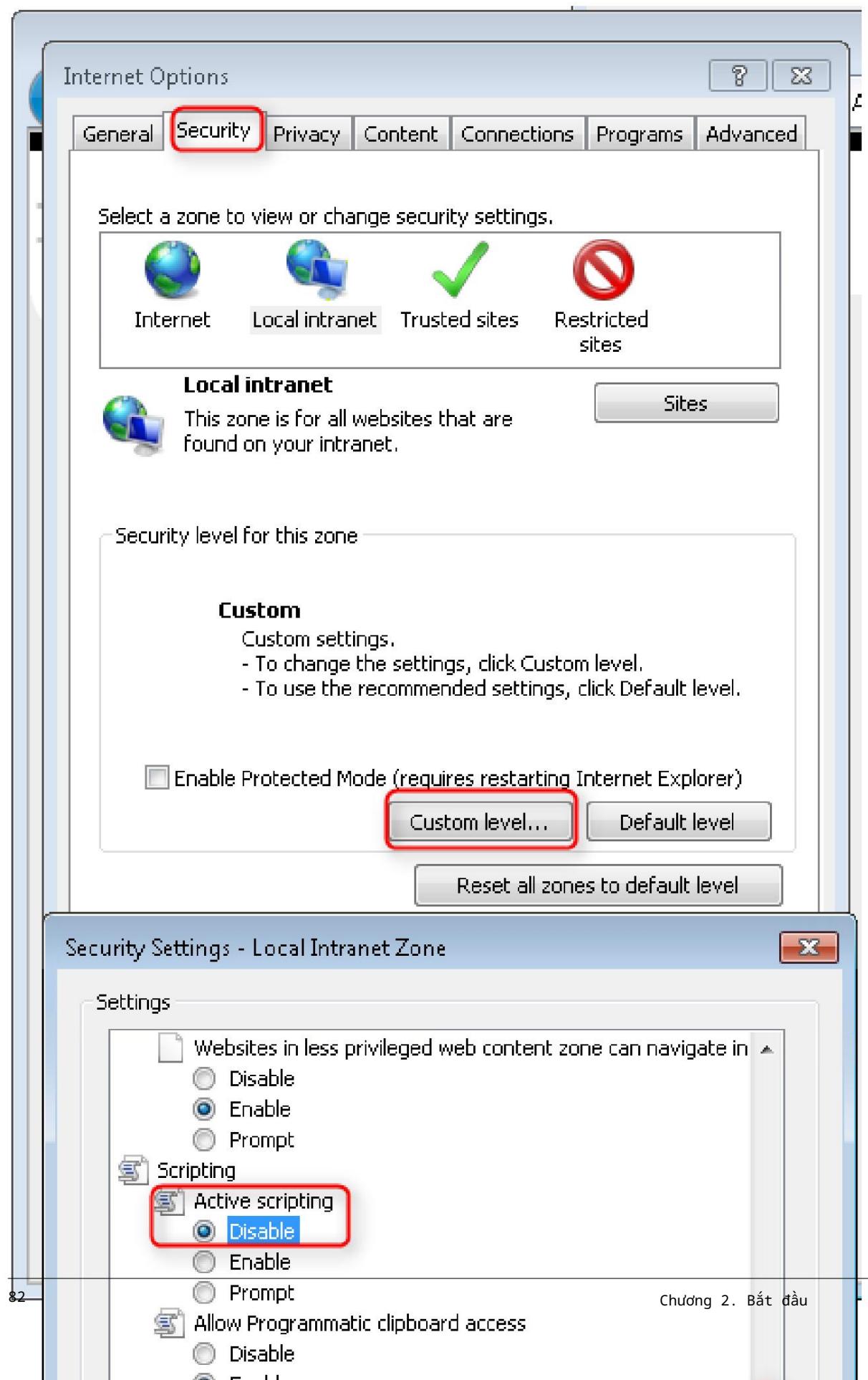
// ĐĂNG: Phim / Tạo
[HttpPost]
[ValidateAntiForgeryToken] public
IActionResult Create ([Bind ("ID, Title, ReleaseDate, Genre, Price, Rating")] Movie movie) {

    if (ModelState.IsValid)
    {
        _context.Movie.Add (phim);
        _context.SaveChanges (); return
        RedirectToAction ("Chỉ mục");

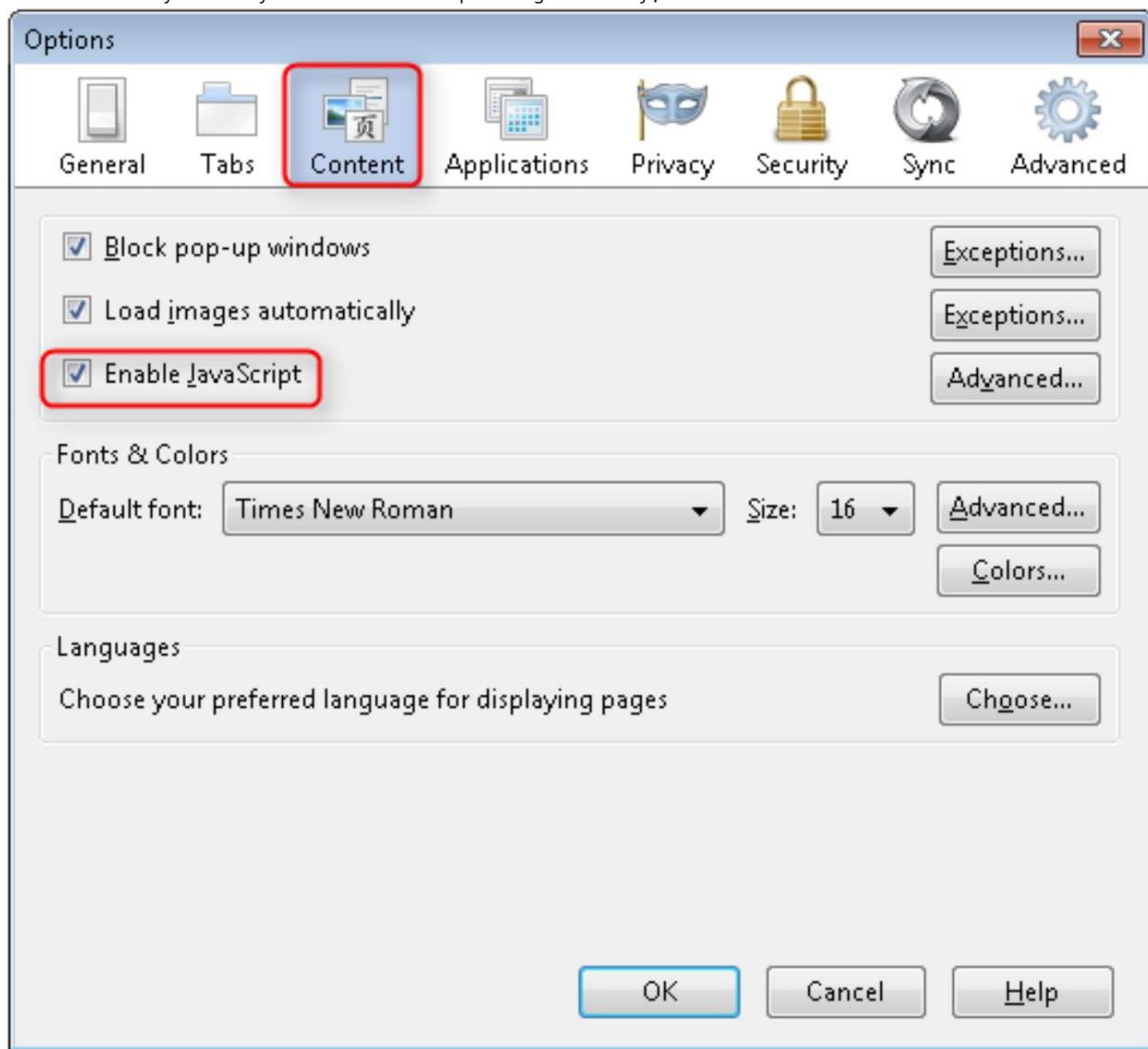
    } return View (phim);
}
```

Phương thức hành động Tạo (HTTP GET) đầu tiên hiển thị biểu mẫu Tạo ban đầu. Phiên bản thứ hai ([HttpPost]) xử lý bài đăng trên biểu mẫu. Phương thức Tạo thứ hai (Phiên bản HttpNotFound) gọi ModelState.IsValid để kiểm tra xem phim có bất kỳ lỗi xác thực nào không. Việc gọi phương thức này sẽ đánh giá bất kỳ thuộc tính xác thực nào đã được áp dụng cho đối tượng. Nếu đối tượng có lỗi xác thực, phương pháp Tạo sẽ hiển thị lại biểu mẫu. Nếu không có lỗi, phương pháp sẽ lưu phim mới trong cơ sở dữ liệu. Trong ví dụ phim của chúng tôi, biểu mẫu không được đăng lên máy chủ khi phát hiện có lỗi xác thực ở phía máy khách; phương thức Tạo thứ hai không bao giờ được gọi khi có lỗi xác thực phía máy khách. Nếu bạn tắt JavaScript trong trình duyệt của mình, xác thực ứng dụng khách sẽ bị vô hiệu hóa và bạn có thể kiểm tra phương thức HTTP POST Create gọi ModelState.IsValid để kiểm tra xem phim có bất kỳ lỗi xác thực nào không

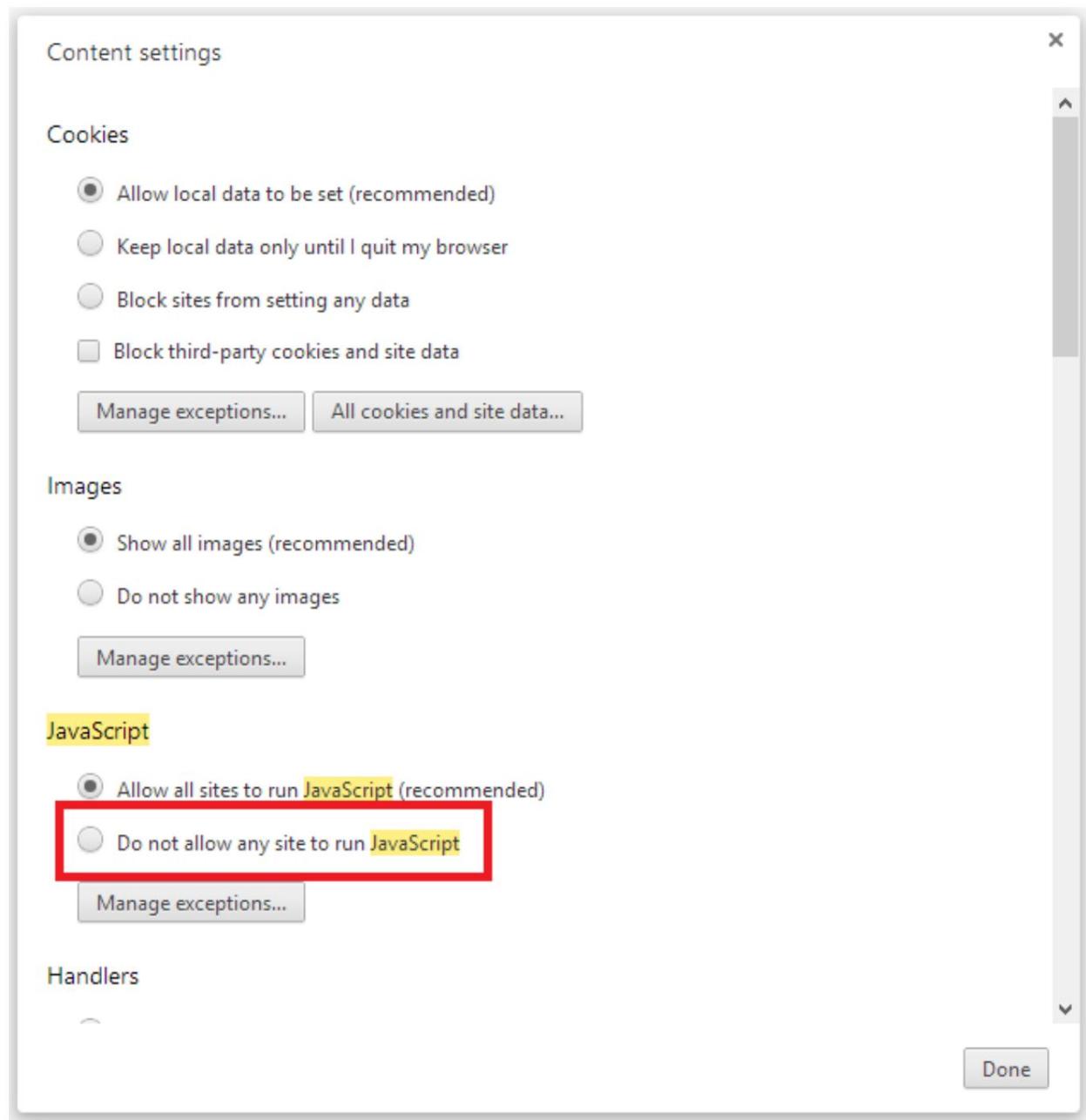
Bạn có thể đặt điểm ngắt trong phương thức Tạo [HttpPost] và xác minh phương thức không bao giờ được gọi, xác thực phía máy khách sẽ không gửi dữ liệu biểu mẫu khi phát hiện lỗi xác thực. Nếu bạn tắt JavaScript trong trình duyệt của mình, sau đó gửi biểu mẫu có lỗi, điểm ngắt sẽ bị ảnh hưởng. Bạn vẫn nhận được xác thực đầy đủ mà không cần JavaScript. Hình ảnh sau đây cho thấy cách tắt JavaScript trong Internet Explorer.



Hình ảnh sau đây cho thấy cách tắt JavaScript trong trình duyệt FireFox.



Hình ảnh sau đây cho thấy cách tắt JavaScript trong trình duyệt Chrome.



Sau khi bạn tắt JavaScript, hãy đăng dữ liệu không hợp lệ và bước qua trình gõ lỗi.

```
// POST: Movies/Create
[HttpPost]
[ValidateAntiForgeryToken]
0 references
public IActionResult Create([Bind("ID,Title,ReleaseDate,Genre")]
{
    if (ModelState.IsValid)
    {
        _context.Movie.Add(movie);
        _context.SaveChanges();
        return RedirectToAction("Index");
    }
    return View(movie);
}
```

Dưới đây là một phần của mẫu ché độ xem Create.cshtml mà bạn đã giới thiệu trước đó trong hướng dẫn. Nó được sử dụng bởi hành động các phương thức hiển thị ở trên vừa để hiển thị biểu mẫu ban đầu vừa để hiển thị lại trong trường hợp có lỗi.

```
1 <form asp-action = "Tạo">
2     <div class = "form-ngang">
3         <h4> Phim </h4>
4         <giờ />
5         <div asp-validation-summary = "ValidationSummary.ModelOnly" class = "text-risk"> </div>
6         <div class = "form-group">
7             <label asp-for = "Genre" class = "col-md-2 control-label"> </label>
8             <div class = "col-md-10">
9                 <input asp-for = "Thể loại" class = "form-control" />
10                <span asp-validation-for = "Thể loại" class = "text-risk" />
11            </div>
12        </div>
13        @ * Đã xóa đánh dấu cho ngắn gọn. * @
14        <div class = "form-group">
15            <label asp-for = "Rating" class = "col-md-2 control-label"> </label>
16            <div class = "col-md-10">
17                <input asp-for = "Xếp hạng" class = "form-control" />
18                <span asp-validation-for = "Xếp hạng" class = "text-risk" />
19            </div>
20        </div>
21        <div class = "form-group">
22            <div class = "col-md-offset-2 col-md-10">
23                <input type = "submit" value = "Tạo" class = "btn btn-default" />
24            </div>
25        </div>
26    </div>
27 </form>
```

Trình trợ giúp thẻ đầu vào sử dụng DataAnnotations các thuộc tính và tạo ra các thuộc tính HTML cần thiết cho jQuery Xác nhận ở phía khách hàng. Trình trợ giúp thẻ xác thực hiển thị một thông báo xác thực.

Điều thực sự hay về cách tiếp cận này là cả bộ điều khiển và mẫu tạo dạng xem đều không biết gì cả về các quy tắc xác thực thực tế đang được thực thi hoặc về các thông báo lỗi cụ thể được hiển thị. Các quy tắc xác thực và các chuỗi lỗi chỉ được chỉ định trong lớp Phim. Các quy tắc xác thực tương tự này được tự động áp dụng cho ché độ xem Chính sửa và bất kỳ mẫu ché độ xem nào khác mà bạn có thể tạo để chỉnh sửa mô hình của mình.

Nếu bạn muốn thay đổi logic xác thực sau này, bạn có thể làm như vậy ở chính xác một nơi bằng cách thêm các thuộc tính xác thực vào mô hình (trong ví dụ này là lớp Phim). Bạn sẽ không phải lo lắng về các phần khác nhau của ứng dụng không phù hợp với cách các quy tắc được thực thi - tất cả logic xác thực sẽ được xác định ở một nơi và được sử dụng ở mọi nơi. Điều này giữ cho mã rất rõ ràng, đồng thời giúp dễ dàng bảo trì và phát triển. Và nó có nghĩa là bạn sẽ hoàn toàn

tôn trọng nguyên tắc KHÔ.

Sử dụng thuộc tính DataType

Mở tệp Movie.cs và kiểm tra lớp Phim. Không gian tên System.ComponentModel.DataAnnotations cung cấp các thuộc tính định dạng ngoài tập hợp các thuộc tính xác thực được tích hợp sẵn. Chúng tôi đã sắp xếp một giá trị liệt kê DataType vào ngày phát hành và các trường giá. Đoạn mã sau hiển thị thuộc tính Ngày phát hành và Giá với thuộc tính Loại dữ liệu thích hợp.

```
1 [DataType(DataType.Date)] 2 public  
DateTime ReleaseDate { get; bộ; }  
3  
4 [DataType(DataType.Currency)] 5 số thập phân  
công khai Giá { get; bộ; }
```

Thuộc tính DataType chỉ cung cấp gợi ý cho công cụ xem để định dạng dữ liệu (và cung cấp các thuộc tính như <a> cho URL và cho email. Bạn có thể sử dụng thuộc tính RegularExpression để xác thực định dạng của dữ liệu. Thuộc tính DataType được sử dụng để chỉ định kiểu dữ liệu cụ thể hơn kiểu nội tại của cơ sở dữ liệu, chúng không phải là thuộc tính xác thực. Trong trường hợp này, chúng tôi chỉ muốn theo dõi ngày chứ không phải thời gian. DataType Chế độ liệt kê cung cấp nhiều kiểu dữ liệu, chẳng hạn như Ngày, Giờ, Số điện thoại, Đơn vị tiền tệ, Địa chỉ email, v.v. Thuộc tính DataType cũng có thể cho phép ứng dụng tự động cung cấp các tính năng dành riêng cho từng loại. Ví dụ: có thể tạo liên kết mailto: cho DataType. EmailAddress và một bộ chọn ngày có thể được cung cấp cho DataType.Date trong các trình duyệt hỗ trợ HTML5. Thuộc tính DataType phát ra thuộc tính dữ liệu HTML 5 (dấu gạch ngang dữ liệu rõ ràng) mà trình duyệt HTML 5 có thể hiểu được. Thuộc tính DataType không cung cấp IDE bất kỳ xác nhận nào.

DataType.Date không chỉ định định dạng của ngày được hiển thị. Theo mặc định, trường dữ liệu được hiển thị theo các định dạng mặc định dựa trên CultureInfo của máy chủ.

Thuộc tính DisplayFormat được sử dụng để chỉ định rõ ràng định dạng ngày:

```
[DisplayFormat(DataFormatString = "{0: yyyy-MM-dd}", ApplyFormatInEditMode = true)] public DateTime EnrollmentDate { get;  
bộ; }
```

Cài đặt ApplyFormatInEditMode chỉ định rằng định dạng cũng nên được áp dụng khi giá trị được hiển thị trong hộp văn bản để chỉnh sửa. (Bạn có thể không muốn điều đó đối với một số trường - ví dụ: đối với giá trị tiền tệ, bạn có thể không muốn ký hiệu tiền tệ trong hộp văn bản để chỉnh sửa.)

Bạn có thể tự mình sử dụng thuộc tính DisplayFormat, nhưng thông thường, bạn nên sử dụng riêng thuộc tính DataType. Thuộc tính DataType truyền tải ngữ nghĩa của dữ liệu thay vì cách hiển thị dữ liệu trên màn hình và cung cấp các lợi ích sau mà bạn không nhận được với DisplayFormat:

- Trình duyệt có thể bật các tính năng HTML5 (ví dụ: để hiển thị điều khiển lịch, trình duyệt phù hợp với ngôn ngữ biểu tượng lần truy cập gần đây, liên kết email, v.v.)
- Theo mặc định, trình duyệt sẽ hiển thị dữ liệu bằng định dạng chính xác dựa trên ngôn ngữ của bạn
- Thuộc tính DataType có thể cho phép MVC chọn mẫu trường phù hợp để kết xuất dữ liệu (DisplayFormat nếu được sử dụng bởi chính nó sẽ sử dụng mẫu chuỗi). Để biết thêm thông tin, hãy xem [Mẫu ASP.NET MVC 2](#) của Brad Wilson . (Mặc dù được viết cho MVC 2, bài viết này vẫn áp dụng cho phiên bản hiện tại của ASP.NET MVC.)

Lưu ý: xác thực jQuery không hoạt động với thuộc tính Range và DateTime. Ví dụ: mã sau sẽ luôn hiển thị lỗi xác thực phía máy khách, ngay cả khi ngày nằm trong phạm vi được chỉ định:

```
[Phạm vi (typeof(DateTime), "1/1/1966", "1/1/2020")]
```

Bạn sẽ cần phải tắt xác thực ngày jQuery để sử dụng thuộc tính Phạm vi với DateTime. Nó thường không tốt thực hành để biên dịch ngày tháng cố định trong các mô hình của bạn, vì vậy không khuyến khích sử dụng thuộc tính Range và DateTime.

Đoạn mã sau cho thấy việc kết hợp các thuộc tính trên một dòng:

```

1 phím cấp công cộng
2 {
3     public int ID {get; bộ; }
4
5     [StringLength (60, MinimumLength = 3)]
6     chuỗi công khai Tiêu đề {get; bộ; }
7
8     [Hiển thị (Tên = "Ngày phát hành"), Kiểu dữ liệu (DataType.Date)]
9     public DateTime ReleaseDate {get; bộ; }
10
11    [Biểu thức chính quy (@ "^[AZ] + [a-zA-Z ' - '\ s] * $"), Bắt buộc, Độ dài chuỗi (30)]
12    chuỗi công khai Thể loại {get; bộ; }
13
14    [Phạm vi (1, 100), Kiểu dữ liệu (DataType.Currency)]
15    giá thập phân công khai {get; bộ; }
16
17    [Biểu thức chính quy (@ "^[AZ] + [a-zA-Z ' - '\ s] * $"), StringLength (5)]
18    chuỗi công khai Đánh giá {get; bộ; }
19 }
```

Trong phần tiếp theo của loạt bài này, chúng tôi sẽ xem xét ứng dụng và thực hiện một số cải tiến đối với Chi tiết và phương pháp Xóa.

Các nguồn bổ sung

- [Toàn cầu hóa và bản địa hóa](#)
- [Giới thiệu về Trình trợ giúp thẻ](#)
- [Trình trợ giúp thẻ tác giả](#)

2.1.10 Kiểm tra chi tiết và phương pháp xóa

Lưu ý: Chúng tôi hiện đang làm việc về chủ đề này.

Chúng tôi hoan nghênh ý kiến đóng góp của bạn để giúp định hình phạm vi và cách tiếp cận. Bạn có thể theo dõi trạng thái và cung cấp thông tin đầu vào về vấn đề này tại [GitHub](#).

Nếu bạn muốn xem lại các bản nháp và phác thảo ban đầu của chủ đề này, vui lòng để lại ghi chú với thông tin liên hệ của bạn trong vấn đề.

[Tìm hiểu thêm về cách bạn có thể đóng góp trên GitHub](#).

2.2 Xây dựng API web đầu tiên của bạn với MVC 6

Bởi [Mike Wasson](#) và [Rick Anderson](#)

HTTP không chỉ để phục vụ các trang web. Nó cũng là một nền tảng mạnh mẽ để xây dựng các API hiển thị các dịch vụ và dữ liệu. HTTP đơn giản, linh hoạt và phổ biến. Hầu hết mọi nền tảng mà bạn có thể nghĩ đến đều có thư viện HTTP, vì vậy Các dịch vụ HTTP có thể tiếp cận nhiều loại khách hàng, bao gồm trình duyệt, thiết bị di động và ứng dụng máy tính để bàn truyền thống.

Trong hướng dẫn này, bạn sẽ xây dựng một API web đơn giản để quản lý danh sách các mục “việc cần làm”. Bạn sẽ không tạo bất kỳ giao diện người dùng nào trong này hướng dẫn.

Các phiên bản trước của ASP.NET bao gồm khung API Web để tạo các API web. Trong ASP.NET 5, tính năng đồng nhất của chức năng này đã được hợp nhất vào khung MVC 6. Việc hợp nhất hai khuôn khổ giúp việc xây dựng các ứng dụng trở nên đơn giản hơn bao gồm cả giao diện người dùng (HTML) và API, vì bây giờ chúng chia sẻ cùng một cơ sở mã và đường dẫn.

Lưu ý: Nếu bạn đang chuyển một ứng dụng Web API hiện có sang MVC 6, hãy xem [Di chuyển từ ASP.NET Web API 2 sang MVC 6](#)

Trong bài viết này:

- [Tổng quan](#)
- [Cài đặt Fiddler](#)
- [Tạo dự án](#)
- [Thêm một lớp mô hình](#)
- [Thêm một lớp kho lưu trữ](#)
- [Đăng ký kho lưu trữ](#)
- [Thêm bộ điều khiển](#)
- [Bắt các hạng mục công việc](#)
- [Sử dụng Fiddler để gọi API](#)
- [Thực hiện các hoạt động CRUD khác](#)
- [Các bước tiếp theo](#)

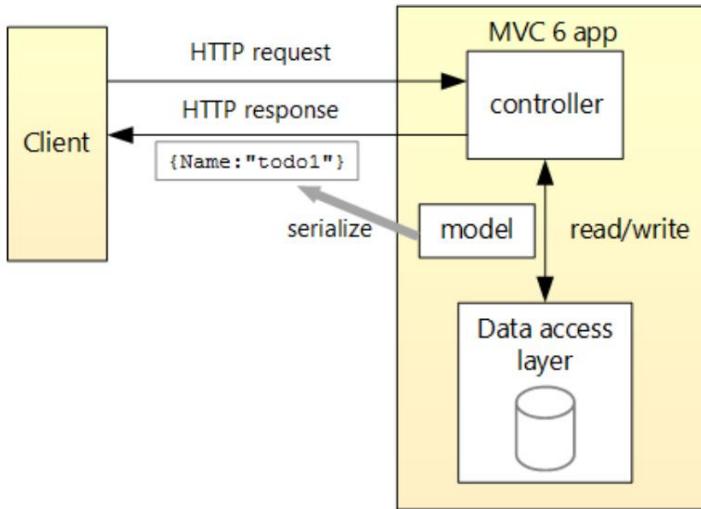
Bạn có thể duyệt mã nguồn cho ứng dụng mẫu trên [GitHub](#).

2.2.1 Tổng quan

Đây là API mà bạn sẽ tạo:

API	Sự mô tả	Yêu cầu cơ bản	Phản hồi cơ bản
GET / api / todo	Nhận tất cả các việc cần làm	Không có	Mảng các việc cần làm
GET / api / todo / {id}	Nhận một mặt hàng theo ID	Không có	Mục việc cần làm
POST / api / todo	Thêm một mặt hàng mới	Mục việc cần làm	Mục việc cần làm
PUT / api / todo / {id}	Cập nhật một mục hiện có	Mục việc cần làm	Không có
DELETE / api / todo / {id}	Xóa một mục.	Không có	Không có

Sơ đồ sau đây cho thấy thiết kế cơ bản của ứng dụng.



- Máy khách là bất cứ thứ gì sử dụng API web (trình duyệt, ứng dụng dành cho thiết bị di động, v.v.). Chúng tôi không viết một khách hàng trong hướng dẫn này.
- Mô hình là một đối tượng đại diện cho dữ liệu trong ứng dụng của bạn. Trong trường hợp này, mô hình duy nhất là một mục việc cần làm. Các mô hình được biểu diễn dưới dạng các lớp C# đơn giản (POCO).
- Bộ điều khiển là một đối tượng xử lý các yêu cầu HTTP và tạo phản hồi HTTP. Ứng dụng này sẽ có một bộ điều khiển.
- Để giữ cho hướng dẫn đơn giản và tập trung vào MVC 6, ứng dụng không sử dụng cơ sở dữ liệu. Thay vào đó, nó chỉ lưu các mục việc cần làm trong bộ nhớ. Nhưng chúng tôi sẽ vẫn bao gồm một lớp truy cập dữ liệu (tầm thường), để minh họa sự tách biệt giữa API web và lớp dữ liệu. Để biết hướng dẫn sử dụng cơ sở dữ liệu, hãy xem [Xây dựng ứng dụng MVC 6 đầu tiên](#) của bạn.

2.2.2 Cài đặt Fiddler

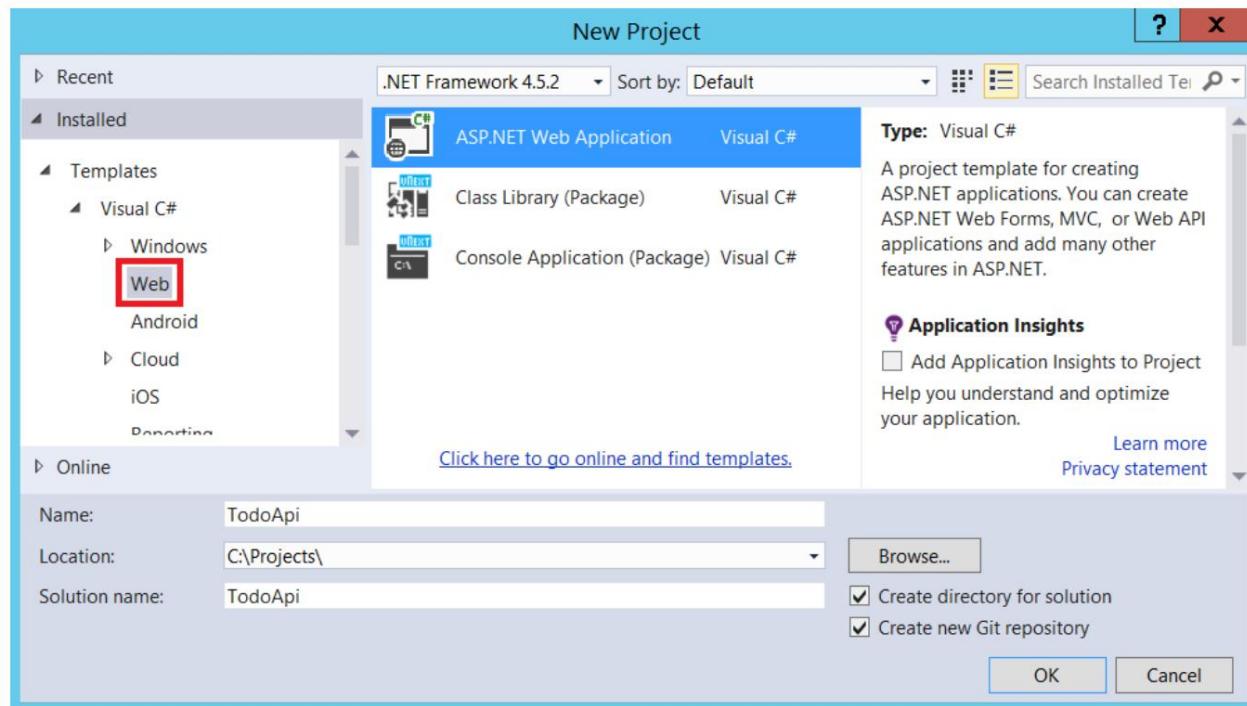
Bước này là tùy chọn nhưng được khuyến nghị.

Bởi vì chúng tôi không xây dựng một ứng dụng khách, chúng tôi cần một cách để gọi API. Trong hướng dẫn này, tôi sẽ chỉ ra điều đó bằng cách sử dụng [Fiddler](#). Fiddler là một công cụ gỡ lỗi web cho phép bạn soạn các yêu cầu HTTP và xem các phản hồi HTTP thô. Fiddler cho phép bạn thực hiện các yêu cầu HTTP trực tiếp tới API khi chúng tôi phát triển ứng dụng.

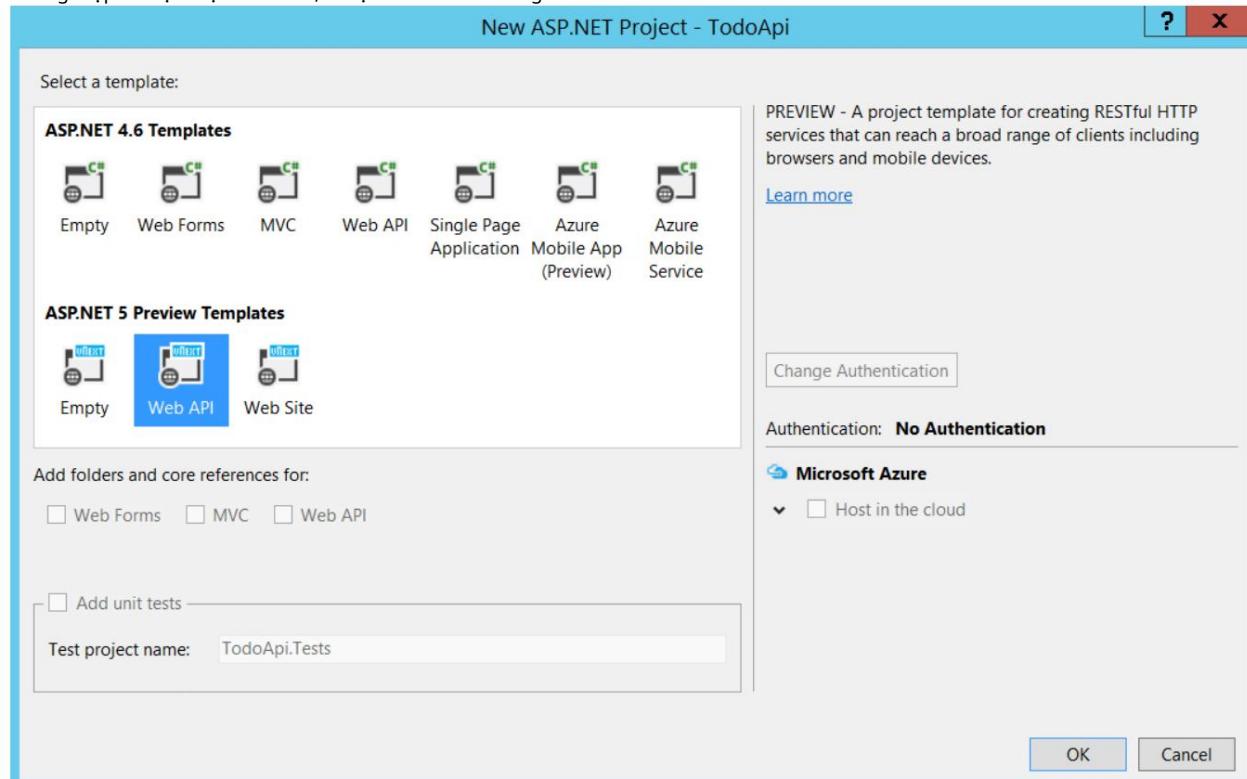
2.2.3 Tạo dự án

Khởi động Visual Studio 2015. Từ menu Tệp, chọn Mới > Dự án.

Chọn mẫu dự án [Ứng dụng Web ASP.NET](#). Đặt tên cho dự án là TodoApi và nhập vào OK.



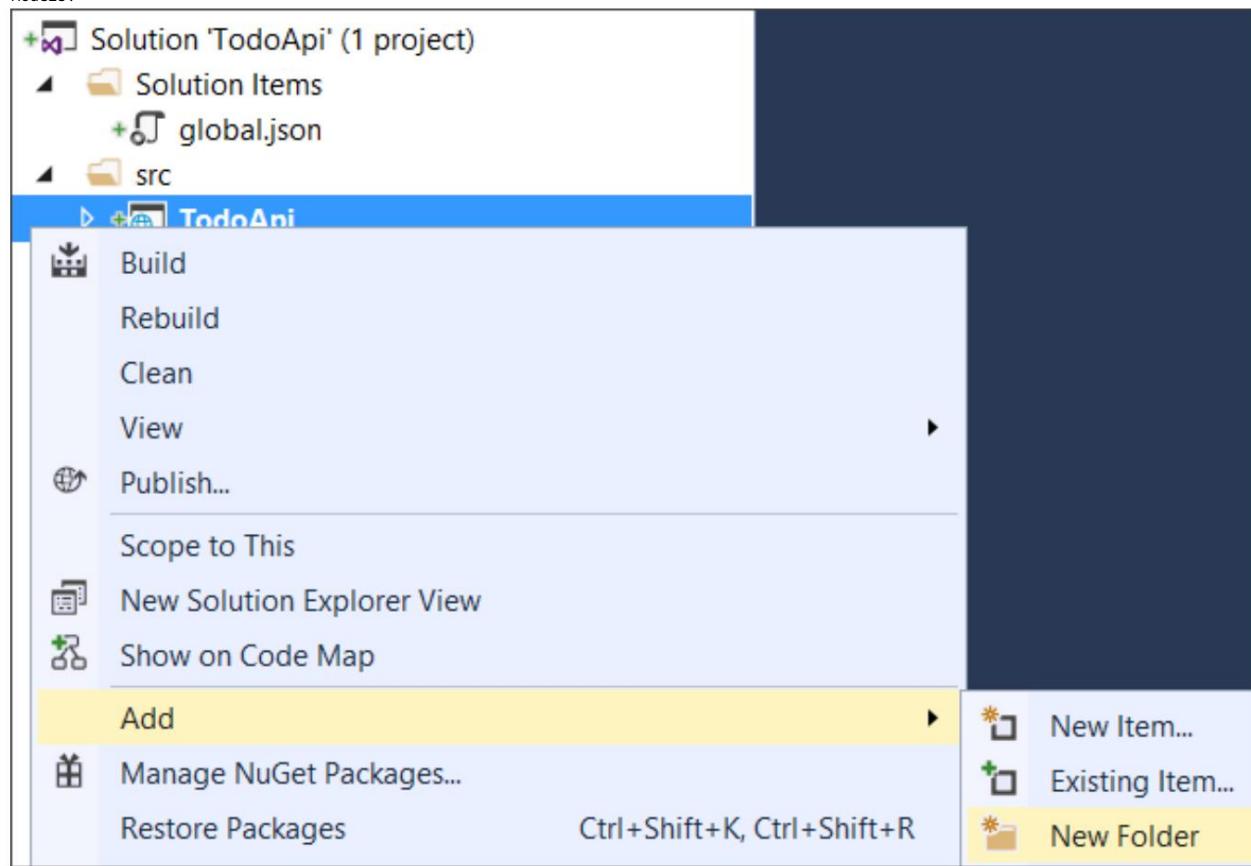
Trong hộp thoại Dự án mới , chọn API Web trong Mẫu xem trước ASP.NET 5. Bấm OK.



2.2.4 Thêm một lớp mô hình

Mô hình là một đối tượng đại diện cho dữ liệu trong ứng dụng của bạn. Trong trường hợp này, mô hình duy nhất là một mục việc cần làm.

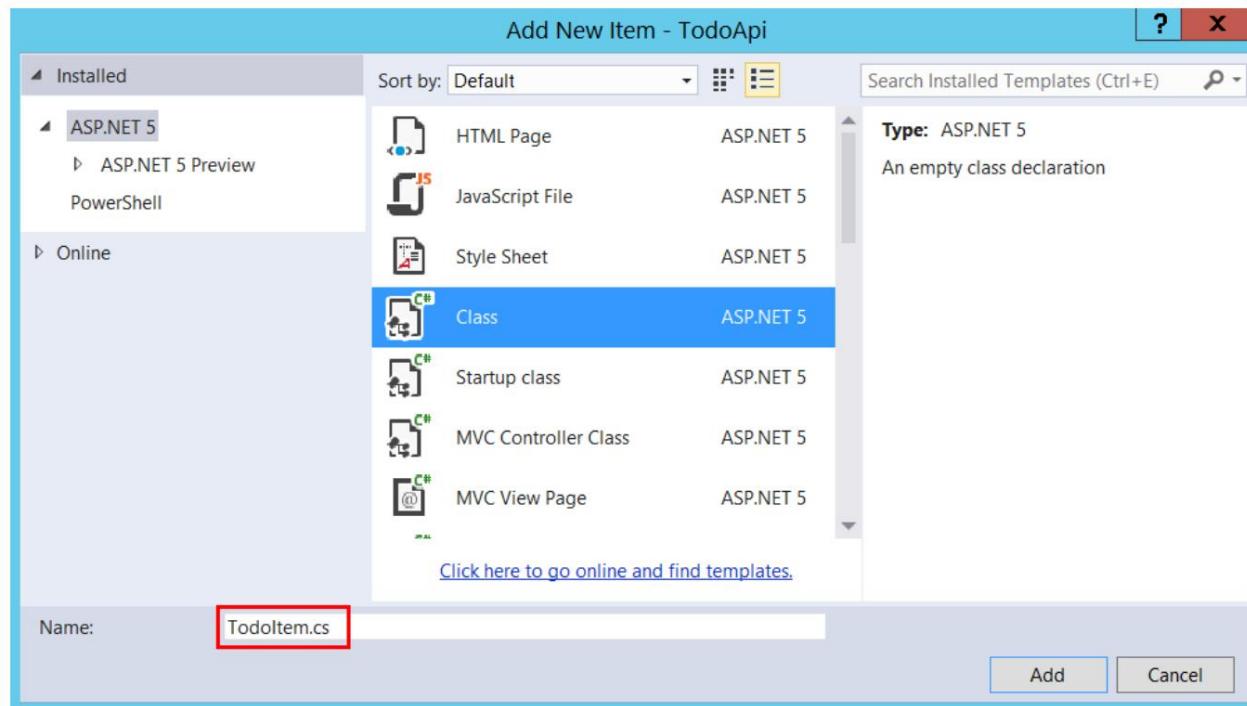
Thêm một thư mục có tên "Mô hình". Trong Giải pháp Explorer, bấm chuột phải vào dự án. Chọn Thêm > Thư mục Mới. Đặt tên cho thư mục là Models.



Lưu ý: Bạn có thể đặt các lớp mô hình ở bất kỳ đâu trong dự án của mình, nhưng thư mục Mô hình được sử dụng theo quy ước.

Tiếp theo, thêm một lớp TodoItem. Nhấp chuột phải vào thư mục Mô hình và chọn Thêm > Mục mới.

Trong hộp thoại Thêm mục mới, hãy chọn mẫu Lớp. Đặt tên cho lớp là TodoItem và nhấp vào OK.



Thay thế mã đã tạo bằng:

```
không gian tên TodoApi.Models {

    lớp công khai TodoItem {

        chuỗi công khai Key { get; bộ; } public
        string Tên { get; bộ; } public bool IsComplete
        { get; bộ; }
    }
}
```

2.2.5 Thêm một lớp kho lưu trữ

Kho lưu trữ là một đối tượng đóng gói lớp dữ liệu và chứa logic để truy xuất dữ liệu và ánh xạ nó tới một mô hình thực thể. Mặc dù ứng dụng mẫu không sử dụng cơ sở dữ liệu, nhưng sẽ rất hữu ích khi xem cách bạn có thể đưa kho lưu trữ vào bộ điều khiển của mình. Tạo mã kho lưu trữ trong thư mục Mô hình.

Bắt đầu bằng cách xác định giao diện kho lưu trữ có tên là `ITodoRepository`. Sử dụng mẫu lớp (Thêm mục mới > lớp).

