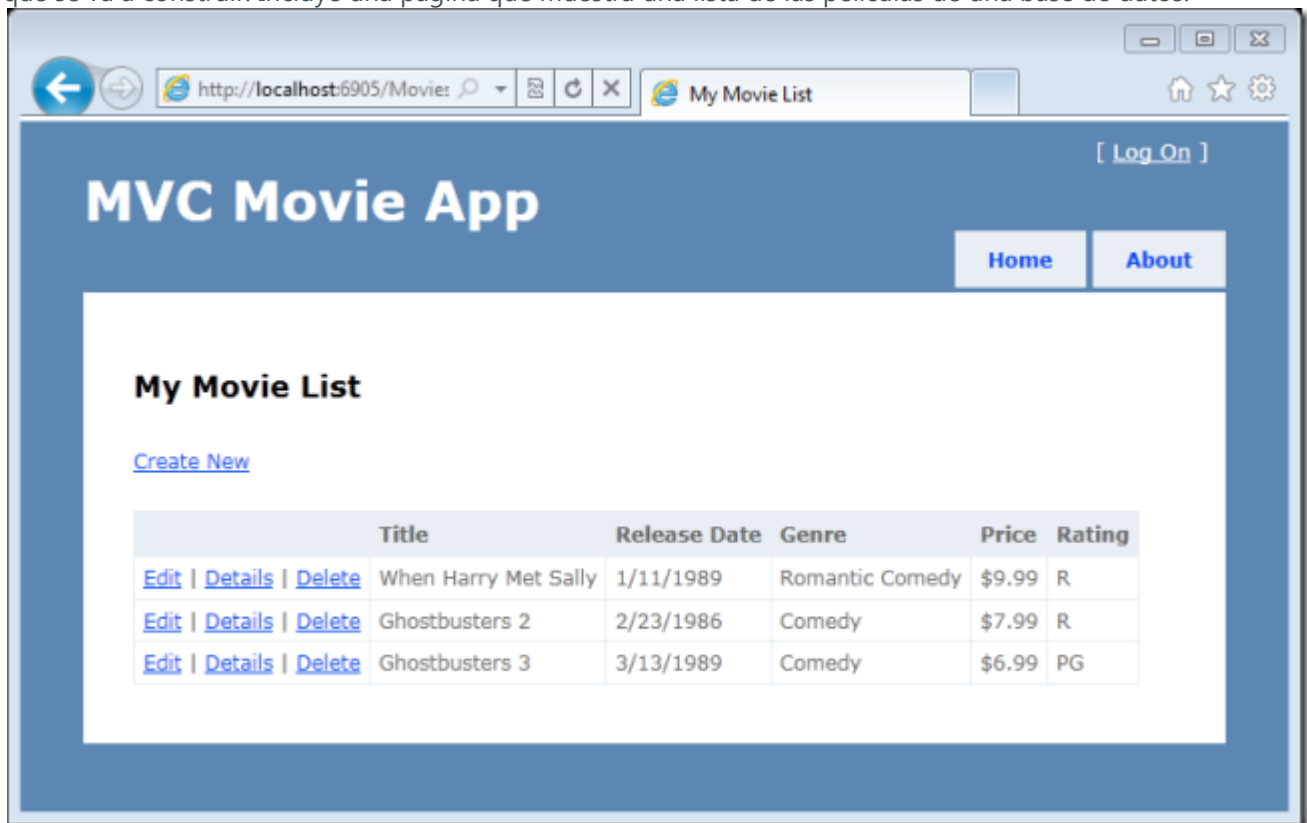


Introducción a ASP.NET MVC 3 (C #)

Lo que va a construir

Vas a poner en práctica una simple película de exclusión de la lista de aplicaciones que permite crear, editar y películas establecimiento de una base de datos. A continuación hay dos capturas de pantalla de la aplicación que se va a construir. Incluye una página que muestra una lista de las películas de una base de datos:



La aplicación también te permite añadir, editar y borrar películas, así como ver información sobre los individuales. Todos los escenarios de entrada de datos incluyen la validación para asegurar que los datos almacenados en la base de datos es correcta.

The screenshot shows a web browser window with the address bar at `http://localhost: Create`. The page title is "MVC Movie App" and there is a "[Log On]" link in the top right. Below the title are two buttons: "Home" and "About". The main content area is titled "Create" and contains a form for adding a new movie. The form has the following fields and values:

- Title:** An empty text box with a red error message "Title is required" to its right.
- ReleaseDate:** A text box containing "1/11/1999".
- Genre:** A text box containing "Comedy".
- Price:** A text box containing "129.99" with a red error message "Price must be between \$1 and \$100" to its right.
- Rating:** A text box containing "PG".

At the bottom of the form is a "Create" button. Below the form is a link labeled "Back to List".

Habilidades que aprenderá

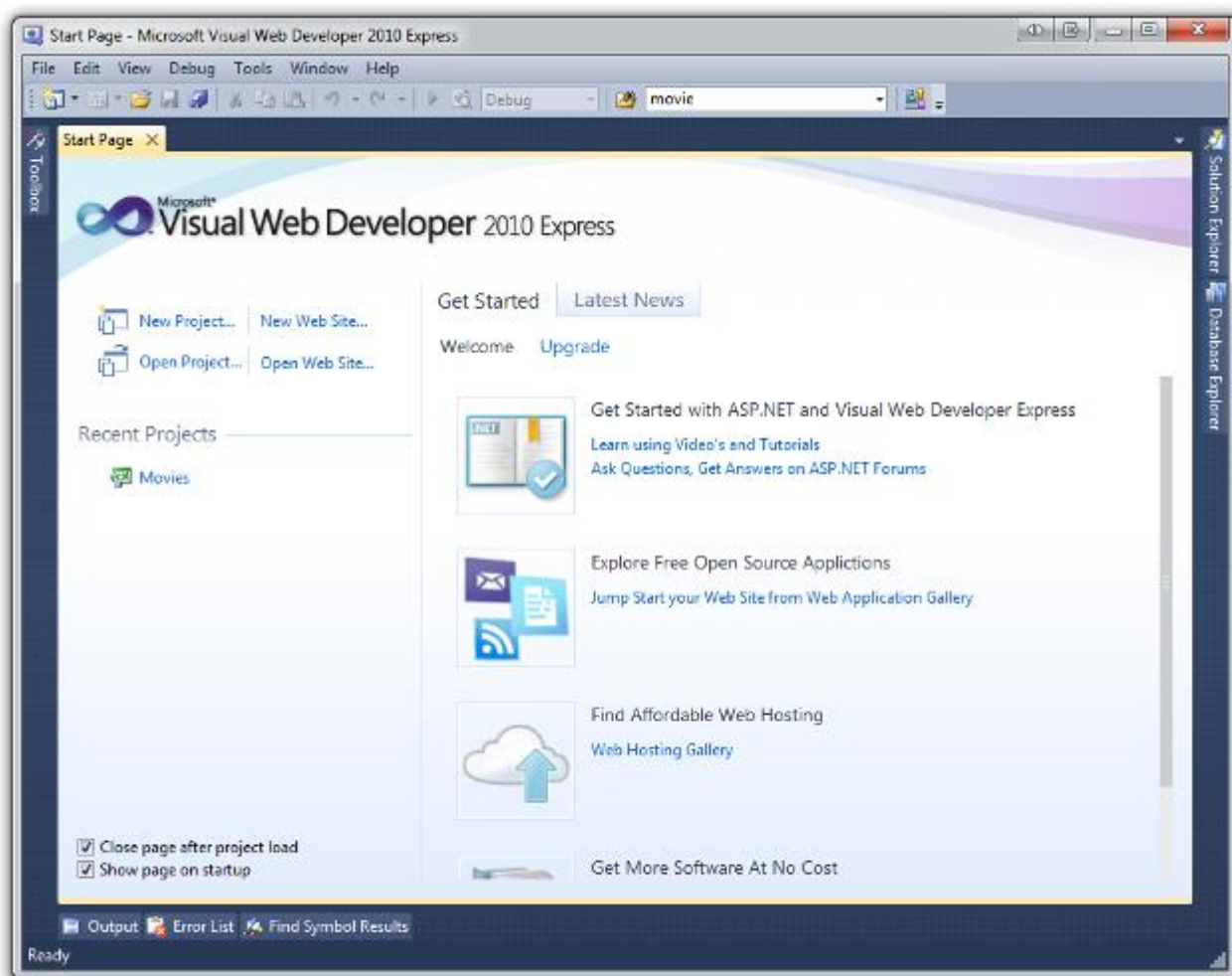
Esto es lo que aprenderá:

- ¿Cómo crear un nuevo proyecto ASP.NET MVC.
- Cómo crear ASP.NET MVC controladores y vistas.
- Cómo crear una nueva base de datos mediante Entity Framework paradigma Primer Código.
- Cómo recuperar y mostrar datos.
- Cómo editar los datos y permitir la validación de datos.

Introducción

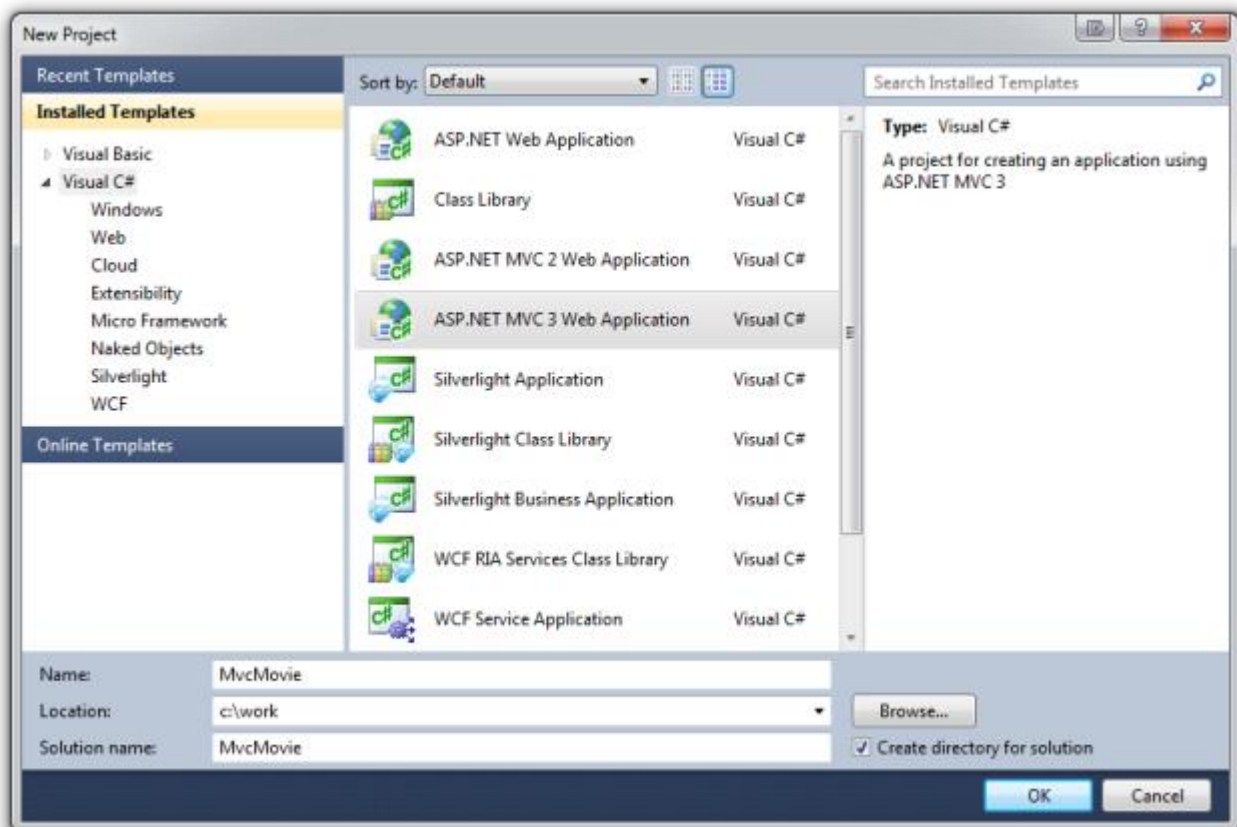
Comience por ejecutar Visual Web Developer 2010 Express ("Visual Web Developer" para abreviar) y seleccione **Nuevo proyecto** en el **inicio** de página.

Visual Web Developer es un IDE o entorno de desarrollo integrado. Al igual que usted utiliza Microsoft Word para escribir documentos, va a utilizar un IDE para crear aplicaciones. En Visual Web Developer hay una barra de herramientas en la parte superior que muestra las diversas opciones disponibles para usted. También hay un menú que proporciona otra forma de realizar las tareas en el IDE. (Por ejemplo, en lugar de seleccionar **Nuevo proyecto** desde el **inicio** página, puede utilizar el menú y seleccione **Archivo > Nuevo proyecto** .)

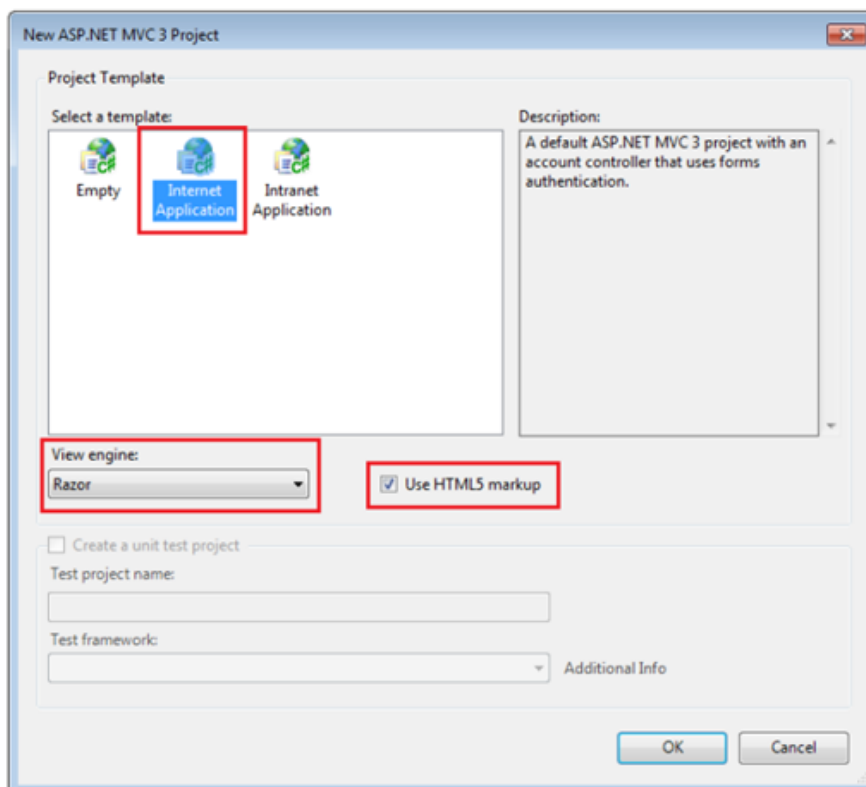


Crear su primera aplicación

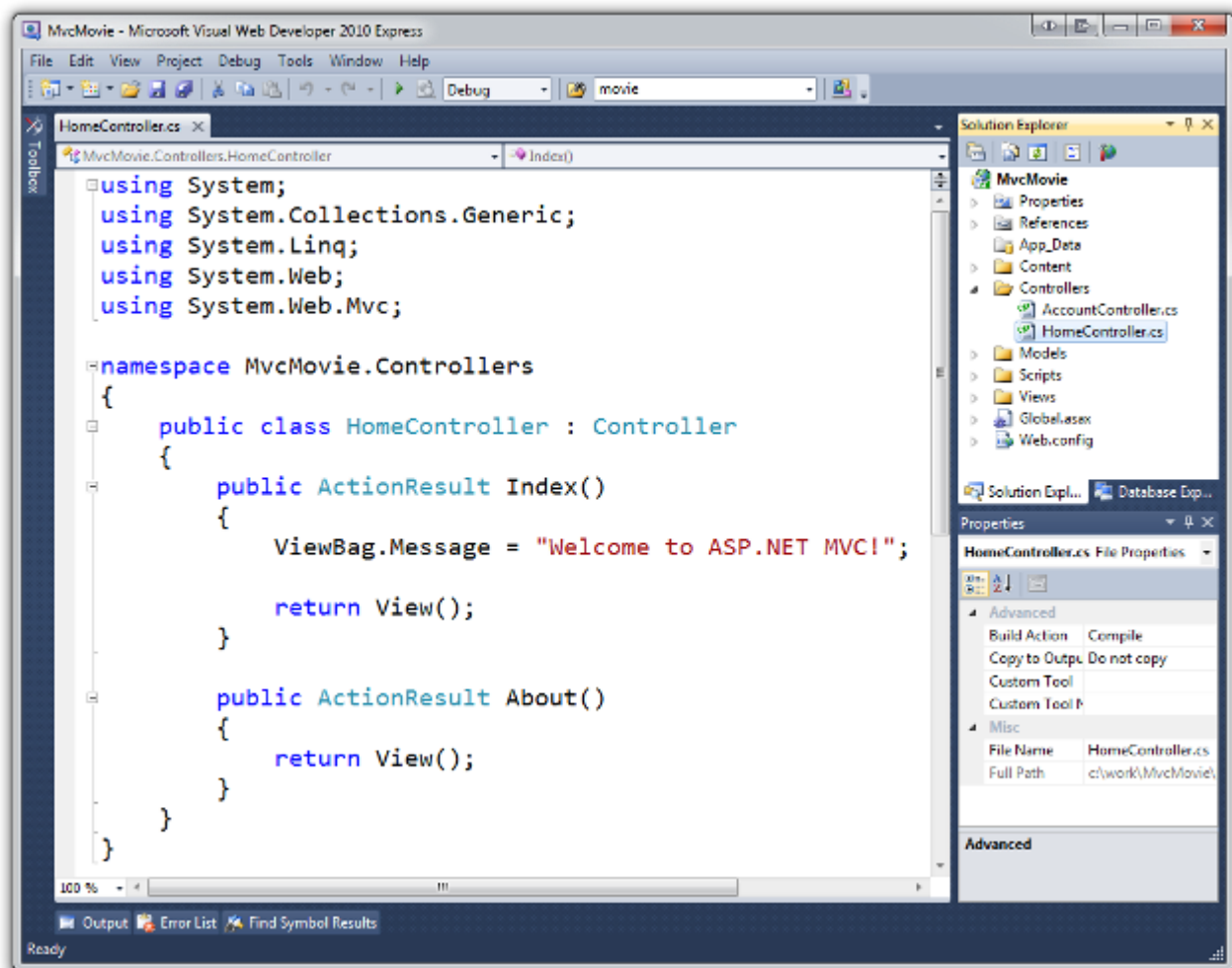
Puede crear aplicaciones utilizando Visual C # o Visual Basic como lenguaje de programación. Seleccione Visual C # a la izquierda y luego seleccione **Aplicación Web ASP.NET MVC 3** . El nombre de su proyecto "MvcMovie" y luego haga clic en **Aceptar** . (Si prefiere Visual Basic, cambiar a la [versión Visual Basic](#) de este tutorial.)



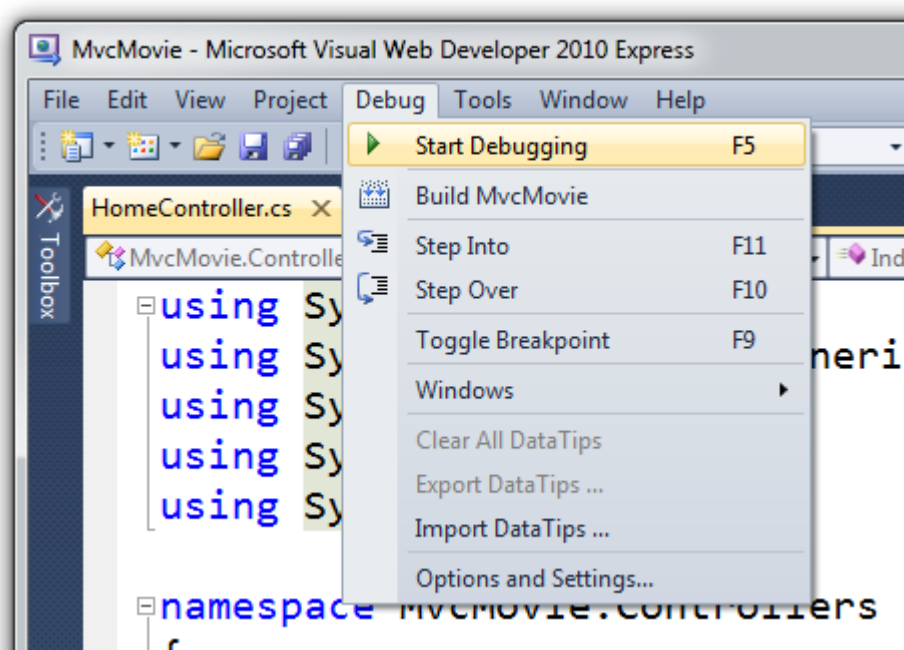
En el **nuevo proyecto de ASP.NET MVC 3** cuadro de diálogo, seleccione **Aplicación de Internet** . Compruebe **Utilice HTML5 marcado** y dejar **Razor** como el motor vista por defecto.



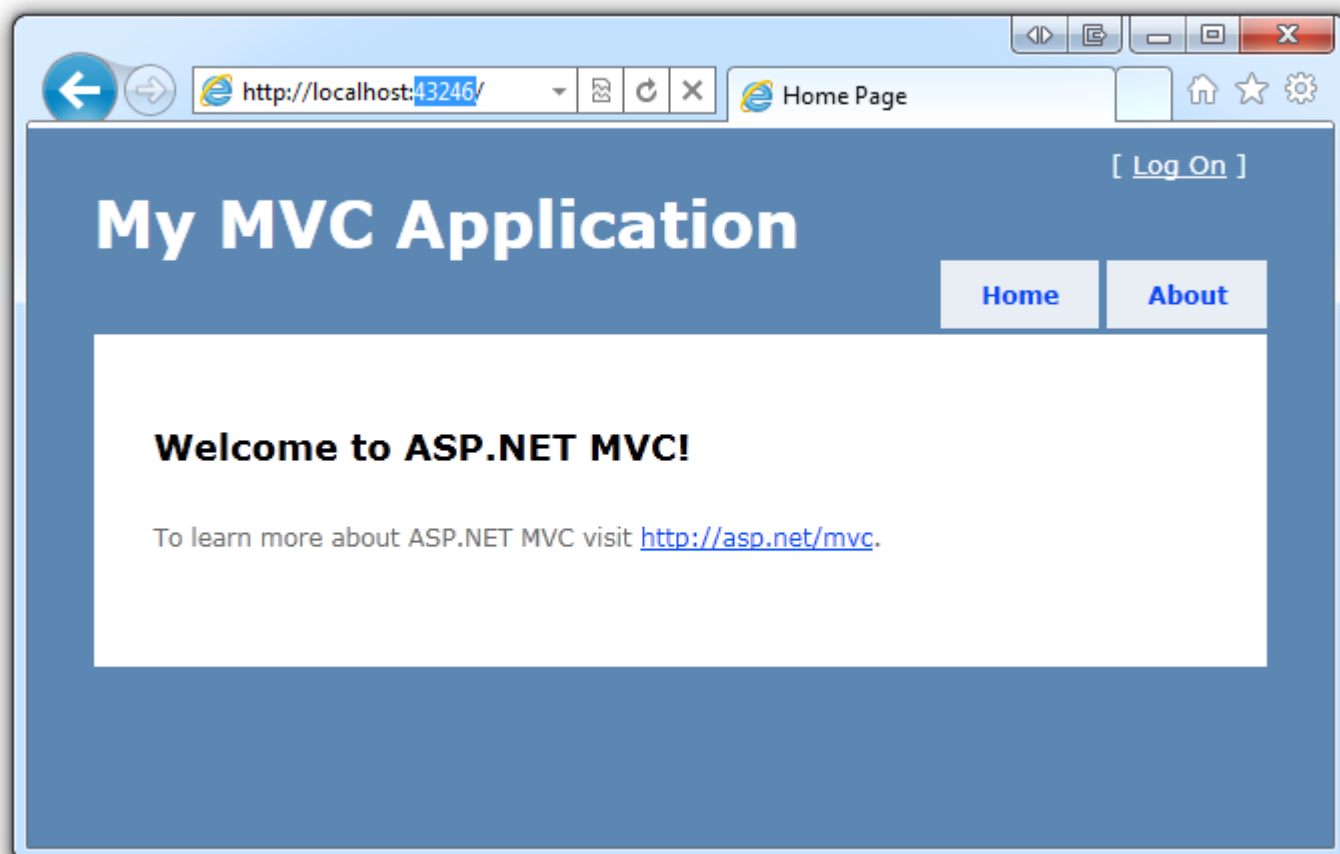
Haga clic en **Aceptar** . Visual Web Developer utiliza una plantilla predeterminada para el proyecto ASP.NET MVC que acaba de crear, por lo que tiene una aplicación que funciona en este momento sin hacer nada! Se trata de un simple "Hello World!" proyecto, y es un buen lugar para iniciar la aplicación.



Desde la **depuración** del menú, seleccione **Iniciar depuración** .



Tenga en cuenta que el acceso directo de teclado para iniciar la depuración es F5. F5 hace que Visual Web Developer para iniciar un servidor web y ejecutar el desarrollo de aplicaciones web. Visual Web Developer entonces inicia un explorador y abre la página de la aplicación en casa. Observe que la barra de direcciones del navegador dice **localhost** algo y no como **example.com** . Esto se debe a **localhost** apunta siempre a su propio equipo local, que en este caso se está ejecutando la aplicación que acaba de crear. Cuando Visual Web Developer ejecuta un proyecto web, un puerto aleatorio se utiliza para el servidor web. En la imagen de abajo, el número de puerto aleatorio es 43246. Al ejecutar la aplicación, es probable que vea un número de puerto diferente.



Nada más sacarlo de la caja de la plantilla por defecto le da dos páginas a visitar y una página de inicio de sesión de base. El siguiente paso es cambiar la forma en que esta aplicación funciona y aprender un poco acerca de ASP.NET MVC en el proceso. Cierre el navegador y vamos a cambiar algo de código.

Adición de un controlador (C #)

Este tutorial le enseñará los fundamentos de la construcción de una aplicación ASP.NET MVC Web utilizando Microsoft Visual Web Developer 2010 Express Service Pack 1, que es una versión gratuita de Microsoft Visual Studio. Antes de empezar, asegúrese de que ha instalado los requisitos previos enumerados a continuación. Puede instalar todos ellos haciendo clic en el siguiente enlace: [Web Platform Installer](#) . Si lo prefiere, puede instalar individualmente los requisitos previos utilizando los siguientes enlaces:

- [Visual Studio Web Developer Express SP1 requisitos previos](#)
- [ASP.NET MVC 3 Herramientas actualización](#)
- [SQL Server Compact 4.0](#) (+ tiempo de ejecución de herramientas de apoyo)

Si está utilizando Visual Studio 2010 en lugar de Visual Web Developer 2010, instale los requisitos previos, haga clic en el siguiente enlace: [requisitos previos de Visual Studio 2010](#) .

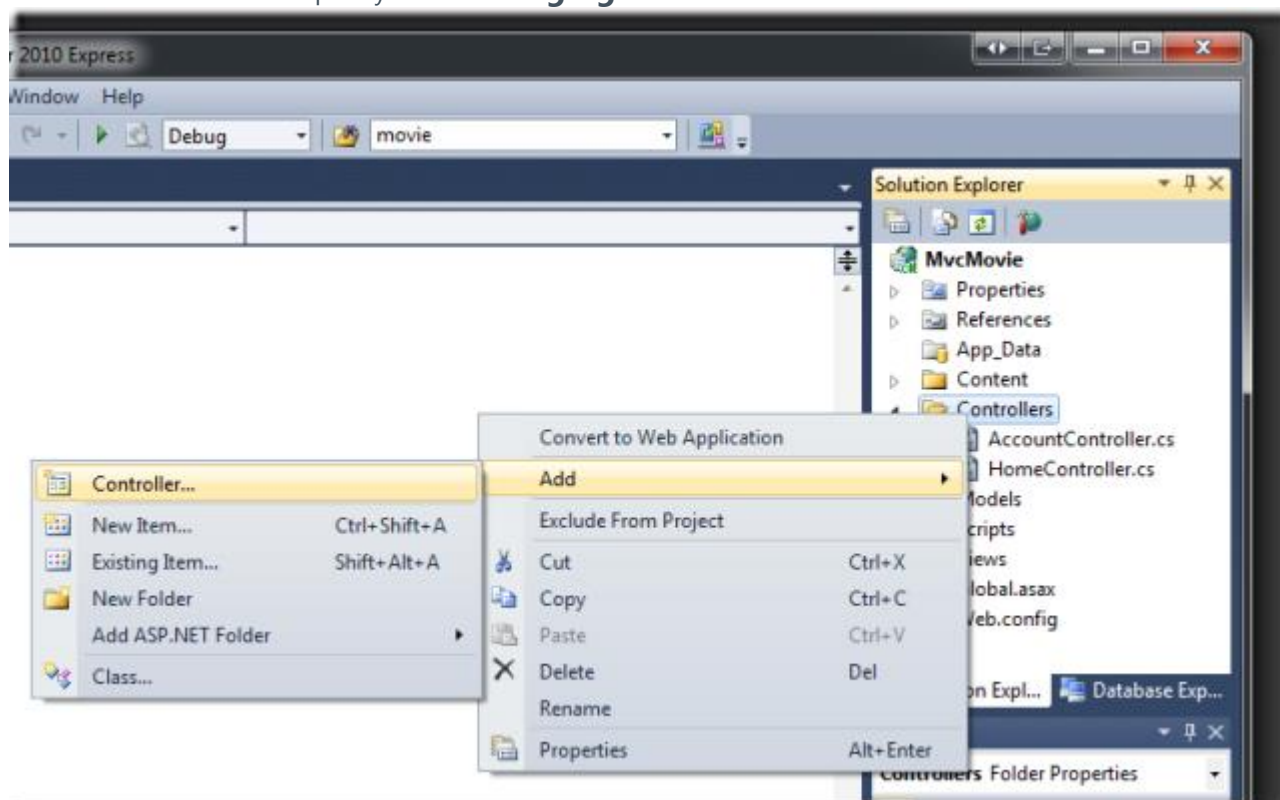
Un proyecto de Visual Web Developer con el código fuente de C # está disponible para este tema. [Descargue la versión C #](#) . Si prefiere Visual Basic, cambiar a la [versión Visual Basic](#) de este tutorial.

MVC es sinónimo de *modelo-vista-controlador* . MVC es un patrón para el desarrollo de aplicaciones que son bien estructurado y fácil de mantener. MVC aplicaciones basadas contener:

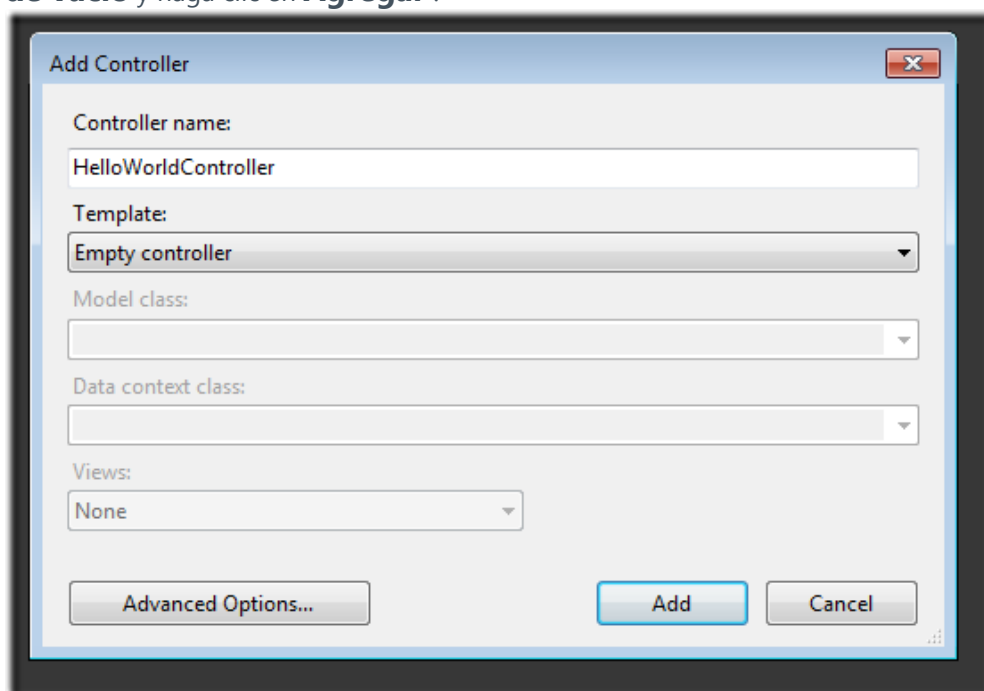
- Controladores: Las clases que manejan las solicitudes de entrada a la aplicación, recuperar los datos del modelo y, a continuación, especificar plantillas de vista que devuelven una respuesta al cliente.
- Modelos: Las clases que representan los datos de la solicitud y que la lógica de validación uso para hacer cumplir las reglas de negocio para los datos.
- Vistas: Archivos de plantilla que utiliza la aplicación para generar dinámicamente HTML respuestas.

Vamos a estar cubriendo todos estos conceptos en esta serie de tutoriales y mostrarle cómo usarlas para crear una aplicación.

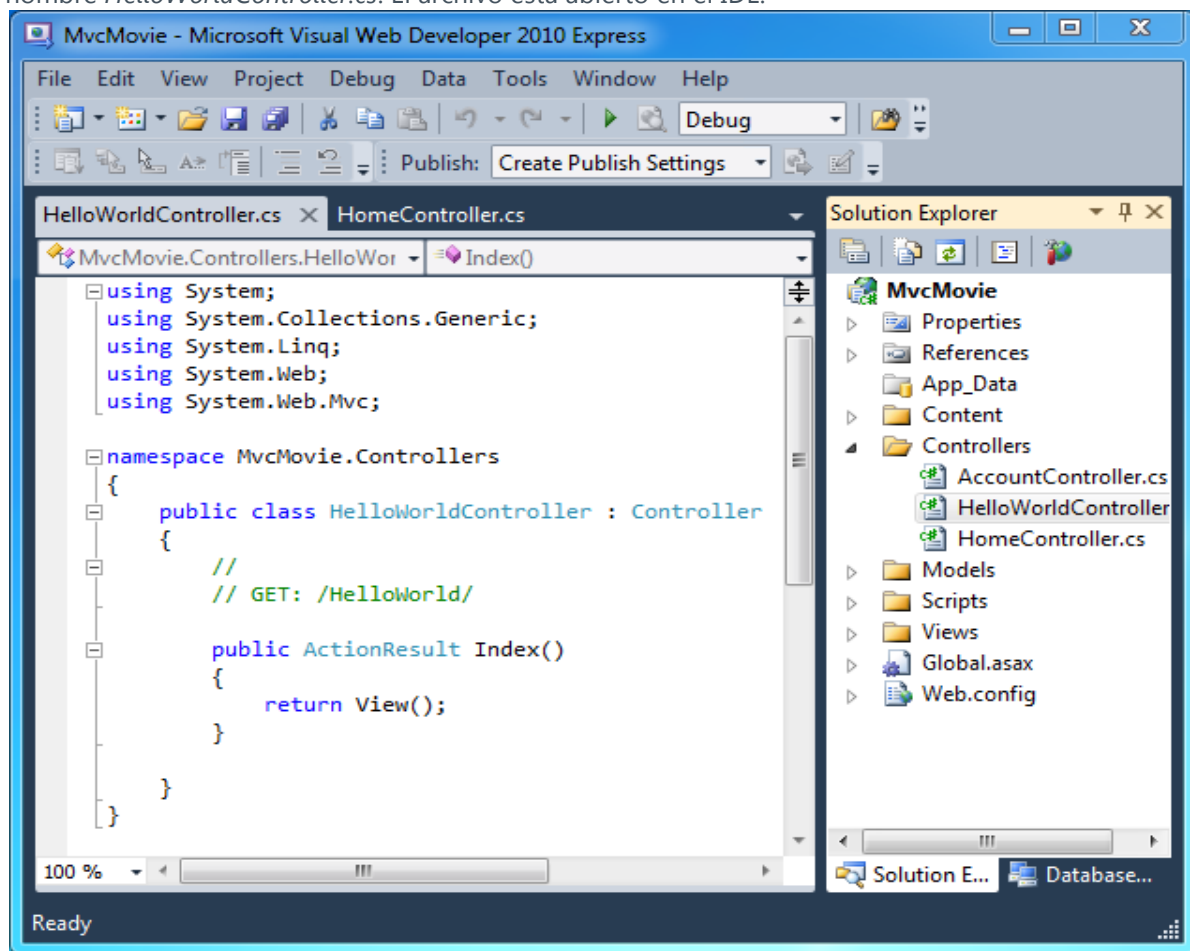
Vamos a empezar creando una clase de controlador. En **el Explorador de soluciones** , haga clic con el botón **Controladores de carpeta** y seleccione **Agregar controlador** .



El nombre de su nuevo controlador "HelloWorldController". Deje la plantilla por defecto como **controlador de vacío** y haga clic en **Agregar**.



Observe en **el Explorador de soluciones** que un nuevo archivo ha sido creado con nombre *HelloWorldController.cs*. El archivo está abierto en el IDE.