```
sử dụng System.Collections.Generic;

không gian tên TodoApi.Models {

    giao diện công cộng ITodoRepository {

        void Thêm (mục TodoItem);
        IEnumerable <TodoItem> GetAll ();
        TodoItem Tìm ( khóa chuỗi );
        TodoItem Remove (khóa chuỗi ); void Cập
        nhật (mục TodoItem);
    }
}
```

Giao diện này xác định các hoạt động CRUD cơ bản. Trong thực tế, bạn có thể có các phương pháp dành riêng cho miền.

Tiếp theo, thêm một lớp TodoRepository triển khai ITodoRepository:

```
sử dụng Hệ thống;
sử dụng System.Collections.Generic; sử dụng
System.Collections.Concurrent;

không gian tên TodoApi.Models {

    lớp công khai TodoRepository : ITodoRepository {

        static ConcurrentDictionary <string, TodoItem> _todos = new ConcurrentDictionary <string, TodoItem>();

        public TodoRepository () {

            Thêm ( TodoItem mới {Tên = "Item1" });
        }

        public IEnumerable <TodoItem> GetAll () {

            return _todos.Values;
        }

        public void Add (TodoItem item) {

            item.Key = Guid.NewGuid ().ToString (); _todos
            [item.Key] = item;
        }

        public TodoItem Find (string key) {

            Mục TodoItem;
            _todos.TryGetValue (key, out item); trả lại hàng;

        }

        public TodoItem Remove (string key) {

            Mục TodoItem;
            _todos.TryGetValue (key, out item); _todos.TryRemove
            (key, out item); trả lại hàng;

        }

        public void Update (TodoItem item) {

            _todos [item.Key] = item;
        }
    }
}
```

Tạo ứng dụng để xác minh rằng bạn không có bất kỳ lỗi nào.

2.2.6 Đăng ký kho lưu trữ

Bằng cách xác định giao diện kho lưu trữ, chúng ta có thể tách lớp kho lưu trữ khỏi bộ điều khiển MVC sử dụng nó. Thay vì tạo mới TodoRepository bên trong bộ điều khiển, chúng tôi sẽ chèn một ITodoRepository, sử dụng ASP.NET

Vùng chứa 5 phụ thuộc tiêm (DI).

Cách tiếp cận này giúp đơn vị kiểm tra bộ điều khiển của bạn dễ dàng hơn. Các bài kiểm tra đơn vị nên đưa vào một phiên bản giả hoặc sơ khai của ITodoRepository. Bằng cách đó, thử nghiệm nhầm mục tiêu hép đến logic bộ điều khiển chứ không phải lớp truy cập dữ liệu.

Để đưa kho lưu trữ vào bộ điều khiển, chúng ta cần đăng ký nó với vùng chứa DI. Mở tệp Startup.cs. Thêm phần sau bằng chỉ thị:

```
sử dụng TodoApi.Models;
```

Trong phương pháp ConfigureServices, hãy thêm mã được đánh dấu:

```
public void ConfigureServices (dịch vụ IServiceCollection) {
    services.AddMvc (); // Thêm
    loại kho lưu trữ của chúng tôi
    services.AddSingleton <ITodoRepository, TodoRepository> ();
}
```

2.2.7 Thêm bộ điều khiển

Trong Giải pháp Explorer, bấm chuột phải vào thư mục Bộ điều khiển. Chọn Thêm > Mục mới. Trong hộp thoại Thêm mục mới, hãy chọn mẫu Lớp bộ điều khiển API Web. Đặt tên lớp là TodoController.

Thay thế mã đã tạo bằng mã sau:

```
sử dụng System.Collections.Generic; sử dụng
Microsoft.AspNet.Mvc; sử dụng TodoApi.Models;

không gian tên SimpleApi.Controllers {

    [Route ("api / [controller]")] public
    class TodoController : Controller {

        [FromServices] công
        khai ITodoRepository TodoItems { get; bộ; }
    }
}
```

Điều này xác định một lớp bộ điều khiển trống. Trong các phần tiếp theo, chúng tôi sẽ thêm các phương thức để triển khai API. [Tệp nạn FromSer] thuộc tính yêu cầu MVC đưa ITodoRepository mà chúng tôi đã đăng ký trong lớp Khởi động.

Xóa tệp ValuesController.cs khỏi thư mục Bộ điều khiển. Mẫu dự án thêm nó làm bộ điều khiển ví dụ, nhưng chúng tôi không cần nó.

2.2.8 Các mục việc cần làm

Để nhận các mục công việc, hãy thêm các phương thức sau vào lớp TodoController.

```
[HttpGet]
public IEnumerable <TodoItem> GetAll () {

    trả về TodoItems.GetAll ();
}

[HttpGet ("{id}", Name = "GetTodo")] public
IActionResult GetById (string id)
```

```
{
    var item = TodoItems.Find (id); if (item ==
    null) {

        trả về HttpNotFound ();

    } trả về ObjectResult mới (item);
}
```

Các phương pháp này thực hiện hai phương thức GET:

- GET / api / todo
- GET / api / todo / {id}

Đây là một phản hồi HTTP mẫu cho phương thức GetAll:

```
HTTP / 1.1 200 OK Nội dung-Loại: ứng dụng / json; charset = utf-8 Máy chủ: Microsoft-IIS / 10.0 Ngày: Thứ Sáu, ngày 18 tháng 6 năm 2015 20:51:10 GMT Nội dung-Độ dài: 82

[{"Khóa": "4f67d7c5-a2a9-4aae-b030-16003dd829ae", "Tên": "Item1", "IsComplete": false}]
```

Phần sau của hướng dẫn, tôi sẽ chỉ cho bạn cách bạn có thể xem phản hồi HTTP bằng công cụ Fiddler.

Định tuyến và đường dẫn URL

[HttpGet] thuộc tính chỉ định rằng đây là các phương thức HTTP GET. Đường dẫn URL cho mỗi phương thức được xây dựng như sau:

- Lấy chuỗi mẫu trong thuộc tính tuyến đường của bộ điều khiển, [Tuyến đường ("api / [bộ điều khiển]])]
- Thay thế “[Bộ điều khiển]” bằng tên của bộ điều khiển, là tên lớp bộ điều khiển trừ đi hậu tố “Bộ điều khiển”. Đối với mẫu này, tên của bộ điều khiển là “todo” (không phân biệt chữ hoa chữ thường). Đối với mẫu này, tên lớp bộ điều khiển là TodoController và tên gốc là “todo”. ASP.NET MVC không phân biệt chữ hoa chữ thường.
- Nếu thuộc tính [HttpGet] cũng có chuỗi mẫu, hãy nối chuỗi đó vào đường dẫn. Mẫu này không sử dụng chuỗi mẫu.

Đối với phương thức GetById, “{id}” là một biến giữ chỗ. Trong yêu cầu HTTP thực tế, khách hàng sẽ sử dụng ID của mục việc cần làm. Trong thời gian chạy, khi MVC gọi GetById, nó sẽ gán giá trị của “{id}” trong URL cho tham số id của phương thức.

Mở tệp src \ TodoApi \ Properties \ launcherSettings.json và thay thế giá trị launchUrl để sử dụng trò chơi lừa đảo. Thay đổi đó sẽ khiến IIS Express gọi bộ điều khiển việc làm khi dự án được bắt đầu.

```
{
    "hỗn hợp": {
        "IIS Express": {
            "commandName": "IISExpress",
            "launcherBrowser": true, "launchUrl": "api / todo", "environmentVariables": {
                "ASPNET_ENV": "Phát triển"
            }
        }
    }
}
```

Để tìm hiểu thêm về định tuyến yêu cầu trong MVC 6, hãy xem Định tuyến đến các hành động của bộ điều khiển.

Trả lại giá trị

Phương thức GetAll trả về một đối tượng CLR. MVC tự động tuần tự hóa đối tượng thành JSON và ghi JSON vào nội dung của thông báo phản hồi. Mã phản hồi cho phương thức này là 200, giả sử không có ngoại lệ nào được xử lý. (Các ngoại lệ chưa được xử lý được chuyển thành lỗi 5xx.)

Ngược lại, phương thức GetById trả về kiểu ActionResult tổng quát hơn, đại diện cho kiểu kết quả chung. Đó là bởi vì GetById có hai kiểu trả về khác nhau:

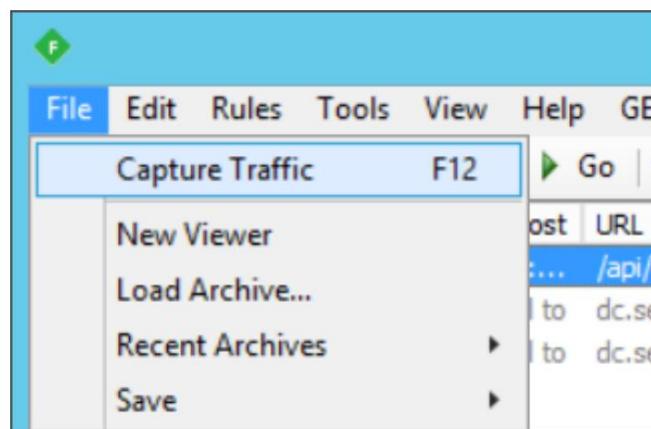
- Nếu không có mục nào khớp với ID được yêu cầu, phương thức này sẽ trả về lỗi 404. Điều này được thực hiện bằng cách trả về HttpNotFound.
- Nếu không, phương thức trả về 200 với phần thân phản hồi JSON. Điều này được thực hiện bằng cách trả về một ActionResult.

2.2.9 Sử dụng Fiddler để gọi API

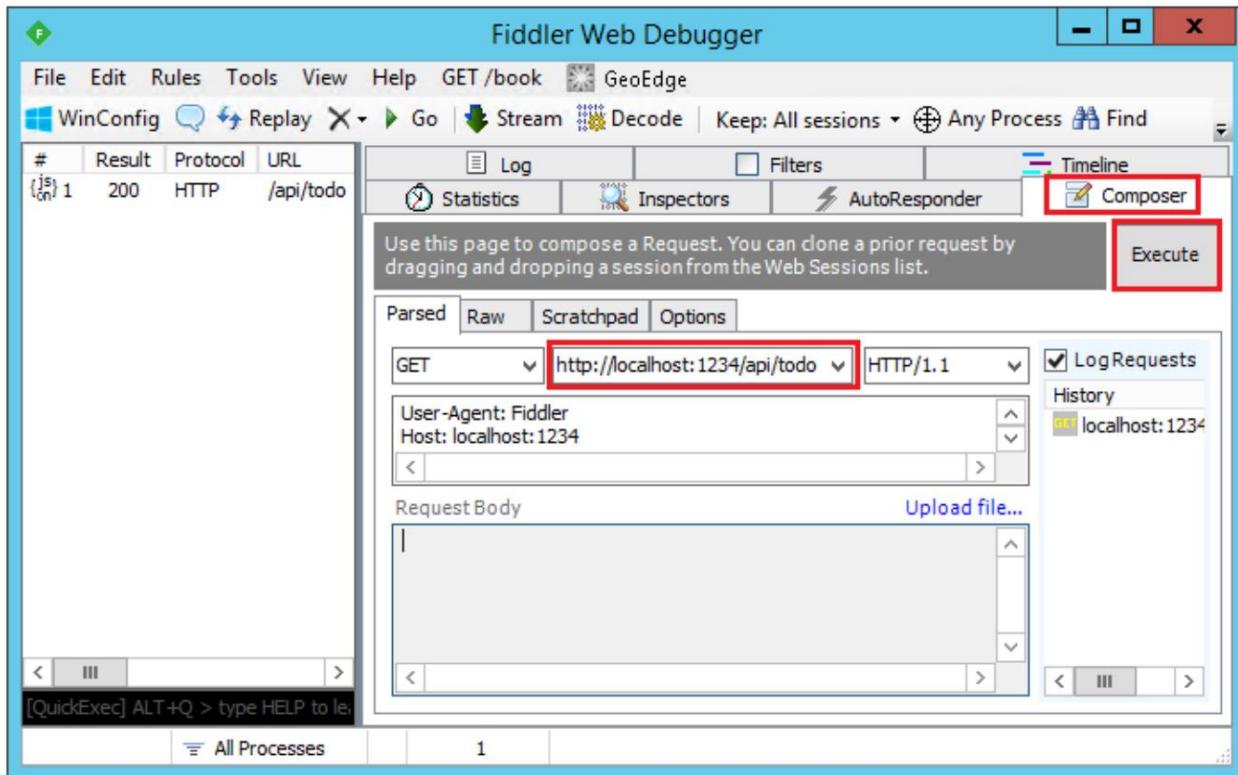
Bước này là tùy chọn, nhưng sẽ hữu ích khi xem các phản hồi HTTP thô từ API web. Trong Visual Studio, nhấn ^ F5 để khởi chạy ứng dụng. Visual Studio khởi chạy trình duyệt và điều hướng đến `http://localhost: port / api / todo`, nơi cổng là số cổng được chọn ngẫu nhiên. Nếu bạn đang sử dụng Chrome, Edge hoặc Firefox, dữ liệu việc cần làm sẽ được hiển thị.

Nếu bạn đang sử dụng IE, IE sẽ nhắc bạn mở hoặc lưu tệp todo.json.

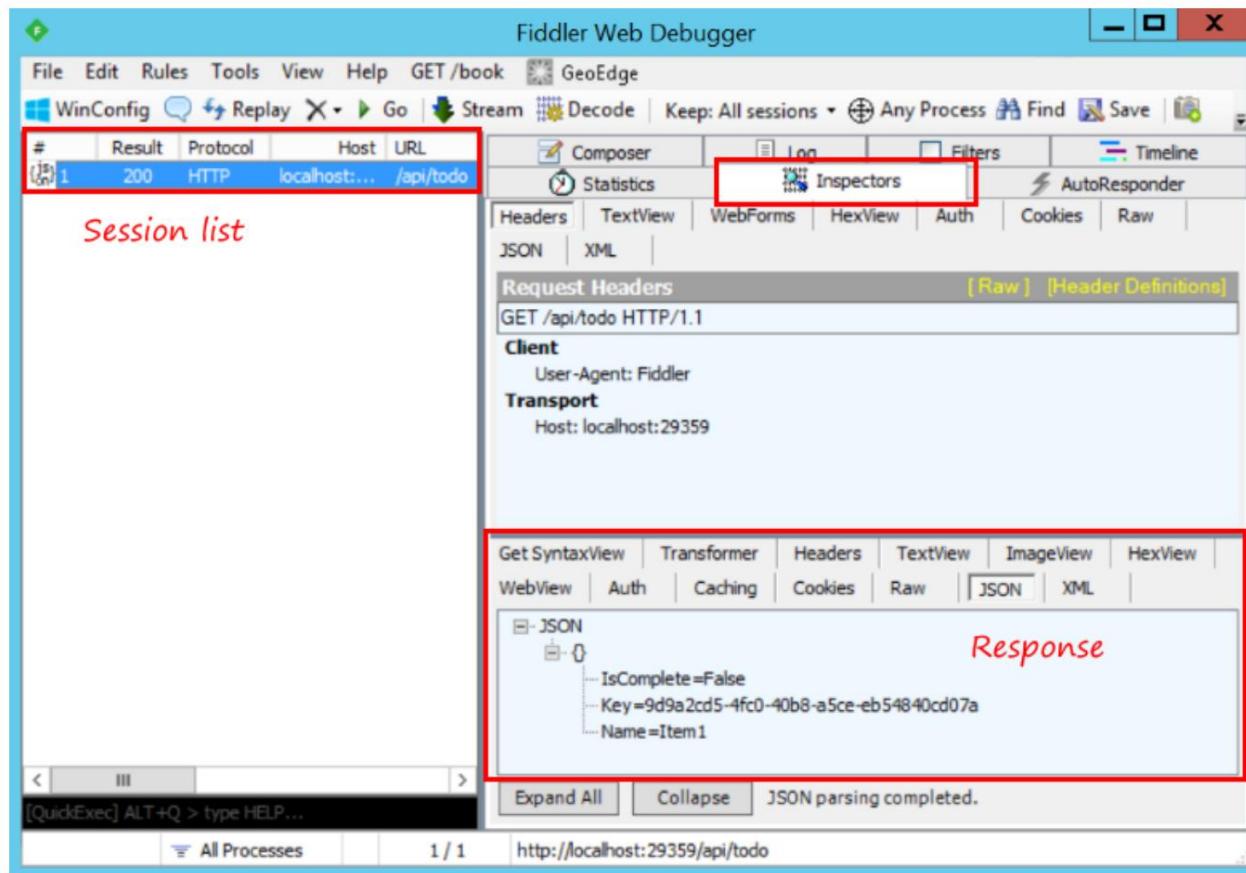
Khởi chạy Fiddler. Từ menu Tệp, bấm chọn tùy chọn Capture Traffic. Thao tác này sẽ tắt tính năng thu lưu lượng truy cập HTTP.



Chọn trang Trình soạn nhạc. Trong tab Đã phân tích cú pháp, nhập `http://localhost: port / api / todo`, trong đó port là số cổng. Nhập vào Thực hiện để gửi yêu cầu.



Kết quả xuất hiện trong danh sách phiên. Mã phản hồi phải là 200. Sử dụng tab Người kiểm tra để xem nội dung của phản hồi, bao gồm cả nội dung phản hồi.



2.2.10 Thực hiện các hoạt động CRUD khác

Bước cuối cùng là thêm các phương thức Tạo, Cập nhật và Xóa vào bộ điều khiển. Các phương pháp này là các biến thể của một chủ đề, vì vậy tôi sẽ chỉ hiển thị mã và nêu bật những điểm khác biệt chính.

Tạo ra

```
[HttpPost]
public IActionResult Create ([FromBody] TodoItem item) {

    if (item == null)
    {
        trả về BadRequest ();
    }
    TodoItems.Add (mục); return
    CreatedAtRoute ("GetTodo", mới {controller = "Todo", id = item.Key}, item);
}
```

Đây là một phương thức HTTP POST, được chỉ ra bởi `[HttpPost]` thuộc tính. `[FromBody]` thuộc tính yêu cầu MVC nhận giá trị của mục việc cần làm từ phần thân của yêu cầu HTTP.

`CreatedAtRoute` – phương thức trả về phản hồi 201, là phản hồi tiêu chuẩn cho phương thức HTTP POST tạo tài nguyên mới trên máy chủ. `CreateAtRoute` cũng thêm tiêu đề Vị trí vào phản hồi. Tiêu đề Vị trí chỉ định URI của mục việc cần làm mới được tạo. Xem 10.2.2 201 Đã tạo.

Chúng tôi có thể sử dụng Fiddler để gửi yêu cầu Tạo:

1. Trong trang Trình soạn nhạc , chọn ĐĂNG từ menu thả xuống.
2. Trong hộp văn bản tiêu đề yêu cầu, hãy thêm tiêu đề Loại-Nội dung với giá trị application / json. Fiddler tự động thêm tiêu đề Nội dung-Độ dài.
3. Trong hộp văn bản nội dung yêu cầu, hãy nhập nội dung sau: {"Tên": "<mục việc cần làm của bạn>"}
4. Nhập vào Thực thi.

The screenshot shows the Fiddler interface with the following details:

- Request Headers:**
 - Method: POST
 - URL: http://localhost:29359/api/todo
 - User-Agent: Fiddler
 - Host: localhost:29359
 - Content-Type: application/json
- Request Body:** { "Name": "Alphabetize paperclips" }

Đây là một phiên HTTP mẫu. Sử dụng tab Raw để xem dữ liệu phiên ở định dạng này.

Lời yêu cầu:

```
ĐĂNG http://localhost:29359/api/todo HTTP/1.1 Tác nhân người
dùng: Fiddler Máy chủ: localhost:29359
```

Loại nội dung: Ứng dụng / json
Nội dung-Độ dài: 33

```
{"Tên": "Xếp thứ tự bảng chữ cái trong bìa giấy"}
```

Phản ứng:

```
HTTP/1.1 201 Đã tạo
Nội dung-Loại: Ứng dụng / json; charset = utf-8 Vị trí: http://
localhost:29359/api/Todo/8fa2154d-f862-41f8-a5e5-a9a3faba0233 Máy chủ: Microsoft-IIS/10.0
```

Ngày: Thứ Sáu, ngày 18 tháng 6 năm 2015 20:51:55 GMT
Nội dung-Độ dài: 97

```
{"Khóa": "8fa2154d-f862-41f8-a5e5-a9a3faba0233", "Tên": "Xếp thứ tự bảng chữ cái", "Incomplete": false}
```

Cập nhật

```
[HttpPut ("{id}")] cập nhật IActionResult công khai (id chuỗi , [FromBody] mục TodoItem) {

    if (item == null || item.Key! = id) {

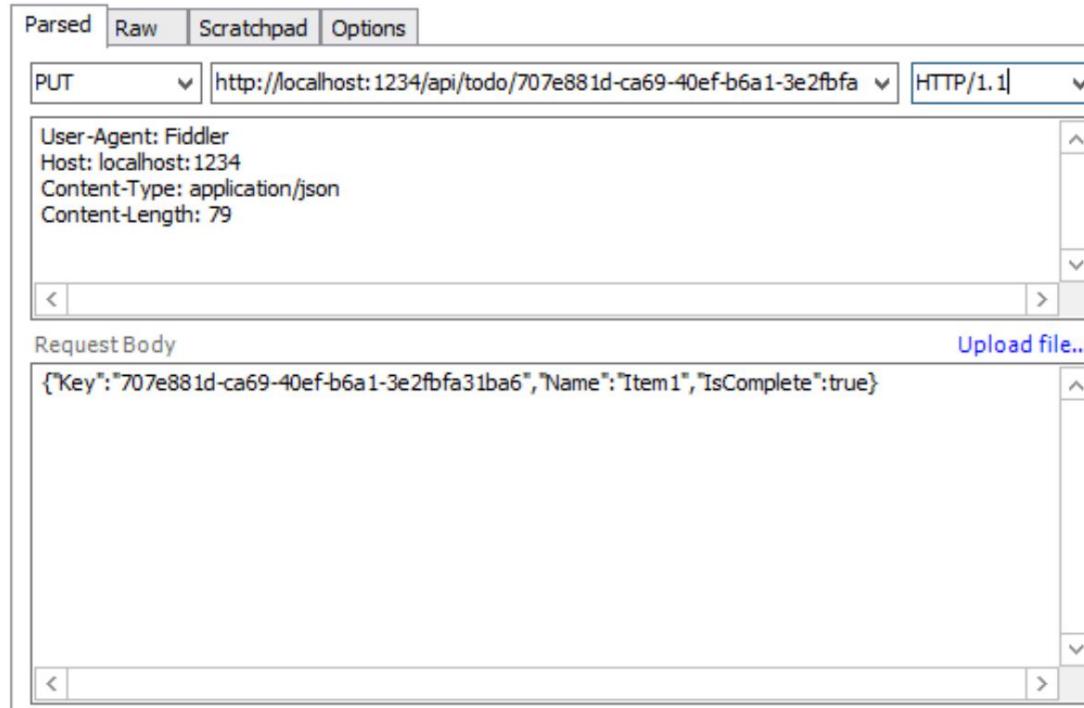
        trả về HttpNotFound ();
    }

    var todo = TodoItems.Find (id); if (todo ==
    null) {

        trả về HttpNotFound ();
    }

    TodoItems.Update (mục); trả về mới
    NoContentResult ();
}
```

Cập nhật tương tự như Tạo, nhưng sử dụng HTTP PUT. Câu trả lời là 204 (Không có nội dung). Theo thông số HTTP, một yêu cầu PUT yêu cầu máy khách gửi toàn bộ thực thể được cập nhật, không chỉ các delta. Để hỗ trợ cập nhật từng phần, hãy sử dụng HTTP PATCH.



Xóa bỏ

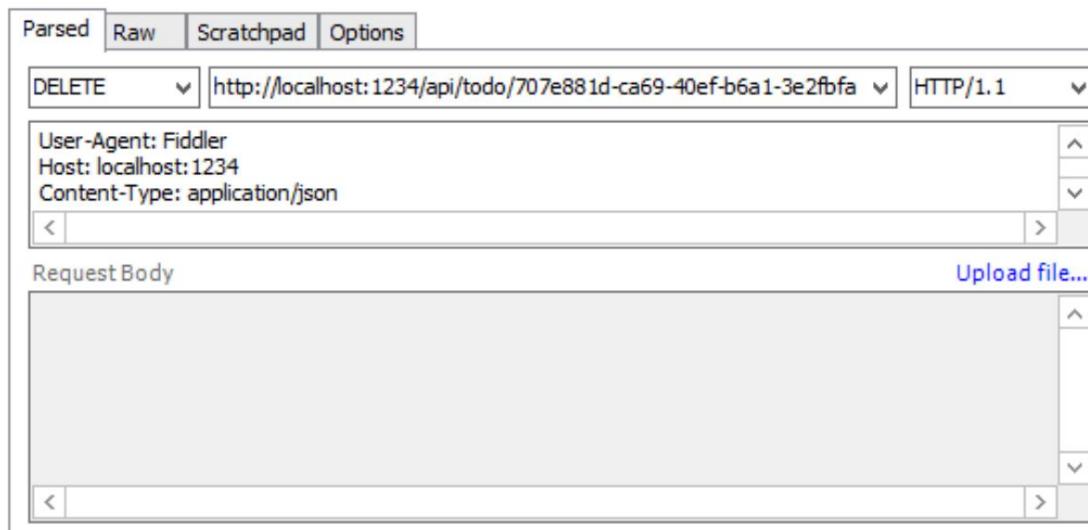
```
[HttpDelete ("{id}")] public void Delete (string id) {

    TodoItems.Remove (id);
}
```

Kiểu trả về void trả về phản hồi 204 (Không có nội dung). Điều đó có nghĩa là khách hàng nhận được 204 ngay cả khi mục đó đã bị xóa hoặc không bao giờ tồn tại. Có hai cách để nghĩ về yêu cầu xóa tài nguyên không tồn tại:

- “Xóa” có nghĩa là “xóa một mục hiện có” và mục đó không tồn tại, vì vậy hãy trả lại 404.
- “Xóa” có nghĩa là “đảm bảo mặt hàng không có trong bộ sưu tập.” Mặt hàng đã không có trong bộ sưu tập, vì vậy hãy trả lại 204.

Một trong hai cách tiếp cận là hợp lý. Nếu bạn trả về 404, máy khách sẽ cần phải xử lý trường hợp đó.



2.2.11 Các bước tiếp theo

Để tìm hiểu về cách tạo chương trình phụ trợ cho ứng dụng di động gốc, hãy xem [Tạo dịch vụ phụ trợ cho ứng dụng di động gốc](#).

Để biết thông tin về việc triển khai API của bạn, hãy xem [Xuất bản và Triển khai](#).

Hướng dẫn

3.1 Hướng dẫn về Cửa hàng âm nhạc

Lưu ý: Chúng tôi hiện đang làm việc về chủ đề này.

Chúng tôi hoan nghênh ý kiến đóng góp của bạn để giúp định hình phạm vi và cách tiếp cận. Bạn có thể theo dõi trạng thái và cung cấp thông tin đầu vào về vấn đề này tại GitHub.

Nếu bạn muốn xem lại các bản nháp và phác thảo ban đầu của chủ đề này, vui lòng để lại ghi chú với thông tin liên hệ của bạn trong số này.

Tìm hiểu thêm về cách bạn có thể đóng góp trên GitHub.

3.2 Tạo dịch vụ phụ trợ cho các ứng dụng di động gốc

Lưu ý: Chúng tôi hiện đang làm việc về chủ đề này.

Chúng tôi hoan nghênh ý kiến đóng góp của bạn để giúp định hình phạm vi và cách tiếp cận. Bạn có thể theo dõi trạng thái và cung cấp thông tin đầu vào về vấn đề này tại GitHub.

Nếu bạn muốn xem lại các bản nháp và phác thảo ban đầu của chủ đề này, vui lòng để lại ghi chú với thông tin liên hệ của bạn trong số này.

Tìm hiểu thêm về cách bạn có thể đóng góp trên GitHub.

Mô hình

4.1 Mô hình ràng buộc

Bởi Rachel Appel

Trong bài viết này

- Giới thiệu về ràng buộc mô hình
- Cách hoạt động của ràng buộc mô hình
- Tùy chỉnh hành vi ràng buộc mô hình với các thuộc tính
- Ràng buộc dữ liệu được định dạng từ phần thân yêu cầu

4.1.1 Giới thiệu về ràng buộc mô hình

Liên kết mô hình trong MVC ánh xạ dữ liệu từ các yêu cầu HTTP đến các tham số của phương thức hành động. Các tham số có thể là các kiểu đơn giản như chuỗi, số nguyên hoặc float hoặc chúng có thể là các kiểu phức tạp. Đây là một tính năng tuyệt vời của MVC vì ánh xạ dữ liệu đến một đối tác là một kịch bản thường lặp lại, bất kể kích thước hoặc độ phức tạp của dữ liệu.

MVC giải quyết vấn đề này bằng cách trích xuất tượng hóa ràng buộc đi để các nhà phát triển không phải tiếp tục viết lại một phiên bản hơi khác của cùng một mã đó trong mọi ứng dụng. Viết văn bản của riêng bạn để nhập mã chuyển đổi là tệ nhạt và dễ xảy ra lỗi.

4.1.2 Cách thức hoạt động của ràng buộc mô hình

Khi MVC nhận được một yêu cầu HTTP, nó sẽ định tuyến nó đến một phương thức hành động cụ thể của bộ điều khiển. Nó xác định phương thức hành động nào sẽ chạy dựa trên những gì có trong dữ liệu tuyến đường, sau đó nó liên kết các giá trị từ yêu cầu HTTP với các tham số của phương thức hành động đó. Ví dụ: hãy xem xét URL sau:

`http://contoso.com/movies/edit/2`

Vì mẫu tuyến đường trông giống như thế này, {controller = Home} / {action = Index} / {id?}, Phim / chỉnh sửa / 2 định tuyến đến trình điều khiển Phim và phương thức hành động Chỉnh sửa của nó. Nó cũng chấp nhận một tham số tùy chọn được gọi là id. Mã cho phương thức hành động sẽ trông giống như sau:

1 `chỉnh sửa IActionResult công khai (int? Id)`

Lưu ý: Các chuỗi trong tuyến URL không phân biệt chữ hoa chữ thường.

MVC sẽ cố gắng liên kết dữ liệu yêu cầu với các tham số hành động theo tên. MVC sẽ tìm kiếm các giá trị cho từng tham số bằng cách sử dụng tên tham số và tên của các thuộc tính có thể thiết lập công khai của nó. Trong ví dụ trên, hành động duy nhất

tham số được đặt tên là id, mà MVC liên kết với giá trị có cùng tên trong các giá trị của tuyến đường. Ngoài các giá trị định tuyến, MVC sẽ ràng buộc dữ liệu từ các phần khác nhau của yêu cầu và nó làm như vậy theo một thứ tự đã định. Dưới đây là danh sách các nguồn dữ liệu theo thứ tự ràng buộc mô hình xem qua chúng:

1. Giá trị biểu mẫu: Đây là các giá trị biểu mẫu đi trong yêu cầu HTTP bằng phương thức POST. (bao gồm jQuery Yêu cầu ĐĂNG).
2. Giá trị tuyến: Tập hợp các giá trị tuyến được cung cấp bởi định tuyến.
3. Chuỗi truy vấn: Phần chuỗi truy vấn của URI.

Lưu ý: Các giá trị biểu mẫu, dữ liệu định tuyến và chuỗi truy vấn đều được lưu trữ dưới dạng cặp tên-giá trị.

Vì liên kết mô hình yêu cầu khóa có tên id và không có id có tên nào trong các giá trị biểu mẫu, nên nó đã chuyển sang các giá trị định tuyến tìm kiếm khóa đó. Trong ví dụ của chúng tôi, đó là một trạn đấu. Ràng buộc xảy ra và giá trị được chuyển đổi thành số nguyên 2. Yêu cầu tương tự sử dụng Chính sửa (chuỗi id) sẽ chuyển đổi thành chuỗi "2".

Cho đến nay ví dụ sử dụng các kiểu đơn giản. Trong MVC, các kiểu đơn giản là bất kỳ kiểu nguyên thủy .NET nào hoặc kiểu có bộ chuyển đổi kiểu chuỗi. Nếu tham số của phương thức hành động là một lớp chẳng hạn như kiểu Phim, chứa cả kiểu plex đơn giản và com plex làm thuộc tính, thì ràng buộc mô hình của MVC sẽ vẫn xử lý nó một cách độc đáo. Nó sử dụng phản xạ và đệ quy để duyệt qua các thuộc tính của các kiểu phức tạp để tìm kiếm các kết quả phù hợp. Liên kết mô hình tìm kiếm tham số mẫu_name.property_name để liên kết các giá trị với thuộc tính. Nếu nó không tìm thấy các giá trị phù hợp của biểu mẫu này, nó sẽ cố gắng liên kết chỉ bằng cách sử dụng tên property. Đối với những kiểu đó, chẳng hạn như Kiểu tập hợp, liên kết mô hình sẽ tìm kiếm các kết quả phù hợp với tên_tham_số [chỉ mục] hoặc chỉ [chỉ mục]. Liên kết mô hình xử lý các loại từ điển tương tự như vậy, yêu cầu tham số_name [khóa] hoặc chỉ [khóa], miễn là khóa của chúng là loại đơn giản. Các khóa được hỗ trợ khớp với tên trường HTML và trình trợ giúp thẻ được tạo cho cùng một loại mô hình. Điều này cho phép các giá trị xoay vòng để các trường biểu mẫu vẫn được diển đầy đủ thông tin đầu vào của người dùng để thuận tiện cho họ, ví dụ: khi dữ liệu liên kết từ một lần tạo hoặc chỉnh sửa không vượt qua được xác thực.

Để ràng buộc xảy ra, các thành viên của các lớp phải là thuộc tính công khai, có thể ghi, có chứa một phương thức khởi tạo công khai mặc định. Các trường công khai không bị ràng buộc. Thuộc tính của kiểu IEnumarable hoặc mảng phải có thẻ thiết lập được và sẽ được diền vào một mảng. Thuộc tính bộ sưu tập của loại ICollection <T> có thẻ ở chế độ chỉ đọc.

Khi một tham số bị ràng buộc, liên kết mô hình ngừng tìm kiếm các giá trị có tên đó và nó chuyển sang ràng buộc tham số tiếp theo. Nếu liên kết không thành công, MVC sẽ không báo lỗi. Bạn có thể truy vấn lỗi trạng thái mô hình bằng cách kiểm tra thuộc tính ModelState.IsValid.

Lưu ý: Mỗi mục nhập trong thuộc tính ModelState của bộ điều khiển là một ModelStateEntry chứa thuộc tính Lỗi. Ít khi cần thiết phải tự mình truy vấn bộ sưu tập này. Sử dụng ModelState.IsValid để thay thế.

Ngoài ra, có một số kiểu dữ liệu đặc biệt mà MVC phải xem xét khi thực hiện ràng buộc mô hình:

- IFormFile, IEnumarable <IFormFile>: Một hoặc nhiều tệp được tải lên là một phần của yêu cầu HTTP.
- CancellationToken: Được sử dụng để hủy bỏ hoạt động trong bộ điều khiển không đồng bộ.

Các kiểu này có thể được liên kết với các tham số hành động hoặc các thuộc tính trên một kiểu lớp.

Sau khi liên kết mô hình hoàn tất, xác thực xảy ra. Ràng buộc mô hình mặc định hoạt động tốt cho phần lớn các kịch bản phát triển. Nó cũng có thể mở rộng nếu bạn có nhu cầu riêng, bạn có thể tùy chỉnh hành vi tích hợp sẵn.

4.1.3 Tùy chỉnh hành vi ràng buộc mô hình với các thuộc tính

MVC chứa một số thuộc tính mà bạn có thể sử dụng để hướng hành vi liên kết mô hình mặc định của nó tới một nguồn khác.

Ví dụ: bạn có thể chỉ định liệu ràng buộc là bắt buộc đối với một thuộc tính hay nếu nó không bao giờ xảy ra bằng cách sử dụng thuộc tính [BindRequired] hoặc [BindNever]. Ngoài ra, bạn có thể ghi đè nguồn dữ liệu mặc định và chỉ định nguồn dữ liệu của chất kết dính mô hình. Dưới đây là danh sách các thuộc tính ràng buộc mô hình:

- [BindRequired]: Thuộc tính này thêm lỗi trạng thái mô hình nếu không thể xảy ra ràng buộc.

- [BindNever]: Cho chất kết dính mô hình không bao giờ liên kết với tham số này.
- [FromHeader], [FromQuery], [FromRoute], [FromForm]: Sử dụng các phần này để chỉ định nguồn liên kết chính xác mà bạn muốn áp dụng.
- [FromServices]: Thuộc tính này sử dụng tính năng **tiêm phụ thuộc** để ràng buộc các tham số từ các dịch vụ.
- [FromBody]: Sử dụng bộ định dạng đã cấu hình để liên kết dữ liệu từ phần thân yêu cầu. Định dạng được chọn dựa trên loại nội dung của yêu cầu.
- [ModelBinder]: Được sử dụng để ghi đè tên và nguồn liên kết mô hình mặc định.

Thuộc tính là công cụ rất hữu ích khi bạn cần ghi đè hành vi mặc định của ràng buộc mô hình.

4.1.4 Ràng buộc dữ liệu được định dạng từ phần thân yêu cầu

Dữ liệu yêu cầu có thể có nhiều định dạng bao gồm JSON, XML và nhiều định dạng khác. Khi bạn sử dụng thuộc tính [FromBody] để chỉ ra rằng bạn muốn liên kết một tham số với dữ liệu trong phần thân yêu cầu, MVC sử dụng một bộ định dạng đã định cấu hình để xử lý dữ liệu yêu cầu dựa trên loại nội dung của nó. Theo mặc định, MVC bao gồm một lớp JsonInputFormatter để xử lý dữ liệu JSON, nhưng bạn có thể thêm các bộ định dạng bổ sung để xử lý XML và các định dạng tùy chỉnh khác.

Lưu ý: JsonInputFormatter là định dạng mặc định và nó dựa trên Json.NET.

ASP.NET chọn bộ định dạng đầu vào dựa trên **Loại-Nội dung tiêu đề** và loại của tham số, trừ khi có một thuộc tính được áp dụng cho nó chỉ định khác. Nếu bạn muốn sử dụng XML hoặc định dạng khác, bạn phải định cấu hình nó trong tệp Startup.cs, nhưng trước tiên bạn có thể phải lấy tham chiếu đến Microsoft.AspNet.Mvc.Formatters.Xml bằng NuGet. Mã khởi động của bạn sẽ trông giống như sau:

```
1 public void ConfigureServices (dịch vụ IServiceProvider)
2 {
3     services.AddMvc ()
4         .AddXmlSerializerFormatters ();
5 }
```

Mã trong tệp Startup.cs chứa phương thức ConfigureServices với đối số dịch vụ mà bạn có thể sử dụng để xây dựng dịch vụ cho ứng dụng ASP.NET của mình. Trong mẫu, chúng tôi đang thêm một định dạng XML làm dịch vụ mà MVC sẽ cung cấp cho ứng dụng này. Đối số tùy chọn được truyền vào phương thức AddMvc cho phép bạn thêm và quản lý bộ lọc, bộ định dạng và các tùy chọn hệ thống khác từ MVC khi khởi động ứng dụng. Sau đó, áp dụng thuộc tính Consumes cho các lớp bộ điều khiển hoặc các phương thức hành động để hoạt động với định dạng bạn muốn.

4.2 Xác thực mô hình

Lưu ý: Chúng tôi hiện đang làm việc về chủ đề này.

Chúng tôi hoan nghênh ý kiến đóng góp của bạn để giúp định hình phạm vi và cách tiếp cận. Bạn có thể theo dõi trạng thái và cung cấp thông tin đầu vào về vấn đề này tại GitHub.

Nếu bạn muốn xem lại các bản nháp và phác thảo ban đầu của chủ đề này, vui lòng để lại ghi chú với thông tin liên hệ của bạn trong số này.

Tìm hiểu thêm về cách bạn có thể đóng góp trên GitHub.

4.3 Định dạng

Lưu ý: Chúng tôi hiện đang làm việc về chủ đề này.

Chúng tôi hoan nghênh ý kiến đóng góp của bạn để giúp định hình phạm vi và cách tiếp cận. Bạn có thể theo dõi trạng thái và cung cấp thông tin đầu vào về vấn đề này tại GitHub.

Nếu bạn muốn xem lại các bản nháp và phác thảo ban đầu của chủ đề này, vui lòng để lại ghi chú với thông tin liên hệ của bạn trong số này.

Tìm hiểu thêm về cách bạn có thể đóng góp trên GitHub.

4.4 Bộ định dạng tùy chỉnh

Lưu ý: Chúng tôi hiện đang làm việc về chủ đề này.

Chúng tôi hoan nghênh ý kiến đóng góp của bạn để giúp định hình phạm vi và cách tiếp cận. Bạn có thể theo dõi trạng thái và cung cấp thông tin đầu vào về vấn đề này tại GitHub.

Nếu bạn muốn xem lại các bản nháp và phác thảo ban đầu của chủ đề này, vui lòng để lại ghi chú với thông tin liên hệ của bạn trong số này.

Tìm hiểu thêm về cách bạn có thể đóng góp trên GitHub.

Lượt xem

5.1 Cú pháp Razor

Lưu ý: Chúng tôi hiện đang làm việc về chủ đề này.

Chúng tôi hoan nghênh ý kiến đóng góp của bạn để giúp định hình phạm vi và cách tiếp cận. Bạn có thể theo dõi trạng thái và cung cấp thông tin đầu vào về vấn đề này tại GitHub.

Nếu bạn muốn xem lại các bản nháp và phác thảo ban đầu của chủ đề này, vui lòng để lại ghi chú với thông tin liên hệ của bạn trong số này.

Tìm hiểu thêm về cách bạn có thể đóng góp trên GitHub.

5.2 Chế độ xem động so với chế độ xem được nhập mạnh

Lưu ý: Chúng tôi hiện đang làm việc về chủ đề này.

Chúng tôi hoan nghênh ý kiến đóng góp của bạn để giúp định hình phạm vi và cách tiếp cận. Bạn có thể theo dõi trạng thái và cung cấp thông tin đầu vào về vấn đề này tại GitHub.

Nếu bạn muốn xem lại các bản nháp và phác thảo ban đầu của chủ đề này, vui lòng để lại ghi chú với thông tin liên hệ của bạn trong số này.

Tìm hiểu thêm về cách bạn có thể đóng góp trên GitHub.

Tìm hiểu thêm về Chế độ xem động so với Chế độ xem được nhập mạnh.

5.3 Trình trợ giúp HTML

Lưu ý: Chúng tôi hiện đang làm việc về chủ đề này.

Chúng tôi hoan nghênh ý kiến đóng góp của bạn để giúp định hình phạm vi và cách tiếp cận. Bạn có thể theo dõi trạng thái và cung cấp thông tin đầu vào về vấn đề này tại GitHub.

Nếu bạn muốn xem lại các bản nháp và phác thảo ban đầu của chủ đề này, vui lòng để lại ghi chú với thông tin liên hệ của bạn trong số này.

Tìm hiểu thêm về cách bạn có thể đóng góp trên GitHub.

5.4 Trình trợ giúp thẻ

5.4.1 Giới thiệu về Trình trợ giúp thẻ

Bởi Rick Anderson

- [Trình trợ giúp Thẻ là gì?](#)
- [Công cụ trợ giúp thẻ cung cấp gì](#)
- [Quản lý phạm vi Trình trợ giúp thẻ](#)
- [Hỗ trợ IntelliSense cho Trình trợ giúp thẻ](#)
- [Trình trợ giúp thẻ so với Trình trợ giúp HTML](#)
- [Tag Helpers so với Web Server Controls](#)
- [Tùy chỉnh phông chữ phần tử Trình trợ giúp thẻ](#)
- [Tài nguyên bổ sung](#)

[Trình trợ giúp thẻ là gì?](#)

Trình trợ giúp thẻ cho phép mã phía máy chủ tham gia vào việc tạo và hiển thị các phần tử HTML trong tệp Razor. Ví dụ: `ImageTagHelper` tích hợp sẵn có thể thêm số phiên bản vào tên hình ảnh. Bất cứ khi nào hình ảnh thay đổi, máy chủ sẽ tạo ra một phiên bản duy nhất mới cho hình ảnh, vì vậy các máy khách được đảm bảo nhận được hình ảnh hiện tại (thay vì một hình ảnh được lưu trong bộ nhớ cache cũ). Có nhiều Trình trợ giúp thẻ được tích hợp sẵn cho các tác vụ phổ biến - chẳng hạn như tạo biểu mẫu, liên kết, tải nội dung và hơn thế nữa - và thậm chí có nhiều hơn nữa trong kho lưu trữ GitHub công khai và dưới dạng gói NuGet. Trình trợ giúp thẻ được tạo bằng C# và chúng nhắm mục tiêu các phần tử HTML dựa trên tên phần tử, tên thuộc tính hoặc thẻ mẹ. Ví dụ: `LabelTagHelper` được tích hợp sẵn có thể nhắm mục tiêu phần tử HTML `<label>` khi các thuộc tính `LabelTagHelper` được áp dụng. Nếu bạn đã quen thuộc với [Trình trợ giúp HTML](#), Trình trợ giúp thẻ giảm chayen đổi rõ ràng giữa HTML và C# trong chế độ xem Razor. [Trình trợ giúp thẻ so với Trình trợ giúp HTML](#) giải thích sự khác biệt chi tiết hơn.

[Những gì Công cụ trợ giúp Thẻ cung cấp](#)

Trải nghiệm phát triển thân thiện với HTML. Đối với hầu hết các phần, đánh dấu Razor sử dụng Trình trợ giúp thẻ trông giống như HTML tiêu chuẩn. Các nhà thiết kế front-end thông thạo HTML / CSS / JavaScript có thể chỉnh sửa Razor mà không cần học cú pháp C# Razor.

Môi trường IntelliSense phong phú để tạo đánh dấu HTML và Razor. Điều này trái ngược hẳn với HTML Helpers, cách tiếp cận trước đây để tạo đánh dấu phía máy chủ trong dạng xem Razor. [Trình trợ giúp thẻ so với Trình trợ giúp HTML](#) giải thích sự khác biệt chi tiết hơn. [Hỗ trợ IntelliSense cho Trình trợ giúp thẻ](#) giải thích môi trường Intel liSense. Ngay cả những nhà phát triển có kinh nghiệm với cú pháp Razor C# cũng có năng suất cao hơn khi sử dụng Trình trợ giúp thẻ hơn là viết đánh dấu C# Razor.

Một cách để giúp bạn làm việc hiệu quả hơn và có thể tạo ra mã mạnh hơn, đáng tin cậy và có thể bảo trì chỉ bằng cách sử dụng thông tin. Ví dụ: trước đây, câu trả lời về việc cập nhật hình ảnh là thay đổi tên của hình ảnh khi bạn thay đổi hình ảnh. Hình ảnh nên được lưu trữ mạnh mẽ vì lý do hiệu suất và trừ khi bạn thay đổi tên của hình ảnh, bạn có nguy cơ khách hàng nhận được một bản sao cũ. Trong lịch sử, sau khi một hình ảnh được chỉnh sửa, tên phải được thay đổi và mỗi tham chiếu đến hình ảnh trong ứng dụng web cần được cập nhật. Điều này không chỉ rất tốn công sức mà còn dễ xảy ra lỗi (Bạn có thể bỏ lỡ một tham chiếu, vô tình nhập sai chuỗi, v.v.) `ImageTagHelper` tích hợp sẵn có thể làm điều này cho bạn một cách tự động. `ImageTagHelper` có thể thêm số phiên bản vào tên hình ảnh, vì vậy bất cứ khi nào hình ảnh thay đổi, máy chủ sẽ tự động tạo một phiên bản duy nhất mới cho hình ảnh. Khách hàng được đảm bảo rằng có được hình ảnh hiện tại. Tính mạnh mẽ và tiết kiệm lao động này về cơ bản hoàn toàn miễn phí bằng cách sử dụng `ImageTagHelper`.

Hầu hết các Trình trợ giúp thẻ tích hợp đều nhầm mục tiêu đến các phần tử HTML hiện có và cung cấp các thuộc tính phía máy chủ cho phần tử. Ví dụ: phần tử `<input>` được sử dụng trong nhiều dạng xem trong thư mục Chế độ xem / Tài khoản có chứa asp-for thuộc tính này trích xuất tên của thuộc tính mô hình được chỉ định vào HTML được hiển thị. Dao cao sau đánh dấu:

```
<label asp-for = "Email"> </label>
```

Tạo HTML sau:

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

Thuộc tính `asp-for` được cung cấp bởi thuộc tính `For` trong `LabelTagHelper`. Xem Trình trợ giúp thẻ tác giả để biết thêm thông tin.

Quản lý phạm vi Trình trợ giúp thẻ

Phạm vi của Trình trợ giúp thẻ được kiểm soát bởi sự kết hợp của `@addTagHelper`, `@removeTagHelper` và `"!"` chọn không tham gia tính cách.

@addTagHelper cung cấp Trình trợ giúp thẻ

Nếu bạn tạo một ứng dụng web ASP.NET 5 mới có tên AuthoringTagHelpers (không có xác thực), như sau:

```
@using AuthoringTagHelpers  
@addTagHelper *, Microsoft.AspNetCore.Mvc.TagHelpers
```

Chi tiết @addTagHelper cung cấp Trình trợ giúp thẻ cho chế độ xem. Trong trường hợp này, tệp chế độ xem là Views / _ViewImports.cshtml, theo mặc định được kế thừa bởi tất cả các tệp chế độ xem trong thư mục Chế độ xem và các thư mục con; làm cho Trình trợ giúp thẻ có sẵn. Đoạn mã trên sử dụng cú pháp ký tự đại diện ("*") để chỉ định rằng tất cả các Trình trợ giúp thẻ trong lặp ráp được chỉ định (`Microsoft.AspNetCore.Mvc.TagHelpers`) sẽ có sẵn cho mọi tệp chế độ xem trong thư mục Chế độ xem hoặc thư mục con. Tham số đầu tiên sau `@addTagHelper` chỉ định Trình trợ giúp thẻ tải (chúng tôi đang sử dụng "*" cho tất cả Tag Helpers) và tham số thứ hai "`Microsoft.AspNetCore.Mvc.TagHelpers`" chỉ định assembly chứa Trình trợ giúp thẻ. `Microsoft.AspNetCore.Mvc.TagHelpers` là tập hợp cho Trình trợ giúp thẻ ASP.NET 5 được tích hợp sẵn.

Để hiển thị tất cả các Trình trợ giúp thẻ trong dự án này (tạo ra một tập hợp có tên AuthoringTagHelpers), bạn sẽ sử dụng như sau:

```
@using AuthoringTagHelpers  
@addTagHelper "*", Microsoft.AspNetCore.Mvc.TagHelpers"  
@addTagHelper "*", AuthoringTagHelpers"
```

Nếu chưa (AuthoringTagHelpers.TagHelpers.EmailTagHelper) tách riêng, tên cung cấp tên đủ điều kiện mặc định không gian tên

(FON) của Trình trơ giúp thè:

```
@using AuthoringTagHelpers  
@addTagHelper *, Microsoft.AspNetCore.Mvc.TagHelpers  
@addTagHelper "AuthoringTagHelpers.TagHelpers.EmailTagHelper, AuthoringTagHelpers"
```

Để thêm Trình trợ giúp thẻ vào chế độ xem bằng FQN, trước tiên bạn thêm FQN (AuthoringTagHelpers.TagHelpers.EmailTagHelper), và sau đó là tên hội (Authoring TagHelpers). Hầu hết các nhà phát triển thích sử dụng cú pháp ký tự đại diện "*". Cú pháp ký tự đại diện cho phép bạn chèn ký tự đại diện "*" làm hậu tố trong FQN. Ví dụ: bất kỳ lệnh nào sau đây sẽ mang lại EmailTagHelper:

```
@addTagHelper "AuthoringTagHelpers.TagHelpers.E*", AuthoringTagHelpers" @addTagHelper
"AuthoringTagHelpers.TagHelpers.Email*", AuthoringTagHelpers"
```

Như đã đề cập trước đây, việc thêm chỉ thị @addTagHelper vào tệp Views / _ViewImports.cshtml làm cho Trình trợ giúp thẻ có sẵn cho tất cả các tệp ché độ xem trong thư mục Ché độ xem và các thư mục con. Bạn có thể sử dụng lệnh @addTagHelper trong các tệp ché độ xem cụ thể nếu bạn muốn chọn tham gia chỉ hiển thị Trình trợ giúp thẻ cho các ché độ xem đó.

@removeTagHelper xóa Trình trợ giúp thẻ

@RemoveTagHelper có hai tham số giống như @addTagHelper và nó xóa Trình trợ giúp thẻ đã được thêm trước đó. Ví dụ: @removeTagHelper được áp dụng cho một ché độ xem cụ thể sẽ xóa Trình trợ giúp thẻ được chỉ định khỏi ché độ xem. Sử dụng @removeTagHelper trong tệp Ché độ xem / Thư mục / _ViewImports.cshtml sẽ xóa Trình trợ giúp thẻ được chỉ định khỏi tất cả các ché độ xem trong Thư mục.

Kiểm soát phạm vi Trình trợ giúp thẻ với tệp _ViewImports.cshtml

Bạn có thể thêm _ViewImports.cshtml vào bất kỳ thư mục ché độ xem nào và công cụ ché độ xem thêm các lệnh từ tệp _ViewImports.cshtml đó vào các lệnh chứa trong tệp Ché độ xem / _ViewImports.cshtml. Nếu bạn đã thêm tệp Views / Home / _ViewImports.cshtml trống cho các ché độ xem Trang chủ, sẽ không có thay đổi nào vì tệp _ViewImports.cshtml là phụ gia. Bất kỳ lệnh @addTagHelper nào bạn thêm vào tệp Ché độ xem / Home / _ViewImports.cshtml (không có trong tệp Ché độ xem / _ViewImports.cshtml mặc định) sẽ chỉ hiển thị các Trình trợ giúp thẻ đó ở ché độ xem trong thư mục Trang chủ.

Chọn không tham gia các yếu tố riêng lẻ

Bạn có thể vô hiệu hóa Trình trợ giúp thẻ ở cấp phần tử bằng ký tự chọn không tham gia Trình trợ giúp thẻ ("!"). Ví dụ: xác thực Email bị tắt trong có ký tự chọn không tham gia Trình trợ giúp thẻ:

```
<! span asp-validation-for = "Email" class = "text-risk"> </! span>
```

Bạn phải áp dụng ký tự chọn không tham gia Trình trợ giúp thẻ cho thẻ mở và thẻ đóng. (Trình chỉnh sửa Visual Studio tự động thêm ký tự chọn không tham gia vào thẻ đóng khi bạn thêm một ký tự vào thẻ mở). Sau khi bạn thêm ký tự chọn không tham gia, phần tử và thuộc tính Trình trợ giúp thẻ không còn được hiển thị bằng phông chữ đặc biệt nữa.

Sử dụng @tagHelperPrefix để làm rõ việc sử dụng Trình trợ giúp thẻ

Chỉ thị @tagHelperPrefix cho phép bạn chỉ định chuỗi tiền tố thẻ để bật hỗ trợ Trình trợ giúp thẻ và làm rõ việc sử dụng Trình trợ giúp thẻ. Trong hình ảnh mã bên dưới, tiền tố Trình trợ giúp thẻ được đặt thành "th:", vì vậy chỉ những phần tử sử dụng tiền tố "th:" mới hỗ trợ Trình trợ giúp thẻ (các phần tử được bật Trình trợ giúp thẻ có phông chữ đặc biệt). Các phần tử <label> và có tiền tố Trình trợ giúp thẻ và được bật Trình trợ giúp thẻ, trong khi phần tử <input> thì không.

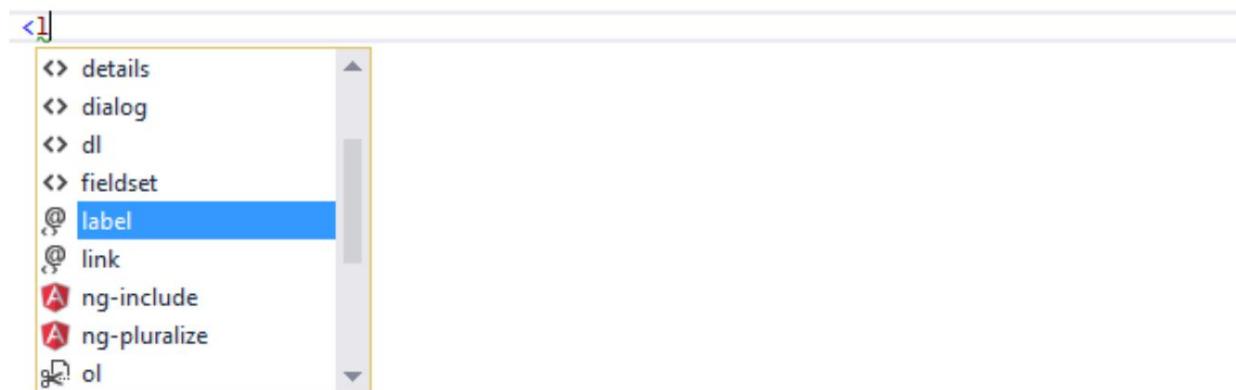
```
@tagHelperPrefix "th:"
<div class="form-group">
    <th:label asp-for="Password" class="col-md-2 control-label"></th:label>
    <div class="col-md-10">
        <input asp-for="Password" class="form-control" />
        <th:span asp-validation-for="Password" class="text-danger"></th:span>
    </div>
</div>
```

Các quy tắc phân cấp tương tự áp dụng cho @addTagHelper cũng áp dụng cho @tagHelperPrefix.

Hỗ trợ IntelliSense cho Trình trợ giúp thẻ

Khi bạn tạo một ứng dụng web ASP.NET mới trong Visual Studio, nó sẽ thêm "Microsoft.AspNet.Tooling.Razor" vào tệp project.json. Đây là gói bổ sung công cụ Trình trợ giúp thẻ.

Cân nhắc viết một phần tử HTML <label>. Ngay sau khi bạn nhập <l vào trình chỉnh sửa Visual Studio, IntelliSense sẽ hiển thị các phần tử phù hợp:



The label element represents a caption in a user interface. The caption can be associated with a specific form control, known as the label element's labeled control, either using the for attribute, or by putting the form control inside the label element itself.

Bạn không chỉ nhận được trợ giúp về HTML mà còn có biểu tượng (biểu tượng "@" với "<>" bên dưới).

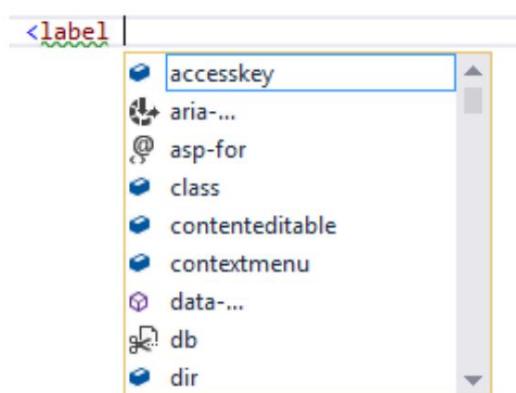


xác định phần tử như được nhắm mục tiêu bởi Trình trợ giúp thẻ. Các phần tử HTML thuần túy (chẳng hạn như tập trường) hiển thị biểu tượng "<>".

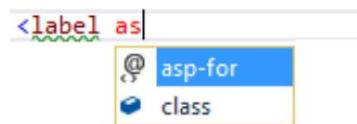
Thẻ <label> HTML thuần túy hiển thị thẻ HTML (với chủ đề màu Visual Studio mặc định) bằng phông chữ màu nâu, các thuộc tính có màu đỏ và các giá trị thuộc tính có màu xanh lam.

<label class="col-md-2">Email</label>

Sau khi bạn nhập nhãn <, IntelliSense liệt kê các thuộc tính HTML / CSS có sẵn và các thuộc tính nhắm mục tiêu của Trình trợ giúp thẻ:



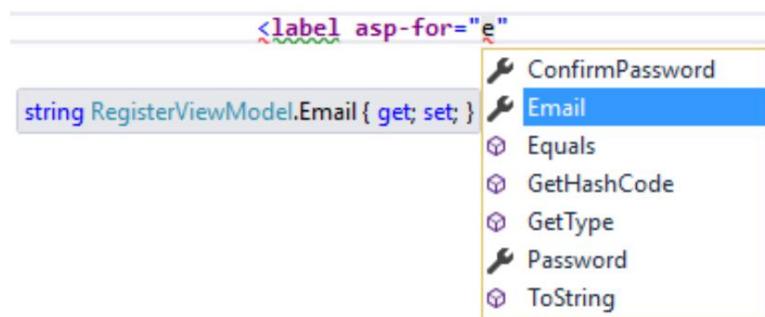
Hoàn thành câu lệnh IntelliSense cho phép bạn nhập phím tab để hoàn thành câu lệnh với giá trị đã chọn:



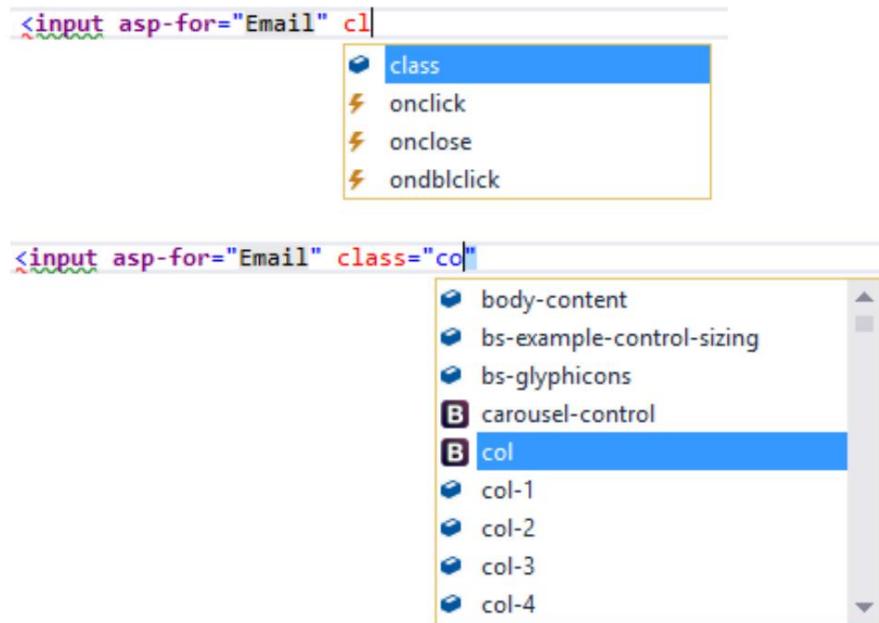
Ngay sau khi thuộc tính Trình trợ giúp thẻ được nhập, thẻ và phông chữ thuộc tính sẽ thay đổi. Sử dụng chủ đề màu "Xanh lam" hoặc "Sáng" mặc định của Visual Studio, phông chữ có màu tím đậm. Nếu bạn đang sử dụng chủ đề "Dark", phông chữ có màu xanh mòng két đậm. Hình ảnh trong tài liệu này được chụp bằng chủ đề mặc định.

<label asp-for

Bạn có thể nhập phím tắt Visual Studio CompleteWord (Ctrl + phím cách là **mặc định**) bên trong dấu ngoặc kép (""), và bây giờ bạn đang học C#, giống như bạn đang học trong một lớp C#. IntelliSense hiển thị tất cả các phương thức và thuộc tính trên mô hình trang. Các phương thức và thuộc tính có sẵn vì kiểu thuộc tính là ModelExpression. Trong hình ảnh bên dưới, tôi đang chỉnh sửa chế độ xem Đăng ký, do đó, Chế độ xem Đăng ký có sẵn.



IntelliSense liệt kê các thuộc tính và phương pháp có sẵn cho mô hình trên trang. Môi trường IntelliSense phong phú giúp bạn chọn lớp CSS:



Trình trợ giúp thẻ so với Trình trợ giúp HTML

Trình trợ giúp thẻ định kèm vào các phần tử HTML trong chế độ xem Razor, trong khi Trình trợ giúp HTML được gọi dưới dạng các phương thức xen kẽ với HTML trong chế độ xem Razor. Hãy xem xét đánh dấu Razor sau đây, tạo nhãn HTML với "chú thích" lớp CSS:

```
@ Html.Label ("FirstName", "First Name:", {@ class = "caption"} mới)
```

Biểu tượng @ () cho Razor biết đây là phần bắt đầu của mã. Hai tham số tiếp theo ("FirstName" và "First Name:") là các chuỗi, vì vậy IntelliSense không thể giúp được. Đôi số cuối cùng:

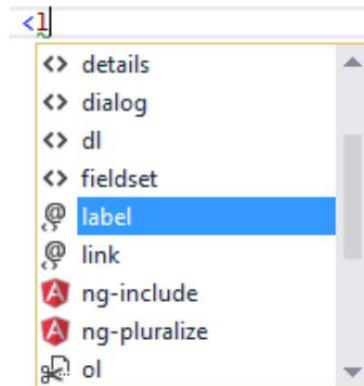
```
mới {@ class = "caption"}
```

Là một đối tượng ẩn danh dùng để đại diện cho các thuộc tính. Vì class là một từ khóa dành riêng trong C #, bạn sử dụng ký hiệu @ để buộc C # phải diễn giải "@ class =" là một ký hiệu (tên thuộc tính). Đối với một nhà thiết kế giao diện người dùng (một người quen thuộc với HTML / CSS / JavaScript và các công nghệ máy khách khác nhưng không quen thuộc với C # và Razor), hầu hết mọi thứ đều là của nước ngoài. Toàn bộ dòng phải được tạo tác giả mà không cần sự trợ giúp từ IntelliSense.

Sử dụng LabelTagHelper, đánh dấu tương tự có thể được viết như sau:

```
<label class="caption" asp-for="FirstName"></label>
```

Với phiên bản Trình trợ giúp thẻ, ngay sau khi bạn nhập <l vào trình chỉnh sửa Visual Studio, IntelliSense sẽ hiển thị các phần tử phù hợp:



The label element represents a caption in a user interface. The caption can be associated with a specific form control, known as the label element's labeled control, either using the for attribute, or by putting the form control inside the label element itself.

IntelliSense giúp bạn viết toàn bộ dòng. LabelTagHelper cũng mặc định đặt nội dung của giá trị thuộc tính asp-for ("FirstName") thành "First Name"; Nó chuyển đổi các thuộc tính có vỏ lác đà thành một câu bao gồm tên thuộc tính với một khoảng trắng trong đó mỗi chữ cái viết hoa mới xuất hiện. Trong đánh dấu sau:

```
<label class="caption" asp-for="FirstName"></label>
```

tạo ra:

```
<label class = "caption" for = "FirstName"> Tên </label>
```

Nội dung dựa trên câu lạc đà không được sử dụng nếu bạn thêm nội dung vào <label>. Ví dụ:

```
<label class="caption" asp-for="FirstName">Name First</label>
```

tạo ra:

```
<label class = "caption" for = "FirstName"> Đặt tên </label>
```

Hình ảnh mã sau đây cho thấy phần Biểu mẫu của chế độ xem Views / Account / Register.cshtml Razor được tạo từ mã ASP.NET 4.5.x MVC kế thừa được bao gồm trong Visual Studio 2015.

```
@using (Html.BeginForm("Register", "Account", FormMethod.Post, new { @class = "form-horizo
{
    @Html.AntiForgeryToken()
    <h4>Create a new account.</h4>
    <hr />
    @Html.ValidationSummary("", new { @class = "text-danger" })
    <div class="form-group">
        @Html.LabelFor(m => m.Email, new { @class = "col-md-2 control-label" })
        <div class="col-md-10">
            @Html.TextBoxFor(m => m.Email, new { @class = "form-control" })
        </div>
    </div>
    <div class="form-group">
        @Html.LabelFor(m => m.Password, new { @class = "col-md-2 control-label" })
        <div class="col-md-10">
            @Html.PasswordFor(m => m.Password, new { @class = "form-control" })
        </div>
    </div>
    <div class="form-group">
        @Html.LabelFor(m => m.ConfirmPassword, new { @class = "col-md-2 control-label" })
        <div class="col-md-10">
            @Html.PasswordFor(m => m.ConfirmPassword, new { @class = "form-control" })
        </div>
    </div>
    <div class="form-group">
        <div class="col-md-offset-2 col-md-10">
            <input type="submit" class="btn btn-default" value="Register" />
        </div>
    </div>
}
```

Trình chỉnh sửa Visual Studio hiển thị mã C # với nền màu xám. Ví dụ: Trình trợ giúp HTML AntiForgeryToken:

```
@ Html.AntiForgeryToken()
```

được hiển thị với nền màu xám. Hầu hết đánh dấu trong dạng xem Đăng ký là C #. So sánh cách tiếp cận đó với cách tiếp cận tương đương bằng Công cụ trợ giúp thẻ:

```

<form asp-controller="Account" asp-action="Register" method="post" class="form-hori
    <h4>Create a new account.</h4>
    <hr />
    <div asp-validation-summary="ValidationSummary.All" class="text-danger"></div>
    <div class="form-group">
        <label asp-for="Email" class="col-md-2 control-label"></label>
        <div class="col-md-10">
            <input asp-for="Email" class="form-control" />
            <span asp-validation-for="Email" class="text-danger"></span>
        </div>
    </div>
    <div class="form-group">
        <label asp-for="Password" class="col-md-2 control-label"></label>
        <div class="col-md-10">
            <input asp-for="Password" class="form-control" />
            <span asp-validation-for="Password" class="text-danger"></span>
        </div>
    </div>
    <div class="form-group">
        <label asp-for="ConfirmPassword" class="col-md-2 control-label"></label>
        <div class="col-md-10">
            <input asp-for="ConfirmPassword" class="form-control" />
            <span asp-validation-for="ConfirmPassword" class="text-danger"></span>
        </div>
    </div>
    <div class="form-group">
        <div class="col-md-offset-2 col-md-10">
            <button type="submit" class="btn btn-default">Register</button>
        </div>
    </div>
</form>

```

Đánh dấu rõ ràng hơn và dễ đọc, chỉnh sửa và bảo trì hơn nhiều so với cách tiếp cận Trình trợ giúp HTML. Mã C # được giảm xuống mức tối thiểu mà máy chủ cần biết. Trình chỉnh sửa Visual Studio hiển thị đánh dấu được nhắm mục tiêu bởi Trình trợ giúp thẻ bằng một phông chữ đặc biệt.

Xem xét nhóm Email:

```

<div class = "form-group">
    <label asp-for = "Email" class = "col-md-2 control-label"> </label>
    <div class = "col-md-10">
        <input asp-for = "Email" class = "form-control" /> <span
        asp-validation-for = "Email" class = "text-risk"> </div> </div>

```

Mỗi thuộc tính “asp-” có giá trị là “Email”, nhưng “Email” không phải là một chuỗi. Trong ngữ cảnh này, “Email” là thuộc tính biểu thức mô hình C # cho RegisterViewModel.

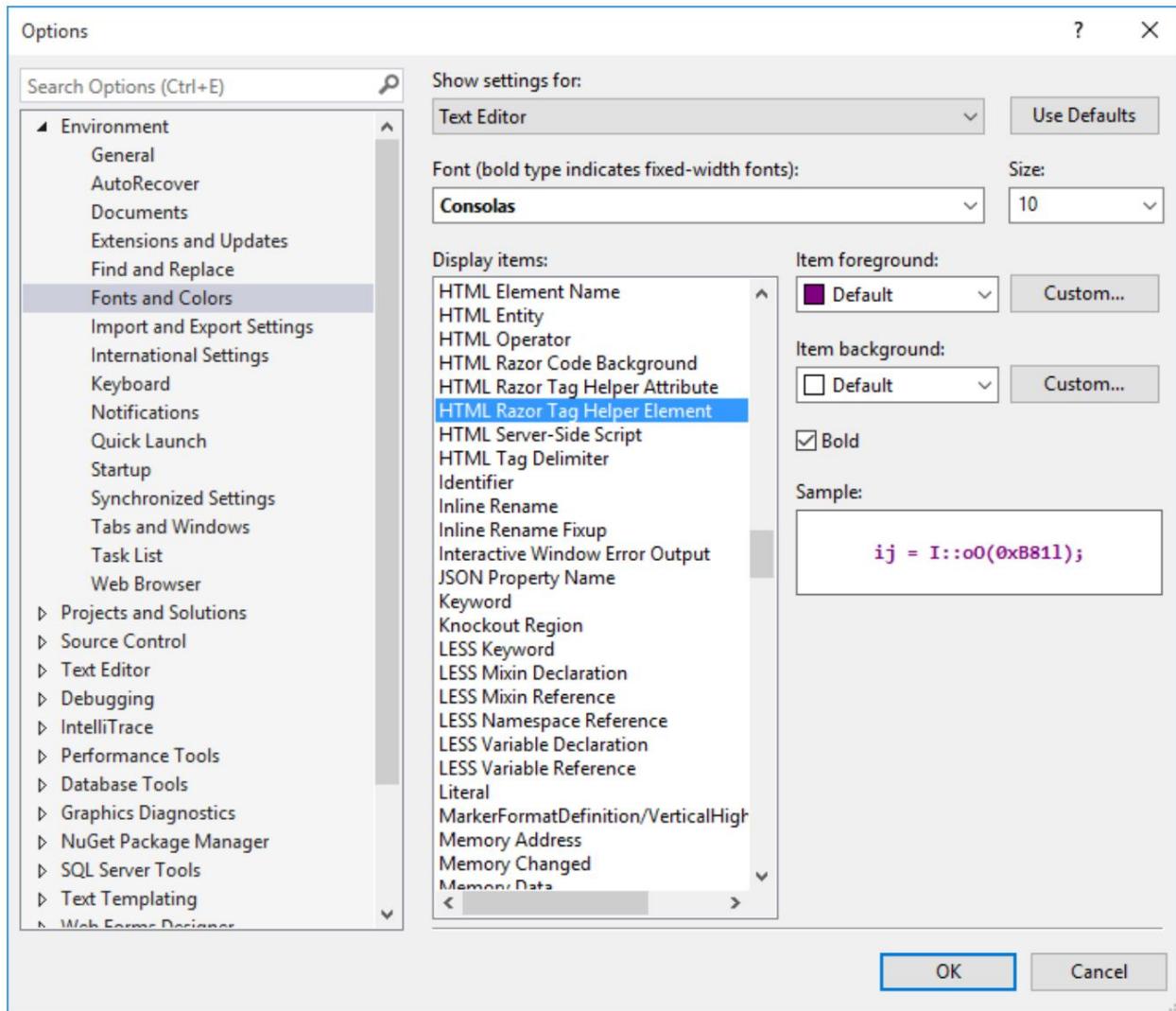
Trình chỉnh sửa Visual Studio giúp bạn viết tắt cả đánh dấu trong cách tiếp cận Trình trợ giúp thẻ của biểu mẫu đăng ký, trong khi Visual Studio không cung cấp trợ giúp cho hầu hết mã trong cách tiếp cận Trình trợ giúp HTML. Hỗ trợ của IntelliSense cho Trình trợ giúp thẻ đi sâu vào cách làm việc với Trình trợ giúp thẻ trong trình chỉnh sửa Visual Studio.

Tag Helpers so với Web Server Controls

- Trình trợ giúp thẻ không sở hữu phần tử mà chúng được liên kết với; họ chỉ đơn giản là tham gia vào việc kết xuất yếu tố và nội dung. Các điều khiển ASP.NET `Web Server` được khai báo và gọi trên một trang.
- Điều khiển Máy chủ Web có vòng đời không tầm thường có thẻ gây khó khăn cho việc phát triển và gỡ lỗi.
- Điều khiển Máy chủ Web cho phép bạn thêm chức năng vào các phần tử Mô hình Đối tượng Tài liệu (DOM) máy khách bằng cách sử dụng một điều khiển máy khách. Trình trợ giúp thẻ không có DOM.
- Điều khiển Máy chủ Web bao gồm phát hiện trình duyệt tự động. Người trợ giúp thẻ không có kiến thức về trình duyệt.
- Nhiều Trình trợ giúp thẻ có thể hoạt động trên cùng một phần tử (xem phần [Tránh xung đột Trình trợ giúp thẻ](#)) trong khi bạn thường không thể soạn các điều khiển Máy chủ Web.
- Trình trợ giúp thẻ có thể sửa đổi thẻ và nội dung của các phần tử HTML mà chúng có phạm vi sử dụng, nhưng không trực tiếp sửa đổi bất kỳ thứ gì khác trên một trang. Các điều khiển Máy chủ Web có phạm vi ít cụ thể hơn và có thể thực hiện các hành động ảnh hưởng đến các phần khác trên trang của bạn; tạo ra những tác dụng phụ ngoài ý muốn.
- Điều khiển Máy chủ Web sử dụng bộ chuyển đổi kiểu để chuyển đổi các chuỗi thành các đối tượng. Với Trình trợ giúp thẻ, bạn làm việc nguyên bản trong C#, vì vậy bạn không cần thực hiện chuyển đổi kiểu.
- Điều khiển Máy chủ Web sử dụng `System.ComponentModel` để thực hiện hành vi thời gian chạy và thời gian thiết kế của các nhà điều hành và kiểm soát của com. `System.ComponentModel` bao gồm các lớp cơ sở và giao diện để triển khai các thuộc tính và bộ chuyển đổi kiểu, liên kết với các nguồn dữ liệu và các thành phần cấp phép. Ngược lại điều đó với Trình trợ giúp thẻ, thường xuất phát từ `TagHelper` và lớp cơ sở `TagHelper` chỉ hiển thị hai phương thức, `Process` và `ProcessAsync`.

Tùy chỉnh phông chữ phần tử Trình trợ giúp thẻ

Bạn có thể tùy chỉnh phông chữ và màu sắc từ Công cụ > Tùy chọn > Môi trường > Phông chữ và Màu sắc:



Tài nguyên bổ sung

- TagHelperSamples trên GitHub chứa các mẫu Trình trợ giúp thẻ để làm việc với Bootstrap.
- Kênh 9 video về Trình trợ giúp thẻ nâng cao. Đây là một video tuyệt vời về các tính năng nâng cao hơn. Đó là một vài phiên bản đã lỗi thời nhưng các bình luận chứa danh sách các thay đổi đối với phiên bản hiện tại. Mã cập nhật có thể được tìm thấy ở đây.

5.4.2 Trình trợ giúp thẻ tác giả

Bởi Rick Anderson

- Bắt đầu với Trình trợ giúp thẻ
- Khởi động Trình trợ giúp thẻ email
- Trình trợ giúp thẻ email đang hoạt động
- Trình trợ giúp thẻ in đậm
- Trình trợ giúp thẻ thông tin trang web

- Trình trợ giúp thẻ điều kiện
- Tránh xung đột Trình trợ giúp thẻ
- Kiểm tra và truy xuất nội dung trẻ em
- Kết thúc và các bước tiếp theo
- Tài nguyên bổ sung

Bạn có thể duyệt mã nguồn cho ứng dụng mẫu được sử dụng trong tài liệu này trên GitHub.

Bắt đầu với Trình trợ giúp thẻ

Hướng dẫn này cung cấp phần giới thiệu về lập trình Trình trợ giúp thẻ. [Giới thiệu về Công cụ trợ giúp thẻ](#) mô tả những lợi ích mà Công cụ hỗ trợ thẻ cung cấp.

Trình trợ giúp thẻ là bất kỳ lớp nào triển khai giao diện `ITagHelper`. Tuy nhiên, khi bạn tạo ra một trình trợ giúp thẻ, bạn thường bắt nguồn từ `TagHelper`, làm như vậy sẽ cung cấp cho bạn quyền truy cập vào phương thức `Process`. Chúng tôi sẽ giới thiệu các phương thức và thuộc tính `TagHelper` khi chúng tôi sử dụng chúng trong hướng dẫn này.

1. Tạo một dự án ASP.NET MVC 6 mới có tên là `AuthoringTagHelpers`. Bạn sẽ không cần xác thực cho việc này dự án.
2. Tạo một thư mục để chứa các Trình trợ giúp Thẻ được gọi là `TagHelpers`. Thư mục `TagHelpers` không bắt buộc, nhưng nó là một quy ước hợp lý. Bởi giờ chúng ta hãy bắt đầu viết một số trợ giúp thẻ đơn giản.

Bắt đầu Trình trợ giúp thẻ email

Trong phần này, chúng tôi sẽ viết một trình trợ giúp thẻ cập nhật thẻ email. Ví dụ:

```
<email> Hỗ trợ </email>
```

Máy chủ sẽ sử dụng trình trợ giúp thẻ email của chúng tôi để chuyển đổi đánh dấu đó thành như sau:

```
<a href="mailto:Support@contoso.com"> Support@contoso.com </a>
```

Đó là, một thẻ liên kết làm cho nó trở thành một liên kết email. Bạn có thể muốn làm điều này nếu bạn đang viết một công cụ blog và cần nó để gửi email tiếp thị, hỗ trợ và các địa chỉ liên hệ khác, tất cả đến cùng một miền.

1. Thêm lớp `EmailTagHelper` sau vào thư mục `TagHelpers`.