Dentro de la **clase HelloWorldController público** bloque, crear dos métodos que se parecen al siguiente código. El controlador volverá una cadena de HTML como ejemplo.

```
using System.Web;
using System.Web.Mvc;

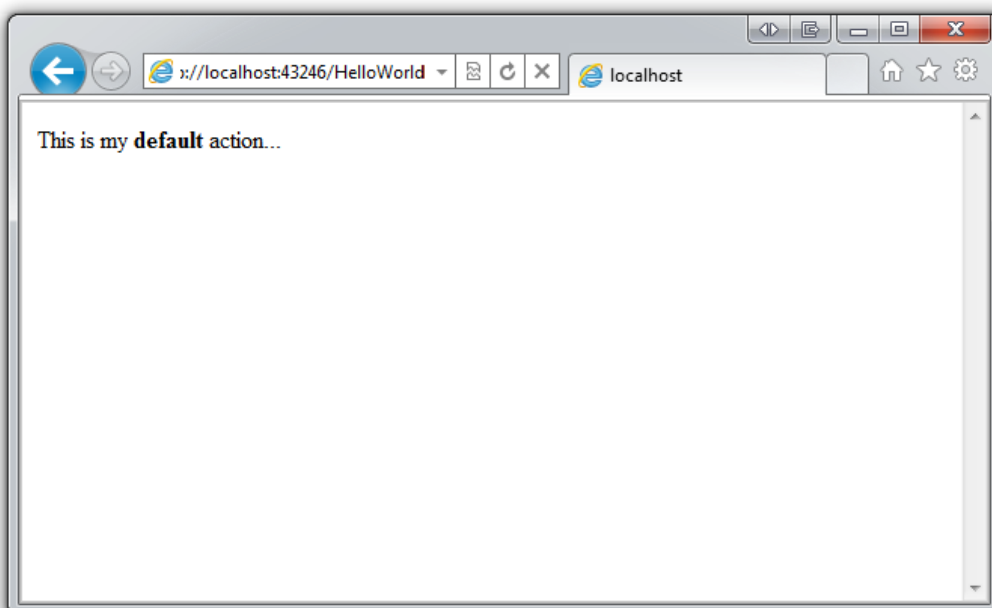
namespace MvcMovie.Controllers
{
    public class HelloWorldController : Controller
    {
        //
        // GET: /HelloWorld/

        public string Index()
        {
            return "This is my <b>default</b> action...";
        }

        //
        // GET: /HelloWorld/Welcome/

        public string Welcome()
        {
            return "This is the Welcome action method...";
        }
    }
}
```

El controlador se nombra **HelloWorldController** y el primer método se denomina **índice**. Vamos a invocar desde un navegador. Ejecutar la aplicación (presione F5 o Ctrl + F5). En el navegador, añadir "HelloWorld" a la ruta en la barra de direcciones. (Por ejemplo, en la ilustración de abajo, es <http://localhost:43246/HelloWorld>.) La página en el navegador se verá como la siguiente captura de pantalla. En el método anterior, el código devuelto una cadena directamente. Usted dijo que el sistema vuelva sólo algo de HTML, y lo hizo!

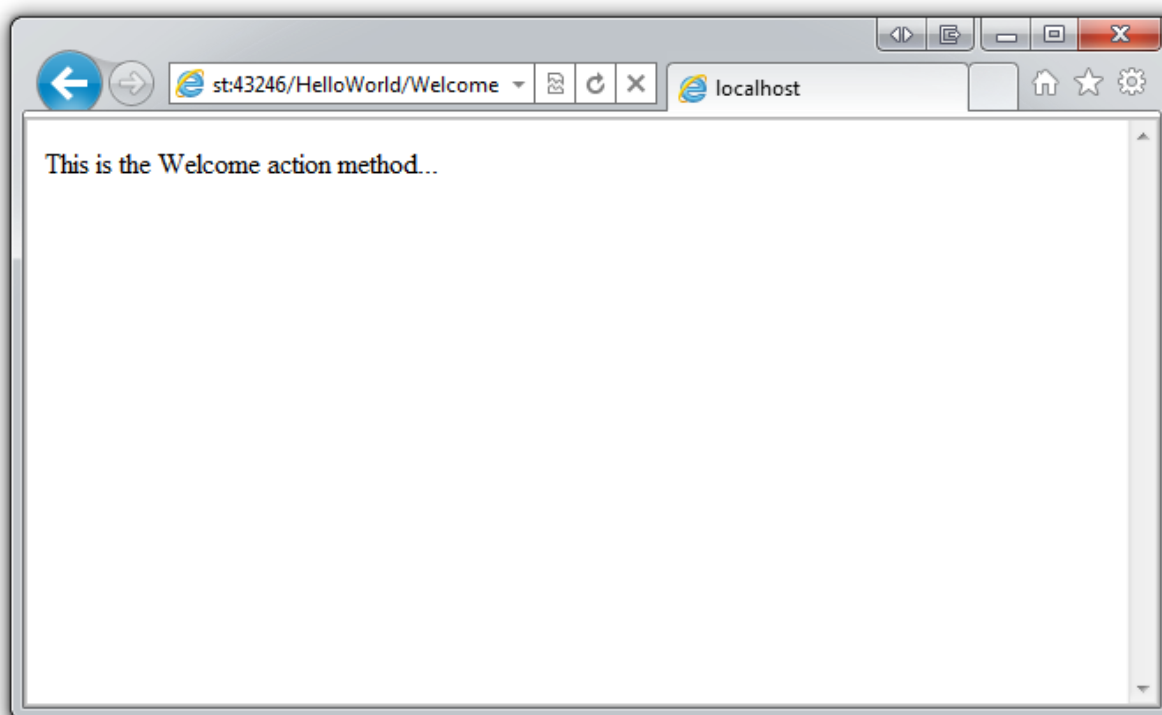


ASP.NET MVC invoca diferentes clases de controlador (y diferentes métodos de acción dentro de ellas) en función de la dirección URL entrante. La lógica de mapeo por defecto utilizado por ASP.NET MVC utiliza un formato como este para determinar cuál es el código para invocar:

/ [Controller] / [actionName] / [Parámetros]

La primera parte de la URL determina la clase de controlador de ejecutar. So / *HelloWorld* se asigna a la *HelloWorldController* clase. La segunda parte de la dirección determina el método de acción en la clase para ejecutar. Entonces / *HelloWorld* / *Índice* haría que el *Índice* método de la *HelloWorldController* clase a ejecutar. Tenga en cuenta que sólo tuvimos que navegar a / *HelloWorld* y el *índice* se utilizó el método por defecto. Esto se debe a un método denominado *índice* es el método por defecto que se llama en un controlador si no se especifica explícitamente.

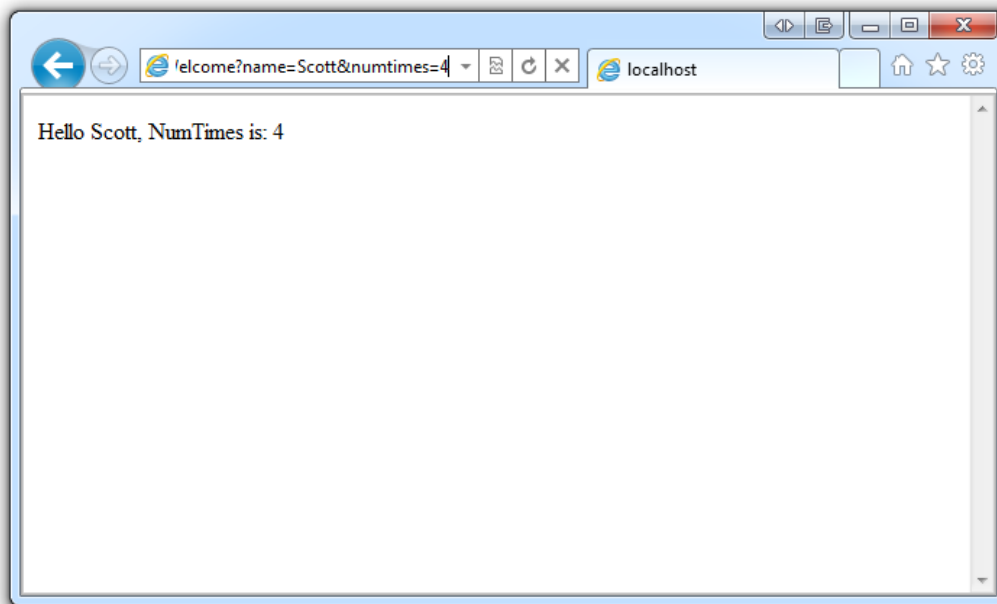
Busque *http://localhost:xxxx/HelloWorld/Welcome* . El *Welcome* método se ejecuta y devuelve la cadena "Este es el método de acción Welcome ...". La asignación predeterminada MVC es / [controller] / [] / [ActionName Parámetros] . Por esta URL, el controlador es *HelloWorld* y *Welcome* es el método de acción. No ha utilizado el [Parámetros] parte de la URL todavía.



Vamos a modificar un poco el ejemplo para que pueda pasar un poco de información sobre los parámetros de la dirección URL para el controlador (por ejemplo, / *HelloWorld* / *Welcome*? *name* = *Scott* & *numtimes* = 4). Cambiar el *Welcome* método para incluir dos parámetros, como se muestra a continuación. Tenga en cuenta que el código utiliza el C # opcional-parámetro característica para indicar que la *numTimes* parámetro en caso de incumplimiento a 1 si ningún valor es pasado para ese parámetro.

```
public string Welcome(string name, int numTimes = 1) {  
    return HttpUtility.HtmlEncode("Hello " + name + ", NumTimes is: " + numTimes);  
}
```

Ejecute la aplicación y vaya a la dirección URL de ejemplo (*http://localhost:xxxx/HelloWorld/Welcome?name=Scott&numtimes=4*) . Puede probar diferentes valores para *nombre* y *numtimes* en la URL. El sistema asigna automáticamente los parámetros con nombre de la cadena de consulta en la barra de direcciones a los parámetros de su método.



En ambos ejemplos, el controlador ha estado haciendo el "VC" parte del MVC - es decir, la visión y el trabajo de controlador. El controlador está volviendo HTML directamente. Por lo general no desea que los controladores vuelven HTML directamente, ya que resulta muy complicado de código. En cambio, por lo general vamos a usar un archivo de vista plantilla independiente para ayudar a generar la respuesta HTML. Vamos a ver que viene en la forma en que puede hacer esto.

Agregar una vista (C #)

En esta sección vas a modificar la **HelloWorldController** clase que se utiliza archivos de vista de la plantilla para encapsular limpiamente el proceso de generar respuestas HTML a un cliente.

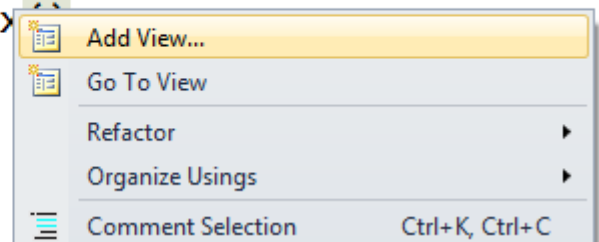
Vamos a crear un archivo de plantilla de vista con el nuevo [motor Razor vista](#) introducido con ASP.NET MVC 3. Basados en plantillas de vista Razor-tienen una `.cshtml` extensión de archivo, y proporcionan una manera elegante de crear una salida HTML usando C #. Razor minimiza el número de caracteres y pulsaciones necesarias para escribir una plantilla de vista, y permite un flujo de trabajo rápido y fluido de codificación.

Comience con una plantilla de vista con el **Índice de** método en la **HelloWorldController** clase. Actualmente, el **Índice** método devuelve una cadena con el mensaje de que está codificada en la clase del controlador. Cambiar el **Índice** método para devolver una **Vista** objeto, como se muestra en la siguiente:

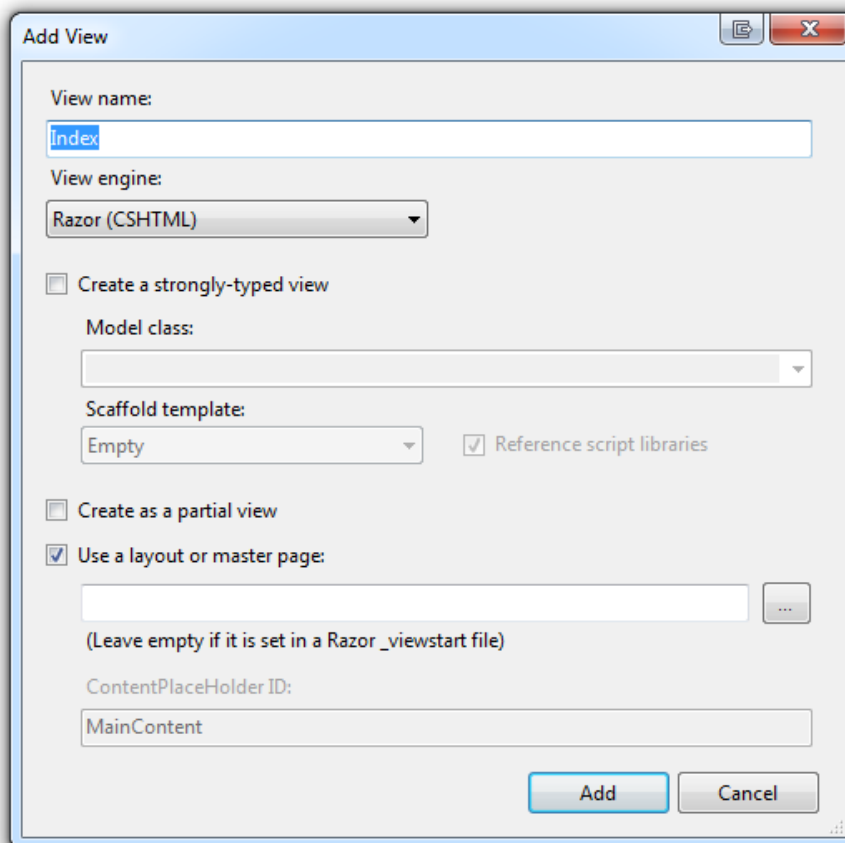
```
público ActionResult Index ()
{
    return View ();
}
```

Este código se utiliza una plantilla de vista para generar una respuesta HTML al navegador. En el proyecto, agregue una plantilla de vista que se puede utilizar con el **Índice de** método. Para ello, haga clic dentro del **Índice de** método y haga clic en **Añadir vista**.

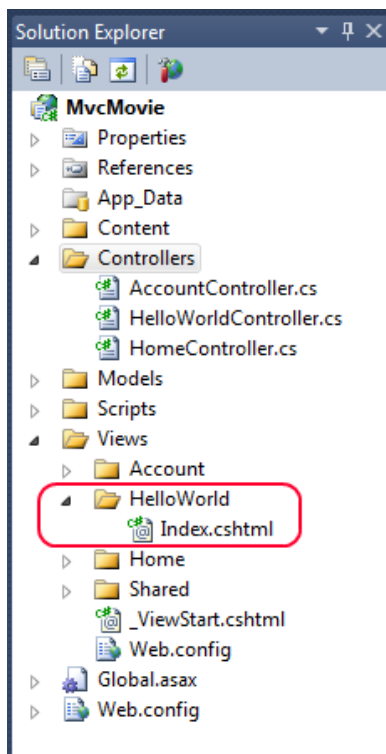
```
public class HelloWorldController : Controller
{
    public ActionResult Index
    {
        return View();
    }
}
```



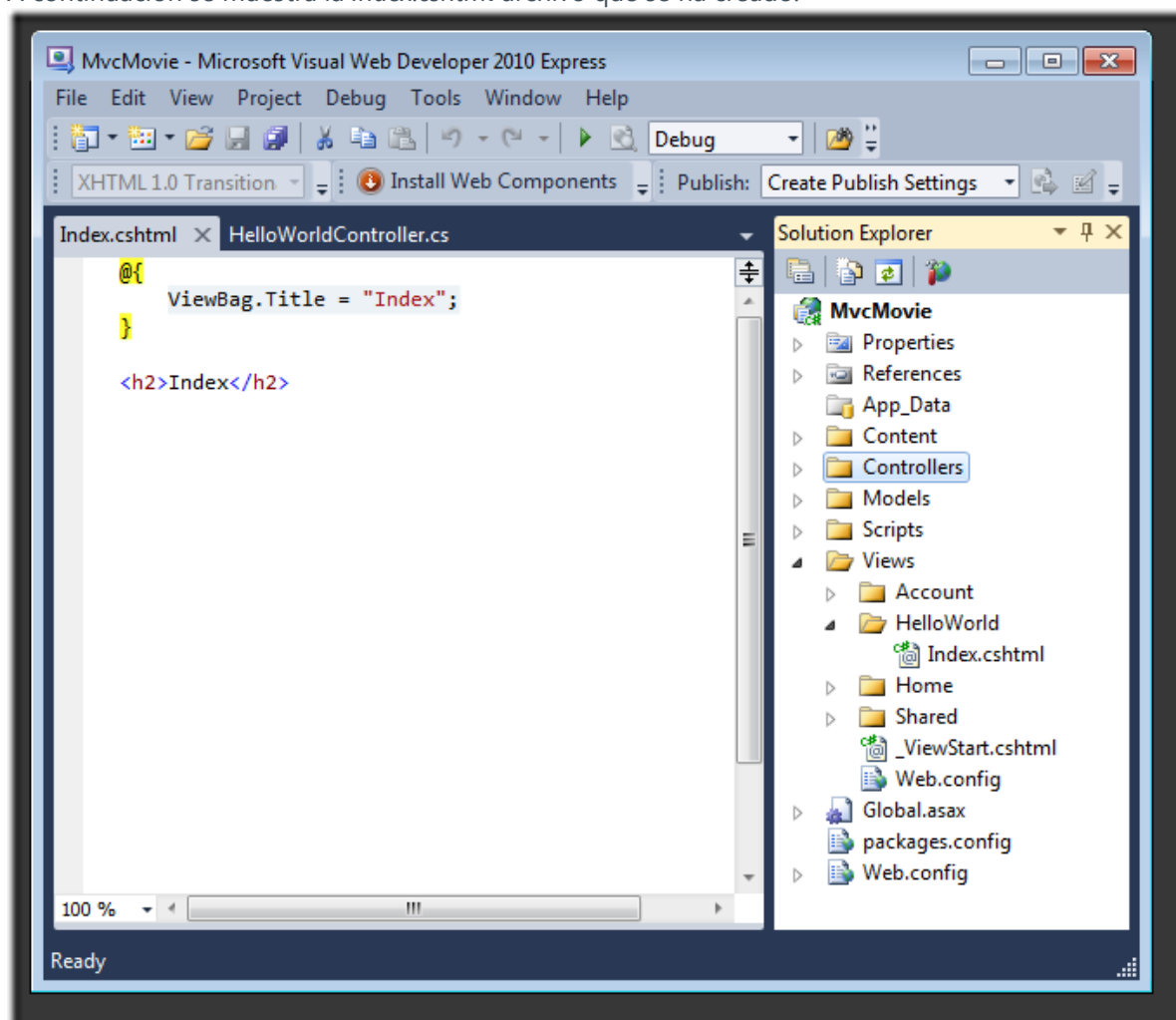
La **Vista Añadir** cuadro de diálogo. Deje los valores por defecto como están y haga clic en **Añadir** botón:



El *MvcMovie \ Views \ HelloWorld* carpeta y el *MvcMovie \ Views \ HelloWorld \ Index.cshtml* archivo se crean. Se los puede ver en el **Explorador de soluciones** :



A continuación se muestra la *Index.cshtml* archivo que se ha creado:



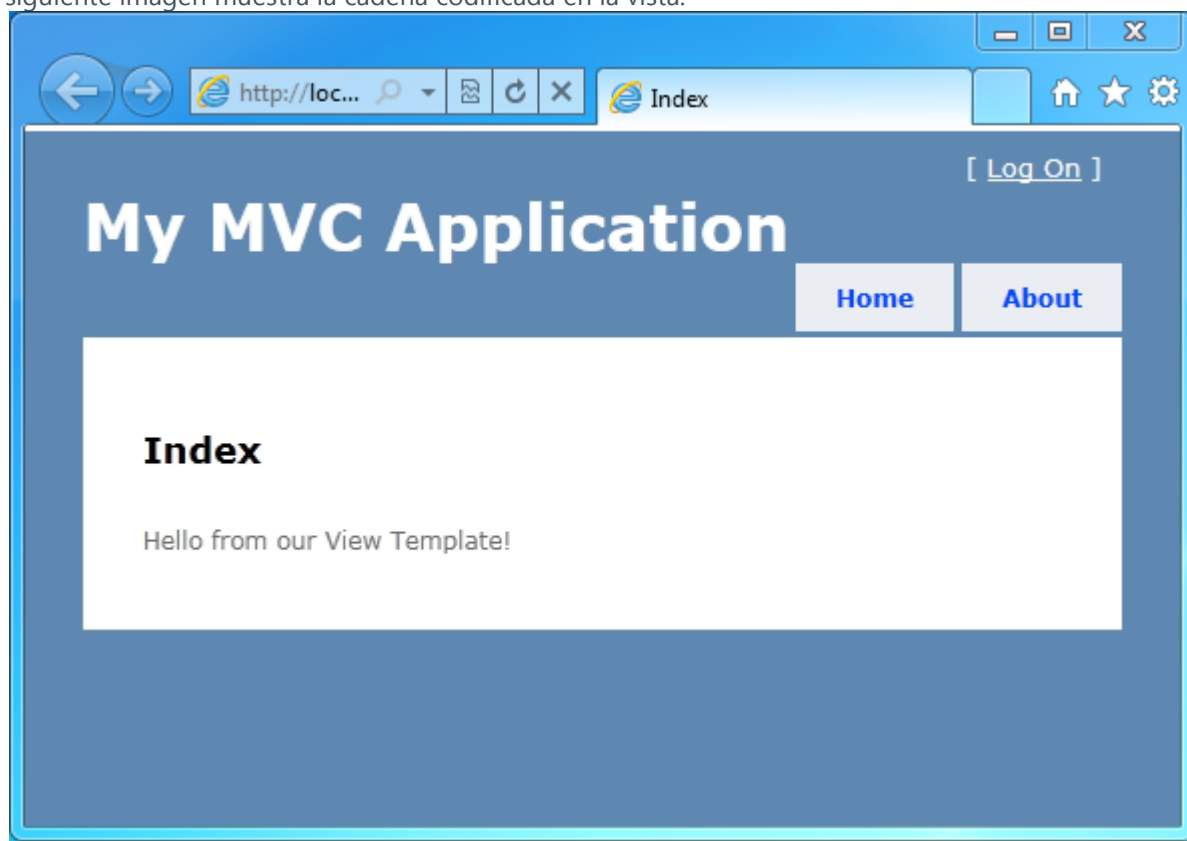
Agregue un poco de HTML en el `<h2>` etiqueta. La modificación *MvcMovie \ Views \ HelloWorld \ Index.cshtml* archivo se muestra a continuación.

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

<h2>Index</h2>

<p>Hello from our View Template!</p>
```

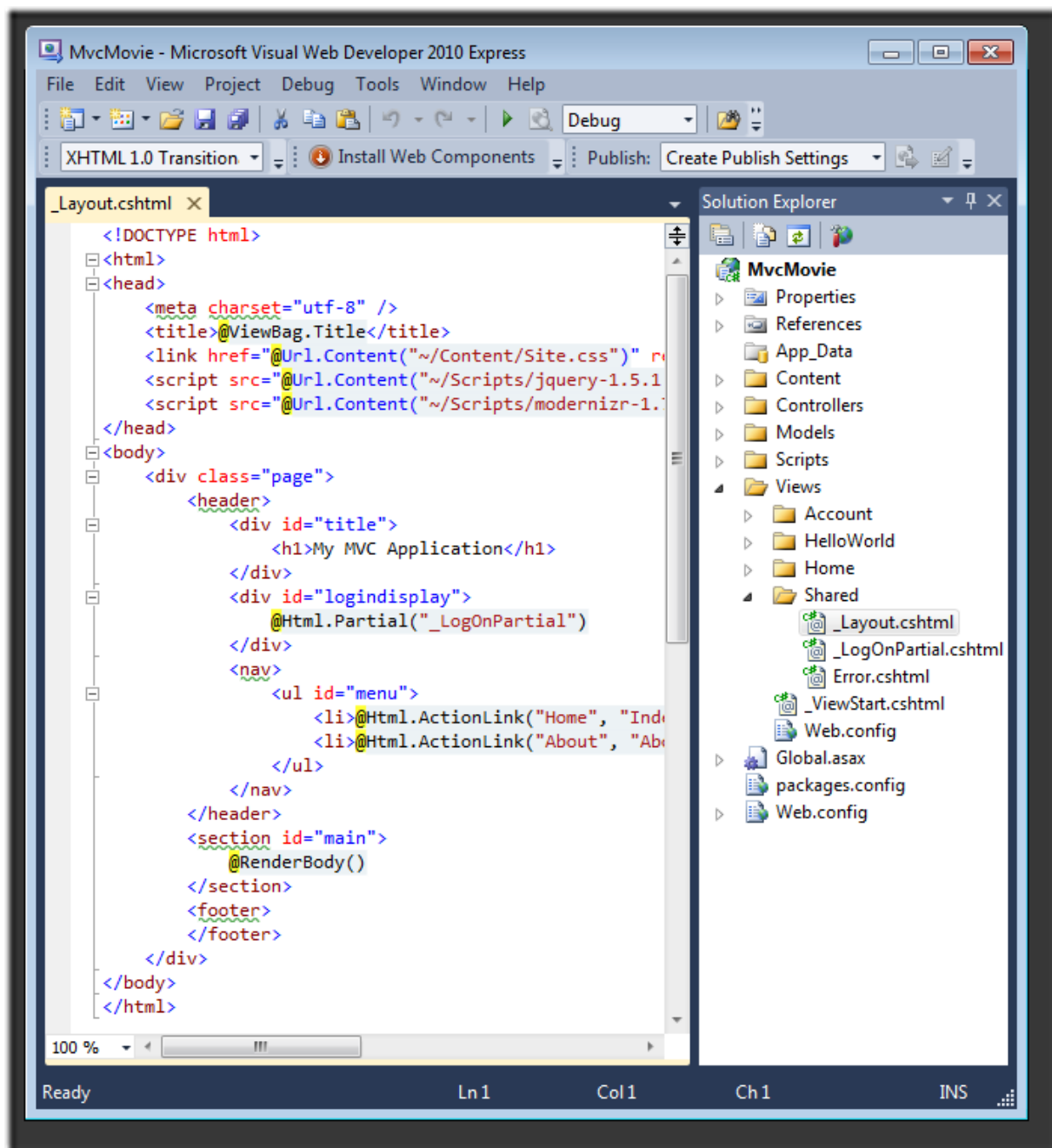
Ejecute la aplicación y busque la **HelloWorld** controlador (<http://localhost:xxxx/HelloWorld>). El **Índice** de método en el controlador no hacer mucho trabajo, sino que simplemente ejecutó la instrucción **View return ()** , que especificaba que el método debe utilizar un archivo de plantilla para hacer una respuesta al navegador. Debido a que no especifica explícitamente el nombre del archivo de plantilla de vista de utilizar, por defecto en ASP.NET MVC utilizando el *Index.cshtml* archivo de vista en el *\ Views \ HelloWorld* carpeta. La siguiente imagen muestra la cadena codificada en la vista.



Se ve muy bien. Sin embargo, observe que la barra del navegador título dice "Index" y el gran título de la página dice: "Mi aplicación MVC." Vamos a cambiar eso.

Vistas cambiantes y páginas de diseño

En primer lugar, usted quiere cambiar el "Mi aplicación MVC" de título en la parte superior de la página. Ese texto es común a todas las páginas. En realidad, se lleva a cabo en un solo lugar en el proyecto, a pesar de que aparece en cada página de la aplicación. Ir a los */ Views / Shared* carpetas en **el Explorador de soluciones** y abra la *_Layout.cshtml* archivo. Este archivo se llama un *diseño de página* y es la que compartimos "shell" que todas las demás páginas usar.



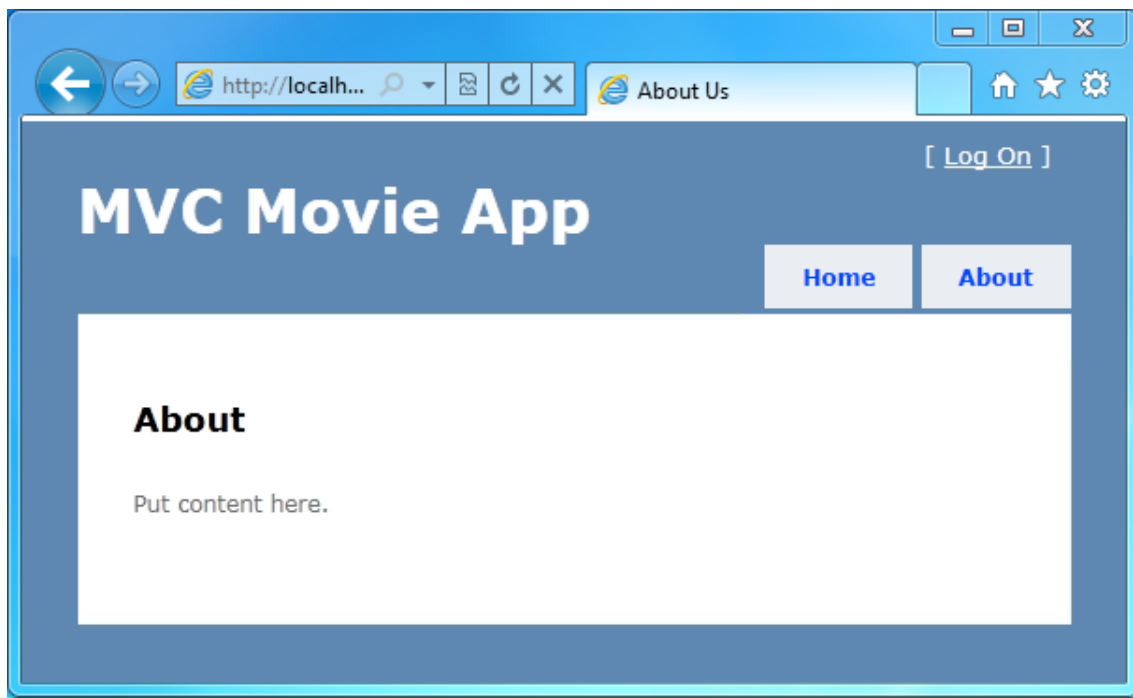
Plantillas de diseño le permiten especificar el diseño contenedor HTML de su sitio en un lugar y luego aplicarlo en varias páginas de su sitio. Tenga en cuenta la `RenderBody @ ()` línea cerca de la parte inferior del archivo. `RenderBody` es un marcador de posición en todas las páginas específicas de la vista que cree se presenta, "envuelto" en la página de diseño. Cambiar el título de la partida en la plantilla de diseño de "mi aplicación MVC" a "Movie MVC App".

```

<div id="title">
  <h1>MVC Movie App</h1>
</div>

```

Ejecute la aplicación y observe que ahora dice "App Movie MVC". Haga clic en el **Acerca de** enlace, y ves cómo esa página aparece "MVC Movie App", también. Hemos sido capaces de hacer el cambio una vez en la plantilla de diseño y tener todas las páginas del sitio refleja el nuevo título.



La completa *_Layout.cshtml* archivo se muestra a continuación:

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8" />
    <title>@ViewBag.Title</title>
    <link href="@Url.Content("~/Content/Site.css")" rel="stylesheet" type="text/css" />
    <script src="@Url.Content("~/Scripts/jquery-1.5.1.min.js")"
type="text/javascript"></script>
    <script src="@Url.Content("~/Scripts/modernizr-1.7.min.js")"
type="text/javascript"></script>
</head>
<body>
    <div class="page">
        <header>
            <div id="title">
                <h1>MVC Movie App</h1>
            </div>
            <div id="logindisplay">
                @Html.Partial("_LogOnPartial")
            </div>
            <nav>
                <ul id="menu">
                    <li>@Html.ActionLink("Home", "Index", "Home")</li>
                    <li>@Html.ActionLink("About", "About", "Home")</li>
                </ul>
            </nav>
        </header>
        <section id="main">
            @RenderBody()
        </section>
    </div>
</body>
</html>
```

```

        </section>
        <footer>
        </footer>
    </div>
</body>
</html>

```

Ahora, vamos a cambiar el título de la página Índice (ver).
 Abierto *MvcMovie \ Views \ HelloWorld \ Index.cshtml* . Hay dos lugares para hacer un cambio: en primer lugar, el texto que aparece en el título del navegador, y luego en el encabezado secundario (el **<h2>** elemento). Te hacen un poco diferente para que pueda ver qué parte de los cambios de código que parte de la aplicación.

```

@ {
    ViewBag.Title = "Lista de películas";
}

<h2> Mi Lista de películas </ h2>

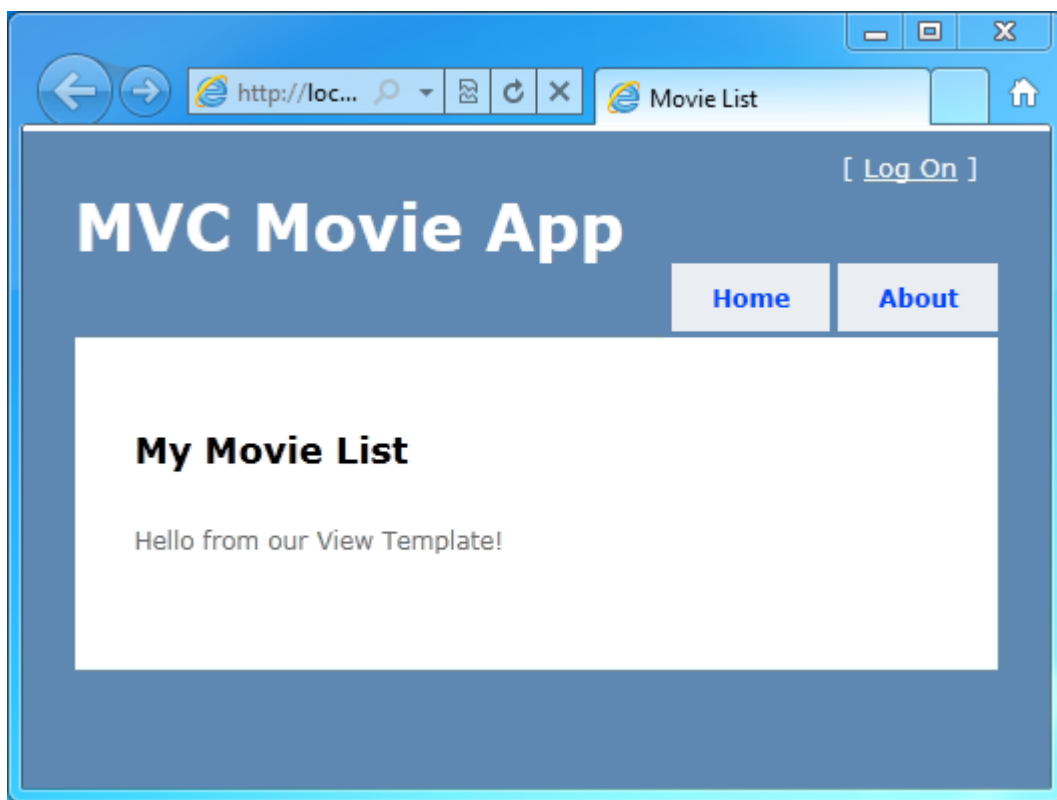
<p> ! Hola de nuestra plantilla View </ p>

```

Para indicar el título del HTML para mostrar el código anterior establece un **Título de** propiedad del **ViewBag** objeto (que se encuentra en la *Index.cshtml* plantilla de vista). Si uno mira hacia atrás en el código fuente de la plantilla de diseño, te darás cuenta de que la plantilla utiliza este valor en el **<title>** elemento como parte de la **<head>** sección del HTML. El uso de este enfoque, se puede pasar fácilmente a otros parámetros entre su plantilla de vista y su archivo de diseño.

Ejecute la aplicación y vaya a *http://localhost:xx/HelloWorld* . Tenga en cuenta que el título del navegador, el título principal, y los títulos secundarios han cambiado. (Si no ve los cambios en el navegador, es posible que esté viendo el contenido almacenado en caché. Presione Ctrl + F5 en su navegador para forzar la respuesta del servidor se va a cargar.)

Observe también cómo el contenido en el *Index.cshtml* plantilla de vista se fusionó con el *_Layout.cshtml* plantilla de vista y una respuesta HTML solo fue enviado al navegador. Plantillas de diseño que sea realmente fácil hacer cambios que se aplican a todas las páginas de la aplicación.



Nuestro poco de "datos" (en este caso el "Hola desde nuestra plantilla View!" Mensaje) es modificable, sin embargo. La aplicación MVC tiene una "V" (ver) y que tenga una "C" (control), pero no "M" (modelo) todavía. En breve, vamos a caminar a través de cómo crear una base de datos y recuperar los datos del modelo de la misma.

Pasar datos desde el Controlador a la Vista

Antes de ir a una base de datos y hablar acerca de los modelos, sin embargo, vamos a hablar primero de la transmisión de información desde el controlador a una vista. Clases de controlador se invoca en respuesta a una petición de URL entrante. Una clase controlador es donde se escribe el código que controla los parámetros de entrada, recupera datos de una base de datos, y en última instancia decide qué tipo de respuesta para enviar de vuelta al navegador. Ver plantillas se pueden utilizar desde un controlador para generar y dar formato a una respuesta HTML al navegador.

Los controladores son responsables de proporcionar lo que los datos u objetos son necesarios para una plantilla de vista para representar una respuesta al navegador. Una plantilla de vista nunca debe realizar la lógica de negocio o interactuar con una base de datos directamente. En su lugar, debería funcionar sólo con los datos que se le había facilitado el controlador. El mantenimiento de esta "separación de preocupaciones" ayuda a mantener el código limpio y más fácil de mantener.

En la actualidad, la **bienvenida** método de acción en el **HelloWorldController** clase tiene un **nombre** y un **numTimes** parámetro y luego envía los valores directamente en el navegador. En lugar de tener el controlador de prestar esta respuesta como una cadena, vamos a cambiar el controlador a utilizar una plantilla de vista en su lugar. La plantilla de vista va a generar una respuesta dinámica, lo que significa que usted necesita para pasar los bits apropiados de los datos del controlador a la vista con el fin de generar la respuesta. Usted puede hacer esto haciendo que el controlador de poner los datos dinámicos que la plantilla de vista necesita en un **ViewBag** objeto de que la plantilla de vista se puede acceder.