```
sử dụng Microsoft.AspNet.Razor.Runtime.TagHelpers; sử dụng
System.Threading.Tasks;

không gian tên AuthoringTagHelpers.TagHelpers {

    lớp công khai EmailTagHelper : TagHelper {

        ghi đè công khai void Process (ngũ cảnh TagHelperContext, đầu ra TagHelperOutput) {

            output.TagName = "a"; // Thay thế thẻ <email> bằng thẻ <a>
        }
    }
}
```

Ghi chú:

- Trình trợ giúp thẻ sử dụng quy ước đặt tên nhầm mục tiêu đến các phần tử của tên lớp gốc (trừ phần `TagHelper` của tên lớp). Trong ví dụ này, tên gốc của `EmailTagHelper` là `email`, vì vậy thẻ `<email>` sẽ được nhầm mục tiêu. Quy ước đặt tên này sẽ phù hợp với hầu hết các trình trợ giúp thẻ, sau này tôi sẽ trình bày cách ghi đè nó.

- Lớp EmailTagHelper bắt nguồn từ TagHelper. Lớp TagHelper cung cấp các phương thức phong phú và các thuộc tính mà chúng tôi sẽ kiểm tra trong hướng dẫn này.
- Phương thức Process bị ghi đè kiểm soát những gì trình trợ giúp thẻ thực hiện khi được thực thi. Lớp TagHelper cũng cung cấp phiên bản không đồng bộ (ProcessAsync) với các tham số tương tự.
- Tham số ngữ cảnh của Process (và ProcessAsync) chứa thông tin liên quan đến việc thực thi thẻ HTML hiện tại.
- Tham số đầu ra cho Process (và ProcessAsync) chứa đại diện phần tử HTML trạng thái của nguồn gốc được sử dụng để tạo thẻ và nội dung HTML.
- Tên lớp của chúng ta có hậu tố là TagHelper, không bắt buộc, nhưng nó được coi là quy ước thực tiễn tốt nhất. Bạn có thể khai báo lớp là:

```
lớp công khai Email : TagHelper
```

2. Để cung cấp lớp EmailTagHelper cho tất cả các chế độ xem Razor của chúng tôi, chúng tôi sẽ thêm chỉ thị addTagHelper vào tệp Views / _ViewImports.cshtml:

```
@using AuthoringTagHelpers
@addTagHelper "*", Microsoft.AspNet.Mvc.TagHelpers" @addTagHelper "*",
AuthoringTagHelpers"
```

Đoạn mã trên sử dụng cú pháp ký tự đại diện để chỉ định tất cả các trình trợ giúp thẻ trong hợp ngữ của chúng tôi sẽ khả dụng. Chuỗi đầu tiên sau @addTagHelper chỉ định trình trợ giúp thẻ để tài (chúng tôi đang sử dụng "*" cho tất cả các trình trợ giúp thẻ) và chuỗi thứ hai "AuthoringTagHelpers" chỉ định lắp ráp mà trình trợ giúp thẻ đang ở trong đó. Ngoài ra, hãy lưu ý rằng dòng thứ hai đưa vào trình trợ giúp thẻ ASP.NET 5 MVC 6 sử dụng cú pháp ký tự đại diện (những trình trợ giúp đó được thảo luận trong [Giới thiệu về Trình trợ giúp thẻ](#).) Đó là chỉ thị @addTagHelper làm cho trình trợ giúp thẻ có sẵn cho chế độ xem Razor. Ngoài ra, bạn có thể cung cấp tên đủ điều kiện (FQN) của trình trợ giúp thẻ như được hiển thị bên dưới:

```
@using AuthoringTagHelpers
@addTagHelper "*", Microsoft.AspNet.Mvc.TagHelpers" @addTagHelper
"AuthoringTagHelpers.TagHelpers.EmailTagHelper, AuthoringTagHelpers"
```

Để thêm trình trợ giúp ~~một~~ dạng xem bằng FQN, trước tiên bạn thêm FQN (AuthoringTagHelpers.TagHelpers.EmailTagHelper), sau đó là tên hợp ngữ (Authoring TagHelpers). Hầu hết các nhà phát triển sẽ thích sử dụng cú pháp ký tự đại diện. [Giới thiệu về Trình trợ giúp thẻ](#) đi vào chi tiết về cú pháp thêm, xóa, phân cấp và ký tự đại diện của trình trợ giúp thẻ.

3. Cập nhật đánh dấu trong tệp Chế độ xem / Trang chủ / Liên hệ.cshtml với những thay đổi sau:

```
@ {
    ViewData ["Title"] = "Liên hệ";
}

} <h2> @ViewData ["Title"]. </h2> <h3>
@ViewData ["Message"] </h3>

<địa chỉ>
    Một cách của Microsoft <br />
    Redmond, WA 98052 <br /> <abbr
        title = "Phone"> P: </abbr> 425.555.0100
    </address>

<địa chỉ>
    <strong> Hồ trợ: </strong> <email> Hồ trợ </email> <br /> <strong> Tiếp
    thị: </strong> <email> Tiếp thị </email>
</address>
```

4. Chạy ứng dụng và sử dụng trình duyệt yêu thích của bạn để xem nguồn HTML để bạn có thể xác minh rằng các thẻ email được thay thế bằng dấu liên kết (Ví dụ: <a> Hỗ trợ). Hỗ trợ và Tiếp thị được hiển thị dưới dạng liên kết, nhưng chúng không có thuộc tính href để làm cho chúng hoạt động. Chúng tôi sẽ khắc phục điều đó trong phần tiếp theo.

Lưu ý: Giống như các thẻ và thuộc tính HTML, các thẻ, tên lớp và thuộc tính trong Razor và C# không phân biệt chữ hoa chữ thường.

Trình trợ giúp thẻ email đang hoạt động

Trong phần này, chúng tôi sẽ cập nhật EmailTagHelper để nó tạo thẻ liên kết hợp lệ cho email. Chúng tôi sẽ cập nhật trình trợ giúp thẻ của mình để lấy thông tin từ chế độ xem Razor (ở dạng thuộc tính mail-to) và sử dụng nó trong việc tạo liên kết.

Cập nhật lớp EmailTagHelper như sau:

```
lớp công khai EmailTagHelper : TagHelper {

    private const string EmailDomain = "contoso.com";

    // Có thẻ được chuyển qua <email mail-to = "..."/>.
    // Chữ hoa Pascal được dịch thành chữ thường-kebab-case. chuỗi công khai MailTo
    { get; bộ; }

    ghi đè công khai void Process (ngữ cảnh TagHelperContext, đầu ra TagHelperOutput) {

        output.TagName = "a"; // Thay thế thẻ <email> bằng thẻ <a>

        var address = MailTo + "@" + EmailDomain; output.Attributes
        ["href"] = "mailto:" + địa chỉ; output.Content.SetContent (địa chỉ);

    }
}
```

Ghi chú:

- Tên thuộc tính và lớp dựa trên Pascal cho trình trợ giúp thẻ được dịch sang chữ hoa kebab dưới của chúng. Do đó, để sử dụng thuộc tính MailTo, bạn sẽ sử dụng <email mail-to = "value" /> tương đương.
- Dòng cuối cùng đặt nội dung đã hoàn thành cho trình trợ giúp thẻ chức năng tối thiểu của chúng tôi.
- Dòng sau hiển thị cú pháp để thêm thuộc tính:

```
ghi đè công khai void Process (ngữ cảnh TagHelperContext, đầu ra TagHelperOutput) {

    output.TagName = "a"; // Thay thế thẻ <email> bằng thẻ <a>

    var address = MailTo + "@" + EmailDomain; output.Attributes
    ["href"] = "mailto:" + địa chỉ; output.Content.SetContent (địa chỉ);

}
```

Cách tiếp cận đó phù hợp với thuộc tính "href" miễn là nó hiện không tồn tại trong bộ sưu tập thuộc tính. Bạn cũng có thể sử dụng phương thức output.Attributes.Add để thêm thuộc tính trình trợ giúp thẻ vào cuối tập hợp các thuộc tính thẻ.

3. Cập nhật đánh dấu trong tệp Chế độ xem / Trang chủ / Liên hệ.cshtml với những thay đổi sau:

```
@ {
    ViewData ["Title"] = "Liên hệ";
}

} <h2> @ViewData ["Tiêu đề"]. </h2>
```

```
<h3> @ViewData["Tin nhắn"] </h3>

<địa chỉ>
    Một cách của Microsoft <br />
    Redmond, WA 98052-6399 <br /> <abbr title
    = "Phone"> P: </abbr>
    425.555.0100
</address>

<địa chỉ>
    <strong> Hỗ trợ: </strong> <email mail-to = "Support"> </email> <br /> <strong> Tiếp thị: </
    strong> <email mail-to = "Tiếp thị"> </email>
</address>
```

4. Chạy ứng dụng và xác minh rằng nó tạo ra các liên kết chính xác.

Lưu ý: Nếu bạn viết thẻ email tự đóng (<email mail-to = "Rick" />), đầu ra cuối cùng cũng sẽ tự đóng. Để kích hoạt khả năng viết thẻ chỉ với thẻ bắt đầu (<email mail-to = "Rick">), bạn phải trang trí lớp bằng những thứ sau:

```
[TargetElement ("email", TagStructure = TagStructure.WithoutEndTag)]
```

Với trình trợ giúp thẻ email tự đóng, đầu ra sẽ là . Các thẻ liên kết tự đóng không phải là HTML hợp lệ, vì vậy bạn sẽ không muốn tạo một thẻ, nhưng bạn có thể muốn tạo một trình trợ giúp thẻ tự đóng. Trình trợ giúp thẻ đặt loại thuộc tính TagMode sau khi đọc thẻ.

Trình trợ giúp email không đồng bộ

Trong phần này, chúng tôi sẽ viết một trình trợ giúp email không đồng bộ.

1. Thay thế lớp EmailTagHelper bằng mã sau:

```
lớp công khai EmailTagHelper : TagHelper {

    private const string EmailDomain = "contoso.com"; ghi đè công khai không
    đồng bộ Task ProcessAsync (ngõ cảnh TagHelperContext, đầu ra TagHelperOutput) {

        = "a"; var content = await context.GetChildContentAsync(); bàngert target output.TagName
        content.GetContent () + "@" + EmailDomain; output.Attributes ["href"] = "mailto:" +
        target; output.Content.SetContent (target);

    }
}
```

Ghi chú:

- Phiên bản này sử dụng phương thức ProcessAsync không đồng bộ. GetChildContentAsync không đồng bộ trả về một Tác vụ có chứa TagHelperContent.
- Chúng tôi sử dụng tham số đầu ra để lấy nội dung của phần tử HTML.

2. Thực hiện thay đổi sau đây với tệp Views / Home / Contact.cshtml để trình trợ giúp thẻ có thể nhận được email mục tiêu.

```
@ {
    ViewData ["Title"] = "Liên hệ";

} <h2> @ViewData ["Title"]. </h2> <h3>
@ViewData ["Message"] </h3>
```

```
<địa chỉ>
    Một cách của Microsoft <br />
    Redmond, WA 98052 <br /> <abbr
        title = "Phone"> P: </abbr> 425.555.0100 </
        address>

<địa chỉ>
    <strong> Hỗ trợ: </strong> <email> Hỗ trợ </email> <br /> <strong> Tiếp thị: </
    strong> <email> Tiếp thị </email>
</address>
```

- Chạy ứng dụng và xác minh rằng nó tạo ra các liên kết email hợp lệ.

Trình trợ giúp thẻ in đậm

- Thêm lớp BoldTagHelper sau vào thư mục TagHelpers.

```
sử dụng Microsoft.AspNetCore.Razor.Runtime.TagHelpers;

không gian tên AuthoringTagHelpers.TagHelpers {

    [TargetElement (Attributes = "bold")] public class
    BoldTagHelper : TagHelper {

        ghi đè công khai void Process (ngữ cảnh TagHelperContext, đầu ra TagHelperOutput) {

            output.Attributes.RemoveAll ("in đậm");
            output.PreContent.SetHtmlContent ("<strong>");
            output.PostContent.SetHtmlContent ("</strong>");
        }
    }
}
```

Ghi chú:

- Thuộc tính [HtmlTargetElement] chuyển một tham số thuộc tính chỉ định rằng bất kỳ phần tử HTML nào có chứa thuộc tính HTML có tên "bold" sẽ khớp và phương thức ghi đè quy trình trong lớp sẽ chạy. Trong mẫu của chúng tôi, phương pháp Process loại bỏ thuộc tính "bold" và bao quanh đánh dấu chứa bằng ` `.
- Vì không muốn thay thế nội dung thẻ hiện có, chúng ta phải viết thẻ `` mở bằng phương thức `PreContent.SetHtmlContent` và thẻ đóng `` bằng phương thức `PostContent.SetHtmlContent`.

- Sửa đổi dạng xem About.cshtml để chứa giá trị thuộc tính in đậm. Mã hoàn thành được hiển thị bên dưới.

```
@ {
    ViewData ["Title"] = "Giới thiệu";

} <h2> @ViewData ["Title"]. </h2> <h3>
@ViewData ["Message"] </h3>

<p bold> Sử dụng khu vực này để cung cấp thêm thông tin. </p>
<bold> Cái này có in đậm không? </bold>
```

3. Chạy ứng dụng. Bạn có thể sử dụng trình duyệt yêu thích của mình để kiểm tra nguồn và xác minh rằng đánh dấu đã thay đổi như đã hứa.

Thuộc tính [HtmlTargetElement] ở trên chỉ nhắm mục tiêu đánh dấu HTML cung cấp tên thuộc tính là "bold".

Phần tử <bold> không được trình trợ giúp thẻ sửa đổi.

4. Nhận xét dòng thuộc tính [HtmlTargetElement] và nó sẽ mặc định nhắm mục tiêu các thẻ <bold>, tức là đánh dấu HTML của biểu mẫu <bold>. Hãy nhớ rằng, quy ước đặt tên mặc định sẽ khớp tên lớp BoldTagHelper với các thẻ <bold>.

5. Chạy ứng dụng và xác minh rằng thẻ <bold> được xử lý bởi trình trợ giúp thẻ.

Việc trang trí một lớp với nhiều thuộc tính [HtmlTargetElement] dẫn đến kết quả là OR của các mục tiêu. Ví dụ: sử dụng mã bên dưới, thẻ in đậm hoặc thuộc tính in đậm sẽ khớp.

```
[TargetElement ("in đậm")]
[TargetElement (Thuộc tính = "bold")]
```

Khi nhiều thuộc tính được thêm vào cùng một câu lệnh, thời gian chạy sẽ coi chúng như một logic-AND. Ví dụ: trong đoạn mã dưới đây, một phần tử HTML phải được đặt tên là "bold" với một thuộc tính có tên là "bold" (<bold bold />) để khớp.

```
[HtmlTargetElement ("bold", Attributes = "bold")]
```

Để có ví dụ điển hình về thanh tiến trình bootstrap nhắm mục tiêu thẻ và thuộc tính, hãy xem [Tạo trình trợ giúp thẻ MVC 6 tùy chỉnh](#).

Bạn cũng có thể sử dụng [HtmlTargetElement] để thay đổi tên của phần tử được nhắm mục tiêu. Ví dụ: nếu bạn muốn BoldTagHelper nhắm mục tiêu các thẻ <MyBold>, bạn sẽ sử dụng thuộc tính sau:

```
[HtmlTargetElement ("MyBold")]
```

Thông tin trang web Trình trợ giúp thẻ

1. Thêm thư mục Mô hình.
2. Thêm lớp WebsiteContext sau vào thư mục Mô hình:

```
sử dụng Hệ thống;
không gian tên AuthoringTagHelpers.Models {

    public class WebsiteContext {

        Phiên bản công khai Phiên bản { get; bộ; } public int
        CopyrightYear { get; bộ; } public bool Đã phê duyệt { get;
        bộ; } public int TagsToShow { get; bộ; }

    }
}
```

3. Thêm lớp WebsiteInformationTagHelper sau vào thư mục TagHelpers.

```
sử dụng Hệ thống;
sử dụng Microsoft.AspNet.Razor.Runtime.TagHelpers; sử dụng
AuthoringTagHelpers.Models;

không gian tên AuthoringTagHelpers.TagHelpers {

    lớp công khai WebsiteInformationTagHelper: TagHelper {
```

```

public WebsiteContext Info {get; set; }

ghi đè công khai void Process (ngữ cảnh TagHelperContext, đầu ra TagHelperOutput) {

    output.TagName = "phần";
    output.Content.SetHtmlContent (
$ @ "<ul> <li> <strong> Phiên bản: </strong> {Info.Version} </li> <li> <strong> Năm
bản quyền: </strong> {Info.CopyrightYear} </li> <li> <strong> Được chấp thuận: </strong>
{Info.Approved} </li> <li> <strong> Số lượng thẻ sẽ hiển thị: </strong> {Info.TagsToShow} </li>
</ul> " );
    output.TagMode = TagMode.StartTagAndEndTag;
}
}
}

```

Ghi chú:

- Như đã đề cập trước đây, trình trợ giúp thẻ dịch tên và thuộc tính lớp C # dựa trên Pascal cho trình trợ giúp thẻ thành **chữ hoa kebab thấp hơn**. Do đó, để sử dụng WebsiteInformationTagHelper trong Razor, bạn sẽ viết `<website-information />`.
- Chúng tôi không xác định rõ ràng phần tử mục tiêu với thuộc tính [HtmlTargetElement], do đó, thông tin trang web mặc định sẽ được nhầm mục tiêu. Nếu bạn đã áp dụng thuộc tính sau (lưu ý rằng nó không phải là chữ hoa kebab mà khớp với tên lớp):

`[HtmlTargetElement ("Thông tin trang web")]`

Thẻ viết hoa kebab dưới `<website-information />` sẽ không khớp.

Nếu bạn muốn sử dụng

Thuộc tính [HtmlTargetElement], bạn sẽ sử dụng trường hợp kebab như hình dưới đây:

`[HtmlTargetElement ("Trang web-Thông tin")]`

- Các phần tử tự đóng không có nội dung. Đối với ví dụ này, đánh dấu Razor sẽ sử dụng thẻ tự đóng, nhưng trình trợ giúp thẻ sẽ tạo một **phần** phần tử (không tự đóng và chúng tôi đang viết nội dung bên trong phần tử phần). Do đó, chúng ta cần đặt TagMode thành StartTagAndEndTag để ghi dấu ra.
Ngoài ra, bạn có thể nhận xét về cách đặt TagMode cho dòng và viết đánh dấu bằng thẻ đóng. (Đánh dấu ví dụ được cung cấp sau trong hướng dẫn này.)
- \$ (ký hiệu đô la) trong dòng sau sử dụng một **chuỗi nội suy**:

`$ @ " Phiên bản: {Info.Version} `

5. Thêm đánh dấu sau vào dạng xem About.cshtml. Đánh dấu được đánh dấu hiển thị thông tin trang web.

```

@using AuthoringTagHelpers.Models @ {

    ViewData ["Title"] = "Giới thiệu";

} <h2> @ViewData ["Title"] </h2> <h3>
@ViewData ["Message"] </h3>

<p bold> Sử dụng khu vực này để cung cấp thêm thông tin. </p>

<b> Cái này có in đậm không? </b>

<h3> Thông tin trang web </h3>
<website-information info = "new WebsiteContext {
    Phiên bản = Phiên bản mới (1, 3),
    CopyrightYear = 1790,
}
}
}

```

```
Đã phê duyệt = true,
TagsToShow = 131} " />
```

Lưu ý: Trong đánh dấu Razor được hiển thị bên dưới:

```
<website-information info = "new WebsiteContext {
Phiên bản = Phiên bản mới (1, 3),
CopyrightYear = 1790,
Đã phê duyệt = true,
TagsToShow = 131} " />
```

Razor biết thuộc tính thông tin là một lớp, không phải một chuỗi và bạn muốn viết mã C#. Bất kỳ thuộc tính trợ giúp thẻ không phải chuỗi nào cũng phải được viết mà không có ký tự @.

6. Chạy ứng dụng và điều hướng đến chế độ xem Giới thiệu để xem thông tin trang web.

Ghi chú:

- Bạn có thể sử dụng đánh dấu sau với thẻ đóng và xóa dòng bằng TagMode.StartTagAndEndTag trong trình trợ giúp thẻ:

```
<website-information info = "new WebsiteContext {
Phiên bản = Phiên bản mới (1, 3),
CopyrightYear = 1790,
Đã phê duyệt = true,
TagsToShow = 131} " >
</website-information>
```

Trình trợ giúp thẻ điều kiện

Trình trợ giúp thẻ điều kiện hiển thị đầu ra khi được chuyển một giá trị đúng.

1. Thêm lớp ConditionTagHelper sau vào thư mục TagHelpers.

```
sử dụng Microsoft.AspNet.Razor.Runtime.TagHelpers;

không gian tên AuthoringTagHelpers.TagHelpers {

    [TargetElement (Attributes = nameof (Condition))] public class
    ConditionTagHelper : TagHelper {

        public bool Điều kiện { get; bộ; }

        ghi đè công khai void Process (ngữ cảnh TagHelperContext, đầu ra TagHelperOutput) {

            nếu (! Điều kiện)
            {
                output.SuppressOutput ();
            }
        }
    }
}
```

2. Thay thế nội dung của tệp Views / Home / Index.cshtml bằng đánh dấu sau:

```
@using AuthoringTagHelpers.Models @model
WebsiteContext

@ {
```

```

        ViewData["Title"] = "Trang chủ";
    }

<div>
    <h3> Thông tin về trang web của chúng tôi (đã lỗi thời): </h3> <Website-
    Information info = Model /> <div condition = "Model.Approved">

        <p>
            Trang web này đã được phê duyệt <strong surround = "em"> @ Model.Approved </strong> .
            Truy cập www.contoso.com để biết thêm thông tin.
        </p> </
    div> </div>

```

3. Thay thế phương thức Chỉ mục trong Bộ điều khiển gia đình bằng mã sau:

```

chỉ số IActionResult công khai (bool đã được chấp thuận = false) {

    return View ( WebsiteContext mới
    {
        Approved = được chấp thuận,
        CopyrightYear = 2015,
        Phiên bản = Phiên bản mới (1, 3, 3, 7),
        TagsToShow = 20 );
    }
}

```

4. Chạy ứng dụng và duyệt đến trang chủ. Đánh dấu trong div có điều kiện sẽ không được hiển thị. Nối chuỗi truy vấn? Approved = true vào URL (ví dụ : <http://localhost:1235/Home/Index?Approved=true>).
Đã phê duyệt được đặt thành true và đánh dấu có điều kiện sẽ được hiển thị.

Lưu ý: Chúng tôi sử dụng tên của toán tử để chỉ định thuộc tính để nhắm mục tiêu thay vì chỉ định một chuỗi như chúng ta đã làm với trình trợ giúp thẻ in đậm:

```

[TargetElement (Thuộc tính = nameof (Điều kiện))]
// [TargetElement (Attributes = "condition")] public class
ConditionTagHelper : TagHelper {

    public bool Điều kiện { get; bộ; }

    ghi đè công khai void Process (ngữ cảnh TagHelperContext, dấu ra TagHelperOutput) {

        if (! Điều kiện) {

            output.SuppressOutput ();
        }
    }
}

```

Tên của nhà điều hành sẽ bảo vệ nó nếu nó được cấu trúc lại (chúng tôi có thể muốn đổi tên thành RedCondition).

Tránh xung đột Trình trợ giúp thẻ

Trong phần này, chúng tôi sẽ viết một cặp trợ giúp thẻ liên kết tự động. Đầu tiên sẽ thay thế đánh dấu có chứa một URL bắt đầu bằng HTTP thành một thẻ liên kết HTML có chứa cùng một URL (và do đó tạo ra một liên kết đến URL). Thao tác thứ hai sẽ làm tương tự đối với URL bắt đầu bằng WWW.

Vì hai trình trợ giúp này có liên quan chặt chẽ với nhau và chúng tôi có thể cấu trúc lại chúng trong tương lai, nên chúng tôi sẽ giữ chúng trong cùng một tệp.

1. Thêm lớp AutoLinker sau vào thư mục TagHelpers.

```
[TargetElement ("p")] lớp công
khai AutoLinkerHttpTagHelper: TagHelper {

    ghi đè công khai không đồng bộ Task ProcessAsync (ngữ cảnh TagHelperContext, đầu ra TagHelperOutput) {

        var childContent = await output.GetChildContentAsync (); // Tìm Url trong nội dung và
        thay thế chúng bằng thẻ liên kết tương đương. output.Content.SetHtmlContent (Regex.Replace (childContent.GetContent (), @
        "\ b (? : https? : // ) (\ S + ) \ b ",

        "<a target=_blank" href=\"$0\" $ 0 </a>"); // phiên bản liên kết http
    }
}
```

Lưu ý: Lớp AutoLinkerHttpTagHelper nhắm mục tiêu phần tử p và sử dụng Regex để tạo mỏ neo.

2. Thêm đánh dấu sau vào cuối tệp Ché độ xem / Trang chủ / Liên hệ.cshtml:

```
@ {
    ViewData ["Title"] = "Liên hệ";

} <h2> @ViewData ["Title"]. </h2> <h3>
@ViewData ["Message"] </h3>

<địa chỉ>
    Một cách của Microsoft <br />
    Redmond, WA 98052 <br /> <abbr title
    = "Phone"> P: </abbr> 425.555.0100

</address>

<địa chỉ>
    <strong> Hồ trợ: </strong> <email> Hồ trợ </email> <br /> <strong> Tiếp thị: </
    strong> <email> Tiếp thị </email> </address>

<p> Ghé thăm chúng tôi tại http://docs.asp.net hoặc tại www.microsoft.com </p>
```

3. Chạy ứng dụng và xác minh rằng trình trợ giúp thẻ hiển thị ký tự liên kết chính xác.

4. Cập nhật lớp AutoLinker để bao gồm AutoLinkerWwwTagHelper sẽ chuyển đổi văn bản www thành một thẻ liên kết cũng chứa văn bản www ban đầu. Mã cập nhật được đánh dấu bên dưới:

```
[TargetElement ("p")] lớp công
khai AutoLinkerHttpTagHelper: TagHelper {

    ghi đè công khai không đồng bộ Task ProcessAsync (ngữ cảnh TagHelperContext, đầu ra TagHelperOutput) {

        var childContent = await output.GetChildContentAsync (); // Tìm Url trong nội dung và
        thay thế chúng bằng thẻ liên kết tương đương. output.Content.SetHtmlContent (Regex.Replace (childContent.GetContent (), @
        "\ b (? : https? : // ) (\ S + ) \ b ",

        "<a target=_blank" href=\"$0\" $ 0 </a>"); // phiên bản liên kết http
    }
}
```

```
[TargetElement ("p")] lớp
công khai AutoLinkerWwwTagHelper: TagHelper {

    ghi đè công khai không đồng bộ Task ProcessAsync (ngữ cảnh TagHelperContext, đầu ra TagHelperOutput) {

        var childContent = await output.GetChildContentAsync (); // Tìm Url trong nội dung
        và thay thế chúng bằng thẻ liên kết tương đương. output.Content.SetHtmlContent (Regex.Replace (childContent.GetContent (),
        @ "\ b (www \.) (\ S +) \ b", "<a target = \" _ blank \ " href = \" http: / / \$ 0 \ " > $ 0 </a>")); // phiên bản www

    }
}
```

5. Chạy ứng dụng. Lưu ý rằng văn bản www được hiển thị dưới dạng một liên kết nhưng văn bản HTTP thì không. Nếu bạn đặt điểm ngắt trong cả hai lớp, bạn có thể thấy rằng lớp trình trợ giúp thẻ HTTP sẽ chạy trước. Ở phần sau của hướng dẫn, chúng ta sẽ xem cách kiểm soát thứ tự chạy của trình trợ giúp thẻ. Vấn đề là đầu ra của trình trợ giúp thẻ được lưu vào bộ nhớ cache và khi trình trợ giúp thẻ WWW chạy, nó sẽ ghi đè đầu ra được lưu trong bộ nhớ cache từ trình trợ giúp thẻ HTTP. Chúng tôi sẽ khắc phục điều đó bằng đoạn mã sau:

```
public class AutoLinkerHttpTagHelper: TagHelper {

    ghi đè công khai không đồng bộ Task ProcessAsync (ngữ cảnh TagHelperContext, đầu ra TagHelperOutput) {

        var childContent = output.Content.IsModified? output.Content.GetContent () :
        (đang chờ đầu ra.GetChildContentAsync ()). GetContent ();

        // Tìm Url trong nội dung và thay thế chúng bằng thẻ liên kết tương đương. output.Content.SetHtmlContent (Regex.Replace
        (childContent, @ "\ b (? : https? : // ) (\ S +) \ b",

                    "<a target=_blank\ " href=\"$0\ \"$0 </a>"); // phiên bản liên kết http
    }
}

[TargetElement ("p")] lớp
công khai AutoLinkerWwwTagHelper: TagHelper {

    ghi đè công khai không đồng bộ Task ProcessAsync (ngữ cảnh TagHelperContext, đầu ra TagHelperOutput) {

        var childContent = output.Content.IsModified? output.Content.GetContent () :
        (đang chờ đầu ra.GetChildContentAsync ()). GetContent ();

        // Tìm Url trong nội dung và thay thế chúng bằng thẻ liên kết tương đương. output.Content.SetHtmlContent (Regex.Replace
        (childContent, @ "\ b (www \.) (\ S +) \ b", "<a target = \" _ blank \ " href = \" http: // \$ 0 \ " > $ 0 </a>")); // //
        phiên bản www

    }
}
```

Lưu ý: Trong phiên bản đầu tiên của trình trợ giúp thẻ liên kết tự động, chúng tôi đã nhận được nội dung của mục tiêu với mã sau:

```
var childContent = await output.GetChildContentAsync ();
```

Đó là, chúng tôi gọi GetChildContentAsync bằng cách sử dụng TagHelperOutput được truyền vào phương thức ProcessAsync. Như đã đề cập trước đây, bởi vì đầu ra được lưu trong bộ nhớ cache, trình trợ giúp thẻ cuối cùng chạy sẽ thắng. Chúng tôi đã khắc phục sự cố đó với mã sau:

```
var childContent = output.Content.IsModified? output.Content.GetContent ():  
    (đang chờ đầu ra.GetChildContentAsync ()). GetContent ();
```

Đoạn mã trên sẽ kiểm tra xem nội dung đã được sửa đổi chưa và nếu có, nó sẽ lấy nội dung từ bộ đệm đầu ra.

7. Chạy ứng dụng và xác minh rằng hai liên kết hoạt động như mong đợi. Mặc dù có thể xuất hiện trình trợ giúp thẻ trình liên kết tự động của chúng tôi là chính xác và đầy đủ, nhưng nó có một vấn đề nhỏ. Nếu trình trợ giúp thẻ WWW chạy trước, các liên kết www sẽ không chính xác. Cập nhật mã bằng cách thêm quá tải Đơn hàng để kiểm soát thứ tự mà thẻ chạy. Thuộc tính Order xác định thứ tự thực thi liên quan đến các trình trợ giúp thẻ khác nhau mục tiêu cùng một phần tử. Giá trị đặt hàng mặc định là 0 và các phiên bản có giá trị thấp hơn sẽ được thực hiện trước.

```
public class AutoLinkerHttpTagHelper: TagHelper {  
  
    // Bộ lọc này phải chạy trước AutoLinkerWwwTagHelper khi nó tìm kiếm và thay thế http và // AutoLinkerWwwTagHelper thêm http vào đánh dấu. ghi đè  
    công khai int Order {  
  
        lấy { return int.MinValue; }  
    }  
}
```

Đoạn mã trên sẽ đảm bảo rằng trình trợ giúp thẻ WWW chạy trước trình trợ giúp thẻ HTTP. Thay đổi Thứ tự thành MaxValue và xác minh rằng đánh dấu được tạo cho thẻ WWW là không chính xác.

Kiểm tra và truy xuất nội dung trẻ em

Trình trợ giúp thẻ cung cấp một số thuộc tính để truy xuất nội dung.

- Kết quả của GetChildContentAsync có thể được thêm vào output.Content.
- Bạn có thể kiểm tra kết quả của GetChildContentAsync bằng GetContent.
- Nếu bạn sửa đổi output.Content, phần thân TagHelper sẽ không được thực thi hoặc hiển thị trừ khi bạn gọi GetChildContentAsync như trong mẫu trình liên kết tự động của chúng tôi:

```
public class AutoLinkerHttpTagHelper: TagHelper {  
  
    ghi đè công khai không đồng bộ Task ProcessAsync (ngữ cảnh TagHelperContext, đầu ra TagHelperOutput) {  
  
        var childContent = output.Content.IsModified? output.Content.GetContent ():  
            (đang chờ đầu ra.GetChildContentAsync ()). GetContent ();  
  
        // Tìm Url trong nội dung và thay thế chúng bằng thẻ liên kết tương đương. output.Content.SetHtmlContent (Regex.Replace  
        (childContent, @ "\ b (? : https? : // ) (\ S + ) \ b ",  
  
            "<a target=_blank " href=\"$0\" > $ 0 </a>"); // phiên bản liên kết http  
    }  
}
```

- Nhiều lệnh gọi tới GetChildContentAsync sẽ trả về cùng một giá trị và sẽ không thực thi lại. Nội dung TagHelper trừ khi bạn truyền tham số sai cho biết không sử dụng kết quả được lưu trong bộ nhớ cache.

Kết thúc và các bước tiếp theo

Hướng dẫn này là phần giới thiệu về trình trợ giúp tác giả thẻ và các mã không nên được coi là hướng dẫn cho các phương pháp hay nhất. Ví dụ: một ứng dụng thực có thẻ sẽ sử dụng một biểu thức chính quy thanh lịch hơn để thay thế cả liên kết HTTP và WWW trong một biểu thức. Trình trợ giúp thẻ ASP.NET 5 MVC 6 cung cấp các ví dụ tốt nhất về trình trợ giúp thẻ được viết tốt.

Tài nguyên bổ sung

- TagHelperSamples trên GitHub chứa các mẫu trình trợ giúp thẻ để làm việc với Bootstrap.
- Kênh 9 video về trình trợ giúp thẻ nâng cao. Đây là một video tuyệt vời về các tính năng nâng cao hơn. Đó là một vài phiên bản đã lỗi thời nhưng các nhận xét chứa danh sách các thay đổi đối với phiên bản hiện tại và bạn có thể tìm thấy mã cập nhật tại đây.

5.4.3 Trình trợ giúp thẻ nâng cao

Lưu ý: Chúng tôi hiện đang làm việc về chủ đề này.

Chúng tôi hoàn nghênh ý kiến đóng góp của bạn để giúp định hình phạm vi và cách tiếp cận. Bạn có thể theo dõi trạng thái và cung cấp thông tin đầu vào về vấn đề này tại GitHub.

Nếu bạn muốn xem lại các bản nháp và phác thảo ban đầu của chủ đề này, vui lòng để lại ghi chú với thông tin liên hệ của bạn trong số này.

Tìm hiểu thêm về cách bạn có thể đóng góp trên GitHub.

5.5 Lượt xem một phần

Lưu ý: Chúng tôi hiện đang làm việc về chủ đề này.

Chúng tôi hoàn nghênh ý kiến đóng góp của bạn để giúp định hình phạm vi và cách tiếp cận. Bạn có thể theo dõi trạng thái và cung cấp thông tin đầu vào về vấn đề này tại GitHub.

Nếu bạn muốn xem lại các bản nháp và phác thảo ban đầu của chủ đề này, vui lòng để lại ghi chú với thông tin liên hệ của bạn trong số này.

Tìm hiểu thêm về cách bạn có thể đóng góp trên GitHub.

5.6 Tiêm dịch vụ vào chế độ xem

Bởi Steve Smith

ASP.NET MVC 6 hỗ trợ tiêm phụ thuộc vào các lượt xem. Điều này có thể hữu ích cho các dịch vụ dành riêng cho chế độ xem, chẳng hạn như bản địa hóa hoặc dữ liệu chỉ cần thiết để nhập các phần tử chế độ xem. Bạn nên cố gắng duy trì sự tách biệt các mối quan tâm giữa bộ điều khiển và chế độ xem của bạn. Hầu hết dữ liệu mà chế độ xem của bạn hiển thị phải được chuyển vào từ bộ điều khiển.

Các phần:

- Một ví dụ đơn giản
- Đang diễn dữ liệu tra cứu
- Dịch vụ ghi đè

Xem các tệp mẫu

5.6.1 Một ví dụ đơn giản

Bạn có thể đưa một dịch vụ vào một dạng xem bằng cách sử dụng lệnh @inject. Bạn có thể coi @inject là thêm một thuộc tính vào chế độ xem của bạn và điền thuộc tính bằng DI.

Cú pháp cho @inject: @inject <type> <name>

Ví dụ về @inject trong hoạt động:

```

1 @using System.Threading.Tasks
2 @using ViewInjectSample.Model
3 @using ViewInjectSample.Model.Services
4 @model IEnumerable<ToDoItem>
5 @inject StatisticsService StatsService
6 <!DOCTYPE html>
7 <html>
8 <head>
9     <title> Các việc cần làm </title>
10 </head>
11 <body>
12     <div>
13         <h1> Các việc cần làm </h1>
14         <ul>
15             <li> Tổng số mục: @await StatsService.GetCount () </li>
16             <li> Đã hoàn thành: @await StatsService.GetCompletedCount () </li>
17             <li> Trung bình Mức độ ưu tiên: @await StatsService.GetAveragePosystem () </li>
18         </ul>
19         <bàn>
20             <tr>
21                 <th> Tên </th>
22                 <th> Ưu tiên </th>
23                 <th> Đã xong chưa? </th>
24             </tr>
25             @foreach (var item trong Model)
26             {
27                 <tr>
28                     <td> @ item.Name </td>
29                     <td> @ item.Pinent </td>
30                     <td> @ item.IsDone </td>
31                 </tr>
32             }
33         </table>
34     </div>
35 </body>
36 </html>

```

Chế độ xem này hiển thị danh sách các phiên bản ToDoItem, cùng với bản tóm tắt hiển thị thống kê tổng thể. Tổngmary được điền từ StatisticsService được đưa vào. Dịch vụ này được đăng ký để tiêm phụ thuộc trong ConfigureServices trong Startup.cs:

```

1 // Để biết thêm thông tin về cách định cấu hình ứng dụng của bạn, hãy truy cập http://go.microsoft.com/fwlink/??
2 public void ConfigureServices (dịch vụ IServiceProvider)
3 {
4     services.AddMvc ();
5
6     services.AddTransient <ITodoItemRepository, TodoItemRepository> ();
7     services.AddTransient <StatisticsService> ();
8     services.AddTransient <ProfileOptionsService> ();
..
```

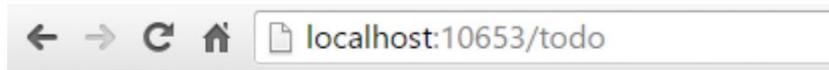
StatisticsService thực hiện một số tính toán trên tập hợp các thẻ hiện ToDoItem, mà nó truy cập thông qua kho:

```

1 bằng cách sử dụng System.Linq;
2 bằng cách sử dụng System.Threading.Tasks;
3 bằng ViewInjectSample.Interfaces;
4
5 không gian tên ViewInjectSample.Model.Services
6 {
7     Public class StatisticsService
8     {
9         riêng tư IToDoItemRepository _toDoItemRepository;
10
11         public StatisticsService (IToDoItemRepository toDoItemRepository)
12         {
13             _toDoItemRepository = toDoItemRepository;
14         }
15
16         Tác vụ không đồng bộ công khai <int> GetCount ()
17         {
18             trả về chờ đợi Task.FromResult (_toDoItemRepository.List (). Count ());
19         }
20
21         Tác vụ không đồng bộ công khai <int> GetCompletedCount ()
22         {
23             trả về dạng chờ Task.FromResult (
24                 _toDoItemRepository.List (). Count (x => x.IsDone));
25         }
26
27         public async Task <double> GetAveragePosystem ()
28         {
29             if (_toDoItemRepository.List (). Count () == 0)
30             {
31                 trả về 0,0;
32             }
33
34             trả về dạng chờ Task.FromResult (
35                 _toDoItemRepository.List (). Average (x => x.Posystem));
36         }
37     }
38 }
```

Kho lưu trữ mẫu sử dụng bộ sưu tập trong bộ nhớ. Việc triển khai được hiển thị ở trên (hoạt động trên tất cả các dữ liệu trong bộ nhớ) không được khuyến nghị cho các tập dữ liệu lớn, được truy cập từ xa.

Mẫu hiển thị dữ liệu từ mô hình được liên kết với chế độ xem và dịch vụ được đưa vào chế độ xem:



To Do Items

- Total Items: 50
- Completed: 17
- Avg. Priority: 3

Name Priority Is Done?

Task 1	1	True
Task 2	2	False
Task 3	3	False
Task 4	4	True
Task 5	5	False

5.6.2 Đang diễn dữ liệu tra cứu

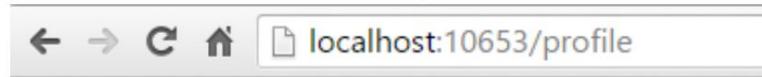
Chế độ xem chèn có thể hữu ích để diễn các tùy chọn trong các phần tử giao diện người dùng, chẳng hạn như danh sách thả xuống. Xem xét một biểu mẫu hồ sơ người dùng bao gồm các tùy chọn để chỉ định giới tính, tiểu bang và các tùy chọn khác. Hiển thị một biểu mẫu như vậy bằng cách sử dụng một tiêu chuẩn. Cách tiếp cận MVC sẽ yêu cầu bộ điều khiển yêu cầu các dịch vụ truy cập dữ liệu cho từng nhóm tùy chọn này, và sau đó diễn vào một mô hình hoặc ViewBag với từng nhóm tùy chọn được ràng buộc.

Một cách tiếp cận thay thế đưa các dịch vụ trực tiếp vào chế độ xem để có được các tùy chọn. Điều này giảm thiểu lượng mã do bộ điều khiển yêu cầu, di chuyển logic xây dựng phần tử chè độ xem này vào chính chế độ xem. Hành động của bộ điều khiển để hiển thị một biểu mẫu chỉnh sửa hồ sơ, chỉ cần chuyển biểu mẫu là trường hợp hồ sơ:

```

1 sử dụng Microsoft.AspNet.Mvc;
2 bằng ViewInjectSample.Model;
3
4 vùng tên ViewInjectSample.Controllers
5 {
6     public class ProfileController : Controller
7     {
8         [Tuyển đường ("Hồ sơ")]
9         Chỉ số IActionResult công khai ()
10        {
11            // VIỆC CẦN LÀM: tra cứu hồ sơ dựa trên người dùng đã đăng nhập
12            var profile = new Profile ()
13            {
14                Tên = "Steve",
15                FavColor = "Xanh lam",
16                Giới tính = "Nam",
17                Bang = Bang mới ("Ohio", "OH")
18            };
19            return View (hồ sơ);
20        }
21    }
22 }
```

Biểu mẫu HTML được sử dụng để cập nhật các tùy chọn này bao gồm danh sách thả xuống cho ba trong số các thuộc tính:



Update Profile

Name:

Gender:

State:

Fav. Color:

Các danh sách này được điều khiển bởi một dịch vụ đã được đưa vào chế độ xem:

```

1 @using System.Threading.Tasks
2 @using ViewInjectSample.Model.Services
3 @model ViewInjectSample.Model.Profile
4 @inject ProfileOptionsService Tùy chọn dịch vụ
5 <!DOCTYPE html>
6 <html>
7 <head>
..   <title> Cập nhật hồ sơ </title>
9 </head>
10 <body>
11 <div>
12     <h1> Cập nhật hồ sơ </h1>
13     Tên: @ Html.TextBoxFor (m => m.Name)
14     <br/>
15     Giới tính: @ Html.DropDownList ("Giới tính",
16         Options.ListGenders () . Chọn (g =>
17             new SelectListItem () {Text = g, Value = g}))
18     <br/>
19
20     Trạng thái: @ Html.DropDownListFor (m => m.State.Code,
21         Options.ListStates () . Chọn (s =>
22             new SelectListItem () {Text = s.Name, Value = s.Code}))
23     <br />
24
25     Yêu thích. Màu: @ Html.DropDownList ("Màu ưa thích",
26         Options.ListColors () . Chọn (c =>
27             new SelectListItem () {Text = c, Value = c}))
28     </div>
29 </body>
30 </html>
```

ProfileOptionsService là một dịch vụ cấp giao diện người dùng được thiết kế để chỉ cung cấp dữ liệu cần thiết cho biểu mẫu này:

```

1 bằng cách sử dụng System.Collections.Generic;
2
3 không gian tên ViewInjectSample.Model.Services
4 {
5     public class ProfileOptionsService
6     {
7         danh sách công khai <string> ListGenders ()
```

```

...
    // giữ cho điều này đơn giản
    return new List <string> () {"Female", "Male"};
}

danh sách công khai <State> ListStates ()
{
    // một vài tiểu bang của Hoa Kỳ
    trả về Danh sách mới <Trạng thái> ()
    {
        Bang mới ("Alabama", "AL"),
        Bang mới ("Alaska", "AK"),
        Bang mới ("Ohio", "OH")
    };
}

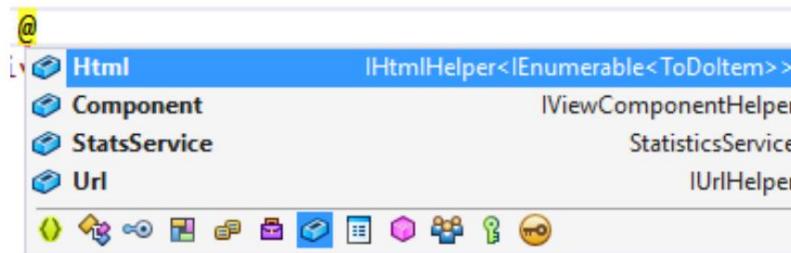
danh sách công khai <string> ListColors ()
{
    return new List <string> () { "Blue", "Green", "Red", "Yellow" };
}
}
}

```

Mèo: Đừng quên đăng ký các loại bạn sẽ yêu cầu thông qua chèn phụ thuộc trong ConfigureServices trong Startup.cs.

5.6.3 Dịch vụ ghi đè

Ngoài việc tiêm các dịch vụ mới, kỹ thuật này cũng có thể được sử dụng để ghi đè các dịch vụ đã tiêm trước đó trên một trang. Hình bên dưới cho thấy tất cả các trường có sẵn trên trang được sử dụng trong ví dụ đầu tiên:



Như bạn có thể thấy, các trường mặc định bao gồm **Html**, **Thành phần** và **Url** (cũng như **StatsService** mà chúng tôi tiêm). Ví dụ: nếu bạn muốn thay thế Trình trợ giúp HTML mặc định bằng Trình trợ giúp HTML của riêng mình, bạn có thể dễ dàng làm như vậy sử dụng `@inject`:

```

1 @using System.Threading.Tasks
2 @using ViewInjectSample.Helpers
3 @inject MyHtmlHelper Html
4 <!DOCTYPE html>
5 <html>
6 <head>
7     <title> Người trợ giúp của tôi </title>
8 </head>
9 <body>
10    <div>
11        Kiểm tra: @ Html.Value

```

```

12   </div>
13 </body> 14
</html>

```

Nếu bạn muốn mở rộng các dịch vụ hiện có, bạn có thể chỉ cần sử dụng kỹ thuật này trong khi kế thừa hoặc kết hợp triển khai hiện có với của riêng bạn.

5.6.4 Xem thêm

- Blog Simon Timms: [Đưa dữ liệu tra cứu vào chế độ xem của bạn](#)

5.7 Xem các thành phần

Bởi Rick Anderson

Trong bài viết này:

- Giới thiệu các thành phần chế độ xem
- Kiểm tra lớp ViewComponent • Kiểm tra chế độ xem thành phần chế độ xem • Thêm InvokeAsync vào thành phần chế độ xem ưu tiên • Chỉ định tên chế độ xem

5.7.1 Giới thiệu các thành phần khung nhìn

Mới đổi với ASP.NET MVC 6, các thành phần chế độ xem tương tự như chế độ xem từng phần, nhưng chúng mạnh hơn nhiều. Các thành phần chế độ xem bao gồm các lợi ích phân tách mối quan tâm và khả năng kiểm tra giống nhau được tìm thấy giữa bộ điều khiển và chế độ xem. Thành phần khung nhìn chịu trách nhiệm hiển thị một đoạn thay vì toàn bộ phản hồi. Bạn có thể sử dụng các thành phần chế độ xem để giải quyết bất kỳ vấn đề nào mà bạn cảm thấy quá phức tạp với một phần, chẳng hạn như:

- Các menu điều hướng động
- Tag đám mây (nơi nó truy vấn cơ sở dữ liệu)
- Bảng đăng nhập
- Giờ hàng
- Các bài báo được xuất bản gần đây
- Nội dung thanh bên trên một blog điển hình

Một cách sử dụng thành phần chế độ xem có thể là tạo bảng đăng nhập sẽ được hiển thị trên mọi trang với chức năng sau:

- Nếu người dùng chưa đăng nhập, một bảng đăng nhập sẽ hiển thị.
- Nếu người dùng đã đăng nhập, các liên kết để đăng xuất và quản lý tài khoản sẽ hiển thị.
- Nếu người dùng ở vai trò quản trị viên, một bảng điều khiển quản trị sẽ được hiển thị.

Bạn cũng có thể tạo thành phần chế độ xem lấy và hiển thị dữ liệu tùy thuộc vào yêu cầu của người dùng. Bạn có thể thêm chế độ xem thành phần dạng xem này vào trang bố cục và để nó nhận và hiển thị dữ liệu dành riêng cho người dùng trong toàn bộ ứng dụng.

Các thành phần dạng xem không sử dụng liên kết mô hình và chỉ phụ thuộc vào dữ liệu bạn cung cấp khi gọi vào nó.

Một thành phần chế độ xem bao gồm hai phần, lớp (thường bắt nguồn từ ViewComponent) và chế độ xem Razor mà gọi các phương thức trong lớp thành phần khung nhìn. Giống như bộ điều khiển, thành phần chế độ xem có thể là POCO, nhưng hầu hết người dùng sẽ muốn tận dụng các phương thức và thuộc tính có sẵn bằng cách lấy từ ViewComponent.

Một lớp thành phần dạng xem có thể được tạo bởi bất kỳ cách nào sau đây:

- Bắt nguồn từ ViewComponent.
- Trang trí lớp bằng thuộc tính [ViewComponent] hoặc bắt nguồn từ một lớp có thuộc tính [ViewComponent].
- Tạo một lớp mà tên kết thúc bằng hậu tố ViewComponent.

Giống như bộ điều khiển, các thành phần của khung nhìn phải là các lớp công khai, không lồng nhau, không trừu tượng.

5.7.2 Kiểm tra lớp ViewComponent

- Kiểm tra tệp src \ TodoList \ ViewComponents \ PriorityListViewComponent.cs:

```

1 bằng cách sử dụng System.Linq;
2 bằng cách sử dụng Microsoft.AspNet.Mvc;
3 bằng cách sử dụng TodoList.Models;
4
5 không gian tên TodoList.ViewComponents
6 {
7     lớp công khai PriorityListViewComponent : ViewComponent
8     {
9         riêng tư chỉ đọc ApplicationDbContext db;
10
11         công khai PriorityListViewComponent (nguyên cảnh ApplicationContext)
12         {
13             db = bối cảnh;
14         }
15
16         public IViewComponentResult Invoke (int maxPosystem)
17         {
18             var items = db.TodoItems.Where (x => x.IsDone == false &&
19                 x. ngoại lệ <= tối đa);
20
21             return View (mục);
22         }
23     }
24 }
```

Ghi chú về mã:

- Xem các lớp thành phần có thể được chứa trong bất kỳ thư mục nào trong dự án.
- Vì tên lớp PriorityListViewComponent kết thúc bằng hậu tố ViewComponent, thời gian chạy sẽ sử dụng chuỗi "PriorityList" khi tham chiếu thành phần lớp từ một dạng xem. Tôi sẽ giải thích điều đó trong chi tiết hơn sau.
- Thuộc tính [ViewComponent] có thể thay đổi tên được sử dụng để tham chiếu một thành phần chế độ xem. Ví dụ, chúng tôi có thể đã đặt tên cho lớp là XYZ và áp dụng thuộc tính ViewComponent:

```

1 [ViewComponent (Name = "PriorityList")]
2 lớp công khai XYZ : ViewComponent
```

- Thuộc tính [ViewComponent] ở trên cho bộ chọn thành phần chế độ xem sử dụng tên PriorityList khi tìm kiếm các chế độ xem được liên kết với thành phần và sử dụng chuỗi "Danh sách ưu tiên" khi tham chiếu

thành phần lớp từ một khung nhìn. Tôi sẽ giải thích điều đó chi tiết hơn sau.

- Thành phần sử dụng phương thức chèn vào phương thức khởi tạo để làm cho ngữ cảnh dữ liệu có sẵn.
- Invoke trình bày một phương thức có thể được gọi từ một khung nhìn và nó có thể nhận một số lượng đối số tùy ý. Phiên bản không đồng bộ, InvokeAsync, có sẵn. Chúng ta sẽ thấy InvokeAsync và nhiều đối số ở phần sau trong hướng dẫn. Trong đoạn mã trên, phương thức Invoke trả về tập hợp các ToDoItems chưa được hoàn thành và có mức độ ưu tiên lớn hơn hoặc bằng tối đa.

5.7.3 Kiểm tra khung nhìn thành phần khung nhìn

1. Kiểm tra nội dung của Views \ Todo \ Components. Thư mục này phải được đặt tên là Thành phần.

Lưu ý: Các dạng xem View Component thường được thêm vào thư mục Views \ Shared \ Components, vì các components view cơm thường không dành riêng cho bộ điều khiển.

2. Kiểm tra thư mục Views \ Todo \ Components \ PriorityList. Tên thư mục này phải khớp với tên của thành phần dạng xem (tên của lớp trừ đi hậu tố nếu chúng ta tuân theo quy ước và sử dụng hậu tố ViewComponent trong tên lớp). Nếu bạn đã sử dụng thuộc tính ViewComponent, tên thư mục sẽ cần phải khớp với chỉ định thuộc tính.

3. Kiểm tra chế độ xem Views \ Todo \ Components \ PriorityList \ Default.cshtml Razor.

```
1 @model IEnumerable<TodoList.Models.TodoItem>
2
3 <h3> Các mục ưu tiên </h3> 4 <ul>
5     @foreach (var todo in Model) {
6
7         <li> @ todo.Title </li>
8
} 9 </ul>
```

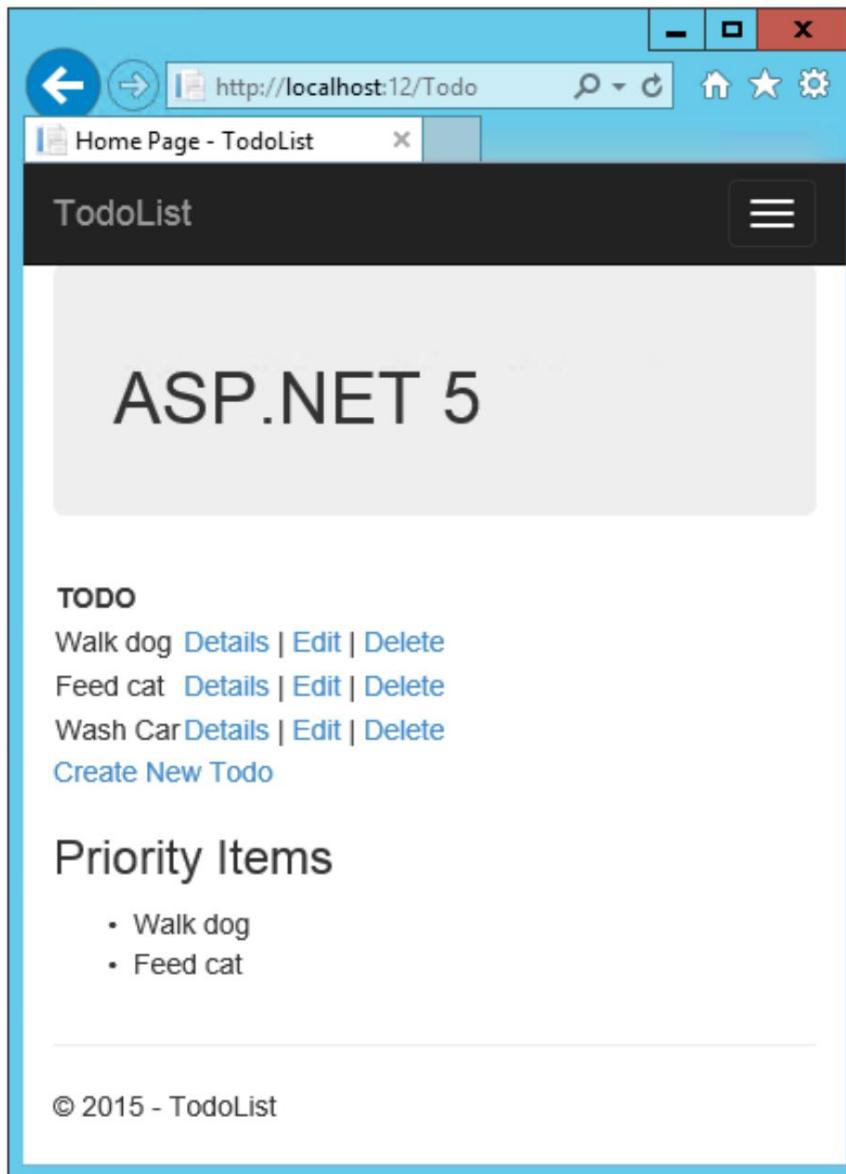
Chế độ xem Razor lấy một danh sách các TodoItems và hiển thị chúng. Nếu phương thức gọi thành phần chế độ xem không chuyển tên của chế độ xem (như trong mẫu của chúng tôi), thì Mặc định được sử dụng cho tên chế độ xem theo quy ước. Phần sau của hướng dẫn, tôi sẽ chỉ cho bạn cách chuyển tên của chế độ xem.

4. Thêm một div chứa lời gọi đến thành phần danh sách ưu tiên vào cuối tệp views \ todo \ index.cshtml:

```
1 @ * Đã xóa đánh dấu cho ngắn gọn * @ 2 <div> @
Html.ActionLink ("Tạo Công cụ mới", "Tạo", "Công việc") </div>
3 <div>
4     <div class = "col-md-4">
5         @ Component.Invoke ("Danh sách ưu tiên", 1) </div> 7
6     </div>
```

Đánh dấu @ Component.Invoke hiển thị cú pháp để gọi các thành phần chế độ xem. Đối số đầu tiên là tên của thành phần mà chúng ta muốn gọi hoặc gọi. Các tham số tiếp theo được chuyển cho nent compo. Trong trường hợp này, chúng tôi đang chuyển "1" làm mức độ ưu tiên mà chúng tôi muốn lọc. Invoke và InvokeAsync có thể nhận một số lượng đối số tùy ý.

Hình ảnh sau đây hiển thị các mục ưu tiên: (đảm bảo bạn có ít nhất một mục ưu tiên 1 mục chưa hoàn thành)



5.7.4 Thêm InvokeAsync vào thành phần chế độ xem ưu tiên

Cập nhật lớp thành phần chế độ xem ưu tiên với mã sau:

Lưu ý: IQueryble kết xuất mảng đồng bộ, không đồng bộ. Đây là một ví dụ đơn giản về cách bạn có thể gọi các phương thức không đồng bộ.

```
1 sử dụng System.Threading.Tasks;
2
3 lớp công khai PriorityListViewComponent : ViewComponent
4 {
5     riêng tư chỉ đọc ApplicationDbContext db;
6
7     công khai PriorityListViewComponent (ngữ cảnh ApplicationDbContext)
8 }
```

```

9     db = bối cảnh;
10    }
11
12    // Đã xóa lời gọi đồng bộ.
13
14    public async Task <IViewComponentResult> InvokeAsync (int maxPosystem, bool isDone)
15    {
16        var items = await GetItemsAsync (maxPosystem, isDone);
17        return View (mục);
18    }
19
20    Tác vụ riêng tư <IQueryable <TodoItem>> GetItemsAsync (int maxPosystem, bool isDone)
21    {
22        return Task.FromResult (GetItems (maxPosystem, isDone));
23    }
24    private IQueryable <TodoItem> GetItems (int maxPosystem, bool isDone)
25    {
26        var items = db.TodoItems.Where (x => x.IsDone == isDone &&
27            x. ngoại lệ <= tối đa);
28
29        string msg = "Priority <= " + maxPosystem.ToString () +
30                    " && isDone == " + isDone.ToString ();
31        ViewBag.PosystemMessage = msg;
32
33        trả lại mặt hàng;
34    }
35

```

Cập nhật chế độ xem Razor view (TodoList \ src \ TodoList \ Views \ ToDo \ Components \ PriorityList \ Default.cshtml) để hiển thị thông báo ưu tiên:

```

1 @model IEnumerable <TodoList.Models.TodoItem>
2
3 <h4> @ ViewBag.PosystemMessage </h4>
4 <ul>
5     @foreach (var todo trong Model)
6     {
7         <li> @ todo.Title </li>
..     }
9 </ul>

```

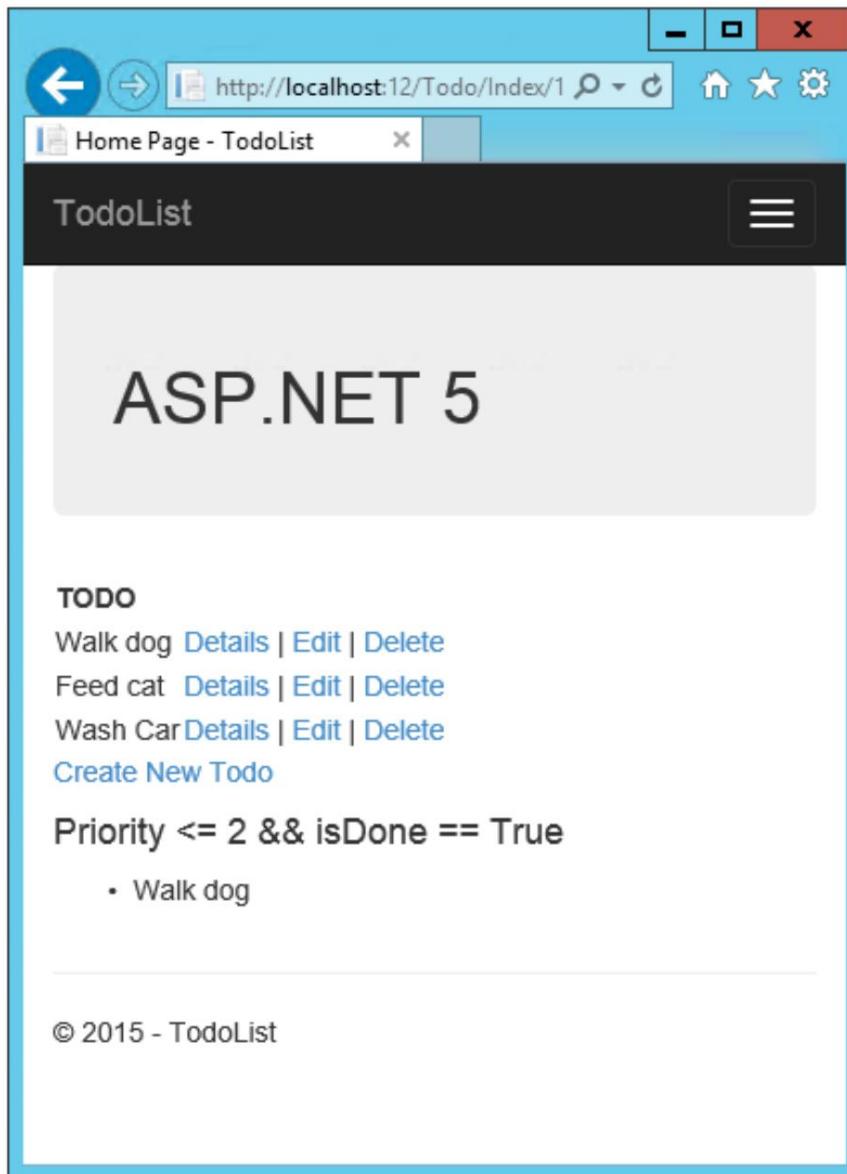
Cuối cùng, cập nhật chế độ xem \ todo \ index.cshtml:

```

1 /* Đã xóa đánh dấu cho ngắn gọn. * @
2 <div class = "col-md-4">
3     @await Component.InvokeAsync ("PriorityList", 2, true)
4 </div>

```

Hình ảnh sau phản ánh những thay đổi chúng tôi đã thực hiện đối với thành phần chế độ xem ưu tiên và chế độ xem chỉ mục:



5.7.5 Chỉ định tên dạng xem

Một thành phần dạng xem phức tạp có thể cần chỉ định dạng xem không mặc định trong một số điều kiện. Các chương trình sau cách chỉ định chế độ xem "PVC" từ phương thức InvokeAsync: Cập nhật phương thức InvokeAsync trong Lớp PriorityListViewComponent.

```

1 Tác vụ không đồng bộ công khai <IViewComponentResult> InvokeAsync (int maxPosystem, bool isDone)
2 {
3     string MyView = "Mặc định";
4     // Nếu yêu cầu tất cả các tác vụ đã hoàn thành, hãy hiển thị bằng chế độ xem "PVC".
5     if (maxPosystem > 3 && isDone == true)
6     {
7         MyView = "PVC";
8     }
9     var items = await GetItemsAsync (maxPosystem, isDone);

```

```
10     return View (MyView, các mục);
11 }
```

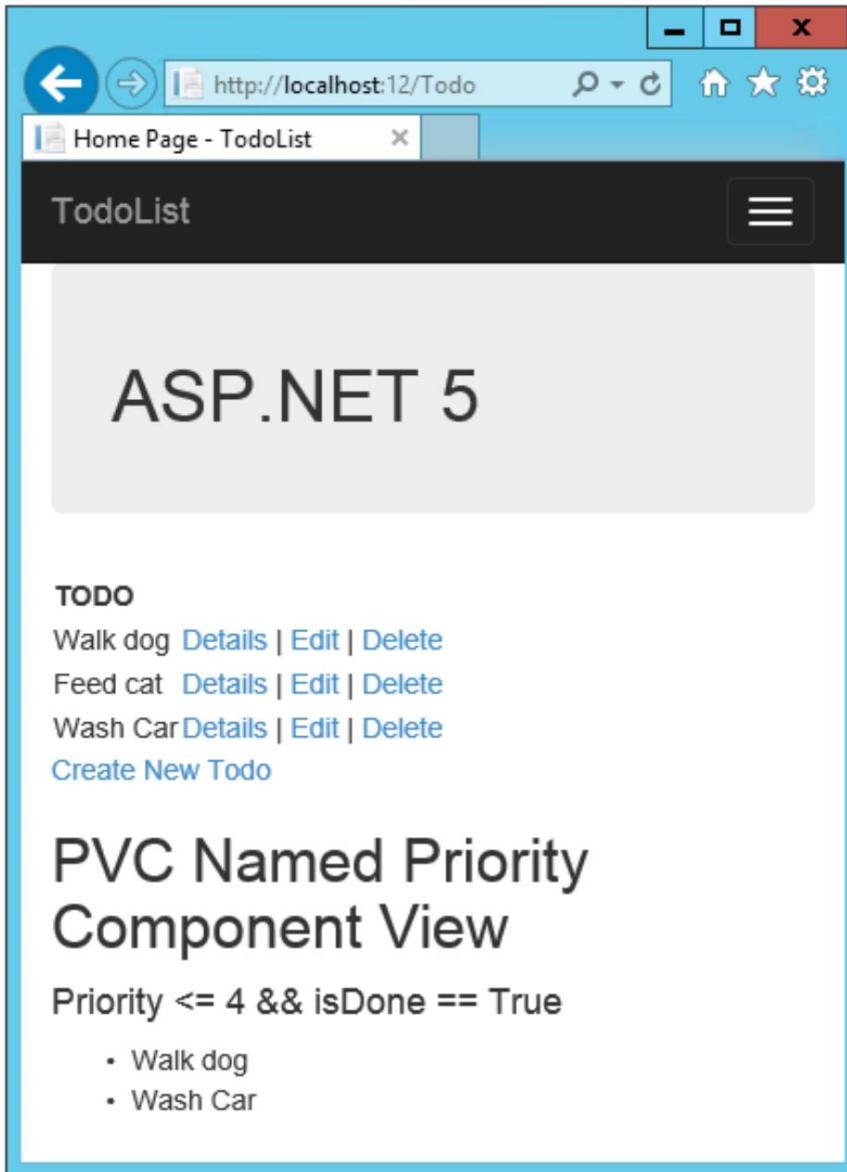
Kiểm tra chế độ xem Views \ Todo \ Components \ PriorityList \ PVC.cshtml. Tôi đã thay đổi chế độ xem PVC để xác minh rằng nó đang được sử dụng:

```
1 @model IEnumerable <TodoList.Models.TodoItem>
2
3 <h2> Chế độ xem thành phần ưu tiên được đặt tên PVC </h2>
4 <h4> @ ViewBag.PosystemMessage </h4>
5 <ul>
6     @foreach (var todo trong Model)
7     {
8         <li> @ todo.Title </li>
9     }
10 </ul>
```

Cuối cùng, cập nhật Views \ TodoIndex.cshtml

```
1 @await Component.InvokeAsync ("PriorityList", 4, true)
```

Chạy ứng dụng và nhấp vào liên kết PVC (hoặc điều hướng đến localhost: <port> / Todo / IndexFinal). Làm mới trang để xem Xem PVC.



5.8 Tạo một công cụ xem tùy chỉnh

Lưu ý: Chúng tôi hiện đang làm việc về chủ đề này.

Chúng tôi hoan nghênh ý kiến đóng góp của bạn để giúp định hình phạm vi và cách tiếp cận. Bạn có thể theo dõi trạng thái và cung cấp thông tin đầu vào về vấn đề này tại GitHub.

Nếu bạn muốn xem lại các bản nháp và phác thảo ban đầu của chủ đề này, vui lòng để lại ghi chú với thông tin liên hệ của bạn trong số này.

Tìm hiểu thêm về cách bạn có thể đóng góp trên GitHub.

5.9 Xây dựng các chế độ xem cụ thể trên thiết bị di động

Lưu ý: Chúng tôi hiện đang làm việc về chủ đề này.

Chúng tôi hoan nghênh ý kiến đóng góp của bạn để giúp định hình phạm vi và cách tiếp cận. Bạn có thể theo dõi trạng thái và cung cấp thông tin đầu vào về vấn đề này tại GitHub.

Nếu bạn muốn xem lại các bản nháp và phác thảo ban đầu của chủ đề này, vui lòng để lại ghi chú với thông tin liên hệ của bạn trong số này.

Tìm hiểu thêm về cách bạn có thể đóng góp trên GitHub.

Bộ điều khiển

6.1 Bộ điều khiển, Hành động và Kết quả Hành động

Bởi Steve Smith

Hành động và kết quả hành động là một phần cơ bản của cách các nhà phát triển xây dựng ứng dụng bằng ASP.NET MVC.

Các phần:

- Bộ điều khiển là gì
- Xác định các hành động

Xem hoặc tải xuống mẫu từ GitHub.

6.1.1 Bộ điều khiển là gì

Trong ASP.NET MVC, Bộ điều khiển được sử dụng để xác định và nhóm một tập hợp các hành động. Một hành động (hoặc phương thức hành động) là một phương thức trên bộ điều khiển xử lý các yêu cầu đến. Bộ điều khiển cung cấp một phương tiện hợp lý để nhóm các hành động tương tự lại với nhau, cho phép các bộ quy tắc chung (ví dụ: định tuyến, bộ nhớ đệm, ủy quyền) được áp dụng chung.

Các yêu cầu đến được ánh xạ tới các hành động thông qua [định tuyến](#).

Trong ASP.NET 5, bộ điều khiển có thể là bất kỳ lớp nào có thể khởi tạo được kết thúc bằng "Bộ điều khiển" hoặc kế thừa từ một lớp kết thúc bằng "Bộ điều khiển". Bộ kiểm soát phải tuân theo [Nguyên tắc phụ thuộc rõ ràng](#) và yêu cầu bất kỳ phụ thuộc nào mà hành động của họ yêu cầu thông qua phương thức khởi tạo bằng cách sử dụng chèn phụ thuộc.

Theo quy ước, các lớp bộ điều khiển:

- nằm trong thư mục "Bộ điều khiển" cấp cơ sở
- inherit từ `Microsoft.AspNetCore.Mvc.Controller`

Hai quy ước này không bắt buộc.

Trong mẫu Model-View-Controller, Bộ điều khiển chịu trách nhiệm xử lý ban đầu yêu cầu và khởi tạo Mô hình. Nói chung, các quyết định kinh doanh phải được thực hiện trong Mô hình.

Lưu ý: Mô hình phải là Đối tượng CLR C# (POCO), không phải là `DbContext` hoặc kiểu liên quan đến cơ sở dữ liệu.

Bộ điều khiển lấy kết quả xử lý của mô hình (nếu có), trả về chế độ xem thích hợp cùng với dữ liệu chế độ xem liên quan. Tìm hiểu thêm: [Tổng quan về ASP.NET MVC và Bắt đầu với ASP.NET MVC 6](#).

Mẹo: Bộ điều khiển là một trùu tượng ở mức giao diện người dùng. Trách nhiệm của nó là đảm bảo dữ liệu yêu cầu đến là hợp lệ và

chọn chế độ xem (hoặc kết quả cho một API) sẽ được trả về. Trong các ứng dụng được kiểm chứng tốt, nó sẽ không trực tiếp bao gồm quyền truy cập dữ liệu hoặc logic nghiệp vụ, mà thay vào đó sẽ ủy quyền cho các dịch vụ xử lý các trách nhiệm này.

6.1.2 Xác định các hành động

Bất kỳ phương thức công khai nào trên một loại bộ điều khiển là một hành động. Các tham số về hành động được ràng buộc để yêu cầu dữ liệu và được xác thực bằng cách sử dụng ràng buộc mô hình.

Cảnh báo: Các phương thức hành động chấp nhận tham số phải xác minh thuộc tính ModelState.IsValid là đúng.

Các phương thức hành động phải chứa logic để ánh xạ một yêu cầu đến với mối quan tâm của doanh nghiệp. Mỗi quan tâm của doanh nghiệp thường phải được thể hiện dưới dạng các dịch vụ mà bộ điều khiển của bạn truy cập thông qua [chèn phụ thuộc](#). Sau đó, các hành động ánh xạ kết quả của hành động kinh doanh sang một trạng thái ứng dụng.

Các hành động có thể trả về bất kỳ thứ gì, nhưng thường sẽ trả về một thể hiện của IActionResult (hoặc Tác vụ <ActionResult>) cho các phương thức không đồng bộ) tạo ra một phản hồi. Phương pháp hành động có trách nhiệm lựa chọn loại phản ứng nào; kết quả hành động thực hiện phản hồi.

Phương pháp trợ giúp bộ điều khiển

Mặc dù không bắt buộc, hầu hết các nhà phát triển sẽ muốn có bộ điều khiển của họ kế thừa từ lớp Bộ điều khiển cơ sở.

Làm như vậy cung cấp cho bộ điều khiển quyền truy cập vào nhiều thuộc tính và các phương pháp hữu ích, bao gồm các phương pháp trình trợ giúp sau được thiết kế để hỗ trợ trả lại các phản hồi khác nhau:

Dạng [xem](#) Trả về dạng xem sử dụng mô hình để hiển thị HTML. Ví dụ: `return View (khách hàng);`

Mã trạng thái HTTP Trả về mã trạng thái HTTP. Ví dụ: `return BadRequest ();`

Phản hồi được định dạng Trả về Json hoặc tương tự để định dạng một đối tượng theo một cách cụ thể. Ví dụ: `trở lại Json (khách hàng);`

Nội dung được thương lượng phản hồi Thay vì trả lại một đối tượng trực tiếp, một hành động có thể trả lại một nội dung được thương lượng lại tài trợ (sử dụng Ok, Created, CreatedAtRoute hoặc CreatedAtAction). Ví dụ: `return Ok ();` hoặc trả về `CreatedAtRoute ("tên tuyến", giá trị, đối tượng mới");`

[Redirect](#) Trả về một chuyển hướng đến một hành động hoặc điểm đến khác (sử dụng Redirect, '' LocalRedirect '', '' RedirectToAction '' hoặc RedirectToRoute). Ví dụ: `return RedirectToAction ("Hoàn thành", mới {id = 123});`

Ngoài các phương thức trên, một hành động cũng có thể trả về một đối tượng đơn giản. Trong trường hợp này, đối tượng sẽ được định dạng dựa trên yêu cầu của khách hàng. [Tìm hiểu thêm về Định dạng](#)

Mối quan tâm xuyên suốt

Trong hầu hết các ứng dụng, nhiều hành động sẽ chia sẻ các phần trong quy trình làm việc của chúng. Ví dụ: hầu hết ứng dụng có thể chỉ có sẵn cho những người dùng đã xác thực hoặc có thể hưởng lợi từ bộ nhớ đệm. Khi bạn muốn thực hiện một số logic trước hoặc sau khi một phương thức hành động chạy, bạn có thể sử dụng một bộ lọc. Bạn có thể giúp giữ cho các hành động của mình không phát triển quá lớn bằng cách sử dụng [Bộ lọc](#) để xử lý những mối quan tâm xuyên suốt này. Điều này có thể giúp loại bỏ sự trùng lặp trong các hành động của bạn, cho phép chúng tuân theo nguyên tắc Không lặp lại bản thân (DRY).

Trong trường hợp ủy quyền và xác thực, bạn có thể áp dụng thuộc tính Authorize cho bất kỳ hành động nào yêu cầu nó. Thêm nó vào một bộ điều khiển sẽ áp dụng nó cho tất cả các hành động trong bộ điều khiển đó. Việc thêm thuộc tính này sẽ đảm bảo bộ lọc thích hợp được áp dụng cho bất kỳ yêu cầu nào cho hành động này. Một số thuộc tính có thể được áp dụng ở cả cấp độ bộ điều khiển và cấp độ hành động để cung cấp khả năng kiểm soát chi tiết đối với hành vi bộ lọc. [Tìm hiểu thêm: Bộ lọc và Bộ lọc ủy quyền.](#)

Các ví dụ khác về mối quan tâm xuyên suốt trong ứng dụng MVC có thể bao gồm:

- [Xử lý lỗi](#)
 - [Bộ nhớ đệm phản hồi](#)
-

Lưu ý: Nhiều mối quan tâm xuyên suốt có thể được xử lý bằng cách sử dụng các bộ lọc trong ứng dụng MVC. Một tùy chọn khác cần ghi nhớ có sẵn cho bất kỳ ứng dụng ASP.NET nào là [phần mềm trung gian tùy chỉnh](#).

6.2 Định tuyến đến các hành động của bộ điều khiển

Lưu ý: Chúng tôi hiện đang làm việc về chủ đề này.

Chúng tôi hoan nghênh ý kiến đóng góp của bạn để giúp định hình phạm vi và cách tiếp cận. Bạn có thể theo dõi trạng thái và cung cấp thông tin đầu vào về vấn đề này tại [GitHub](#).

Nếu bạn muốn xem lại các bản nháp và phác thảo ban đầu của chủ đề này, vui lòng để lại ghi chú với thông tin liên hệ của bạn trong số này.

[Tim hiểu thêm về cách bạn có thể đóng góp trên GitHub](#).

6.3 Xử lý lỗi

Lưu ý: Chúng tôi hiện đang làm việc về chủ đề này.

Chúng tôi hoan nghênh ý kiến đóng góp của bạn để giúp định hình phạm vi và cách tiếp cận. Bạn có thể theo dõi trạng thái và cung cấp thông tin đầu vào về vấn đề này tại [GitHub](#).

Nếu bạn muốn xem lại các bản nháp và phác thảo ban đầu của chủ đề này, vui lòng để lại ghi chú với thông tin liên hệ của bạn trong số này.

[Tim hiểu thêm về cách bạn có thể đóng góp trên GitHub](#).

6.4 Bộ lọc

Lưu ý: Chúng tôi hiện đang làm việc về chủ đề này.

Chúng tôi hoan nghênh ý kiến đóng góp của bạn để giúp định hình phạm vi và cách tiếp cận. Bạn có thể theo dõi trạng thái và cung cấp thông tin đầu vào về vấn đề này tại [GitHub](#).

Nếu bạn muốn xem lại các bản nháp và phác thảo ban đầu của chủ đề này, vui lòng để lại ghi chú với thông tin liên hệ của bạn trong số này.

[Tim hiểu thêm về cách bạn có thể đóng góp trên GitHub](#).

6.5 Bộ điều khiển và tiêm phụ thuộc

Bởi Steve Smith

Bộ điều khiển ASP.NET MVC 6 nên yêu cầu các phụ thuộc của chúng một cách rõ ràng thông qua các hàm tạo của chúng. Trong một số trường hợp, các hành động của bộ điều khiển riêng lẻ có thể yêu cầu một dịch vụ và có thể không hợp lý nếu yêu cầu ở cấp bộ điều khiển. Trong này trong trường hợp này, bạn cũng có thể chọn đưa một dịch vụ làm tham số vào phương thức hành động.

Trong bài viết này:

- [Tiêm phụ thuộc](#)
- [Chèn ép khôi xây dựng](#)
- [Action Injection với FromServices](#)
- [Truy cập Cài đặt từ Bộ điều khiển](#)

Xem hoặc tải xuống mẫu từ [GitHub](#).

6.5.1 Tiêm phụ thuộc

Đưa vào phụ thuộc là một kỹ thuật tuân theo Nguyên tắc Đảo ngược Phụ thuộc, cho phép các ứng dụng được cấu tạo bởi các mô-đun được ghép nối lỏng lẻo. ASP.NET 5, mà ASP.NET MVC 6 được xây dựng, có hỗ trợ tích hợp cho tiêm phụ thuộc (tim hiểu thêm), và mong đợi các ứng dụng được xây dựng cho ASP.NET 5 để triển khai kỹ thuật này (đúng hơn là hơn truy cập tĩnh hoặc khởi tạo trực tiếp).

Mẹo: Điều quan trọng là bạn phải hiểu rõ về cách ASP.NET 5 triển khai Dependency Injection (DI). Nếu bạn chưa làm như vậy, vui lòng đọc phần chèn phụ thuộc trong [ASP.NET 5 Cơ bản](#).

6.5.2 Chèn ép khôi xây dựng

Hỗ trợ tích hợp của ASP.NET 5 để tiêm phụ thuộc dựa trên phương thức khởi tạo mở rộng cho các bộ điều khiển ASP.NET MVC 6. Qua chỉ cần thêm một loại dịch vụ vào bộ điều khiển của bạn làm tham số khởi tạo, ASP.NET sẽ cố gắng giải quyết loại đó bằng cách sử dụng vùng chứa dịch vụ được tích hợp sẵn của nó. Các dịch vụ thường, nhưng không phải luôn luôn, được xác định bằng cách sử dụng các giao diện. Ví dụ, nếu ứng dụng của bạn có logic nghiệp vụ phụ thuộc vào thời gian hiện tại, bạn có thể đưa vào một dịch vụ truy xuất thời gian (chứ không phải mã hóa cứng), điều này sẽ cho phép các bài kiểm tra của bạn vượt qua trong các triển khai sử dụng một thời gian nhất định.