Volver al **HelloWorldController.cs** archivo y cambiar el **Welcome** método para agregar un **mensaje** y **NumTimes** valor a la **ViewBag** objeto. **ViewBag** es un objeto dinámico, lo que significa que puedes poner lo que quieras para ella, la **ViewBag** objeto no tiene propiedades definidas hasta usted pone algo en su interior. La completa **HelloWorldController.cs** archivo es el siguiente:

```
using System.Web;
using System.Web.Mvc;

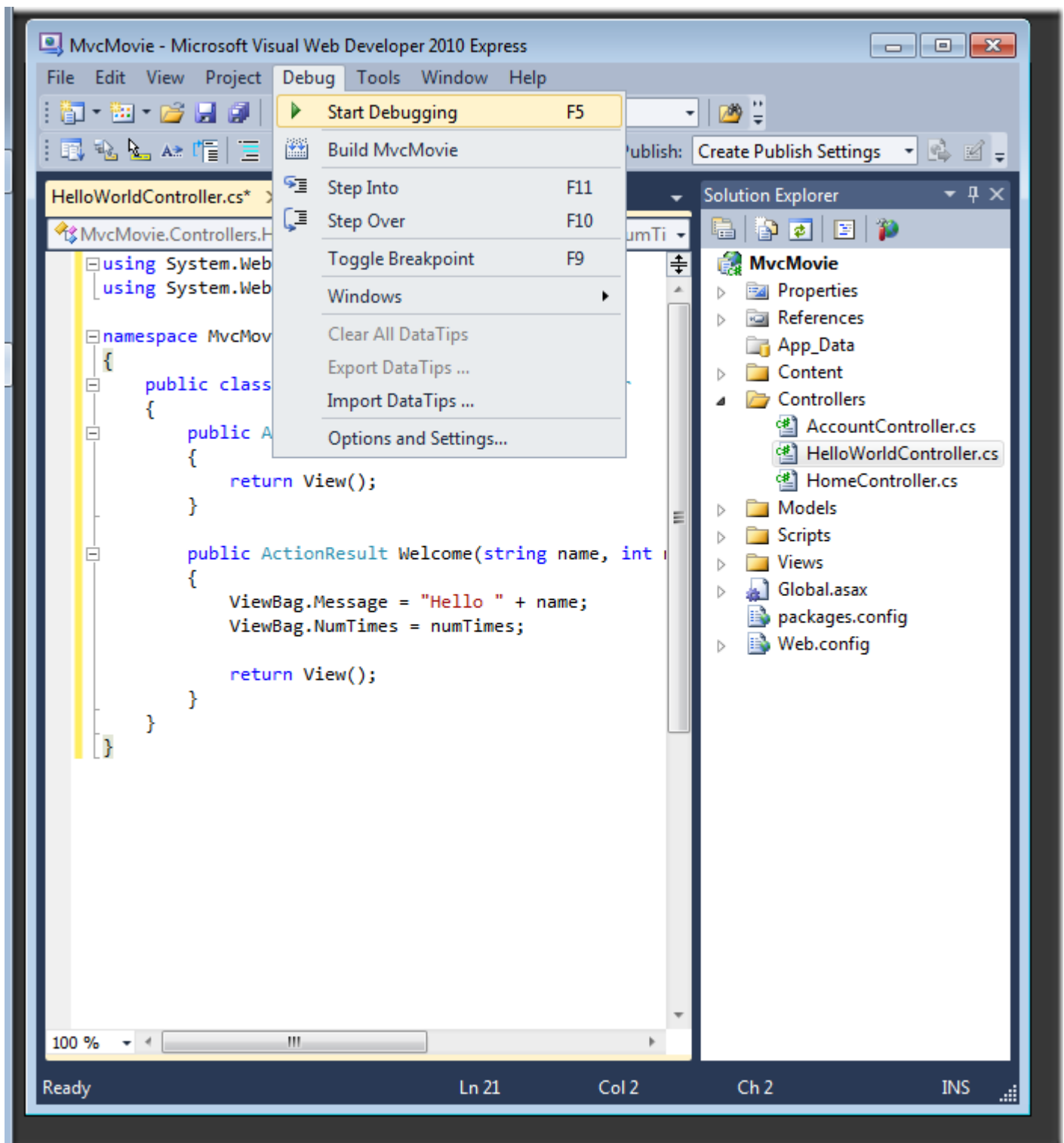
namespace MvcMovie.Controllers
{
    public class HelloWorldController : Controller
    {
        public ActionResult Index()
        {
            return View();
        }

        public ActionResult Welcome(string name, int numTimes = 1)
        {
            ViewBag.Message = "Hello " + name;
            ViewBag.NumTimes = numTimes;

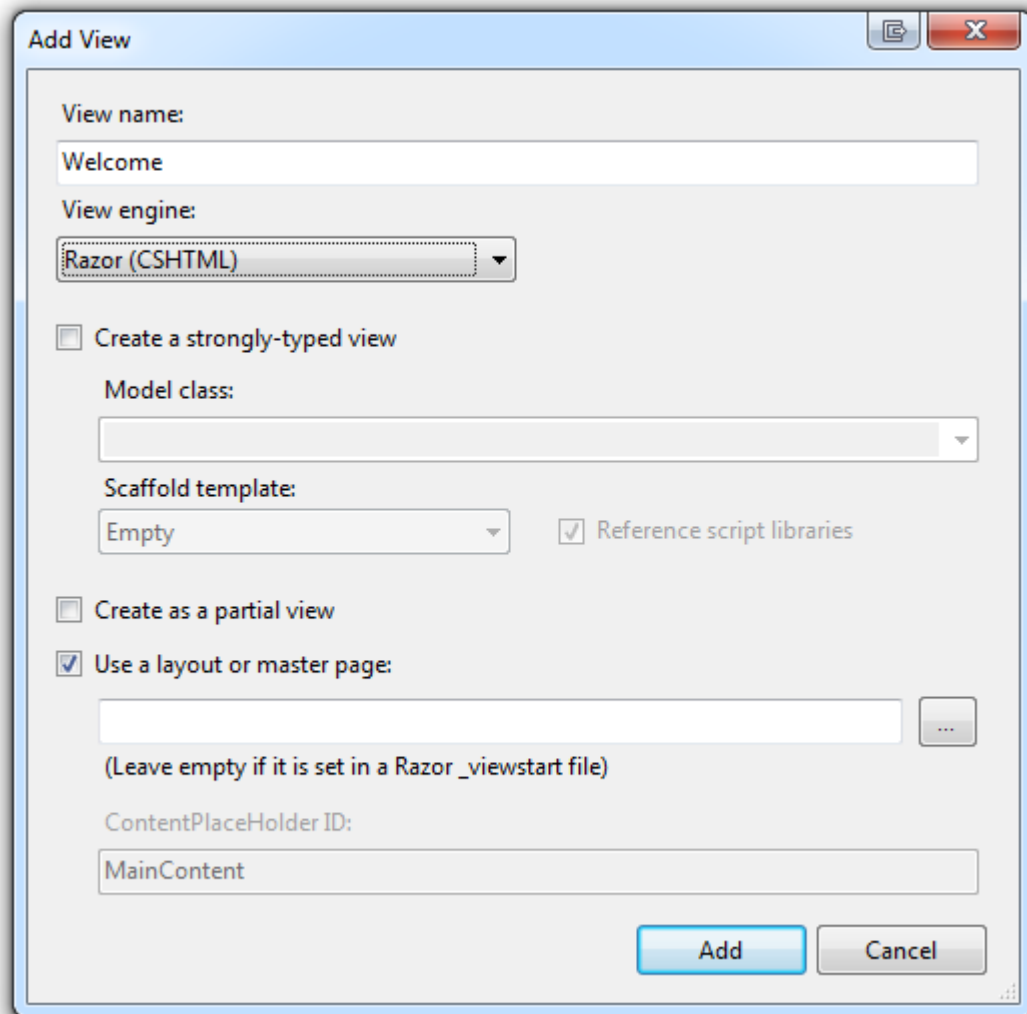
            return View();
        }
    }
}
```

Ahora el **ViewBag** objeto contiene datos que se pasarán a la vista de forma automática.

Después, usted necesita una plantilla de vista Bienvenido! En la **depuración** del menú, seleccione **Crear MvcMovie** para asegurar que el proyecto se compila.



A continuación, haga clic dentro del **Welcome** método y haga clic en **Añadir vista** . Esto es lo que el **Add View** cuadro de diálogo se parece a:



Haga clic en **Agregar** y, a continuación, agregue el código siguiente en la `<h2>` elemento en el nuevo `Welcome.cshtml` archivo. Vamos a crear un bucle que dice "Hello" tantas veces como el usuario dice que debería. La completa `Welcome.cshtml` archivo se muestra a continuación.

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

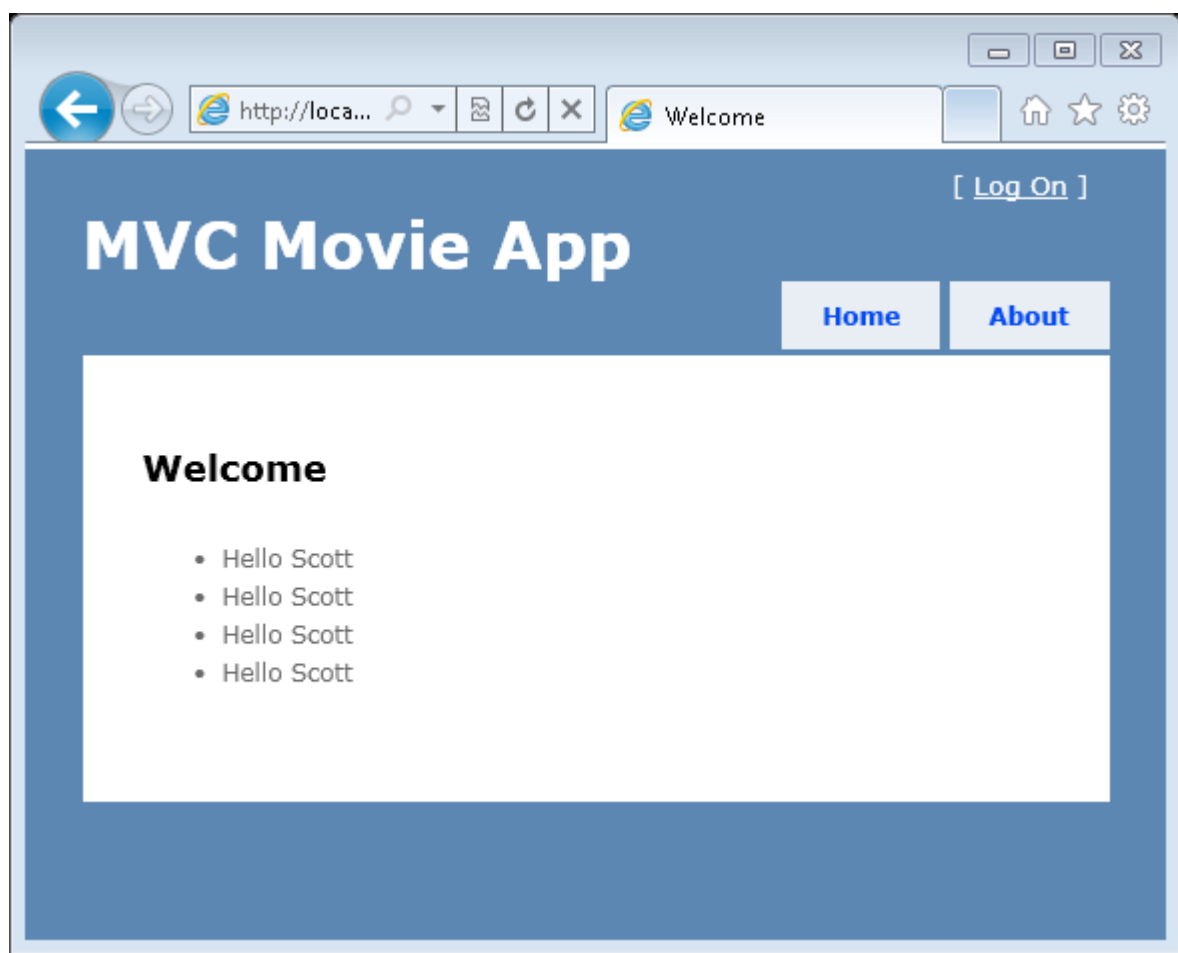
<h2>Welcome</h2>

<ul>
    @for (int i=0; i < ViewBag.NumTimes; i++) {
        <li>@ViewBag.Message</li>
    }
</ul>
```

Ejecute la aplicación y vaya a la siguiente URL:

<http://localhost:xx/HelloWorld/Welcome?name=Scott&numtimes=4>

Ahora los datos se toman a partir de la URL y se pasa al controlador automáticamente. Los paquetes de controlador de los datos en un **ViewBag** objeto y pasa ese objeto a la vista. La vista a continuación, muestra los datos como HTML para el usuario.



Bueno, eso fue una especie de "M" para el modelo, pero no es el tipo de base de datos. Vamos a tomar lo que hemos aprendido y crear una base de datos de películas.

Agregar un modelo (C #)

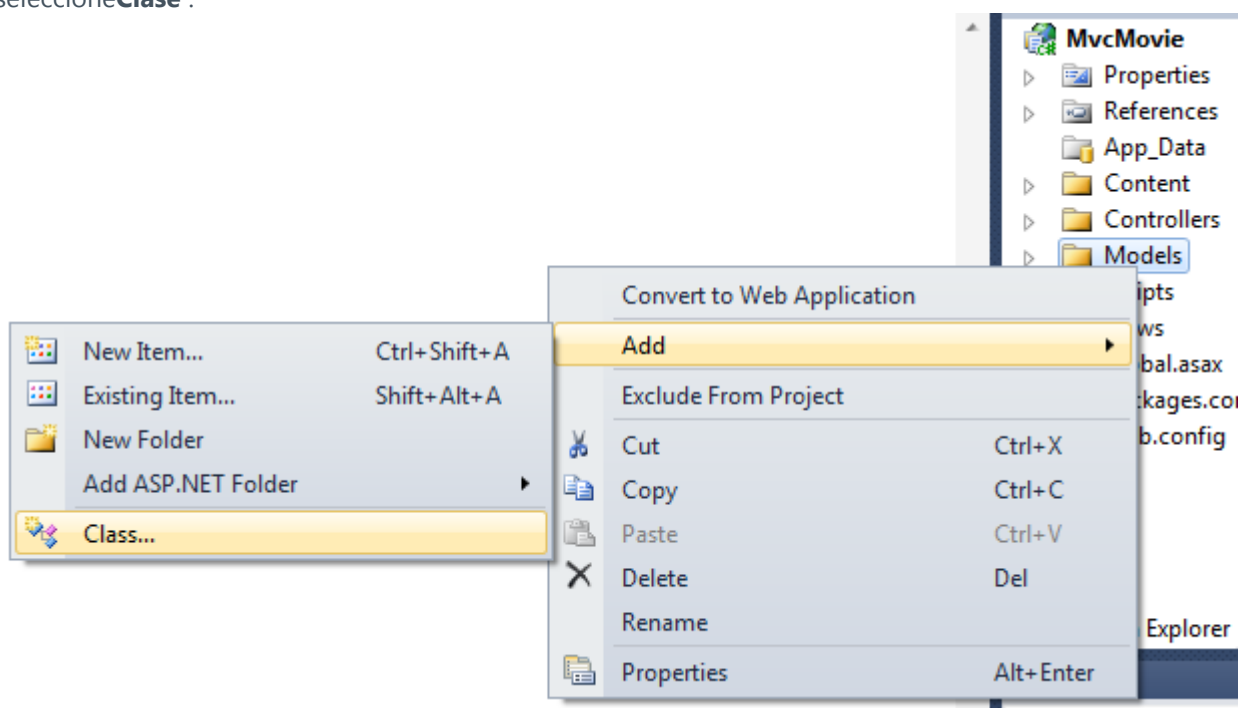
Agregar un modelo

En esta sección vamos a añadir algunas clases para administrar películas en una base de datos. Estas clases serán el "modelo" de la aplicación ASP.NET MVC.

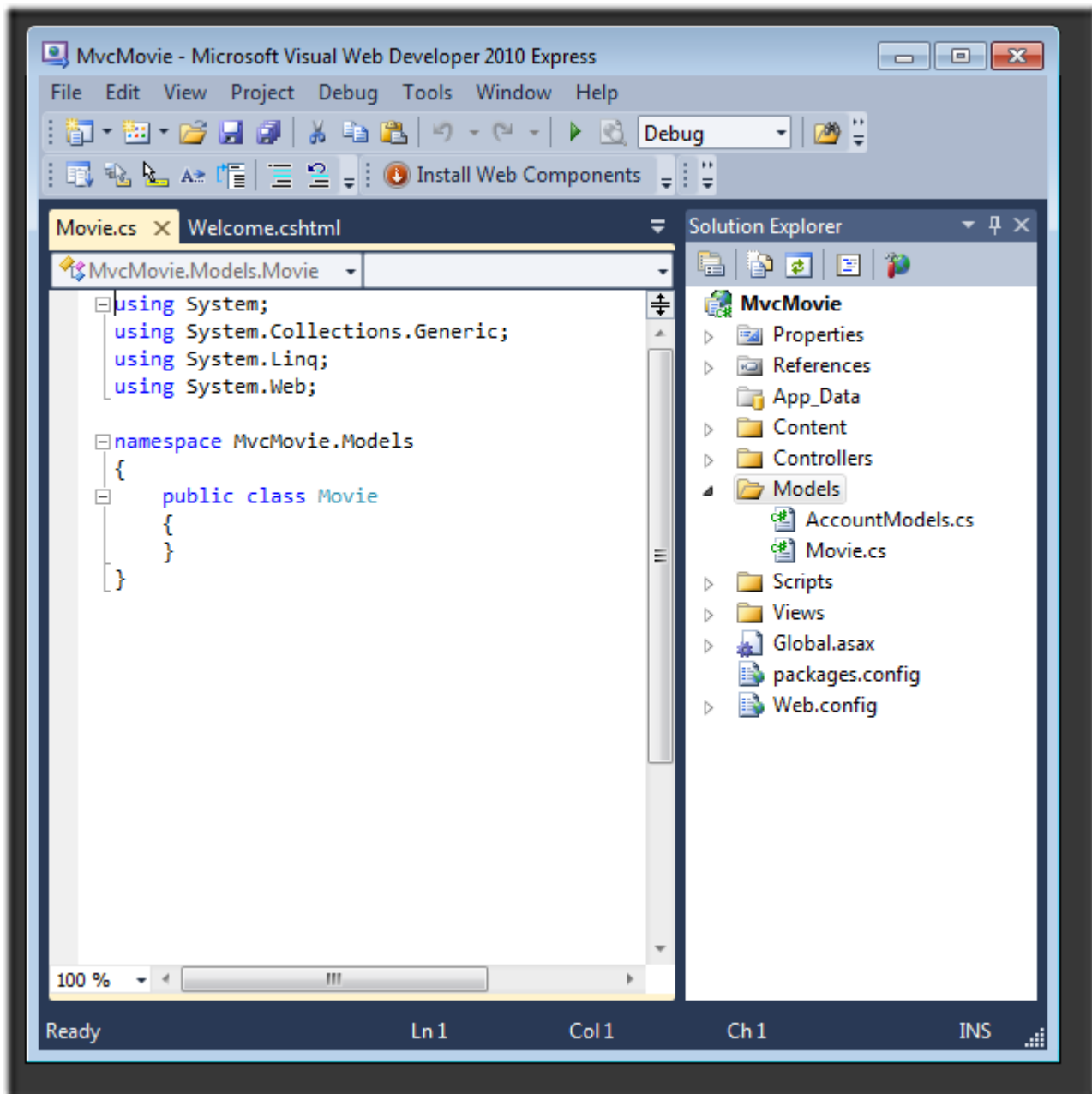
Vamos a usar una .NET Framework de acceso a datos tecnología conocida como Entity Framework para definir y trabajar con estas clases del modelo. Entity Framework (a menudo denominado como EF) es compatible con un modelo de desarrollo llamado *Primer Código* . Primer Código permite crear objetos de modelo de clases de escritura simple. (Estos también se conocen como clases POCO, de "llanura de edad, objetos CLR"). A continuación, puede tener la base de datos creada sobre la marcha de sus clases, lo que permite un flujo de trabajo de desarrollo muy limpia y rápida.

Adición de clases del modelo

En el **Explorador de soluciones** , haga clic con el *Modelos* carpeta, seleccione **Agregar** y, a continuación, seleccione **Clase** .



Nombre de la *clase* "Película".



Agregue los siguientes cinco propiedades a la **película** de clase:

```
public class Movie
{
    public int ID { get; set; }
    public string Title { get; set; }
    public DateTime ReleaseDate { get; set; }
    public string Genre { get; set; }
    public decimal Price { get; set; }
}
```

Usaremos la **película** clase para representar películas en una base de datos. Cada instancia de una **película** objeto se corresponde con una fila dentro de una tabla de base de datos, y cada propiedad de la **Película** clase asignada a una columna en la tabla.

En el mismo archivo, agregue el siguiente **MovieDbContext** clase:

```
public class MovieDbContext : DbContext
{
    public DbSet<Movie> Movies { get; set; }
}
```

El **MovieDbContext** clase representa el Entity Framework película contexto de base de datos, que se encarga ir a buscar, almacenar y actualizar **Película** instancias de clase en una base de datos. El **MovieDbContext** deriva de la **DbContext** clase base proporcionada por el Entity Framework. Para obtener más información acerca de **DbContext** y **DbSet**, consulte [Mejora de la Productividad para Entity Framework](#).

Con el fin de ser capaz de referencia **DbContext** y **DbSet**, es necesario agregar la siguiente **utilizando** declaración al principio del archivo:

```
utilizando System . datos . Entidad ;
```

La completa *Movie.cs* archivo se muestra a continuación.

```
using System;
using System.Data.Entity;

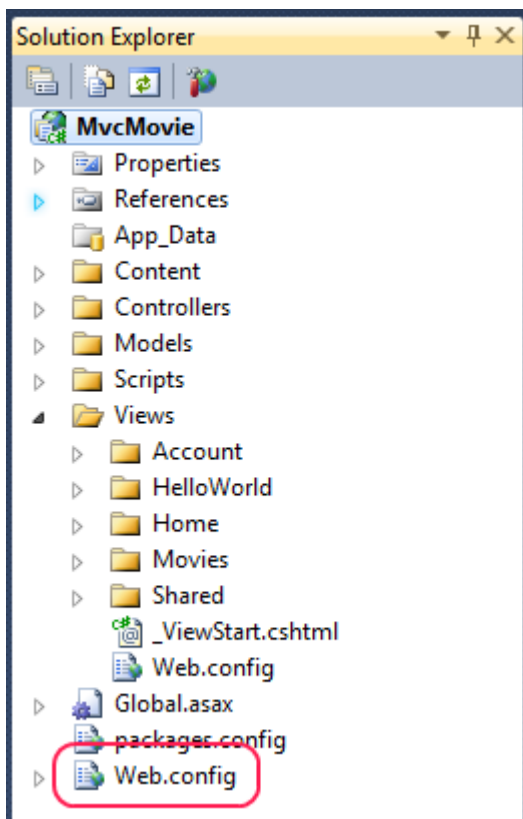
namespace MvcMovie.Models
{
    public class Movie
    {
        public int ID { get; set; }
        public string Title { get; set; }
        public DateTime ReleaseDate { get; set; }
        public decimal Price { get; set; }
        public string Genre { get; set; }
    }

    public class MovieDbContext : DbContext
    {
        public DbSet<Movie> Movies { get; set; }
    }
}
```

Crear una cadena de conexión y trabajar con SQL Server Compact

El **MovieDbContext** clase que creó encarga de la tarea de la conexión a la base de datos y el mapeo de **películas** objetos a los registros de la base de datos. Una de las preguntas que usted puede preguntar, sin embargo, es cómo especificar qué base de datos va a conectar. Que va a hacer que al añadir la información de conexión en el *archivo Web.config* de archivo de la aplicación.

Abra la raíz de la aplicación *Web.config* archivo. (No es el *Web.config* archivo en el *Reproducciones* carpeta) La imagen de abajo muestra tanto *Web.config* archivos, abra el *archivo Web.config* archivo de un círculo rojo.



Agregue la siguiente cadena de conexión a la `<connectionStrings>` elemento en el *Web.config* archivo.

```
<add name="MovieDBContext"
      connectionString="Data Source=|DataDirectory|Movies.sdf"
      providerName="System.Data.SqlClient" />
```

El ejemplo siguiente muestra una porción de la *Web.config* archivo con la nueva cadena de conexión añadido:

```
<configuration>
  <connectionStrings>
    <add name="MovieDBContext"
          connectionString="Data Source=|DataDirectory|Movies.sdf"
          providerName="System.Data.SqlClient" />
    <add name="ApplicationServices"
          connectionString="data source=.\SQLEXPRESS;Integrated
Security=SSPI;AttachDBFilename=|DataDirectory|aspnetdb.mdf;User Instance=true"
          providerName="System.Data.SqlClient" />
  </connectionStrings>
```

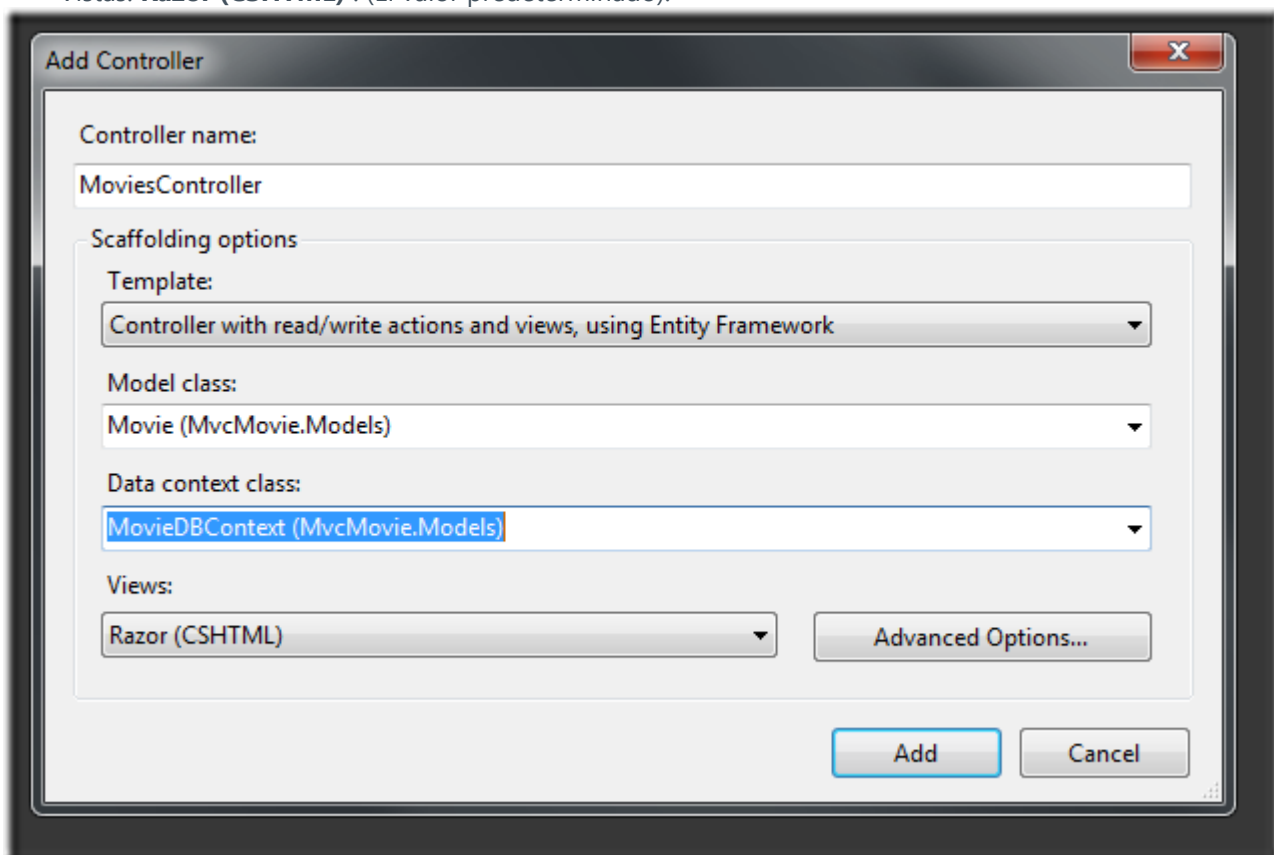
Esta pequeña cantidad de código XML y es todo lo que necesita para escribir con el fin de representar y almacenar los datos de la película en una base de datos.

A continuación, usted construirá un nuevo **MoviesController** clase que se puede utilizar para mostrar los datos de la película y que los usuarios puedan crear los nuevos anuncios de cine.

Acceso a los datos de su modelo desde un controlador (C #)

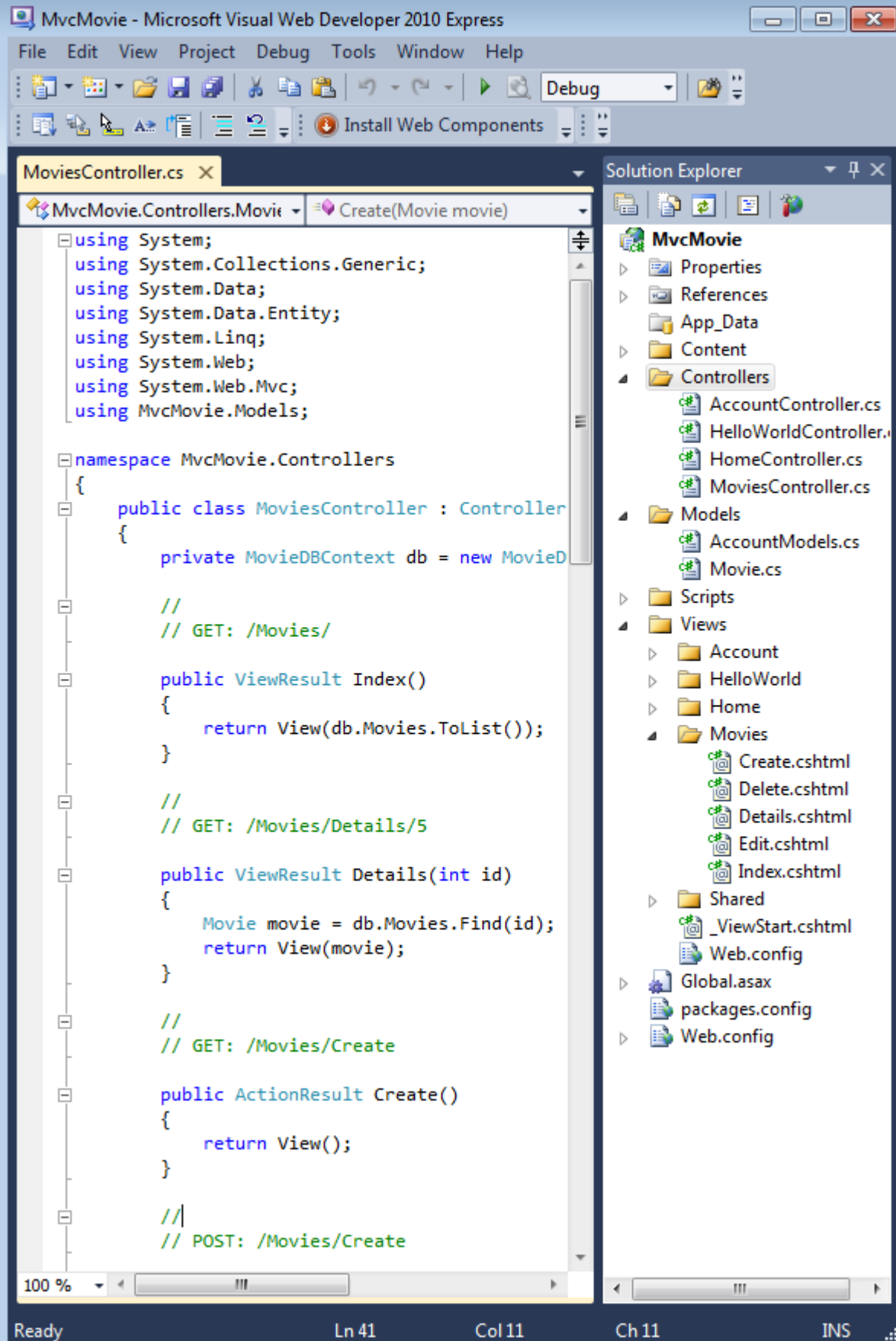
Haga clic con el *Controllers* carpeta y crear una nueva **MoviesController** controlador. Seleccione las siguientes opciones:

- Nombre del controlador: **MoviesController** . (Este es el valor predeterminado.)
- Plantilla: **Controlador de lectura / escritura de las acciones y puntos de vista, utilizando Entity Framework**.
- Modelo de clases: **Película (MvcMovie.Models)** .
- Datos clase de contexto: **MovieDbContext (MvcMovie.Models)** .
- Vistas: **Razor (CSHTML)** . (El valor predeterminado).



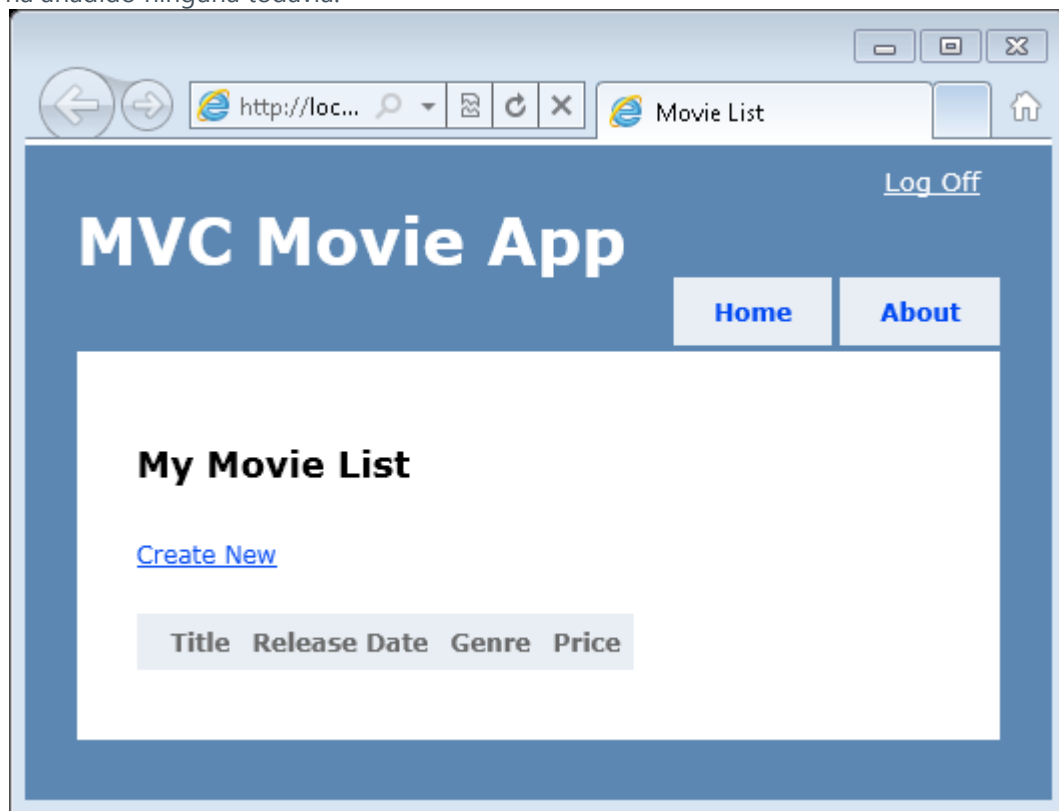
Haga clic en **Agregar** . Visual Web Developer crea los siguientes archivos y carpetas:

- A *MoviesController.cs* archivo en el proyecto de *Controladores* carpeta.
- Un *Películas* carpeta en el proyecto *Vistas* carpeta.
- *Create.cshtml*, *Delete.cshtml*, *Details.cshtml*, *Edit.cshtml* y *Index.cshtml* en los nuevos *Vistas \ Movies* carpeta.



El ASP.NET MVC 3 mecanismo de scaffolding crea automáticamente el CRUD (crear, leer, actualizar y eliminar) los métodos de acción y puntos de vista para usted. Ahora tiene una aplicación web totalmente funcional que le permite crear, listar, editar y eliminar entradas de cine.

Ejecute la aplicación y busque la **Película** controlador añadiendo */Cine* para la dirección URL en la barra de direcciones de su navegador. Debido a que la aplicación depende de la ruta por defecto (definido en el *Global.asax* file), la solicitud del explorador *http://localhost:xxxxx/Movies* se dirige a los valores de **Índice** de método de acción del **Cine** controlador. En otras palabras, la petición del navegador *http://localhost:xxxxx/Movies* es efectivamente el mismo que la petición del navegador *http://localhost:xxxxx/Movies/Index* . El resultado es una lista vacía de películas, porque no se ha añadido ninguna todavía.



Creación de una película

Selecciona la **creación de un nuevo** enlace. Introduce algunos detalles sobre una película y luego haga clic en el **Crear** botón.

http://loc... Create

[Log On]

MVC Movie App

Home About

Create

Movie

Title
When Harry Met Sally

ReleaseDate
1/1/1989

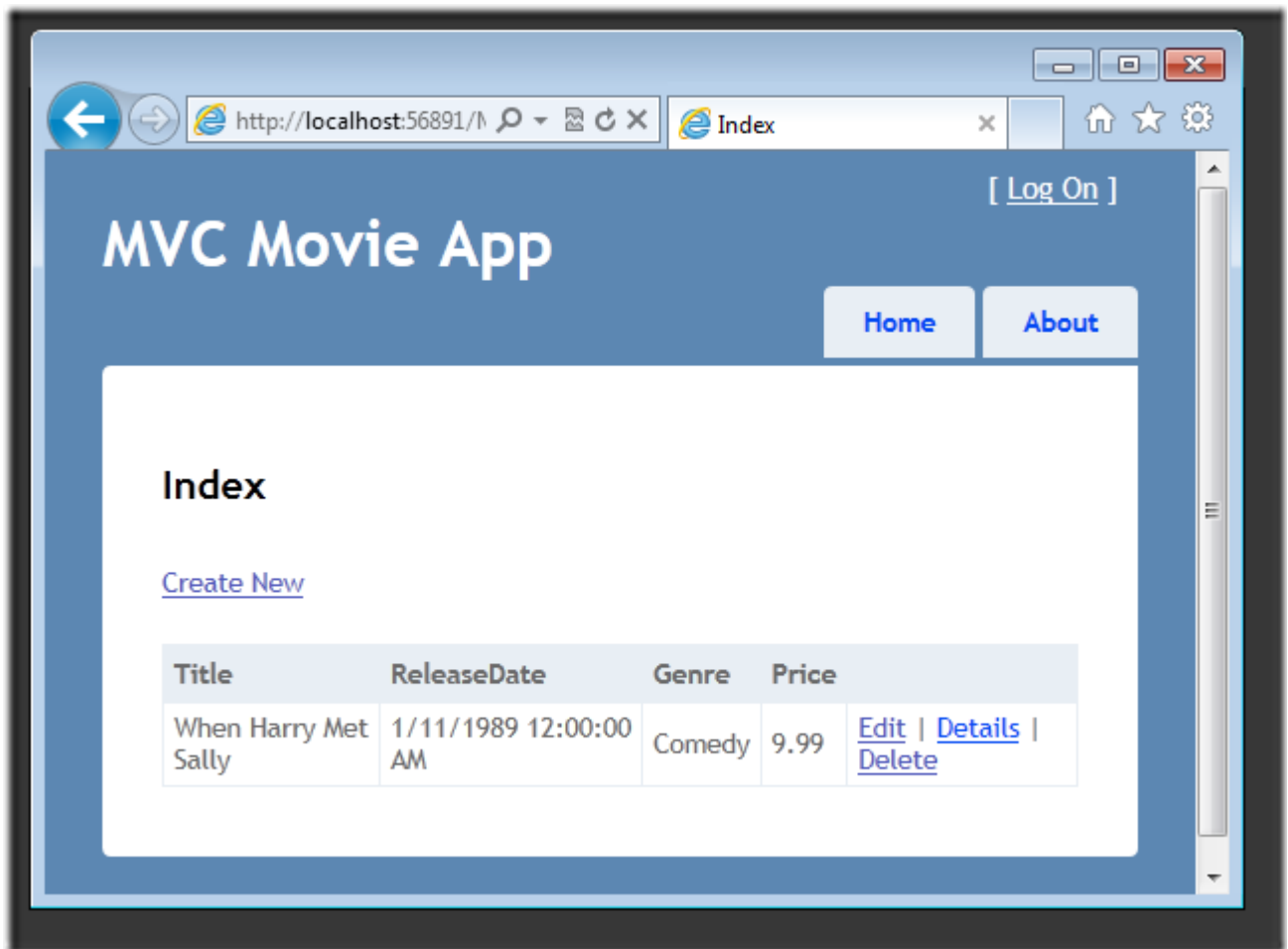
Genre
Comedy

Price
9.99

Create

[Back to List](#)

Al hacer clic en el **Crear** botón hace que el formulario se envía al servidor, donde se guarda la información de la película en la base de datos. Usted está entonces redirigido a la */ Películas* URL, donde se puede ver la película recién creado en el listado.



Crear un par de entradas de cine. Pruebe las **Editar**, **Detalles** y **Borrar** vínculos, que son todos funcionales.

Examinando el código generado

Abrir los *controladores* \ *MoviesController.cs* archivos y examinar el generado por el **Índice** de método. Una parte del controlador de la película con el **Índice** de método se muestra a continuación.