```

1 đang sử dụng Hệ thống;
2
3 vùng điều khiển không gian tênDI.Interfaces
4 {
5     giao diện công cộng IDateTime
6     {
7         DateTime Now { get; }
8     }
9 }
```

Việc triển khai một giao diện như thế này để nó sử dụng đồng hồ hệ thống trong thời gian chạy là điều không cần thiết:

```

1 đang sử dụng Hệ thống;
2 sử dụng ControllerDI.Interfaces;
3
4 bộ điều khiển không gian tênDI.
5 {
6     lớp công khai SystemDateTime : IDateTime
7     {
8         công khai DateTime Now
9         {
10            lấy { return DateTime.Now; }
11        }
12 }
```

```
12     }
13 }
```

Với điều này tại chỗ, chúng tôi có thể sử dụng dịch vụ trong bộ điều khiển của mình. Trong trường hợp này, chúng tôi đã thêm một số logic vào Phương thức HomeController Index để hiển thị lời chào cho người dùng dựa trên thời gian trong ngày.

```
1 sử dụng ControllerDI.Interfaces;
2 bằng Microsoft.AspNet.Mvc;
3
4 bộ điều khiển không gian tênDI.
5 {
6     lớp công khai HomeController : Bộ điều khiển
7     {
8         private readonly IDateTime _dateTime;
9
10        public HomeController (IDateTime dateTime)
11        {
12            _dateTime = dateTime;
13        }
14
15        Chỉ số IActionResult công khai ()
16        {
17            var serverTime = _dateTime.Now;
18            if (serverTime.Hour < 12)
19            {
20                ViewData ["Message"] = "Buổi sáng ở đây - Chào buổi sáng!";
21            }
22            else if (serverTime.Hour < 17)
23            {
24                ViewData ["Message"] = "Buổi chiều ở đây - Chào buổi chiều!";
25            }
26            khác
27            {
28                ViewData ["Message"] = "Ở đây là buổi tối - Chào buổi tối!";
29            }
30            return View ();
31        }
32    }
33 }
```

Nếu chúng ta chạy ứng dụng ngay bây giờ, rất có thể chúng ta sẽ gặp phải lỗi:

Đã xảy ra ngoại lệ chưa được xử lý trong khi xử lý yêu cầu.

InvalidOperationException: Không thể giải quyết dịch vụ cho loại 'ControllerDI.Interfaces.IDateTime' trong khi cố gắng 'ControllerDI.ControllerExtensions.DependencyInjection.ActivatorUtilities.GetService (Loại IServiceProvider sp, loại, Loại bắt buộcBy, Boolean isDefaultParameterRequired)'
to cross-cutting extensions DependencyInjection.ActivatorUtilities.GetService (Loại IServiceProvider sp, loại, Loại bắt buộcBy, Boolean isDefaultParameterRequired)

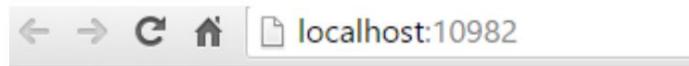
Lỗi này xảy ra khi chúng tôi chưa định cấu hình dịch vụ trong phương thức ConfigureServices trong Khởi động của chúng tôi lớp. Để chỉ định rằng các yêu cầu cho IDatetime phải được giải quyết bằng cách sử dụng một phiên bản của SystemDateTime, hãy thêm dòng được đánh dấu trong danh sách bên dưới cho phương thức ConfigureServices của bạn:

```
1 public void ConfigureServices (dịch vụ IServiceCollection)
2 {
3     services.AddMvc ();
4
5     // Thêm các dịch vụ ứng dụng.
6     services.AddTransient <IDateTime, SystemDateTime> ();
```

7

Lưu ý: Dịch vụ cụ thể này có thể được triển khai bằng cách sử dụng bất kỳ tùy chọn lâu dài nào khác nhau (Tạm thời, Phạm vi, hoặc Singleton). Hãy chắc chắn rằng bạn hiểu mỗi tùy chọn phạm vi này sẽ ảnh hưởng như thế nào đến hành vi của Dịch vụ. [Tìm hiểu thêm.](#)

Khi dịch vụ đã được định cấu hình, việc chạy ứng dụng và điều hướng đến trang chủ sẽ hiển thị thông báo dựa trên thời gian như mong đợi:



A Message From The Server

It's afternoon here - Good Afternoon!

Mẹo: Để xem cách yêu cầu phụ thuộc một cách rõ ràng trong bộ điều khiển giúp kiểm tra mã dễ dàng hơn, tìm hiểu thêm về [đơn vị thử nghiệm](#) các ứng dụng ASP.NET 5.

Việc tiêm phụ thuộc tích hợp trong ASP.NET 5 hỗ trợ chỉ có một phương thức khởi tạo duy nhất cho các lớp yêu cầu dịch vụ. Nếu bạn có nhiều hơn một hàm tạo, bạn có thể nhận được một ngoại lệ nêu rõ:

Đã xảy ra ngoại lệ chưa được xử lý trong khi xử lý yêu cầu.

Không hợp lệ trong hoạt động ngoại lệ: Đã tìm thấy nhiều hàm tạo chấp nhận tất cả các loại đối số nhất định trong loại 'ControllerDI.Controllers.HomeController'. Chỉ nên có một hàm tạo có thể áp dụng.

`Microsoft.Extensions.DependencyInjection.ActivatorUtilities.FindApplicableConstructor (Loại phiên bản kiểu, Loại [] đối sốTypes, ConstructorInfo & matchConstructor, Nullable'1 [] & tham sốMap)`

Khi thông báo lỗi nêu rõ, bạn có thể khắc phục sự cố này chỉ với một hàm tạo duy nhất. Bạn cũng có thể [thay thế](#) hỗ trợ chèn phụ thuộc mặc định với triển khai bên thứ ba, nhiều trong số đó hỗ trợ nhiều hàm tạo.

6.5.3 Chèn hành động với FromServices

Đôi khi bạn không cần một dịch vụ cho nhiều hơn một hành động trong bộ điều khiển của mình. Trong trường hợp này, nó có thể có ý nghĩa để đưa dịch vụ làm tham số vào phương thức hành động. Điều này được thực hiện bằng cách đánh dấu tham số với thuộc tính `[FromServices]` như được hiển thị ở đây:

```
1 công khai IActionResult About ([FromServices] IDateTime dateTime)
2 {
3     ViewData ["Message"] = "Hiện tại trên máy chủ, thời gian là"
4                                         + dateTime.Now;
5
6     return View ();
}
```

6.5.4 Truy cập cài đặt từ Bộ điều khiển

Khá phổ biến là bạn có thể muốn truy cập một số cài đặt ứng dụng hoặc cấu hình từ bộ điều khiển. Làm như vậy nên sử dụng `mẫu`. Tùy chọn được mô tả trong [cấu hình](#). Để yêu cầu cài đặt cấu hình thông qua phụ thuộc tiêm từ bộ điều khiển của bạn, bạn không nên yêu cầu cấu hình hoặc loại cài đặt trực tiếp. Thay vào đó, hãy yêu cầu một

`IOptions <T>` instance, trong đó `T` là lớp cấu hình bạn cần. Để làm việc với mẫu tùy chọn, bạn cần tạo một lớp đại diện cho các tùy chọn, chẳng hạn như lớp này:

```
1 vùng điều khiển không gian tên DI.Model
2 {
3     lớp công khai SampleWebSettings
4     {
5         chuỗi công khai Tiêu đề { get; bộ; }
6         public int Updates { get; bộ; }
7     }
.. }
```

Sau đó, bạn cần định cấu hình ứng dụng để sử dụng mô hình tùy chọn và thêm lớp cấu hình của bạn vào các dịch vụ bộ sưu tập trong `ConfigureServices`:

```
1 Startup công khai ()
2 {
3     var builder = new ConfigurationBuilder ()
4         .AddJsonFile ("samplewebsettings.json");
5     Cấu hình = builder.Build ();
6 }
7
8 public IConfigurationRoot Configuration { get; bộ; }
9
10 // Phương thức này được gọi bởi thời gian chạy. Sử dụng phương pháp này để thêm dịch vụ vào vùng chứa.
11 // Để biết thêm thông tin về cách cấu hình ứng dụng của bạn, hãy truy cập http://go.microsoft.com/fwlink/?
12 public void ConfigureServices (dịch vụ IServiceCollection)
13 {
14     // Bắt buộc để sử dụng mẫu Tùy chọn <T>
15     services.AddOptions ();
16
17     // Thêm cài đặt từ cấu hình
18     services.Configure <SampleWebSettings> (Cấu hình);
19
20     // Bỏ ghi chú để thêm cài đặt từ mã
21     //services.Configure<SampleWebSettings>(settings =>
22     // {
23     //     settings.Updates = 17;
24     // });
25
26     services.AddMvc ();
27
28     // Thêm các dịch vụ ứng dụng.
29     services.AddTransient <IDateTime, SystemDateTime> ();
30 }
```

Lưu ý: Trong danh sách trên, chúng tôi đang định cấu hình ứng dụng để đọc cài đặt từ tệp có định dạng JSON. Bạn cũng có thể định cấu hình cài đặt hoàn toàn bằng mã, như được hiển thị trong đoạn mã đã nhận xét ở trên. [Tim hiểu thêm về ASP.NET Tùy chọn cấu hình](#)

Khi bạn đã chỉ định một đối tượng cấu hình được đánh máy mạnh (trong trường hợp này là `SampleWebSettings`) và thêm nó đối với bộ sưu tập dịch vụ, bạn có thể yêu cầu nó từ bất kỳ phương thức Bộ điều khiển hoặc Hành động nào bằng cách yêu cầu một phiên bản của `IOptions <T>` (trong trường hợp này là `IOptions <SampleWebSettings>`). Đoạn mã sau đây cho thấy cách một người sẽ yêu cầu cài đặt từ bộ điều khiển:

```
1     public class SettingsController : Controller
2     {
3         riêng tư chỉ đọc SampleWebSettings _settings;
```

```

4
5     public SettingsController (IOptions <SampleWebSettings> settingsOptions)
6     {
7         _settings = settingsOptions.Value;
8     }
9
10    Chi số IActionResult công khai ()
11    {
12        ViewData ["Tiêu đề"] = _settings.Title;
13        ViewData ["Bản cập nhật"] = _settings.Updates;
14        return View ();
15    }
16}

```

Làm theo mẫu Tùy chọn cho phép tách các cài đặt và cấu hình khỏi nhau và đảm bảo kiểm soát viên đang theo đuổi [mỗi quan tâm](#), vì nó không cần biết làm thế nào hoặc ở đâu để tìm thông tin cài đặt. Nó cũng giúp bộ điều khiển kiểm tra đơn vị dễ dàng hơn, vì không có [bám tính](#) hoặc khởi tạo trực tiếp các cài đặt các lớp trong lớp bộ điều khiển.

6.6 Kiểm tra lôgic của bộ điều khiển

Lưu ý: Chúng tôi hiện đang làm việc về chủ đề này.

Chúng tôi hoan nghênh ý kiến đóng góp của bạn để giúp định hình phạm vi và cách tiếp cận. Bạn có thể theo dõi trạng thái và cung cấp thông tin đầu vào về vấn đề này tại [GitHub](#).

Nếu bạn muốn xem lại các bản nháp và phác thảo ban đầu của chủ đề này, vui lòng để lại ghi chú với thông tin liên hệ của bạn trong [vấn đề](#).

Tìm hiểu thêm về cách bạn có thể [đóng góp](#) trên GitHub.

6.7 Khu vực

Bởi Tom Archer

Các [khu vực](#) cung cấp một cách để tách một ứng dụng MVC lớn thành các nhóm mô hình, chế độ xem và bộ điều khiển. Hãy xem một ví dụ để minh họa cách các Vùng được tạo và sử dụng. Giả sử bạn có một cửa hàng ứng dụng có hai nhóm bộ điều khiển và chế độ xem riêng biệt: Sản phẩm và Dịch vụ.

Thay vì có tất cả các bộ điều khiển nằm trong thư mục Bộ điều khiển và tất cả các chế độ xem nằm trong thư mục chính Chế độ xem, bạn có thể sử dụng Khu vực để nhóm các chế độ xem và bộ điều khiển của mình theo khu vực (hoặc lôgic nhóm) mà chúng được liên kết với.

- Tên dự án
 - Các khu vực
 - * Các sản phẩm
 - Bộ điều khiển
 - HomeController.cs
 - Lượt xem
 - Nhà

- Index.cshtml
- * Dịch vụ
 - Bộ điều khiển
 - HomeController.cs
 - Lượt xem
 - Nhà
 - Index.cshtml

Xem xét ví dụ về phân cấp thư mục trước đó, có một số nguyên tắc cần ghi nhớ khi xác định các khu vực:

- Một thư mục được gọi là Khu vực phải tồn tại như một thư mục con của dự án.
- Thư mục Khu vực chứa một thư mục con cho từng khu vực trong dự án của bạn (Sản phẩm và Dịch vụ, trong này thí dụ).
- Bộ điều khiển của bạn phải được định vị như sau: /Areas/[area]/Controllers/[controller].cs
- Chế độ xem của bạn sẽ được đặt như sau: /Areas/[area]/Views/[controller]/[action].cshtml

Lưu ý rằng nếu bạn có chế độ xem được chia sẻ trên các bộ điều khiển, thì chế độ xem đó có thể được đặt ở một trong các vị trí sau:

- /Areas/[area]/Views/Shared/[action].cshtml
- /Views/Shared/[action].cshtml

Khi bạn đã xác định phân cấp thư mục, bạn cần cho MVC biết rằng mỗi bộ điều khiển được liên kết với một khu vực. Bạn làm điều đó bằng cách trang trí tên bộ điều khiển với thuộc tính [Khu vực].

```
...
không gian tên MyStore.Areas.Products.Controllers {
    [Area ("Products")]
    public class HomeController : Controller {
        // GET: / <controller> /
        public IActionResult Index () {
            return View ();
        }
    }
}
```

Bước cuối cùng là thiết lập định nghĩa tuyến đường phù hợp với các khu vực mới tạo của bạn. Bài viết Routing to Controller Actions đi sâu vào chi tiết về cách tạo các định nghĩa tuyến, bao gồm cả việc sử dụng các tuyến thông thường so với các tuyến thuộc tính. Trong ví dụ này, chúng tôi sẽ sử dụng một tuyến đường thông thường. Để làm như vậy, chỉ cần mở tệp Startup.cs và sửa đổi nó bằng cách thêm định nghĩa tuyến đường được đánh dấu bên dưới.

```
...
app.UseMvc (các tuyến đường => {
    route.MapRoute (name: "areaRoute",
        template: "{area: being} / {controller = Home} / {action = Index}");

    route.MapRoute
        (name: "default",
        template: "{controller = Home} / {action = Index}");
});
```

Bây giờ, khi người dùng duyệt đến `http://<yourApp>/products`, phương thức hành động Index của HomeController trong vùng Products sẽ được gọi.

6.7.1 Liên kết giữa các khu vực

Để liên kết giữa các khu vực, bạn chỉ cần chỉ định khu vực mà bộ điều khiển được xác định. Nếu bộ điều khiển không phải là một phần của một khu vực, hãy sử dụng một chuỗi trống.

Đoạn mã sau đây cho biết cách liên kết đến một hành động của bộ điều khiển được xác định trong một khu vực có tên Sản phẩm.

```
@ Html.ActionLink ("Xem Trang chủ Sản phẩm", "Chi mục", "Trang chủ", mới {area = "Products" }, null)
```

Để liên kết đến một hành động của bộ điều khiển không phải là một phần của một khu vực, chỉ cần chỉ định một chuỗi trống cho khu vực đó.

```
@ Html.ActionLink ("Đi tới Trang chủ", "Chi mục", "Trang chủ", mới {area = "" }, null)
```

6.7.2 Tóm tắt

Các khu vực là một công cụ rất hữu ích để nhóm các bộ điều khiển và hành động liên quan đến ngữ nghĩa trong một thư mục mẹ chung. Trong bài viết này, bạn đã học cách thiết lập hệ thống phân cấp thư mục để hỗ trợ Khu vực, cách chỉ định thuộc tính [Khu vực] để biểu thị bộ điều khiển thuộc một khu vực cụ thể và cách xác định các tuyến đường của bạn với các khu vực.

6.8 Làm việc với Mô hình Ứng dụng

Lưu ý: Chúng tôi hiện đang làm việc về chủ đề này.

Chúng tôi hoan nghênh ý kiến đóng góp của bạn để giúp định hình phạm vi và cách tiếp cận. Bạn có thể theo dõi trạng thái và cung cấp thông tin đầu vào về vấn đề này tại GitHub.

Nếu bạn muốn xem lại các bản nháp và phác thảo ban đầu của chủ đề này, vui lòng để lại ghi chú với thông tin liên hệ của bạn trong số này.

Tìm hiểu thêm về cách bạn có thể đóng góp trên GitHub.

Màn biểu diễn

7.1 Bộ nhớ đệm phản hồi

Bởi Steve Smith

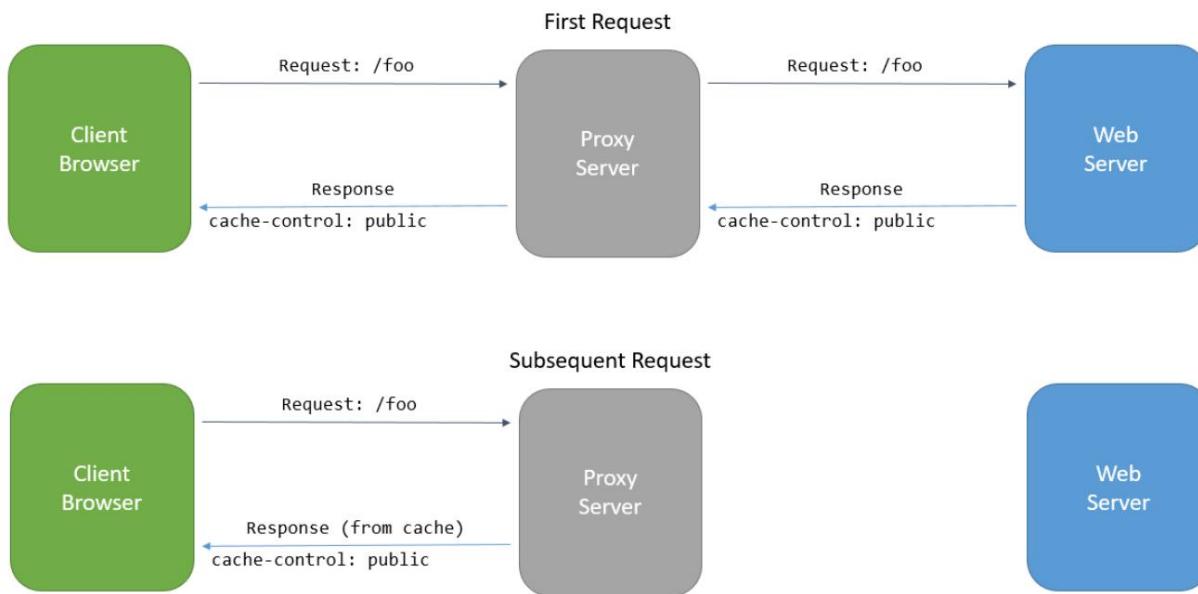
Trong bài viết này:

- Bộ nhớ đệm phản hồi là gì
- Thuộc tính ResponseCache

Xem hoặc tải xuống mẫu từ GitHub.

7.1.1 Bộ nhớ đệm phản hồi là gì

Bộ nhớ đệm phản hồi đề cập đến việc chỉ định các tiêu đề liên quan đến bộ nhớ cache trên các phản hồi HTTP được thực hiện bởi các hành động ASP.NET MVC. Các tiêu đề này chỉ định cách bạn muốn các máy khách và máy trung gian (proxy) lưu vào bộ nhớ cache các phản hồi đối với các yêu cầu nhất định (nếu có). Điều này có thể làm giảm số lượng yêu cầu mà máy khách hoặc proxy thực hiện đối với máy chủ web, vì các yêu cầu tương tự trong tương lai có thể được phục vụ từ bộ đệm ẩn của máy khách hoặc proxy. Trong trường hợp này, yêu cầu không bao giờ được đưa ra trên web người phục vụ.



Tiêu đề HTTP chính được sử dụng để lưu vào bộ nhớ đệm là Cache-Control. Đặc tả HTTP 1.1 chi tiết nhiều tùy chọn cho chỉ thị này. Ba chỉ thị phổ biến là:

```
public Chỉ ra rằng phản hồi có thể được lưu vào bộ nhớ đệm.

private Cho biết phản hồi dành cho một người dùng duy nhất và không được lưu trong bộ nhớ đệm được chia sẻ. Phản hồi vẫn có thể được lưu trong bộ đệm riêng (ví dụ: bởi trình duyệt của người dùng).

no-cache Cho biết phản hồi không được sử dụng bởi bộ đệm để đáp ứng bất kỳ yêu cầu tiếp theo nào (không thành công xác thực lại với máy chủ gốc).
```

Lưu ý: Bộ nhớ đệm phản hồi không lưu vào bộ nhớ cache các phản hồi trên máy chủ web. Nó khác với bộ nhớ đệm đầu ra, sẽ lưu vào bộ nhớ cache các phản hồi trong bộ nhớ trên máy chủ trong các phiên bản ASP.NET và ASP.NET MVC trước đó. Phần mềm trung gian bộ nhớ đệm đầu ra dự kiến sẽ được thêm vào ASP.NET MVC 6 trong một bản phát hành trong tương lai.

Các tiêu đề HTTP bổ sung được sử dụng để lưu vào bộ nhớ đệm bao gồm Pragma và Vary, được mô tả bên dưới. Tìm hiểu thêm về Cache trong HTTP từ thông số kỹ thuật.

7.1.2 Thuộc tính ResponseCache

Thuộc tính `ResponseCache` được sử dụng để chỉ định cách đặt tiêu đề của hành động bộ điều khiển để kiểm soát hành vi bộ nhớ cache của nó. Thuộc tính có các thuộc tính sau, tất cả đều là tùy chọn trừ khi có ghi chú khác.

`Thời lượng int` Thời lượng tối đa (tính bằng giây) phản hồi phải được lưu vào bộ nhớ đệm. Bắt buộc trừ khi `NoStore` là thật.

`Location ResponseCacheLocation` Vị trí nơi phản hồi có thể được lưu vào bộ nhớ cache. Có thể là `Bất kỳ`, `Không có`, hoặc `Khách hàng`. Mặc định là `Bất kỳ`.

`NoStore bool` Xác định xem giá trị có nên được lưu trữ hay không và ghi đè các giá trị thuộc tính khác. Khi đúng, Thời lượng bị bỏ qua và Vị trí bị bỏ qua cho các giá trị khác với Không có.

`Chuỗi VaryByHeader` Khi được đặt, một tiêu đề phản hồi khác nhau sẽ được viết cùng với phản hồi.

`Chuỗi CacheProfileName` Khi được đặt, hãy xác định tên của cấu hình bộ đệm để sử dụng.

`Order int` Thứ tự của bộ lọc (từ `IOrderedFilter`).

`ResponseCacheAttribute` được sử dụng để cấu hình và tạo (qua `IFilterFactory`) một `ResponseCacheFilter` thực hiện công việc ghi các tiêu đề HTTP thích hợp cho phản hồi. Bộ lọc trước tiên sẽ loại bỏ bất kỳ tiêu đề hiện có nào cho `Vary`, `Cache-Control` và `Pragma`, sau đó sẽ viết ra các tiêu đề thích hợp dựa trên các thuộc tính được đặt trong `ResponseCacheAttribute`.

Tiêu đề Vary

Tiêu đề này chỉ được viết khi để xuất `VaryByHeader` được đặt, trong trường hợp này, nó được đặt thành giá trị của thuộc tính đó.

NoStore và Location.None

`NoStore` là một thuộc tính đặc biệt ghi đè hầu hết các thuộc tính khác. Khi thuộc tính này là `true`, tiêu đề `Cache-Control` sẽ được đặt thành `"no-store"`. Ngoài ra, nếu Vị trí được đặt thành `Không`, thì `Cache-Control` sẽ được đặt thành `"no-store, no-cache"` và `Pragma` cũng được đặt thành `no-cache`. (Nếu `NoStore` là `false` và `Location` là `None`, thì cả `Cache-Control` và `Pragma` sẽ được đặt thành `no-cache`).

Một tình huống tốt để đặt `NoStore` thành `true` là các trang lỗi. Có thể bạn sẽ không muốn phản hồi yêu cầu của người dùng với phản hồi lỗi mà người dùng khác đã tạo trước đó và những phản hồi như vậy có thể bao gồm dấu vết ngăn xếp và thông tin nhạy cảm khác không được lưu trữ trên máy chủ `intermediate`. Ví dụ:

```
[ResponseCache (Location = ResponseCacheLocation.None, NoStore = true)]
public IActionResult Error ()
{
    return View ();
}
```

Điều này sẽ dẫn đến các tiêu đề sau:

```
Cache-Control: no-store, no-cache
Pragma: no-cache
```

Vị trí và Thời lượng

Để bật bộ nhớ đệm, Thời lượng phải được đặt thành giá trị dương và Vị trí phải là Bất kỳ (mặc định) hoặc Khách hàng. Trong trường hợp này, tiêu đề Cache-Control sẽ được đặt thành giá trị vị trí, theo sau là "max-age" của Phản hồi.

Lưu ý: Các tùy chọn của Vị trí của Bất kỳ và Khách hàng chuyển thành các giá trị tiêu đề Kiểm soát bộ nhớ cache của công khai và tư nhân, tương ứng. Như đã lưu ý trước đó, đặt Vị trí thành Không sẽ đặt cả Kiểm soát bộ nhớ cache và Tiêu đề Pragma để không có bộ nhớ cache.

Dưới đây là ví dụ hiển thị các tiêu đề được tạo bằng cách đặt Thời lượng và giá trị Vị trí mặc định.

```
[ResponseCache (Duration = 60)]
```

Tạo các tiêu đề sau:

```
Cache-Control: public, max-age = 60
```

Cấu hình bộ nhớ đệm

Thay vì sao chép cài đặt ResponseCache trên nhiều thuộc tính hành động của bộ điều khiển, cấu hình bộ đệm có thể được coi là tùy chọn khi thiết lập MVC trong phương thức ConfigureServices trong Khởi động. Các giá trị được tìm thấy trong một cấu hình bộ đệm được tham chiếu sẽ được sử dụng làm giá trị mặc định bởi thuộc tính ResponseCache và sẽ bị ghi đè bởi bất kỳ thuộc tính được chỉ định trên thuộc tính.

Thiết lập cấu hình bộ nhớ cache:

```
1 public void ConfigureServices (dịch vụ IServiceCollection)
2 {
3     services.AddMvc (tùy chọn =>
4     {
5         options.CacheProfiles.Add ("Mặc định",
6             new CacheProfile ()
7             {
8                 Thời lượng = 60
9             });
10        options.CacheProfiles.Add ("Không bao giờ",
11            new CacheProfile ()
12            {
13                Vị trí = ResponseCacheLocation.None,
14                NoStore = true
15            });
16 }
```

```
16     });
17 }
```

Tham chiếu cấu hình bộ nhớ cache:

```
[ResponseCache (CacheProfileName = "Default")]
```

Mẹo: Thuộc tính ResponseCache có thể được áp dụng cho cả hành động (phương thức) cũng như bộ điều khiển (lớp). Các thuộc tính cấp phương pháp sẽ ghi đè các cài đặt được chỉ định trong các thuộc tính cấp lớp.

Trong ví dụ sau, thuộc tính cấp lớp chỉ định khoảng thời gian là 30 trong khi thuộc tính cấp phương pháp tham chiếu đến cấu hình bộ nhớ cache với thời lượng được đặt thành 60.

```
1 [Bộ nhớ đệm đáp ứng (Thời lượng = 30)]
2 lớp công khai HomeController : Bộ điều khiển
3 {
4     [ResponseCache (CacheProfileName = "Default")]
5     Chỉ số IActionResult công khai ()
6     {
7         return View ();
.. }
```

Tiêu đề kết quả:

```
Cache-Control: public, max-age = 60
```

Bảo vệ

8.1 Bộ lọc ủy quyền

Lưu ý: Chúng tôi hiện đang làm việc về chủ đề này.

Chúng tôi hoan nghênh ý kiến đóng góp của bạn để giúp định hình phạm vi và cách tiếp cận. Bạn có thể theo dõi trạng thái và cung cấp thông tin đầu vào về vấn đề này tại GitHub.

Nếu bạn muốn xem lại các bản nháp và phác thảo ban đầu của chủ đề này, vui lòng để lại ghi chú với thông tin liên hệ của bạn trong số này.

Tìm hiểu thêm về cách bạn có thể đóng góp trên GitHub.

8.2 Thực thi SSL

Lưu ý: Chúng tôi hiện đang làm việc về chủ đề này.

Chúng tôi hoan nghênh ý kiến đóng góp của bạn để giúp định hình phạm vi và cách tiếp cận. Bạn có thể theo dõi trạng thái và cung cấp thông tin đầu vào về vấn đề này tại GitHub.

Nếu bạn muốn xem lại các bản nháp và phác thảo ban đầu của chủ đề này, vui lòng để lại ghi chú với thông tin liên hệ của bạn trong số này.

Tìm hiểu thêm về cách bạn có thể đóng góp trên GitHub.

8.3 Chống yêu cầu giả mạo

Lưu ý: Chúng tôi hiện đang làm việc về chủ đề này.

Chúng tôi hoan nghênh ý kiến đóng góp của bạn để giúp định hình phạm vi và cách tiếp cận. Bạn có thể theo dõi trạng thái và cung cấp thông tin đầu vào về vấn đề này tại GitHub.

Nếu bạn muốn xem lại các bản nháp và phác thảo ban đầu của chủ đề này, vui lòng để lại ghi chú với thông tin liên hệ của bạn trong số này.

Tìm hiểu thêm về cách bạn có thể đóng góp trên GitHub.

8.4 Xác định chính sách CORS

Bởi Mike Wasson

Bảo mật trình duyệt ngăn một trang web thực hiện các yêu cầu AJAX tới một miền khác. Hạn chế này được gọi là chính sách cùng nguồn gốc và ngăn một trang web đọc dữ liệu nhạy cảm từ một trang web khác. Tuy nhiên, đôi khi bạn có thể muốn cho phép các trang web khác thực hiện các yêu cầu nguồn gốc chéo đối với ứng dụng web của bạn.

[Chia sẻ tài nguyên nguồn gốc chéo](#) là một tiêu chuẩn W3C cho phép máy chủ nói lời chính sách nguồn gốc. Sử dụng CORS, máy chủ có thể cho phép rõ ràng một số yêu cầu có nguồn gốc chéo trong khi từ chối những yêu cầu khác. Chủ đề này hướng dẫn cách kích hoạt CORS trong ứng dụng ASP.NET MVC 6 của bạn. (Để biết thông tin cơ bản về CORS, hãy xem [Cách hoạt động của CORS](#).)

8.4.1 Thêm gói CORS

Trong tệp project.json của bạn, hãy thêm thông tin sau:

```
"phụ thuộc": {
    "Microsoft.AspNet.Cors": "6.0.0-beta8"
},
```

8.4.2 Định cấu hình CORS

Để định cấu hình CORS, hãy gọi `AddCors` trong phương thức `ConfigureServices` của lớp `Khởi động` của bạn, như được hiển thị ở đây:

```
public void ConfigureServices (dịch vụ IServiceCollection) {

    services.AddMvc ();
    services.AddCors (tùy chọn => {

        // Xác định một hoặc nhiều tùy chọn chính sách
        CORS.AddPolicy ("AllowSpecificOrigin",
            người xây dựng =>
        {
            builder.WithOrigins ("http://example.com");
        });
    });
}
```

Ví dụ này xác định chính sách CORS có tên là `"AllowSpecificOrigin"` cho phép các yêu cầu có nguồn gốc chéo từ `"http://example.com"` và không có nguồn gốc khác. Lambda nhận một đối tượng `CorsPolicyBuilder`. Để tìm hiểu thêm về các cài đặt chính sách CORS khác nhau, hãy xem [các tùy chọn chính sách CORS](#).

8.4.3 Áp dụng chính sách CORS

Bước tiếp theo là áp dụng các chính sách. Bạn có thể áp dụng chính sách CORS cho mỗi hành động, mỗi bộ điều khiển hoặc trên toàn cầu cho tất cả bộ điều khiển trong ứng dụng của mình.

[Mỗi hành động](#)

Thêm thuộc tính `[EnableCors]` vào hành động. Chỉ định tên chính sách.

```
public class HomeController : Controller {  
  
    [EnableCors ("AllowSpecificOrigin")] chỉ số  
    IActionResult công khai () {  
  
        return View ();  
    }  
}
```

Mỗi bộ điều khiển

Thêm thuộc tính [EnableCors] vào lớp bộ điều khiển. Chỉ định tên chính sách.

```
[EnableCors ("AllowSpecificOrigin")] lớp công khai  
HomeController : Bộ điều khiển {  
}
```

Toàn cầu

Thêm bộ lọc CorsAuthorizationFilterFactory vào bộ sưu tập bộ lọc chung:

```
public void ConfigureServices (dịch vụ IServiceCollection) {  
  
    services.AddMvc ();  
    services.Configure <MvcOptions> (options => {  
  
        options.Filters.Add ( CorsAuthorizationFilterFactory mới ("AllowSpecificOrigin"));  
    });  
}
```

Thứ tự ưu tiên là: Hành động, bộ điều khiển, toàn cầu. Các chính sách cấp hành động được ưu tiên hơn các chính sách cấp kiểm soát và các chính sách cấp kiểm soát được ưu tiên hơn các chính sách toàn cầu.

Tắt CORS

Để tắt CORS cho bộ điều khiển hoặc hành động, hãy sử dụng thuộc tính [DisableCors].

```
[DisableCors]  
public IActionResult About () {  
  
    return View ();  
}
```


Di cư

9.1 Di chuyển từ ASP.NET MVC 5 sang MVC 6

Bởi Rick Anderson, Daniel Roth, Steve Smith, và Scott Addie

Bài viết này hướng dẫn cách bắt đầu di chuyển một dự án ASP.NET MVC 5 sang ASP.NET MVC 6. Trong quá trình này, nó làm nổi bật nhiều điều đã thay đổi từ MVC 5 sang MVC 6. Di chuyển từ MVC 5 sang MVC 6 là một quy trình nhiều bước và bài viết này bao gồm thiết lập ban đầu, bộ điều khiển và chế độ xem cơ bản, nội dung tĩnh và phụ thuộc máy khách. Các bài viết bổ sung đề cập đến việc di chuyển các mô hình ASP.NET Identity, mã khởi động và cấu hình được tìm thấy trong nhiều dự án MVC 5.

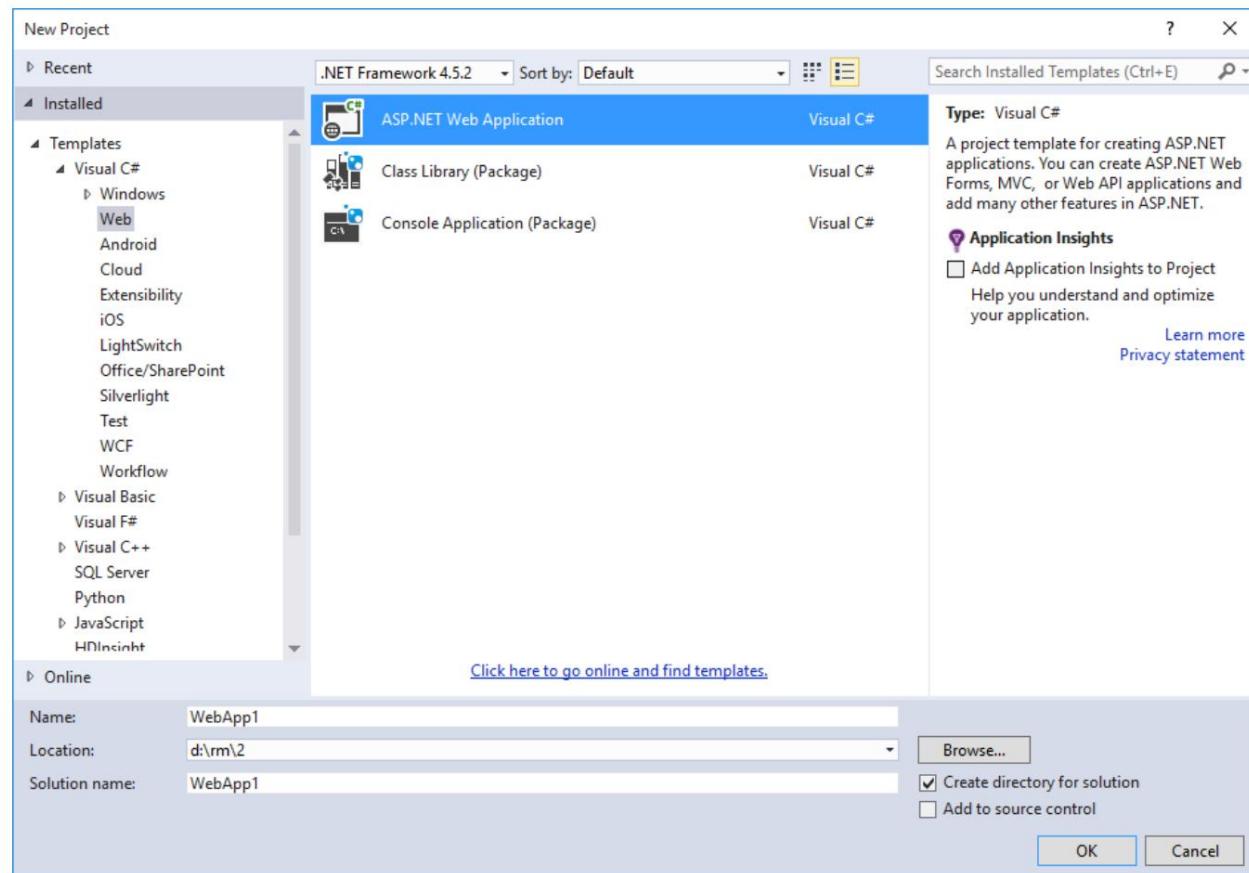
Trong bài viết này:

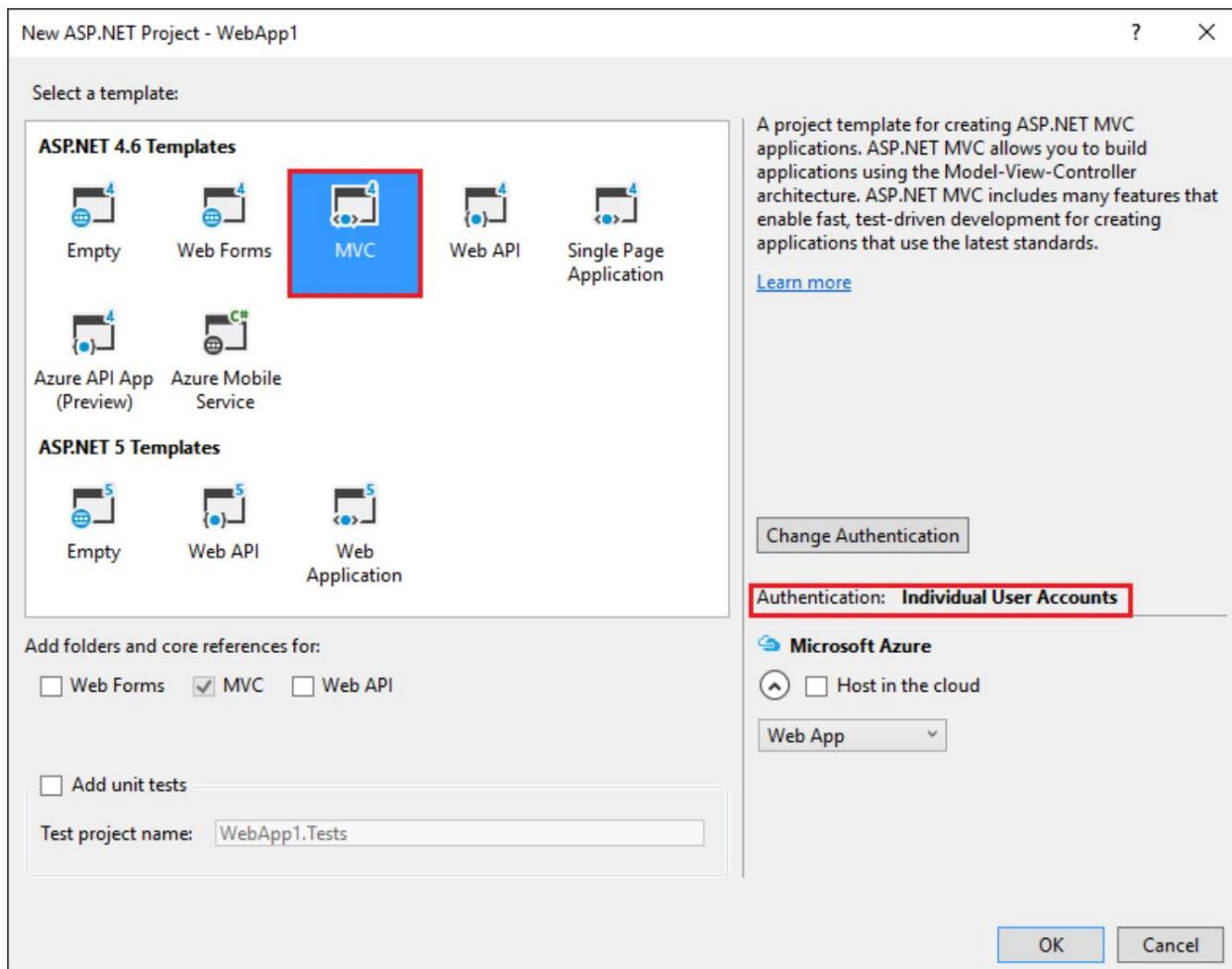
- Tạo dự án MVC 5 khởi động
- Tạo dự án MVC 6
- Định cấu hình trang web để sử dụng MVC
- Thêm bộ điều khiển và chế độ xem
- Bộ điều khiển và chế độ xem
- Nội dung tĩnh
- Gulp
- NPM
- Di chuyển tệp bổ sung
- Định cấu hình gói
- Tài nguyên bổ sung

9.1.1 Tạo dự án MVC 5 khởi động

Để chứng minh nâng cấp, chúng tôi sẽ bắt đầu bằng cách tạo một ứng dụng ASP.NET MVC 5 mới. Tạo nó với tên WebApp1 để không tên sẽ khớp với dự án MVC 6 mà chúng ta tạo trong bước tiếp theo.

Tài liệu ASP.NET MVC 6, Bản phát hành

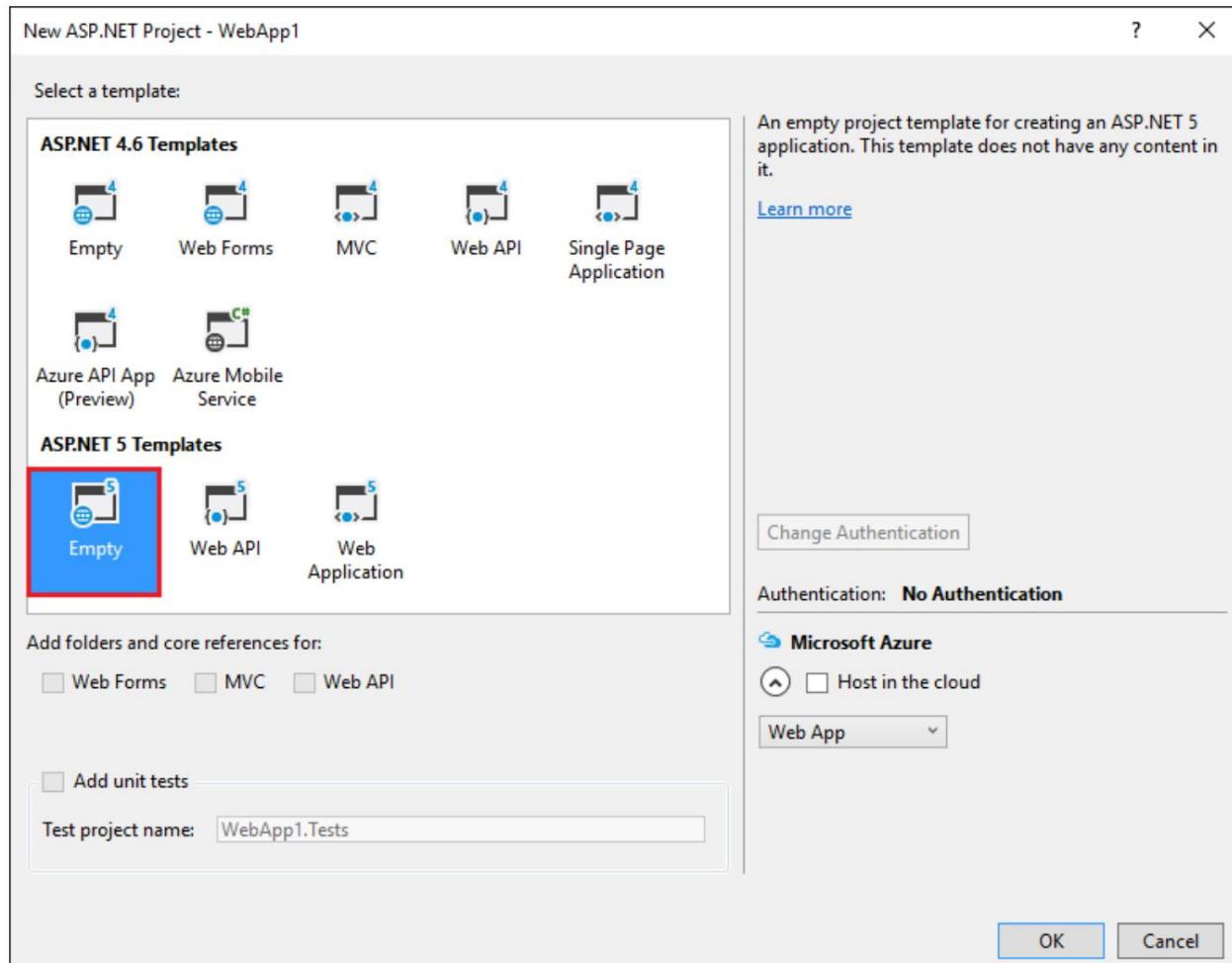




Tùy chọn: Thay đổi tên của Giải pháp từ WebApp1 thành Mvc5. Visual Studio sẽ hiển thị tên giải pháp mới (Mvc5), điều này sẽ giúp bạn dễ dàng phân biệt dự án này với dự án tiếp theo. Bạn có thể cần phải thoát khỏi Visual Studio và sau đó tải lại dự án để xem tên mới.

9.1.2 Tạo dự án MVC 6

Tạo một ứng dụng web MVC 6 trống mới có cùng tên với dự án trước đó (WebApp1) để không gian tên trong hai dự án khớp với nhau. Có cùng một không gian tên giúp sao chép mã giữa hai dự án dễ dàng hơn. Bạn sẽ phải tạo dự án này trong một thư mục khác với dự án trước đó để sử dụng cùng một tên.



- Tùy chọn: Tạo ứng dụng MVC 6 mới có tên WebApp1 với xác thực được đặt thành Tài khoản người dùng cá nhân. Đổi tên ứng dụng này FullMVC6. Tạo dự án này sẽ giúp bạn tiết kiệm thời gian chuyển đổi. Bạn có thể nhìn vào mẫu được tạo mã để xem kết quả cuối cùng hoặc để sao chép mã vào dự án chuyển đổi. Nó cũng hữu ích khi bạn gặp khó khăn ở bước chuyển đổi để so sánh với dự án đã tạo mẫu.

9.1.3 Định cấu hình trang web để sử dụng MVC

- Mở tệp project.json và thêm Microsoft.AspNet.Mvc và Microsoft.AspNet.StaticFiles vào thuộc tính phụ thuộc và phần tập lệnh như được đánh dấu bên dưới:

```

1  {
2      "phiên bản": "1.0.0-*",
3      "compilationOptions": {
4          "releaseEntryPoint": true
5      },
6
7      "phụ thuộc": {
8          "Microsoft.AspNet.StaticFiles": "1.0.0-rc1-final",
9          "Microsoft.AspNet.Mvc": "6.0.0-rc1-final",
10         "Microsoft.AspNet.IISPlatformHandler": "1.0.0-rc1-final",
11         "Microsoft.AspNet.Server.Kestrel": "1.0.0-rc1-final"
12     },
13

```

```

14     "lệnh": {
15         "web": "Microsoft.AspNet.Server.Kestrel"
16     },
17
18     "khuôn khổ": {
19         "dnx451": {},
20         "dnxcore50": {}
21     },
22
23     "loại trừ": [
24         "wwwroot",
25         "node_modules"
26     ],
27     "PublishingExclude": [
28         "**.người sử dụng",
29         "**. vspscc"
30     ],
31     "script": {
32         "prepublish": [ "npm install", "bower install", "gulp clean", "gulp min" ]
33     }
34 }

```

`Microsoft.AspNet.StaticFiles` là trình xử lý tệp tĩnh. Thời gian chạy ASP.NET là mô-đun và bạn phải chọn tham gia một cách rõ ràng để cung cấp tệp tĩnh (xem [Làm việc với tệp tĩnh](#)).

Phần tập lệnh được sử dụng để biểu thị thời điểm các tập lệnh tự động hóa xây dựng được chỉ định sẽ chạy. Visual Studio hiện có hỗ trợ tích hợp để chạy các tập lệnh trước và sau các sự kiện cụ thể. Phần tập lệnh ở trên chỉ định NPM, Bower và Gulp script sẽ chạy ở giai đoạn trước khi xuất bản. Chúng ta sẽ nói về NPM, Bower và Gulp sau trong phần hướng dẫn. Lưu ý dấu vết `,"` được thêm vào cuối phần xuất bản Loại trừ.

Để biết thêm thông tin, hãy xem [project.json](#) và [Giới thiệu .NET Core](#).

- Mở tệp Startup.cs và thay đổi mã để khớp với mã sau:

```

1 lớp công khai Khởi động
2 {
3     // Phương thức này được gọi bởi thời gian chạy. Sử dụng phương pháp này để thêm dịch vụ vào vùng chứa.
4     // Để biết thêm thông tin về cách định cấu hình ứng dụng của bạn, hãy truy cập http://go.microsoft.com/fwlink
5     public void ConfigureServices (dịch vụ IServiceProvider)
6     {
7         services.AddMvc ();
8     }
9
10    // Phương thức này được gọi bởi thời gian chạy. Sử dụng phương pháp này để định cấu hình pipeline yêu cầu HTTP
11    public void Configure (ứng dụng IApplicationBuilder)
12    {
13        app.UseIISPlatformHandler ();
14
15        app.UseStaticFiles ();
16
17        app.UseMvc (các tuyến đường =>
18        {
19            các tuyến đường.MapRoute (
20                tên: "mặc định",
21                mẫu: "{controller = Home} / {action = Index} / {id?}");
22        });
23    }

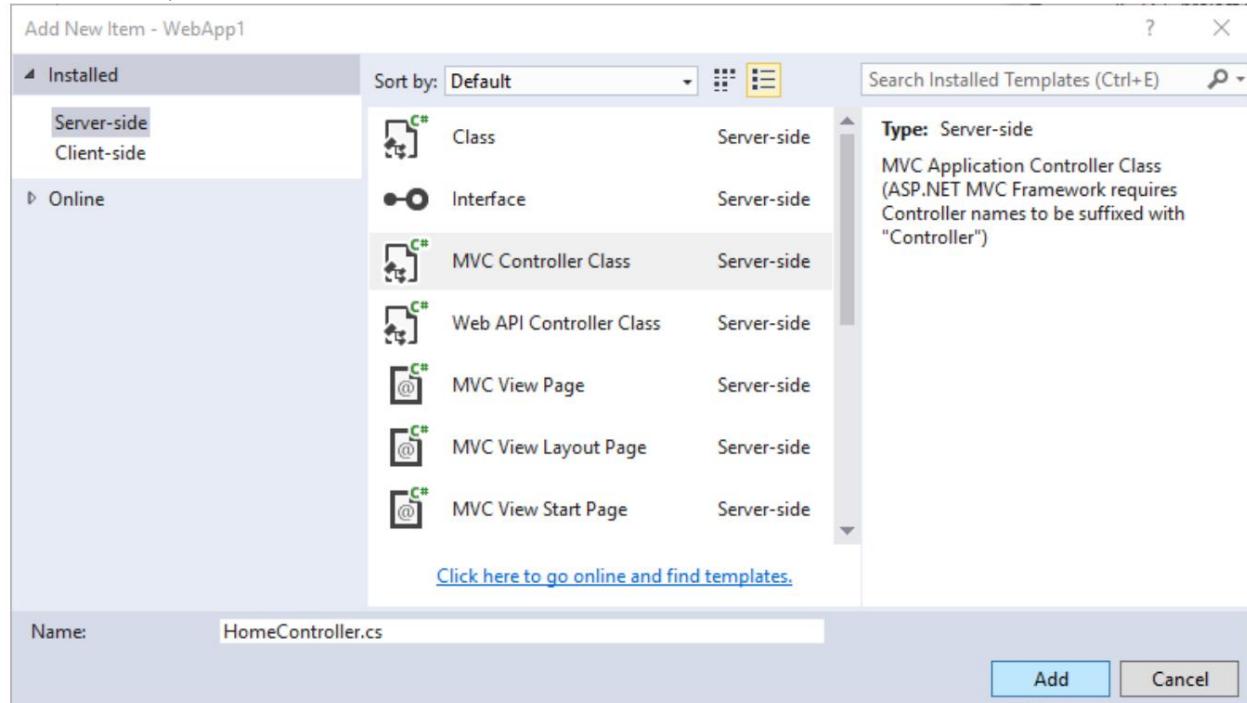
```

UseStaticFiles thêm trình xử lý tệp tĩnh. Như đã đề cập trước đây, thời gian chạy ASP.NET là mô-đun và bạn phải chọn tham gia một cách rõ ràng để cung cấp các tệp tĩnh. Để biết thêm thông tin, hãy xem [Khởi động ứng dụng và Định tuyến](#).

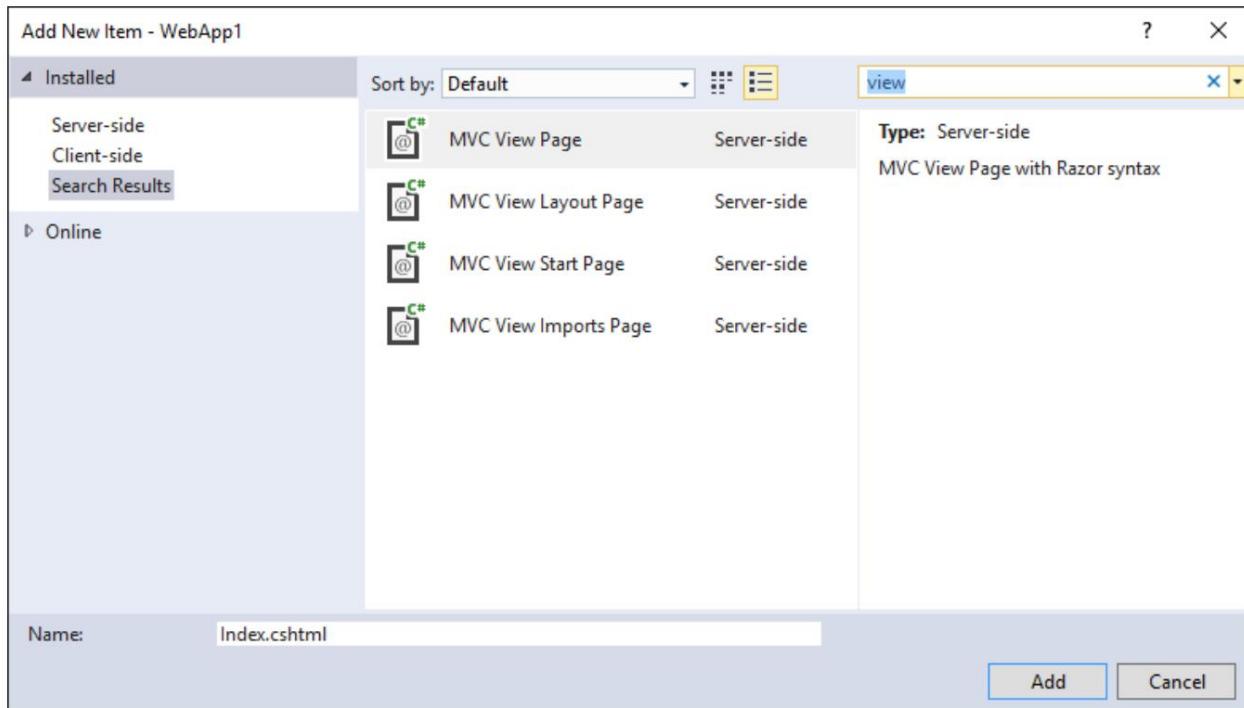
9.1.4 Thêm bộ điều khiển và chế độ xem

Trong phần này, bạn sẽ thêm một bộ điều khiển và chế độ xem tối thiểu để phục vụ nhu cầu các trình giữ chỗ cho bộ điều khiển MVC 5 và các chế độ xem bạn sẽ di chuyển trong phần tiếp theo.

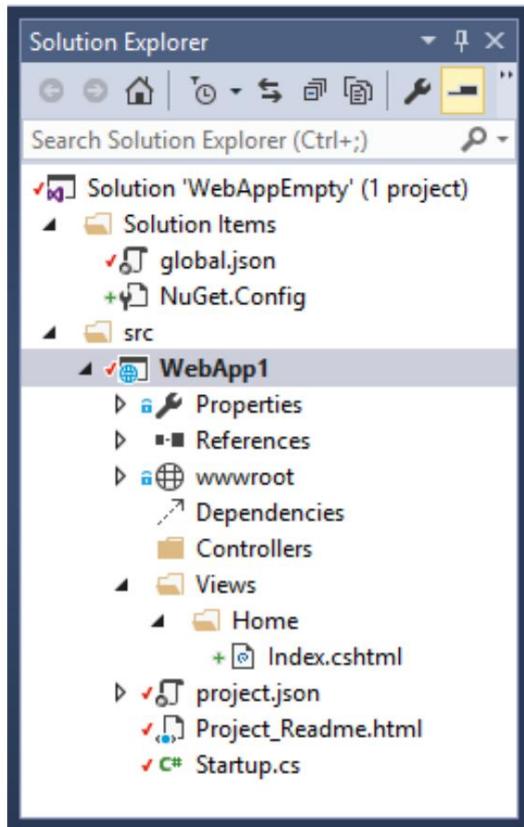
- Thêm thư mục Bộ điều khiển.
- Thêm một lớp bộ điều khiển MVC với tên HomeController.cs vào thư mục Bộ điều khiển.



- Thêm thư mục Chế độ xem.
- Thêm thư mục Lượt xem / Trang chủ.
- Thêm trang xem Index.cshtml MVC vào thư mục Lượt xem / Trang chủ.



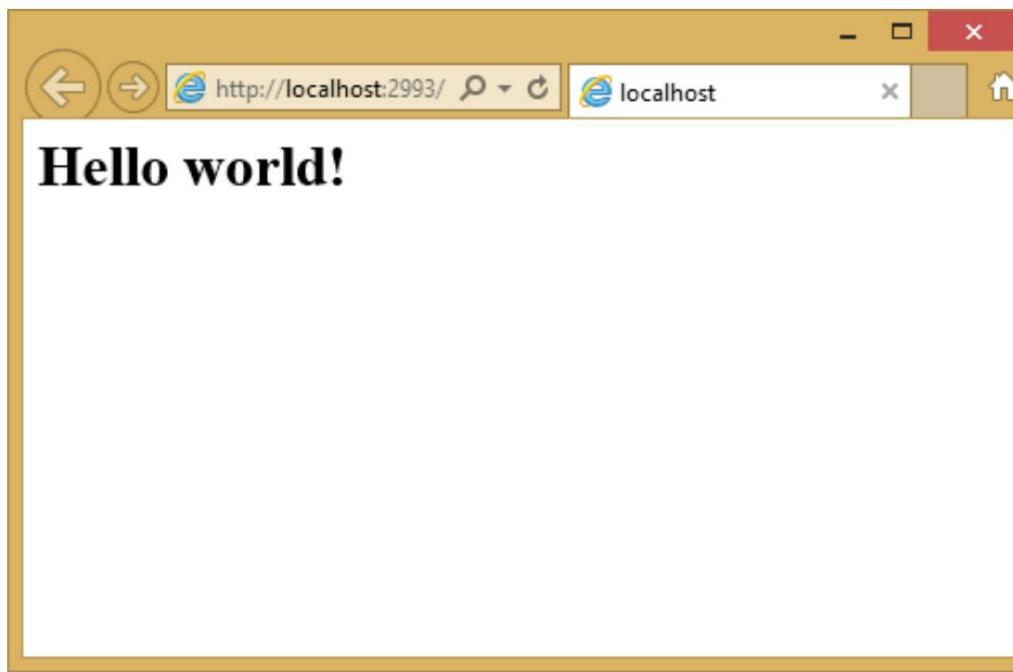
Cấu trúc dự án được hiển thị dưới đây:



Thay thế nội dung của tệp Views / Home / Index.cshtml bằng nội dung sau:

```
<h1> Xin chào thế giới! </h1>
```

Chạy ứng dụng.



Xem Bộ điều khiển và Chế độ xem để biết thêm thông tin.

Bây giờ chúng ta có một dự án MVC 6 hoạt động tối thiểu, chúng ta có thể bắt đầu di chuyển chức năng từ dự án MVC 5. Chúng tôi sẽ cần di chuyển những điều sau:

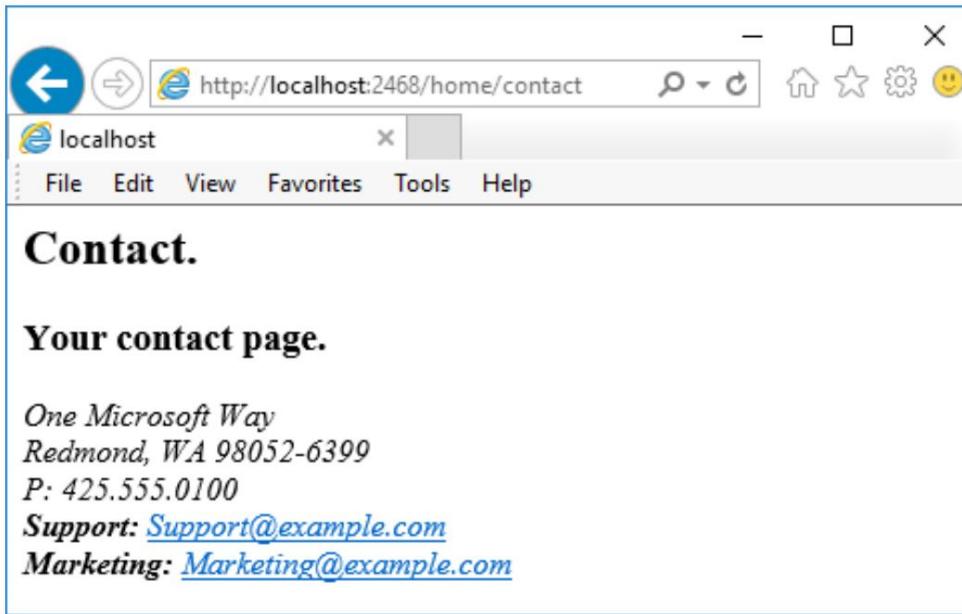
- nội dung phía khách (CSS, phông chữ và tập lệnh)
 - bộ điều khiển
 - lượt xem
 - mô hình
 - đóng gói
 - bộ lọc
- Đăng nhập / đăng xuất, danh tính (Điều này sẽ được thực hiện trong phần hướng dẫn tiếp theo.)

9.1.5 Bộ điều khiển và khung nhìn

- Sao chép từng phương thức từ HomeController MVC 5 sang HomeController MVC 6. Lưu ý rằng trong MVC 5, kiểu trả về phương thức hành động của bộ điều khiển của mẫu tích hợp sẵn là `ActionResult`; trong MVC 6, các mẫu tạo ra `IActionResult` các phương pháp. `ActionResult` là cách triển khai duy nhất của `IActionResult`, vì vậy không cần thay đổi kiểu trả về của các phương thức hành động của bạn.
- Xóa chế độ xem `Views / Home / Index.cshtml` trong dự án MVC 6.
- Sao chép các tệp xem `About.cshtml`, `Contact.cshtml` và `Index.cshtml` Razor từ dự án MVC 5 sang MVC 6 dự án.
- Chạy ứng dụng MVC 6 và kiểm tra từng phương pháp. Chúng tôi chưa di chuyển tệp bô cục hoặc các kiểu, vì vậy các dạng xem được kết xuất sẽ chỉ chứa nội dung trong tệp dạng xem. Bạn sẽ không có các liên kết được tạo tệp bô cục cho

Ché độ xem Giới thiệu và Liên hệ, vì vậy bạn sẽ phải gọi chúng từ trình duyệt (thay thế 2468 bằng số cổng được sử dụng trong dự án của bạn).

- http://localhost:2468/home/about
- http://localhost:2468/home/contact



Lưu ý thiếu các mục kiểu dáng và menu. Chúng tôi sẽ khắc phục điều đó trong phần tiếp theo.

9.1.6 Nội dung tinh

Trong các phiên bản trước của MVC (bao gồm cả MVC 5), nội dung tinh được lưu trữ từ thư mục gốc của dự án web và được trộn lẫn với các tệp phía máy chủ. Trong MVC 6, nội dung tinh được lưu trữ trong thư mục wwwroot. Bạn sẽ muốn sao chép nội dung tinh từ ứng dụng MVC 5 của mình vào thư mục wwwroot trong dự án MVC 6 của bạn. Trong chuyển đổi mẫu này:

- Sao chép tệp favicon.ico từ dự án MVC 5 vào thư mục wwwroot trong dự án MVC 6.

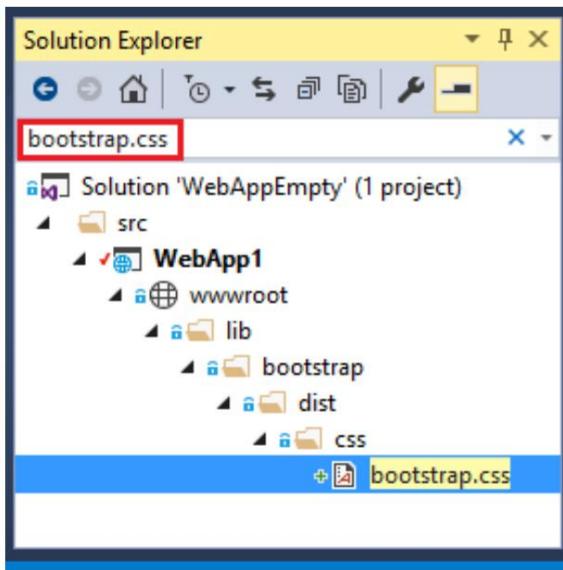
Dự án MVC 5 sử dụng [Bootstrap](#) để tạo kiểu và lưu trữ các tệp Bootstrap trong các thư mục Nội dung và Tập lệnh. Dự án MVC 5 được tạo theo mẫu tham chiếu Bootstrap trong tệp bố cục (Views / Shared / _Layout.cshtml). Bạn có thể sao chép các tệp bootstrap.js và bootstrap.css từ dự án MVC 5 vào thư mục wwwroot trong dự án mới, nhưng cách tiếp cận đó không sử dụng cơ chế cải tiến để quản lý các phụ thuộc phía máy khách trong ASP.NET 5.

Trong dự án mới, chúng tôi sẽ thêm hỗ trợ cho Bootstrap (và các thư viện phía máy khách khác) bằng cách sử dụng [Bower](#):

- Thêm một Bower tệp cấu hình có tên bower.json vào gốc dự án (Bấm chuột phải vào dự án, sau đó Thêm > Mục mới > Tệp cấu hình Bower). Thêm [Bootstrap](#) và [jquery](#) vào tệp (xem các dòng được đánh dấu bên dưới).

```
{
  "name": "ASP.NET",
  "private": true,
  "dependencies": {
    "bootstrap": "3.3.5",
    "jquery": "2.1.4"
  }
}
```

Sau khi lưu tệp, Bower sẽ tự động tải các phần phụ thuộc vào thư mục wwwroot / lib. Bạn có thể sử dụng hộp Trình khám phá giải pháp tìm kiếm để tìm đường dẫn của nội dung.



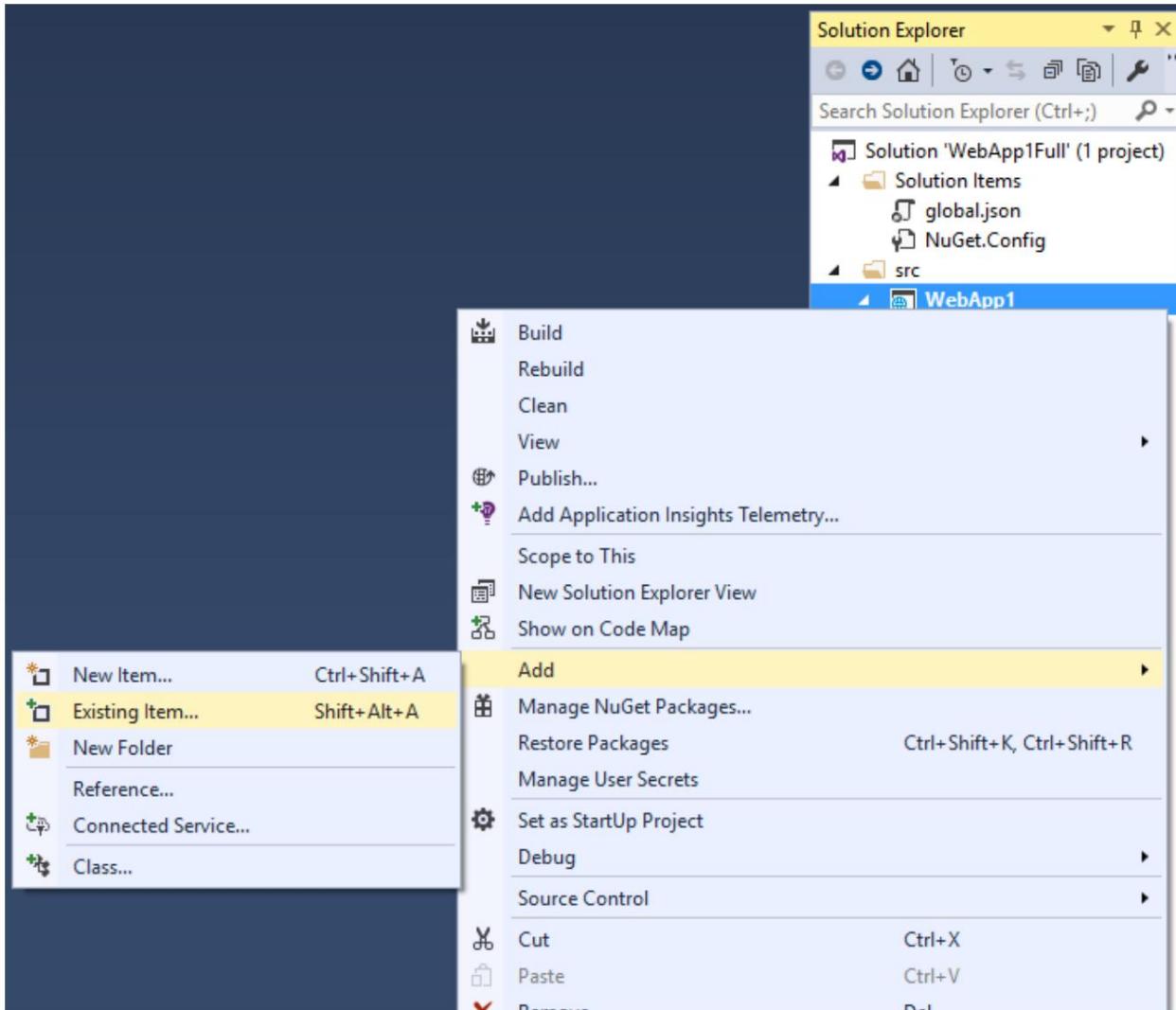
Xem Quản lý các gói phía máy khách với Bower để biết thêm thông tin.

9.1.7 Gulp

Khi bạn tạo ứng dụng web mới bằng mẫu **Ứng dụng web ASP.NET 5**, dự án được thiết lập để sử dụng **Gulp**.

Gulp là một hệ thống xây dựng phát triển trực tuyến cho mã phía máy khách (HTML, LESS, SASS, v.v.). Gulpfile.js được bao gồm trong dự án chứa JavaScript xác định một tập hợp các tác vụ gulp mà bạn có thể đặt để chạy tự động trên các sự kiện xây dựng hoặc bạn có thể chạy theo cách thủ công bằng Task Runner Explorer trong Visual Studio. Trong phần này, chúng tôi sẽ hướng dẫn cách sử dụng tệp gulpfile.js được tạo từ mẫu MVC 6 để đóng gói và rút gọn các tệp JavaScript và CSS trong dự án.

Nếu bạn đã tạo dự án FullMVC6 tùy chọn (ứng dụng web ASP.NET MVC 6 mới với Tài khoản Người dùng Cá nhân), hãy thêm gulpfile.js từ dự án đó vào dự án mà chúng tôi đang cập nhật. Trong Solution Explorer, bấm chuột phải vào dự án **Ứng dụng web** và chọn **Thêm> Mục hiện có**.



Điều hướng đến gulpfile.js từ ứng dụng web ASP.NET MVC 6 mới với Tài khoản Người dùng Cá nhân và thêm tệp thêm gulpfile.js. Ngoài ra, nhấp chuột phải vào dự án Ứng dụng web và chọn Thêm> Mục mới. Chọn Tệp cấu hình Gulp và đặt tên tệp là gulpfile.js. Thay thế nội dung của tệp gulp bằng nội dung sau:

```
/// <xsd:element name="Clean" type="xsd:string">
<xsd:complexType>
    <xsd:sequence>
        <xsd:element name="value" type="xsd:string" />
    </xsd:sequence>
</xsd:complexType>
</xsd:element>

<xsd:element name="CopyTo" type="xsd:string">
<xsd:complexType>
    <xsd:sequence>
        <xsd:element name="source" type="xsd:string" />
        <xsd:element name="destination" type="xsd:string" />
    </xsd:sequence>
</xsd:complexType>
</xsd:element>
```

```

path.concatJsDest = path.webroot + "js / site.min.js";
path.concatCssDest = path.webroot + "css / site.min.css";

gulp.task ("clean: js", function (cb) {rimraf
    (path.concatJsDest, cb);
});

gulp.task ("clean: css", function (cb) {rimraf
    (path.concatCssDest, cb);
});

gulp.task ("sạch", ["sạch: js", "sạch: css"]);

gulp.task ("min: js", function () {
    return gulp.src ([path.js, "!" + path.minJs], {base: ".."})
        .pipe (concat (path.concatJsDest)) .pipe
        (uglify ()) .pipe (gulp.dest ("."));
});

gulp.task ("min: css", function () {
    return gulp.src ([path.css, "!" + path.minCss])
        .pipe (concat (path.concatCssDest)) .pipe
        (cssmin ()) .pipe (gulp.dest ("."));
});

gulp.task ("min", ["min: js", "min: css"]);

```

Đoạn mã trên thực hiện các chức năng sau:

- Xóa (xóa) các tệp đích.
- Thu nhỏ các tệp JavaScript và CSS.
- Gói (nối) các tệp JavaScript và CSS.

Xem [Sử dụng Gulp với ASP.NET 5 và Visual Studio](#).

9.1.8 NPM

NPM (Node Package Manager) là một trình quản lý gói được sử dụng để có được công cụ như Bower và Gulp; và, nó được hỗ trợ đầy đủ trong Visual Studio 2015. Chúng tôi sẽ sử dụng NPM để quản lý các phần phụ thuộc của Gulp.

Nếu bạn đã tạo dự án FullMVC6 tùy chọn, hãy thêm tệp NPM package.json từ dự án đó vào dự án chúng tôi đang cập nhật. Tệp NPM package.json liệt kê các phần phụ thuộc cho các quy trình xây dựng máy khách được xác định trong gulpfile.js. Nhấp chuột phải vào dự án ứng dụng web, chọn Thêm > Mục hiện có và thêm tệp NPM package.json. Ngoài ra, bạn có thể thêm tệp cấu hình NPM mới như sau:

1. Trong Solution Explorer, bấm chuột phải vào dự án.
2. Chọn Thêm > Mục mới.
3. Chọn Tệp cấu hình NPM.
4. Đặt tên mặc định: package.json.
5. Nhấp vào Thêm.

Mở tệp package.json và thay thế nội dung bằng nội dung sau:

```
{
  "name": "ASP.NET",
  "version": "0.0.0",
  "devDependencies":
  { "gulp": "3.8.11",
    "gulp-concat": "2.5.2", "gulp-
    cssmin": "0.1.7", "gulp-
    uglify": "1.2.0", "rimraf
    ": "2.2.8"
  }
}
```

Nhấp chuột phải vào gulpfile.js và chọn Task Runner Explorer. Bấm đúp vào một nhiệm vụ để chạy nó.

Để biết thêm thông tin, hãy xem [Phát triển phía máy khách trong ASP.NET 5](#).

9.1.9 Di chuyển tệp bối cảnh

- Sao chép tệp _ViewStart.cshtml từ thư mục Views của dự án MVC 5 vào thư mục Views của dự án MVC 6.
Tệp _ViewStart.cshtml không thay đổi trong MVC 6.
- Tạo thư mục Lượt xem / Chia sẻ.
- Sao chép tệp _Layout.cshtml từ thư mục Lượt xem / Chia sẻ của dự án MVC 5 vào thư mục của dự án MVC 6
Lượt xem / Thư mục chia sẻ.

Mở tệp _Layout.cshtml và thực hiện các thay đổi sau (mã hoàn thành được hiển thị bên dưới):

- Thay thế @ Styles.Render ("~/Content/css") bằng một phần tử <link> để tải bootstrap.css (xem bên dưới)
- Xóa @ Scripts.Render ("~/packles/modernizr")
- Nhận xét dòng @ Html.Partial ("_LoginPartial") (bao quanh dòng bằng @ * ... * @) - chúng tôi sẽ quay lại nó trong một hướng dẫn trong tương lai
- Thay thế @ Scripts.Render ("~/Bundles/jquery") bằng một phần tử <script> (xem bên dưới)
- Thay thế @ Scripts.Render ("~/Bundles/bootstrap") bằng một phần tử <script> (xem bên dưới)

Liên kết CSS thay thế:

```
<link rel = "stylesheet" href = "~/lib/bootstrap/dist/css/bootstrap.css" />
```

Các thẻ tập lệnh thay thế:

```
<script src = "~/lib/jquery/dist/jquery.js"> </script> <script
src = "~/lib/bootstrap/dist/js/bootstrap.js"> </script>
```

Tệp _Layout.cshtml được cập nhật được hiển thị bên dưới:

```
<!DOCTYPE html>
<html>
<head>
  <meta charset = "utf-8" /> <meta
  name = "viewport" content = "width = device-width, initial-scale = 1.0"> <title> @ ViewBag.Title
  - Ứng dụng ASP.NET của tôi </title> <link rel = "stylesheet" href = "~/lib/bootstrap/
  dist/css/bootstrap.css" /> </head> <body>
```

```

<div class = "navbar navbar-inverse navbar-fixed-top">
    <div class = "container">
        <div class = "navbar-header">
            <button type = "button" class = "navbar-toggle" data-toggle = "sập" data-target = ". nav
                <span class = "icon-bar"> <span class =
                "icon-bar"> <span class = "icon-bar"> </
            button> @ Html.ActionLink ("Tên ứng dụng",
            "Chỉ mục", "Trang chủ", mới {area = </div> <div
            class = "navbar-sập sụp đỡ">
                ""}, mới {@clas

            <ul class = "nav navbar-nav">
                <li> @ Html.ActionLink ("Trang chủ", "Chỉ mục", "Trang chủ") </li> <li>
                @ Html.ActionLink ("Giới thiệu", "Giới thiệu", "Trang chủ") </li> <li> @
                Html.ActionLink ("Liên hệ", "Liên hệ", "Trang chủ") </li> </ul> @ * @ Html.Partial
                ("_ LoginPartial") * @ </div> </div> </ div > <div class = "container body-content">

                @RenderBody ()
                <hr /> <footer>

                    <p> & sao chép; @ DateTime.Now.Year - Ứng dụng ASP.NET của tôi </p> </footer> </div>

                <script src = "~ / lib / jquery / dist / jquery.js"> </script> <script src
                = "~ / lib / bootstrap / dist / js / bootstrap.js"> </script> @RenderSection (" script
                ", bắt buộc: false) </body> </html>

```

Xem trang web trong trình duyệt. Bây giờ nó sẽ tải một cách chính xác, với các kiểu mong đợi đã có sẵn.

9.1.10 Cấu hình gói

Mẫu web khởi động MVC 5 đã sử dụng hỗ trợ thời gian chạy MVC để đóng gói. Trong ASP.NET MVC 6, tính năng thay đổi chức năng này được thực hiện như một phần của quá trình xây dựng bằng Gulp. Trước đây chúng tôi đã định cấu hình tính năng gói và thu nhỏ; tất cả những gì còn lại là thay đổi các tham chiếu đến Bootstrap, jQuery và các nội dung khác để sử dụng các phiên bản được gói và rút gọn. Bạn có thể thấy điều này được thực hiện như thế nào trong tệp bô cục (Lượt xem / Chia sẻ / _Layout.cshtml) của dự án mẫu đầy đủ. Xem [Gom lại và tối thiểu hóa](#) để biết thêm thông tin.

9.1.11 Tài nguyên bổ sung

- [Di chuyển ứng dụng ASP.NET MVC 5 sang ASP.NET 5](#)
- [Sử dụng Gulp](#)
- [Phát triển phía khách hàng trong ASP.NET 5](#)
- [Quản lý các gói phía khách hàng với Bower](#)
- [Bó và nhỏ](#)
- [Bootstrap cho ASP.NET 5](#)

9.2 Di chuyển cấu hình từ ASP.NET MVC 5 sang MVC 6

Bởi Steve Smith, Scott Addie

Trong bài trước, chúng ta đã bắt đầu di chuyển dự án ASP.NET MVC 5 sang MVC 6. Trong bài này, chúng ta chuyển tính năng cấu hình từ ASP.NET MVC 5 sang ASP.NET MVC 6.

Trong bài viết này:

- Thiết lập cấu hình •
- Di chuyển cài đặt cấu hình từ web.config • Tóm tắt

Bạn có thể tải xuống mã nguồn đã hoàn thành từ dự án được tạo trong bài viết này [tại đây](#).

9.2.1 Cấu hình thiết lập

ASP.NET 5 và ASP.NET MVC 6 không còn sử dụng các tệp Global.asax và web.config mà các phiên bản trước của ASP.NET đã sử dụng.

Trong các phiên bản trước của ASP.NET, logic khởi động ứng dụng được đặt trong một phương thức Application_StartUp trong Global.asax. Sau đó, trong ASP.NET MVC 5, một tệp Startup.cs đã được đưa vào thư mục gốc của dự án; và, nó được gọi bằng OwinStartupAttribute khi ứng dụng bắt đầu.

ASP.NET 5 (và ASP.NET MVC 6) đã hoàn toàn áp dụng cách tiếp cận này, đặt tất cả logic khởi động trong tệp Startup.cs.

Tệp web.config cũng đã được thay thế trong ASP.NET 5. Giờ đây, bản thân cấu hình có thể được định cấu hình, như một phần của quy trình khởi động ứng dụng được mô tả trong Startup.cs. Cấu hình vẫn có thể sử dụng tệp XML, nhưng thông thường các dự án ASP.NET 5 sẽ đặt các giá trị cấu hình trong tệp có định dạng JSON, chẳng hạn như appsettings.json. Hệ thống cấu hình của ASP.NET 5 cũng có thể dễ dàng truy cập các biến môi trường, có thể cung cấp vị trí an toàn và mạnh mẽ hơn cho các giá trị dành riêng cho môi trường. Điều này đặc biệt đúng đối với các bí mật như chuỗi kết nối và khóa API không nên kiểm tra trong kiểm soát nguồn.

Đối với bài viết này, chúng tôi đang bắt đầu với dự án ASP.NET MVC 6 được di chuyển một phần từ [bài viết trước](#). Để thiết lập cấu hình bằng cách sử dụng cài đặt MVC 6 mặc định, hãy thêm hàm tạo và thuộc tính sau vào lớp Startup.cs nằm trong thư mục gốc của dự án:

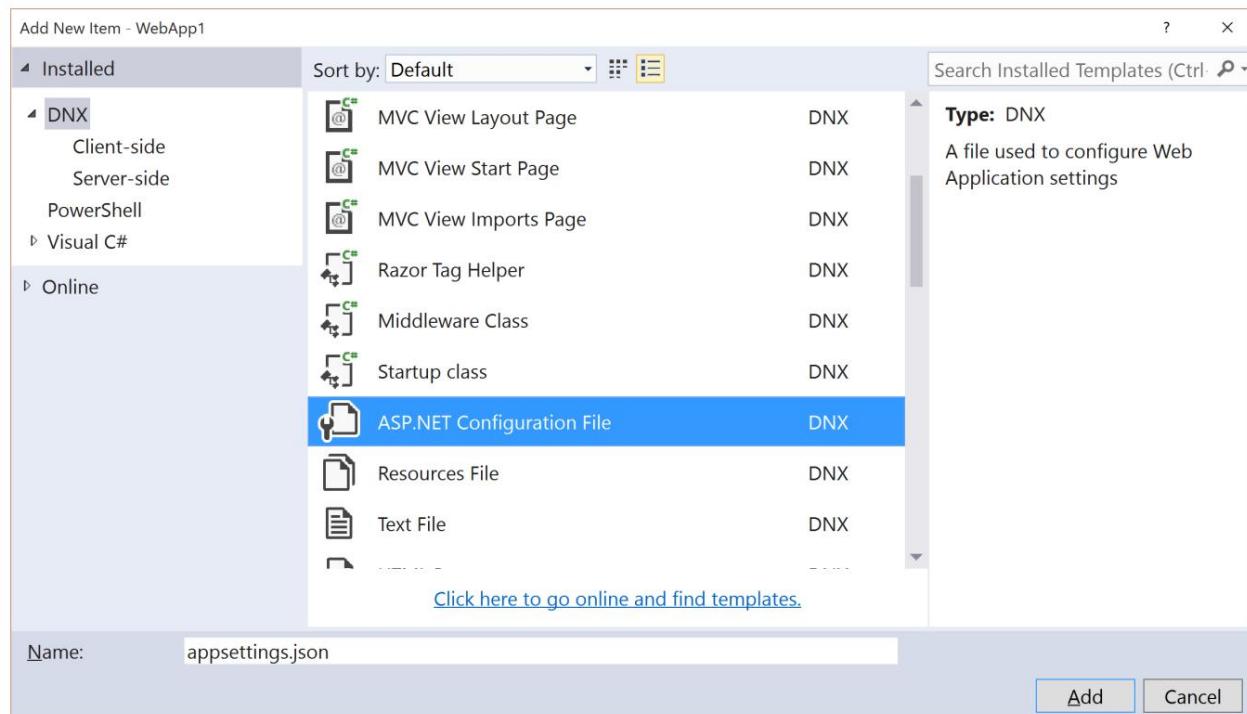
```

1 Startup công khai (IHostingEnosystem env)
2 {
3     // Thiết lập các nguồn cấu hình. var builder =
4     new ConfigurationBuilder ()
5         .AddJsonFile ("appsettings.json")
6         .AddEnosystemVariables (); Cấu hình =
7     builder.Build ();
8 }
9
10 public IConfigurationRoot Configuration { get; bộ; }
```

Lưu ý rằng tại thời điểm này, tệp Startup.cs sẽ không biên dịch, vì chúng tôi vẫn cần thêm câu lệnh using sau:

sử dụng Microsoft.Extensions.Configuration;

Thêm tệp appsettings.json vào thư mục gốc của dự án bằng cách sử dụng mẫu mục thích hợp:



9.2.2 Di chuyển cài đặt cấu hình từ web.config

Dự án ASP.NET MVC 5 của chúng tôi bao gồm chuỗi kết nối cơ sở dữ liệu bắt buộc trong web.config, trong phần tử <connectionStrings>. Trong dự án MVC 6 của chúng tôi, chúng tôi sẽ lưu trữ thông tin này trong tệp appsettings.json. Mở appsettings.json và lưu ý rằng nó đã bao gồm những điều sau:

```

1  {
2      "Dữ liệu": {
3          "DefaultConnection": {
4              "Chuỗi kết nối": "Máy chủ = (localdb) \\ MSSQLLocalDB; Cơ sở dữ liệu = _CHANGE_ME; Trus
5          }
6      }
7 }
```

Trong dòng được đánh dấu được mô tả ở trên, hãy thay đổi tên của cơ sở dữ liệu từ _CHANGE_ME. Chúng tôi sẽ chỉ sang cơ sở dữ liệu mới, sẽ được đặt tên là NewMvc6Project để khớp với tên dự án đã di chuyển của chúng tôi.

9.2.3 Tóm tắt

ASP.NET 5 đặt tất cả logic khởi động cho ứng dụng trong một tệp duy nhất, trong đó các dịch vụ và phụ thuộc cần thiết có thể được định nghĩa và cấu hình. Nó thay thế tệp web.config bằng một tính năng cấu hình linh hoạt có thể tận dụng nhiều định dạng tệp, chẳng hạn như JSON, cũng như các biến môi trường.

9.3 Di chuyển từ ASP.NET Web API 2 sang MVC 6

Bởi Steve Smith, Scott Addie

ASP.NET Web API 2 tách biệt với ASP.NET MVC 5, với mỗi thư sử dụng thư viện riêng để cảnh báo cộng hưởng phụ thuộc, trong số những thứ khác. Trong MVC 6, Web API đã được hợp nhất với MVC, cung cấp một cách duy nhất, nhất quán xây dựng các ứng dụng web. Trong bài viết này, chúng tôi trình bày các bước cần thiết để di chuyển từ ASP.NET Web API 2 dự án MVC 6.

Trong bài viết này:

- Xem lại Dự án Web API 2
- Tạo dự án đích
- Di chuyển cấu hình
- Di chuyển mô hình và bộ điều khiển

Bạn có thể xem nguồn đã hoàn thành từ dự án được tạo trong bài viết này trên GitHub.

9.3.1 Xem lại dự án Web API 2

Bài viết này sử dụng dự án mẫu, ProductsApp, được tạo trong bài viết [Bắt đầu với ASP.NET Web API 2 \(C#\)](#) như điểm khởi đầu của nó. Trong dự án đó, một dự án Web API 2 đơn giản được cấu hình như sau.

Trong Global.asax.cs, một cuộc gọi được thực hiện tới WebApiConfig. Đăng ký:

```

1 đang sử dụng Hệ thống;
2 bằng cách sử dụng System.Collections.Generic;
3 bằng cách sử dụng System.Linq;
4 bằng cách sử dụng System.Web;
5 bằng cách sử dụng System.Web.Http;
6 sử dụng System.Web.Routing;

7
8 không gian tên Sản phẩm
9 {
10     lớp công khai WebApplication : System.Web.HttpApplication
11     {
12         void Application_Start () được bảo vệ
13         {
14             GlobalConfiguration.Configure (WebApiConfig.Register);
15         }
16     }
17 }
```

WebApiConfig được định nghĩa trong App_Start và chỉ có một phương thức Đăng ký tĩnh:

```

1 đang sử dụng Hệ thống;
2 bằng cách sử dụng System.Collections.Generic;
3 bằng cách sử dụng System.Linq;
4 bằng cách sử dụng System.Web;
5
6 không gian tên Sản phẩm
7 {
8     public static class WebApiConfig
9     {
10         public static void Register (cấu hình HttpConfiguration)
11         {
12             // Cấu hình và dịch vụ API Web
13
14             // Các tuyến API web
15             config.MapHttpAttributeRoutes ();
16         }
17 }
```

```

16     config.Routes.MapHttpRoute (
17         tên: "DefaultApi",
18         routeTemplate: "api / {controller} / {id}",
19         mặc định: new { id = RouteParameter.Optional }
20     );
21 }
22 }
23 }
24 }
```

Lớp này định cấu hình **định tuyến** thuộc tính, mặc dù nó không thực sự được sử dụng trong dự án. Nó cũng định cấu hình **định tuyến** bằng được sử dụng bởi Web API 2. Trong trường hợp này, Web API sẽ yêu cầu các URL khớp với định dạng / api / {controller} / {id}, với {id} là tùy chọn.

Dự án ProductsApp chỉ bao gồm một bộ điều khiển đơn giản, kế thừa từ ApiController và đưa ra hai phương pháp:

```

1 sử dụng ProductsApp.Models;
2 sử dụng Hệ thống;
3 bằng cách sử dụng System.Collections.Generic;
4 bằng cách sử dụng System.Linq;
5 bằng cách sử dụng System.Net;
6 sử dụng System.Web.Http;
7
8 không gian tên Sản phẩm Ứng dụng.
9 {
10     public class ProductsController : ApiController
11     {
12         Sản phẩm [] sản phẩm = Sản phẩm mới []
13         {
14             Sản phẩm mới {Id = 1, Name = "Tomato Soup", Category = "Groceries", Price = 1 },
15             Sản phẩm mới {Id = 2, Name = "Yo-yo", Category = "Toys", Price = 3,75M },
16             Sản phẩm mới {Id = 3, Name = "Hammer", Category = "Hardware", Price = 16.99M }
17         };
18
19         public IEnumerable <Product> GetAllProducts ()
20         {
21             trả lại sản phẩm;
22         }
23
24         public IHttpActionResult GetProduct (int id)
25         {
26             var product = products.FirstOrDefault ((p) => p.Id == id);
27             if (product == null)
28             {
29                 trả về NotFound ();
30             }
31             trả lại Ok (sản phẩm);
32         }
33     }
34 }
```

Cuối cùng, mô hình, Sản phẩm, được sử dụng bởi ProductsApp, là một lớp đơn giản:

```

1 không gian tên ProductsApp.Models
2 {
3     sản phẩm hạng công cộng
4     {
```

```

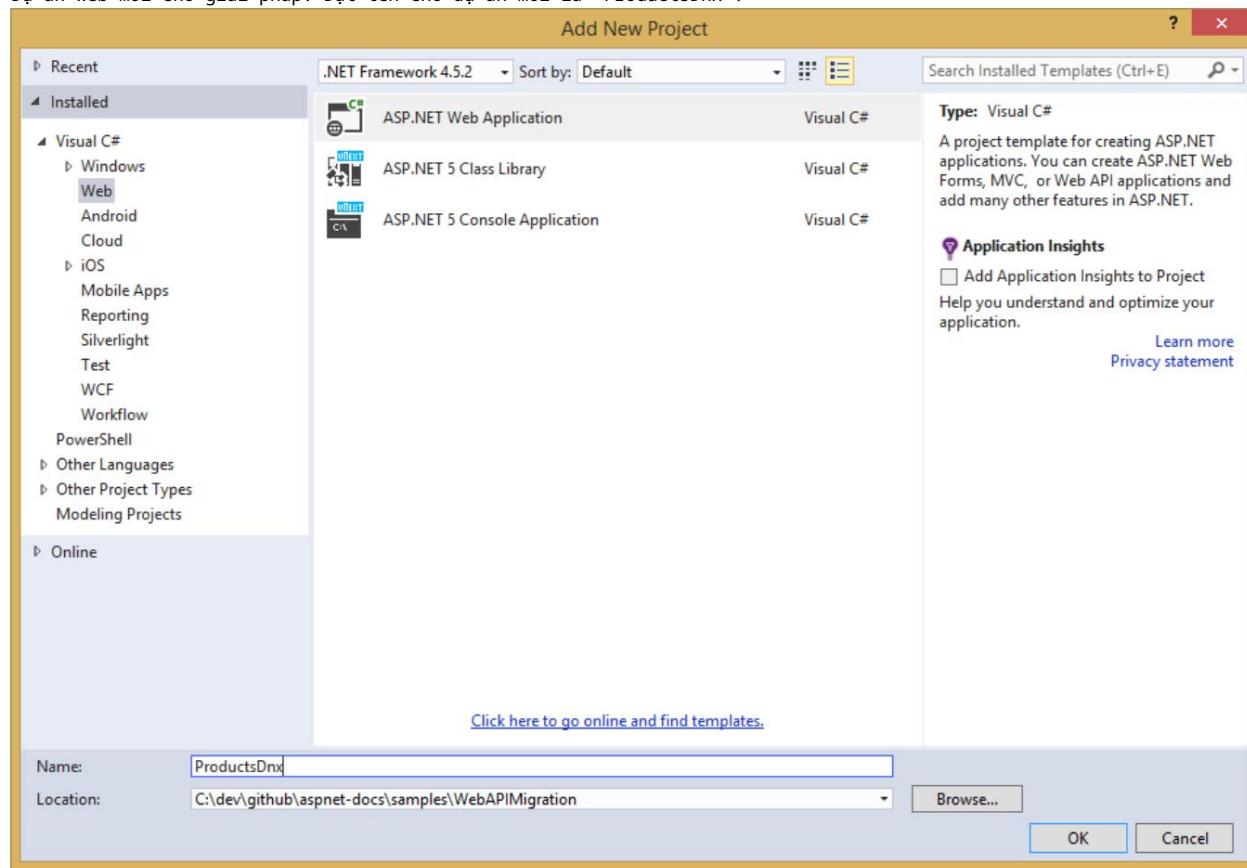
5     public int Id { get; bộ; }
6     chuỗi công khai Tên { get; bộ; }
7     chuỗi công khai Category { get; bộ; }
8     giá thập phân công khai { get; bộ; }
9   }
10 }

```

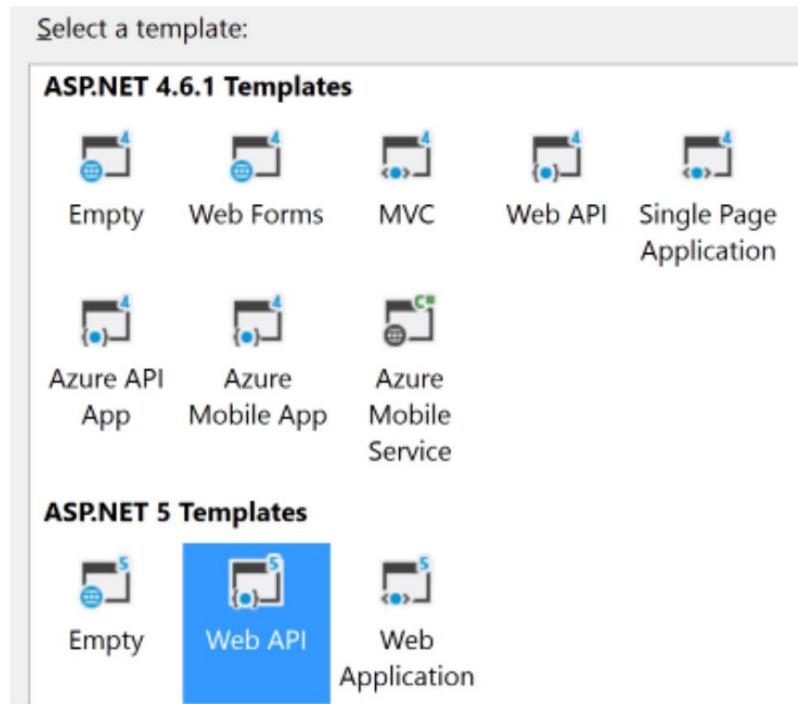
Bây giờ chúng ta có một dự án đơn giản để bắt đầu, chúng ta có thể trình bày cách di chuyển dự án Web API 2 này sang ASP.NET MVC 6.

9.3.2 Tạo dự án đích

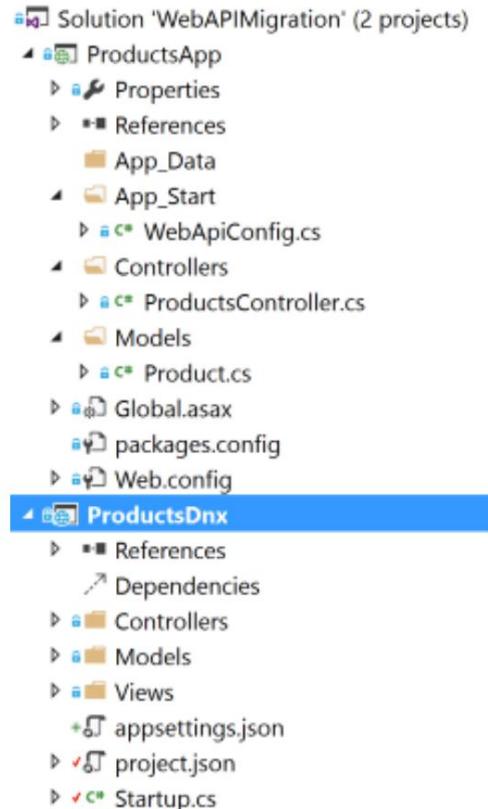
Sử dụng Visual Studio 2015, tạo một giải pháp mới, trong và thêm dự án ProductsApp hiện có vào nó. Sau đó, thêm một Dự án Web mới cho giải pháp. Đặt tên cho dự án mới là 'ProductsDnx'.



Tiếp theo, chọn mẫu dự án API Web ASP.NET 5. Chúng tôi sẽ chuyển nội dung ProductsApp sang dự án mới này.



Xóa tệp Project_Readme.html khỏi dự án mới. Giải pháp của bạn bây giờ sẽ giống như sau:



9.3.3 Di chuyển cấu hình

ASP.NET 5 không còn sử dụng các thư mục Global.asax, web.config hoặc App_Start. Thay vào đó, tất cả các tác vụ khởi động được thực hiện trong Startup.cs trong thư mục gốc của dự án và các tệp cấu hình tĩnh có thể được kết nối từ đó nếu cần (tìm hiểu thêm về [Khởi động ứng dụng ASP.NET 5](#)). Vì Web API hiện đã được tích hợp vào MVC 6, nên ít cần phải định cấu hình nó hơn. Định tuyến dựa trên thuộc tính hiện được bao gồm theo mặc định khi UseMvc () được gọi và đây là phương pháp được đề xuất để định cấu hình các tuyến API Web (và là cách dự án khởi động API Web xử lý việc định tuyến).

```
1 đang sử dụng Hệ thống;
2 bằng cách sử dụng System.Collections.Generic;
3 bằng cách sử dụng System.Linq;
4 sử dụng System.Threading.Tasks;
5 sử dụng Microsoft.AspNetCore.Builder;
6 sử dụng Microsoft.AspNetCore.Hosting;
7 sử dụng Microsoft.Extensions.Configuration;
8 sử dụng Microsoft.Extensions.DependencyInjection;
9 bằng cách sử dụng Microsoft.Extensions.Logging;
10
11 không gian tên Sản phẩmDnx
12 {
13     Khởi động lớp công cộng
14     {
15         Khởi động công khai (IHostingEnosystem env)
16         {
17             // Thiết lập các nguồn cấu hình.
18             var builder = new ConfigurationBuilder ()
19                 .AddJsonFile ("appsettings.json")
20                 .AddEnosystemVariables ();
21             Cấu hình = builder.Build ();
22         }
23
24         public IConfigurationRoot Configuration { get; bộ; }
25
26         // Phương thức này được gọi bởi thời gian chạy. Sử dụng phương pháp này để thêm dịch vụ vào vùng chứa.
27         public void ConfigureServices (dịch vụ IServiceCollection)
28         {
29             // Thêm các dịch vụ khung.
30             services.AddMvc ();
31         }
32
33         // Phương thức này được gọi bởi thời gian chạy. Sử dụng phương pháp này để định cấu hình pip yêu cầu HTTP
34         public void Configure (ứng dụng IApplicationBuilder, IHostingEnosystem env, ILoggerFactory logge
35         {
36             loggerFactory.AddConsole (Configuration.GetSection ("Ghi nhật ký"));
37             loggerFactory.AddDebug ();
38
39             app.UseIISPlatformHandler ();
40
41             app.UseStaticFiles ();
42
43             app.UseMvc ();
44         }
45
46         // Điểm vào của ứng dụng.
47         public static void Main (string [] args) => WebApplication.Run <Khởi động> (args);
48     }
49 }
```

Giả sử bạn muốn sử dụng định tuyến thuộc tính trong dự án của mình về sau, không cần câu hình bổ sung. Chỉ cần áp dụng các thuộc tính cần thiết cho bộ điều khiển và hành động của bạn, như được thực hiện trong ValuesController mẫu lớp được bao gồm trong dự án khởi động API Web:

```

1 đang sử dụng Hệ thống;
2 bằng cách sử dụng System.Collections.Generic;
3 bằng cách sử dụng System.Linq;
4 sử dụng Microsoft.AspNet.Mvc;
5
6 không gian tên Sản phẩmDnx.Controllers
7 {
8     [Tuyến đường ("api / [controller]")]
9     public class ValuesController : Controller
10    {
11        // GET: api / giá trị
12        [HttpGet]
13        public IEnumerable <string> Get ()
14        {
15            trả về chuỗi mới [] { "value1", "value2" };
16        }
17
18        // NHẬN api / giá trị / 5
19        [HttpGet ("{id}")]
20        chuỗi công khai Get (int id)
21        {
22            trả về "giá trị";
23        }
24
25        // ĐĂNG api / giá trị
26        [HttpPost]
27        public void Post (giá trị chuỗi [FromBody] )
28        {
29        }
30
31        // PUT api / giá trị / 5
32        [HttpPut ("{id}")]
33        public void Put (int id, [FromBody] string value)
34        {
35        }
36
37        // XÓA api / giá trị / 5
38        [HttpDelete ("{id}")]
39        public void Delete (int id)
40        {
41        }
42    }
43}

```

Lưu ý sự hiện diện của [controller] trên dòng 8. Định tuyến dựa trên thuộc tính hiện hỗ trợ một số mã thông báo nhất định, chẳng hạn như [controller] và hành động]. Các mã thông báo này được thay thế trong thời gian chạy bằng tên của bộ điều khiển hoặc hành động, tương ứng, thuộc tính đã được áp dụng. Điều này phục vụ để giảm số lượng chuỗi ma thuật trong dự án và nó đảm bảo các tuyến đường sẽ được đồng bộ hóa với bộ điều khiển và hành động tương ứng của chúng khi tái cấu trúc đổi tên tự động đã áp dụng.

Để di chuyển bộ điều khiển API Sản phẩm, trước tiên chúng ta phải sao chép ProductsController vào dự án mới. Sau đó, chỉ cần bao gồm thuộc tính tuyến đường trên bộ điều khiển:

```
[Tuyến đường ("api / [controller]")]
```

Bạn cũng cần thêm thuộc tính `[HttpGet]` vào hai phương thức, vì cả hai đều phải được gọi qua HTTP Get. Bao gồm kỳ vọng của thông số "id" trong thuộc tính cho `GetProduct ()`:

```
// / api / products
[HttpGet]
...

// / api / products / 1
[HttpGet ("{id}")]
```

Tại thời điểm này, định tuyến đã được định cấu hình chính xác; tuy nhiên, chúng tôi vẫn chưa thể kiểm tra nó. Các thay đổi bổ sung phải được thực hiện trước khi `ProductsController` sẽ biên dịch.

9.3.4 Di chuyển mô hình và bộ điều khiển

Bước cuối cùng trong quá trình di chuyển cho dự án Web API đơn giản này là sao chép qua Bộ điều khiển và bất kỳ Mô hình nào mà chúng sử dụng. Trong trường hợp này, chỉ cần sao chép `Controllers / ProductsController.cs` từ dự án gốc sang dự án mới. Sau đó, sao chép toàn bộ thư mục Mô hình từ dự án gốc sang dự án mới. Điều chỉnh không gian tên để khớp với tên dự án mới (`ProductsDnx`). Tại thời điểm này, bạn có thể xây dựng ứng dụng và bạn sẽ tìm thấy một số lỗi biên dịch. Chúng thường thuộc các loại sau:

- `ApiController` không tồn tại
- Không gian tên `System.Web.Http` không tồn tại
- `IHttpActionResult` không tồn tại
- `NotFound` không tồn tại
- `Ok` không tồn tại

May mắn thay, tất cả những điều này đều rất dễ sửa:

- Thay đổi `ApiController` thành `Controller` (bạn có thể cần thêm bằng `Microsoft.AspNet.Mvc`)
- Xóa bất kỳ câu lệnh sử dụng nào tham chiếu đến `System.Web.Http`
- Thay đổi bất kỳ phương thức nào trả về `IHttpActionResult` để trả về `IActionResult`
- Thay đổi `NotFound` thành `NotFound`
- Thay đổi `Ok` (sản phẩm) thành `ObjectResult` (sản phẩm) mới

Khi những thay đổi này đã được thực hiện và các câu lệnh sử dụng không được sử dụng bị xóa, lớp `ProductsController` được di chuyển sẽ trông giống như sau:

```
1 sử dụng Microsoft.AspNet.Mvc; 2 bằng cách
2 sử dụng ProductsDnx.Models; 3 bằng cách sử
3 dụng System.Collections.Generic; 4 bằng cách sử dụng
4 System.Linq;
5
6 không gian tên Sản phẩmDnx.Controllers
7 {
8     [Route ("api / [controller]")] public
9     class ProductsController : Controller {
10
11         Sản phẩm [] sản phẩm = Sản phẩm mới []
```

```

12     {
13         Sản phẩm mới {Id = 1, Name = "Tomato Soup", Category = "Groceries", Price = 1 },
14         Sản phẩm mới {Id = 2, Name = "Yo-yo", Category = "Toys", Price = 3,75M },
15         Sản phẩm mới {Id = 3, Name = "Hammer", Category = "Hardware", Price = 16.99M }
16     };
17
18     // / api / products
19     [HttpGet]
20     public IEnumerable <Product> GetAllProducts ()
21     {
22         trả lại sản phẩm;
23     }
24
25     // / api / products / 1
26     [HttpGet ("{id}")]
27     public IActionResult GetProduct (int id)
28     {
29         var product = products.FirstOrDefault ((p) => p.Id == id);
30         if (product == null)
31         {
32             trả về HttpNotFound ();
33         }
34         trả về ObjectResult (sản phẩm) mới;
35     }
36 }
37

```

Bây giờ bạn có thể chạy dự án đã di chuyển và duyệt đến / api / products; và, bạn sẽ thấy danh sách đầy đủ 3 Mỹ phẩm. Duyệt đến / api / products / 1 và bạn sẽ thấy sản phẩm đầu tiên.

9.3.5 Tóm tắt

Việc chuyển một dự án Web API 2 đơn giản sang MVC 6 khá đơn giản, nhờ sự hợp nhất của Web API vào MVC 6 trong ASP.NET 5. Các phần chính mà mọi dự án Web API 2 sẽ cần di chuyển là các tuyến, bộ điều khiển và các mô hình, cùng với các bản cập nhật cho các loại được sử dụng bởi các bộ điều khiển và hành động MVC 6.

9.3.6 Tài nguyên liên quan

Tạo một API Web trong MVC 6

9.4 Di chuyển xác thực và nhận dạng từ ASP.NET MVC 5 sang MVC 6

Bởi Steve Smith

Trong bài viết trước, chúng tôi đã di chuyển cấu hình từ dự án ASP.NET MVC 5 sang MVC 6. Trong bài viết này, chúng tôi di chuyển các tính năng đăng ký, đăng nhập và quản lý người dùng.

Bài viết này bao gồm các chủ đề sau:

- Định cấu hình danh tính và tư cách thành viên
- Đăng ký di chuyển và Logic đăng nhập
- Di chuyển các tính năng quản lý người dùng

Bạn có thể tải xuống mã nguồn đã hoàn thành từ dự án được tạo trong bài viết này TẠI ĐÂY (VIỆC CẦN LÀM).

9.4.1 Định cấu hình danh tính và tư cách thành viên

Trong ASP.NET MVC 5, các tính năng xác thực và nhận dạng được định cấu hình trong Startup.Auth.cs và IdentityConfig.cs, nằm trong thư mục App_Start. Trong MVC 6, các tính năng này được cấu hình trong Startup.cs. Trước khi kéo các dịch vụ cần thiết vào và định cấu hình chúng, chúng ta nên thêm các phần phụ thuộc bắt buộc vào dự án. Mở project.json và thêm "Microsoft.AspNet.Identity.EntityFramework" và "Microsoft.AspNet.Identity.Cookies" vào danh sách các phần phụ thuộc:

```
"phụ thuộc":
    {"Microsoft.AspNet.Server.IIS": "1.0.0-beta3", "Microsoft.AspNet.Mvc": "6.0.0-beta3", "Microsoft.Framework.ConfigurationModel.Json": "1.0.0-beta3", "Microsoft.AspNet.Identity.EntityFramework": "3.0.0-beta3", "Microsoft.AspNet.Security.Cookies": "1.0.0-beta3"
},
}
```

Bây giờ, hãy mở Startup.cs và cập nhật phương thức ConfigureServices () để sử dụng các dịch vụ Entity Framework và Identity:

```
public void ConfigureServices (dịch vụ IServiceCollection) {
    // Thêm dịch vụ EF vào vùng chứa dịch vụ. services.AddEntityFramework
    (Cấu hình)
        .AddSqlServer ()
        .AddDbContext <ApplicationDbContext> ();

    // Thêm dịch vụ Identity vào vùng chứa dịch vụ. services.AddIdentity
    <ApplicationUser, IdentityRole> (cấu hình)
        .AddEntityFrameworkStores <ApplicationDbContext> ();

    services.AddMvc ();
}
```

Tại thời điểm này, có hai loại được tham chiếu trong đoạn mã trên mà chúng tôi vẫn chưa di chuyển từ dự án MVC 5: ApplicationDbContext và ApplicationUser. Tạo một thư mục Models mới trong dự án MVC 6 và thêm hai lớp vào nó tương ứng với các kiểu này. Bạn sẽ tìm thấy phiên bản MVC 5 của các lớp này trong /Models/IdentityModels.cs, nhưng chúng tôi sẽ sử dụng một tệp cho mỗi lớp trong dự án đã di chuyển vì điều đó rõ ràng hơn.

ApplicationUser.cs:

```
sử dụng Microsoft.AspNet.Identity;
không gian tên NewMvc6Project.Models {

    public class ApplicationUser : IdentityUser {}

}
```

ApplicationDbContext.cs:

```
sử dụng Microsoft.AspNet.Identity.EntityFramework; sử dụng
Microsoft.Data.Entity;
không gian tên NewMvc6Project.Models {
```

```

public class ApplicationDbContext : IdentityDbContext <ApplicationUser> {

    private static bool _create = false; public
    ApplicationDbContext () {

        // Tạo cơ sở dữ liệu và lược đồ nếu nó không tồn tại
        // Đây là một giải pháp tạm thời để tạo cơ sở dữ liệu cho đến khi Entity Framework // được hỗ trợ trong ASP.NET 5
        if (! _Create) {

            Database.MigrationsEnabled (). ApplyMigrations (); _create = true;

        }
    }

    ghi đè được bảo vệ void OnConfiguring (tùy chọn DbContextOptions) {

        tùy chọn.UseSqlServer ();
    }
}
}

```

Dự án Web MVC 5 Starter không bao gồm nhiều tùy chỉnh của người dùng hoặc ApplicationDbContext. Khi di chuyển một ứng dụng thực, bạn cũng sẽ cần phải di chuyển tất cả các thuộc tính và phương thức tùy chỉnh của người dùng ứng dụng của bạn và các lớp DbContext, cũng như bất kỳ lớp Model nào khác mà ứng dụng của bạn sử dụng (ví dụ: nếu DbContext của bạn có DbSet <Album>, tất nhiên bạn sẽ cần phải di chuyển lớp Album).

Với các tệp này tại chỗ, tệp Startup.cs có thể được thực hiện để biên dịch bằng cách cập nhật các câu lệnh của nó bằng cách sử dụng:

```

sử dụng Microsoft.Framework.ConfigurationModel; sử dụng
Microsoft.AspNet.Hosting; sử dụng NewMvc6Project.Models; sử dụng
Microsoft.AspNet.Identity;

```

Ứng dụng của chúng tôi hiện đã sẵn sàng hỗ trợ các dịch vụ xác thực và nhận dạng - ứng dụng chỉ cần để những tính năng này hiển thị với người dùng.

9.4.2 Di chuyển Đăng ký và Logic Đăng nhập

Với các dịch vụ nhận dạng được định cấu hình cho ứng dụng và quyền truy cập dữ liệu được định cấu hình bằng Entity Framework và SQL Server, chúng tôi hiện đã sẵn sàng thêm hỗ trợ đăng ký và đăng nhập vào ứng dụng. Nhớ lại rằng trước đó trong quá trình di chuyển, chúng tôi đã nhận xét về một tham chiếu đến _LoginPartial trong _Layout.cshtml. Nay giờ đã đến lúc quay lại mã đó, bỏ ghi chú và thêm các bộ điều khiển và chế độ xem cần thiết để hỗ trợ chức năng đăng nhập.

Cập nhật _Layout.cshtml; bỏ ghi chú dòng @ Html.

```

<li> @ Html.ActionLink ("Liên hệ", "Liên hệ", "Trang chủ") </li>
</ul>
@ * @ Html.Partial ("_ LoginPartial") * @
</div>
</div>

```

Bây giờ, thêm một Trang xem MVC mới có tên _LoginPartial vào thư mục Lượt xem / Chia sẻ:

Cập nhật _LoginPartial.cshtml bằng mã sau (thay thế tất cả nội dung của nó):

```
@using System.Security.Principal

@if (User.Identity.IsAuthenticated) {

    using (Html.BeginForm ("LogOff", "Account", FormMethod.Post, new {id = "logoutForm", @class = "n {

        @ Html.AntiForgeryToken () <ul
        class = "nav navbar-nav navbar-right"
        <li>
            @ Html.ActionLink ("Xin chào " + User.Identity.GetUserName () + "!", "Quản lý", "Tài khoản",
        </li>
        <li> <a href="javascript:document.getElementById('logoutForm').submit()"> Đăng xuất </a> </li> </ul>

    }
}

} khác
{
    <ul class = "nav navbar-nav navbar-right">
        <li> @ Html.ActionLink ("Đăng ký", "Đăng ký", "Tài khoản", routeValues: null, htmlAttributes: n <li> @ Html.ActionLink ("Đăng nhập",
        "Đăng nhập", "Tài khoản", routeValues: null, htmlAttributes: new {
    </ul>
}
```

Tại thời điểm này, bạn sẽ có thể làm mới trang web trong trình duyệt của mình.

9.4.3 Tóm tắt

ASP.NET 5 và MVC 6 giới thiệu các thay đổi đối với các tính năng ASP.NET Identity 2 đi kèm với ASP.NET MVC 5. Trong bài viết này, bạn đã biết cách di chuyển các tính năng xác thực và quản lý người dùng của dự án ASP.NET MVC 5 sang MVC 6.

Đóng góp

Tài liệu trên trang web này là thủ công của nhiều cộng tác viên của chúng tôi.

Chúng tôi chấp nhận yêu cầu kéo! Nhưng nhiều khả năng bạn sẽ được chấp nhận nếu bạn tuân theo các nguyên tắc sau:

1. Đọc <https://github.com/aspnet/Docs/blob/master/CONTRIBUTING.md>
2. Làm theo Hướng dẫn kiểu tài liệu ASP.NET