```
public class MoviesController : Controller
{
    private MovieDbContext db = new MovieDbContext();

    //
    // GET: /Movies/

    public ActionResult Index()
    {
        return View(db.Movies.ToList());
    }
}
```

La siguiente línea de la *MoviesController* clase crea una instancia de contexto de base de datos de películas, tal como se describe anteriormente. Usted puede utilizar el contexto de base de datos de película para consultar, editar y borrar películas.

```
private MovieDbContext db = new MovieDbContext();
```

Una petición al **Películas** controlador devuelve todas las entradas en el **Cine** tabla de la base de datos de cine y luego pasa los resultados al **Índice** de vista.

Modelos de tipos fuertes y la palabra clave modelo @

Al principio de este tutorial, vimos cómo un controlador puede pasar datos u objetos a una plantilla de vista con la **ViewBag** objeto. El **ViewBag** es un objeto dinámico que proporciona un cómodo tiempo de ejecución manera de pasar información a un punto de vista.

ASP.NET MVC también ofrece la posibilidad de pasar datos fuertemente tipados u objetos a una plantilla de vista. Este enfoque inflexible de tipos permite un mejor control en tiempo de compilación de su código y rica IntelliSense en el editor de Visual Web Developer. Estamos utilizando este enfoque con

el **MoviesController** clase y **Index.cshtml** plantilla de vista.

Observe cómo el código crea una **lista de** objeto cuando se llama a la **Vista de** método auxiliar en el **Índice de** método de acción. El código pasa entonces esta **Películas** lista desde el controlador a la vista:

```
public ViewResult Index()
{
    return View(db.Movies.ToList());
}
```

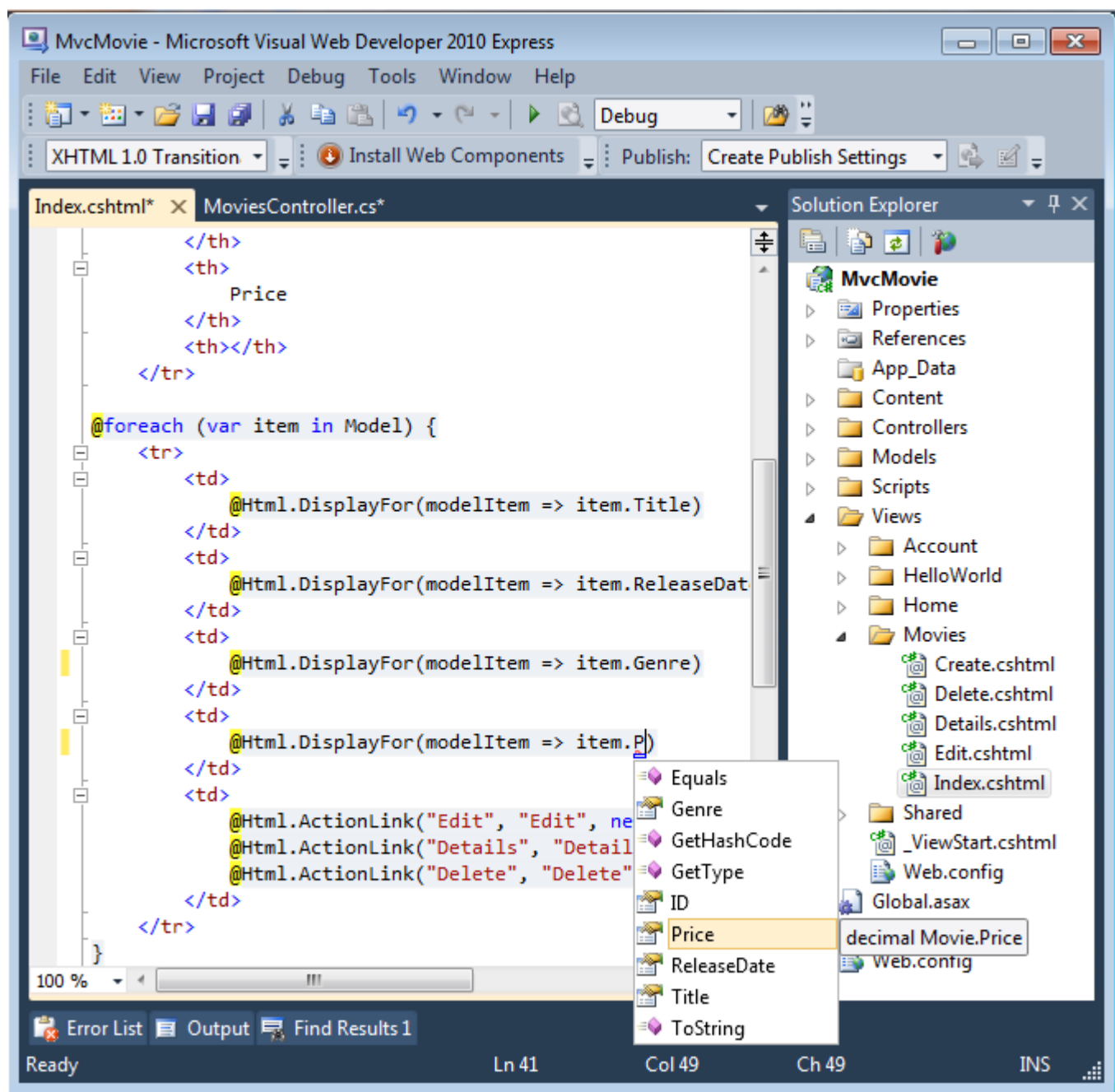
Mediante la inclusión de un **modelo @** declaración en la parte superior del archivo de plantilla de vista, se puede especificar el tipo de objeto que la vista se espera. Cuando se creó el controlador de la película, Visual Web Developer incluye automáticamente los siguientes **@ modelo** declaración en la parte superior de la **Index.cshtml** del archivo:

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

Este **modelo @** directiva le permite acceder a la lista de películas que el controlador pasa a la vista mediante el uso de un **modelo de** objeto que está fuertemente tipado. Por ejemplo, en el **Index.cshtml** plantilla, el código recorre el cine haciendo un **foreach** declaración sobre el establecimiento inflexible de tipos **modelo** objeto:

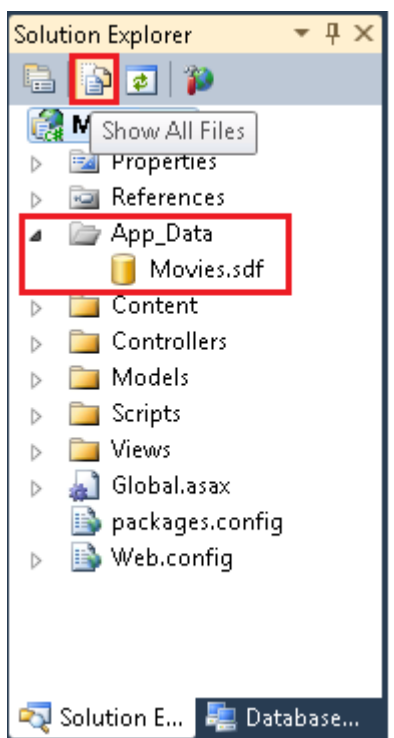
```
@ Foreach (var elemento en el modelo) {
    <tr>
        <td>
            @ Html.DisplayFor (ModelItem => item.Title)
        </ td>
        <td>
            @ Html.DisplayFor (ModelItem => item.ReleaseDate)
        </ td>
        <td>
            @ Html.DisplayFor (ModelItem => item.Genre)
        </ td>
        <td>
            @ Html.DisplayFor (ModelItem => item.Price)
        </ td>
        <td>
            @ Html.ActionLink ("Edit", "Editar", nuevo {id = item.ID}) |
            @ Html.ActionLink ("Detalles" y "Detalles", nuevo {id = item.ID}) |
            @ Html.ActionLink ("Borrar", "Borrar", nuevo {id = item.ID})
        </ td>
    </ tr>
}
```

Debido a que el **modelo de** objeto es fuertemente tipado (como **<Película> IEnumerable** de objetos), cada **elemento de** objeto en el circuito es de tipo **película** . Entre otras ventajas, esto significa que usted obtiene en tiempo de compilación comprobación del código y compatible con IntelliSense en el editor de código:



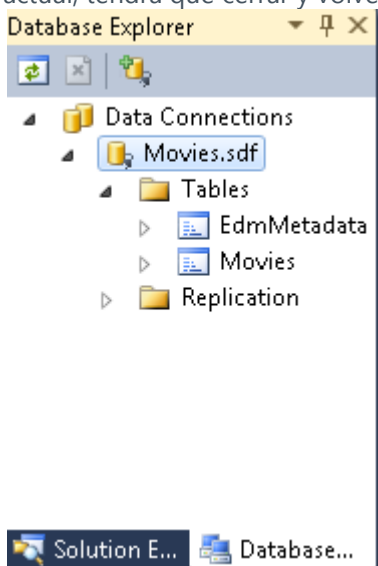
Trabajar con SQL Server Compact

Código de Entity Framework Primera detectado que la cadena de conexión de base de datos que se proporcionó señaló una **Movies** base de datos que aún no existía, por lo que Primer Código creado la base de datos de forma automática. Usted puede verificar que lo ha creado mirando en el *App_Data* carpeta. Si usted no ve el *Movies.sdf* archivo, haga clic en el **Mostrar todos los archivos** en el botón **Explorador de soluciones** barra de herramientas, haga clic en el **Refresh** botón, a continuación, expanda el *App_Data* carpeta.



Haga doble clic en *Movies.sdf* para abrir el **Explorador de servidores**. A continuación, expanda la **Tablas** carpeta para ver las tablas que se han creado en la base de datos.

Nota Si recibe un error cuando se hace doble clic con el botón *Movies.sdf*, asegúrese de haber instalado [SQL Server Compact 4.0](#) (+ tiempo de ejecución de herramientas de apoyo). (Para los enlaces con el software, consulte la lista de requisitos previos en la parte 1 de esta serie de tutoriales.) Si instala la versión actual, tendrá que cerrar y volver a abrir Visual Web Developer.



Hay dos tablas, una para la **película de** conjunto de entidades y luego el **EdmMetadata** mesa. El **EdmMetadata** tabla es utilizada por Entity Framework para determinar cuándo el modelo y la base de datos no están sincronizados.

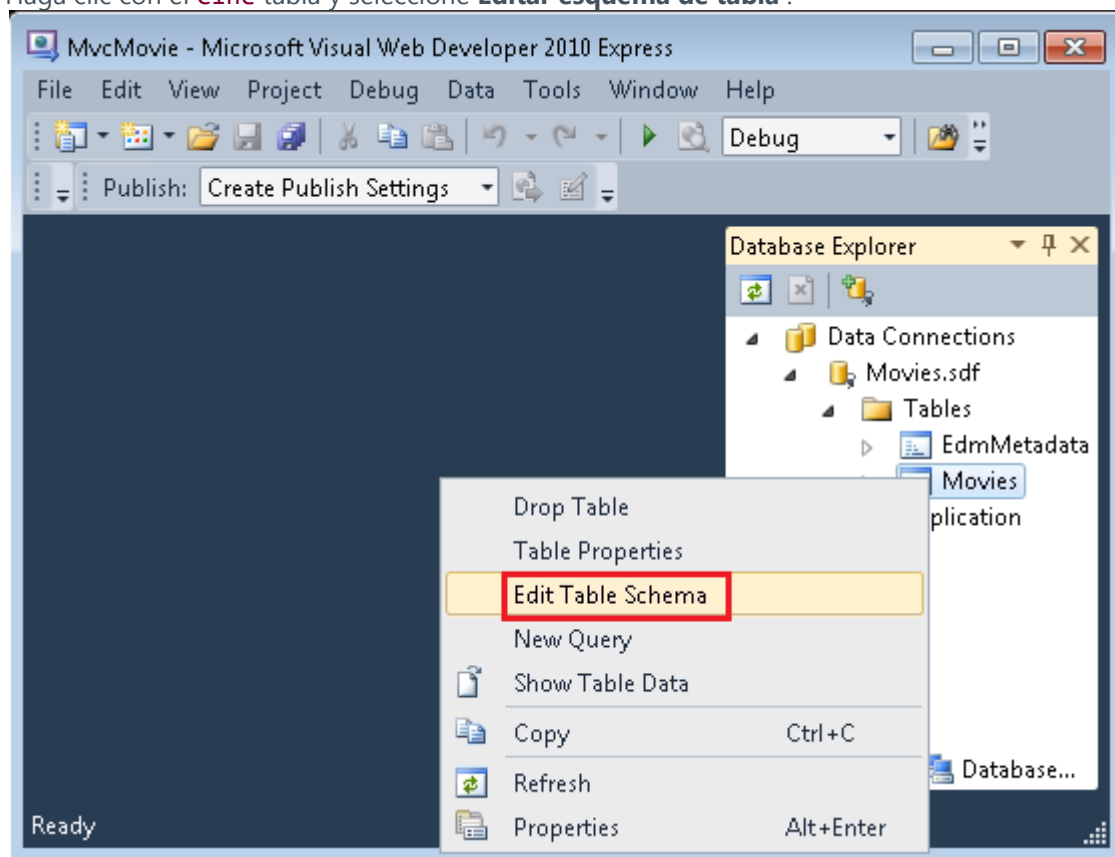
Haga clic con el **Cine** tabla y seleccione **Mostrar datos de tabla** para ver los datos que ha creado.

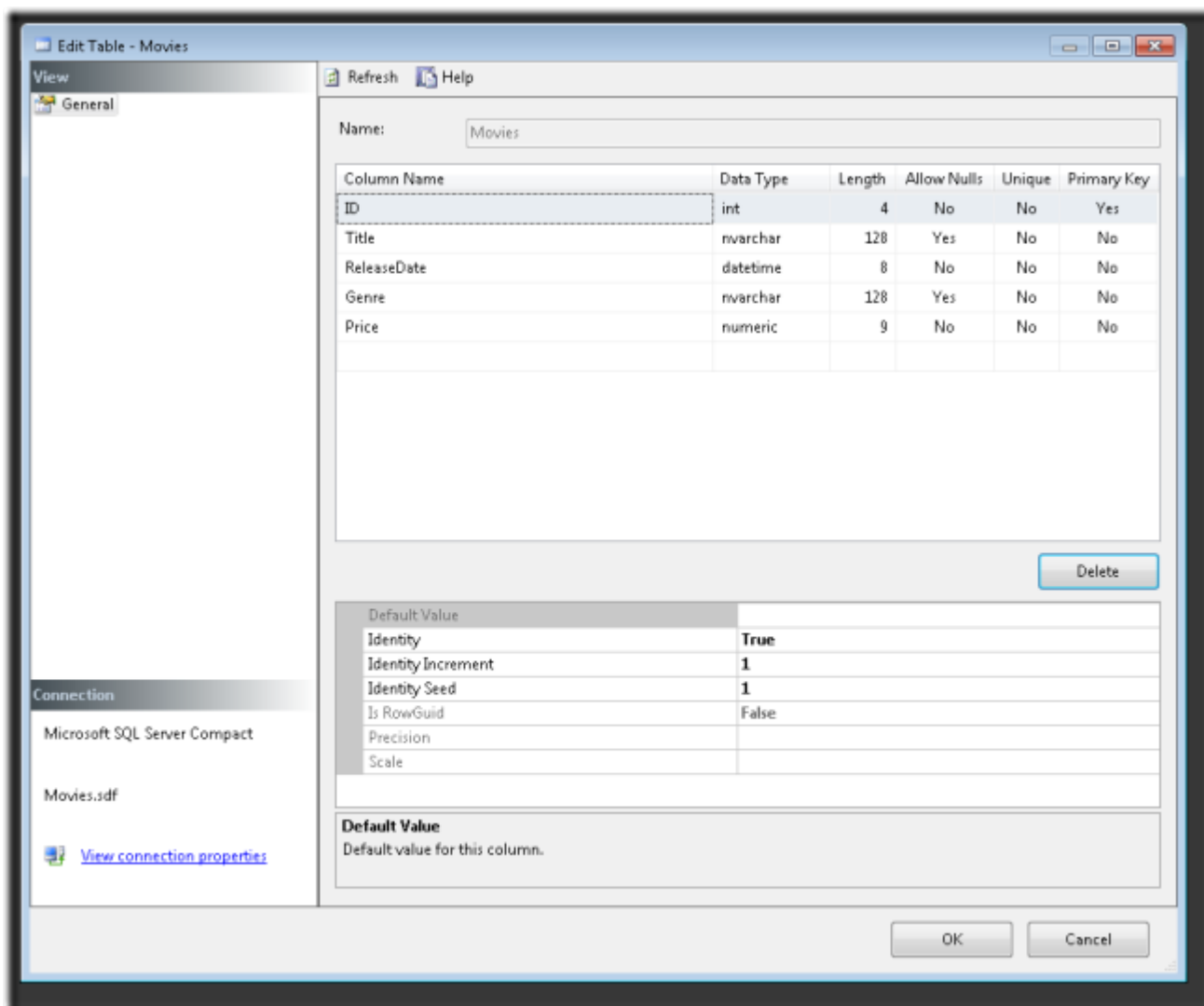
Movies: Query(C:\t...p_Data\Movies.sdf) X

	ID	Title	ReleaseDate	Genre
	1	When Harry Me...	1/11/1989 12:00...	Romantic Com...
	2	Ghostbusters	1/14/1984 12:00...	Comedy
	3	Ghostbusters 2	2/23/1986 12:00...	Comedy
▶	4	Rio Bravo	4/15/1959 12:00...	Western
*	NULL	NULL	NULL	NULL

4 of 4

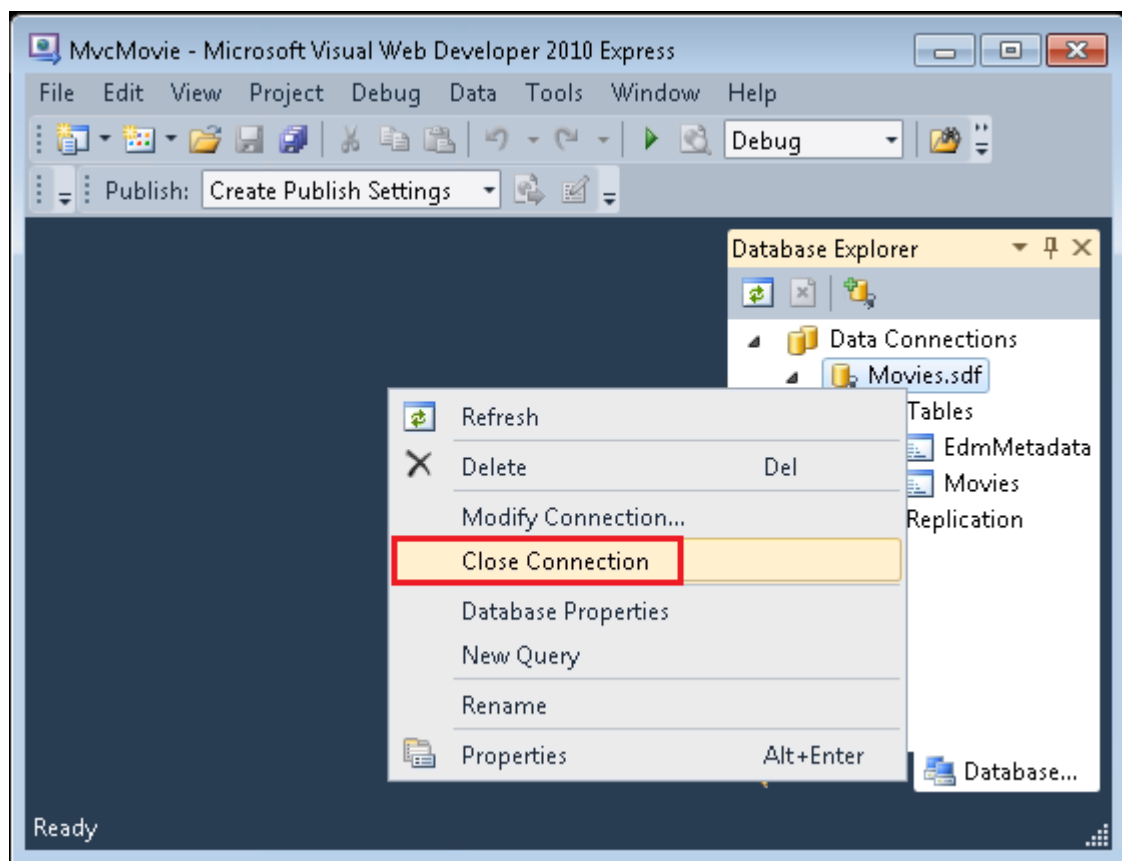
Haga clic con el **Cine** tabla y seleccione **Editar esquema de tabla**.





Observe cómo el esquema de las películas tabla asigna a la película de clase que creó anteriormente. Código de Entity Framework Primero crea automáticamente este esquema para usted basado en su película de clase.

Cuando haya terminado, cierre la conexión. (Si no cierra la conexión, es posible que obtenga un error la próxima vez que ejecute el proyecto).

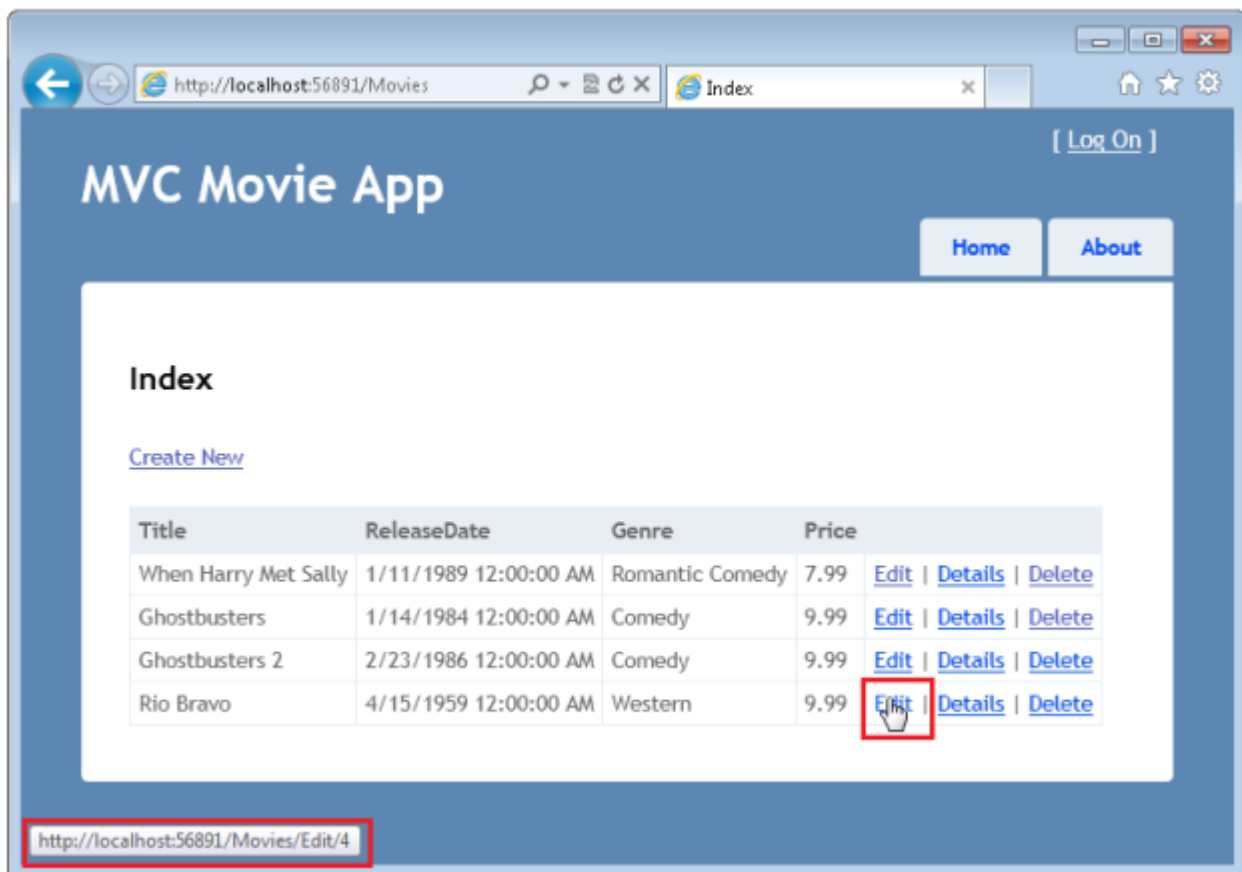


Ahora tiene la base de datos y una página simple enumeración para mostrar el contenido de la misma. En el siguiente tutorial, vamos a examinar el resto del código andamiaje y añadir un **SearchIndex** método y un **SearchIndex** vista que le permite buscar películas en esta base de datos.

Examen de los métodos Edit y Edit View (C #)

En esta sección, examinaremos los métodos de acción generados y puntos de vista para el controlador de vídeo. A continuación, vamos a añadir una página de búsqueda personalizada.

Ejecute la aplicación y busque la **Película** controlador añadiendo */Cine* para la dirección URL en la barra de direcciones de su navegador. Mantenga el puntero del ratón sobre un **Edit** enlace para ver la dirección URL que lo vincula a.



El **Editar** vínculo generado por el `Html.ActionLink` método en la `Views \ Movies \ Index.cshtml` vista:

```
@ Html.ActionLink ("Edit", "Editar", new {id = item.ID})
```

```
<td>
```

```
@Html.ActionLink("Edit Me", "Edit", new { id=item.ID }) |
```

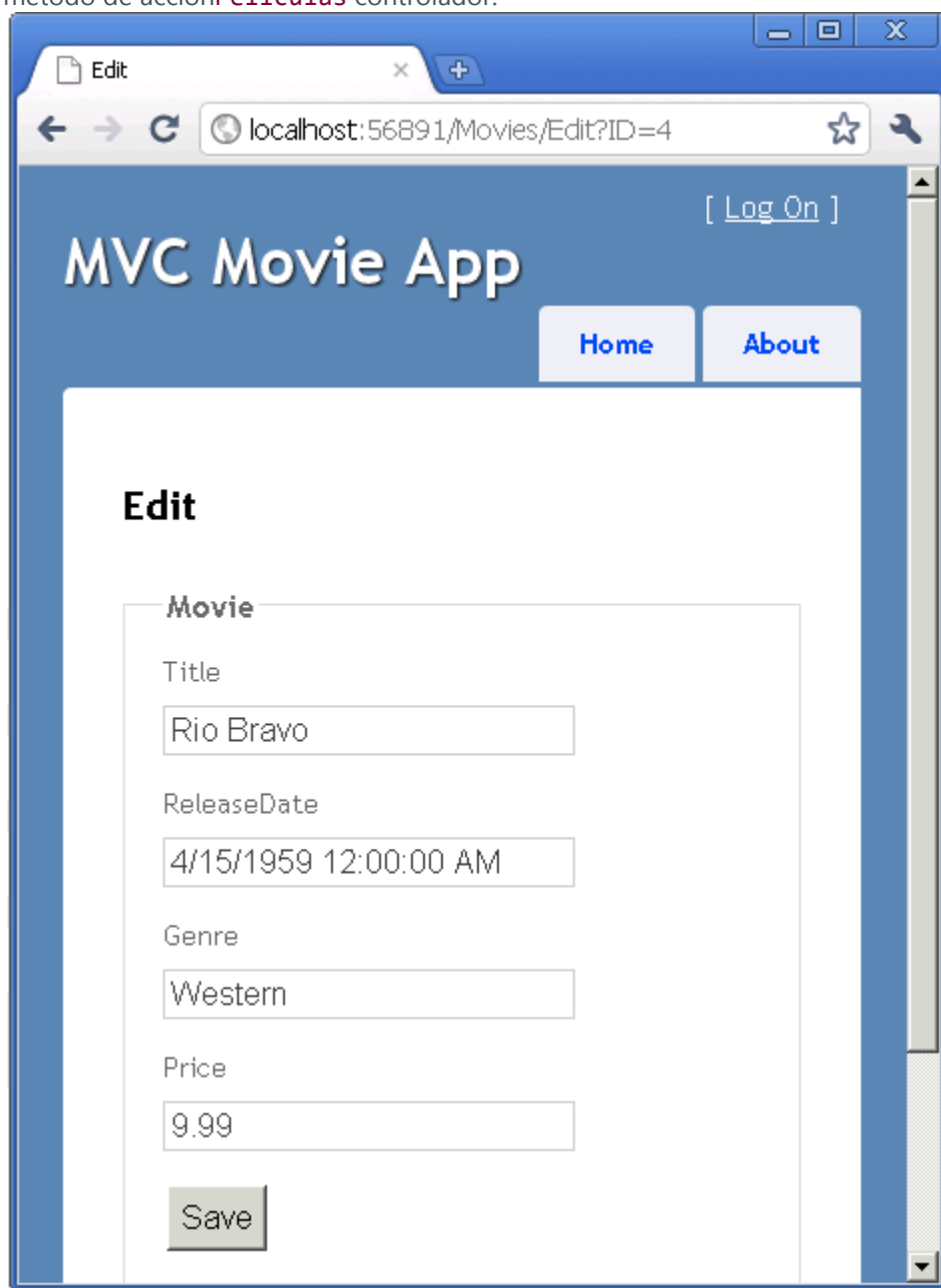
```
@Html. (extension) MvcHtmlString HtmlHelper.ActionLink(string linkText, string actionName, object routeValues)  
@Html. Returns an anchor element (a element) that contains the virtual path of the specified action.
```

Exceptions:
System.ArgumentException

El `Html` objeto es un ayudante que se exponen mediante una propiedad en la `WebViewPage` clase base. El `ActionLink` método del ayudante hace que sea fácil de generar dinámicamente HTML hipervínculos que enlazan con métodos de acción de los controladores. El primer argumento de la `ActionLink` método es el texto del enlace para hacer (por ejemplo, `<a> Me Editar </ a>`). El segundo argumento es el nombre del método de acción a invocar. El último argumento es un **objeto anónimo** que genera los datos de ruta (en este caso, el ID de 4).

El enlace generado se muestra en la imagen anterior es *http://localhost:xxxxx/Movies/Edit/4* . La ruta por defecto toma el patrón de URL `{controller} / {action} / {id}` . Por lo tanto, ASP.NET traduce *http://localhost:xxxxx/Movies/Edit/4* en una solicitud a la **edición** del método de acción **Películas** controlador con el parámetro **ID** igual a 4.

También puede pasar parámetros de acción Utilización de una cadena de consulta. Por ejemplo, la URL *http://localhost:xxxxx/Movies/Edit?ID=4* también pasa el parámetro **ID** de 4 a la **edición** del método de acción **Películas** controlador.



The screenshot shows a web browser window with the title 'Edit'. The address bar displays 'localhost:56891/Movies/Edit?ID=4'. The page features a blue header with the text 'MVC Movie App' and a '[Log On]' link. Below the header, there are two buttons: 'Home' and 'About'. The main content area is titled 'Edit' and contains a form for editing a movie. The form has four input fields: 'Title' with the value 'Rio Bravo', 'ReleaseDate' with the value '4/15/1959 12:00:00 AM', 'Genre' with the value 'Western', and 'Price' with the value '9.99'. A 'Save' button is located at the bottom of the form.

Abra el **Cine** controlador. Los dos **Editar** métodos de acción se muestra a continuación.

```
//  
// GET: /Movies/Edit/5  
  
public ActionResult Edit(int id)  
{  
    Movie movie = db.Movies.Find(id);
```

```

        return View(movie);
    }

    //
    // POST: /Movies/Edit/5

    [HttpPost]
    public ActionResult Edit(Movie movie)
    {
        if (ModelState.IsValid)
        {
            db.Entry(movie).State = EntityState.Modified;
            db.SaveChanges();
            return RedirectToAction("Index");
        }
        return View(movie);
    }

```

Note la segunda **edición** método de acción es precedida por la [HttpPost](#) atributo. Este atributo especifica que esa sobrecarga de la **edición** método puede ser invocado sólo para peticiones POST. Se podría aplicar el [HttpGet](#) atributo al primer método de edición, pero eso no es necesario porque es el valor predeterminado. (Vamos a hacer referencia a métodos de acción que se asigna implícitamente la [HttpGet](#) atributo como [HTTPGet](#) métodos.)

El [HttpGet](#) **Editar** método toma el parámetro ID de cine, mira la película con Entity Framework [Encuentra](#) método y devuelve la película seleccionada para la vista Editar. Cuando el sistema de andamiaje creado la vista Editar, examinó la **Película** clase y creó el código para hacer `<label>` y `<input>` elementos para cada propiedad de la clase. El ejemplo siguiente muestra la vista de edición que se ha generado:

```

@model MvcMovie.Models.Movie

@{
    ViewBag.Title = "Edit";
}

<h2>Edit</h2>

<script src="@Url.Content("~/Scripts/jquery.validate.min.js")"
type="text/javascript"></script>
<script src="@Url.Content("~/Scripts/jquery.validate.unobtrusive.min.js")"
type="text/javascript"></script>

@using (Html.BeginForm()) {
    @Html.ValidationSummary(true)
    <fieldset>
        <legend>Movie</legend>

        @Html.HiddenFor(model => model.ID)

        <div class="editor-label">
            @Html.LabelFor(model => model.Title)
        </div>

```

```

        <div class="editor-field">
            @Html.EditorFor(model => model.Title)
            @Html.ValidationMessageFor(model => model.Title)
        </div>

        <div class="editor-label">
            @Html.LabelFor(model => model.ReleaseDate)
        </div>
        <div class="editor-field">
            @Html.EditorFor(model => model.ReleaseDate)
            @Html.ValidationMessageFor(model => model.ReleaseDate)
        </div>

        <div class="editor-label">
            @Html.LabelFor(model => model.Genre)
        </div>
        <div class="editor-field">
            @Html.EditorFor(model => model.Genre)
            @Html.ValidationMessageFor(model => model.Genre)
        </div>

        <div class="editor-label">
            @Html.LabelFor(model => model.Price)
        </div>
        <div class="editor-field">
            @Html.EditorFor(model => model.Price)
            @Html.ValidationMessageFor(model => model.Price)
        </div>

        <p>
            <input type="submit" value="Save" />
        </p>
    </fieldset>
}

<div>
    @Html.ActionLink("Back to List", "Index")
</div>

```

Observe cómo la plantilla de vista tiene un `modelo MvcMovie.Models.Movie @` declaración en la parte superior del archivo - especifica que la opinión espera que el modelo de la plantilla de la vista como del tipo **de películas**.

El código de andamiaje utiliza varios *métodos de ayuda* para agilizar el formato

HTML. El `Html.LabelFor` ayudante muestra el nombre del campo ("Título", "RELEASEDATE", "Género", o "Precio"). El `Html.EditorFor` muestra un ayudante

HTML `<input>` elemento. El `Html.ValidationMessageFor` auxiliar muestra los mensajes de validación asociados a dicha propiedad.

Ejecute la aplicación y vaya a la `/ Películas` URL. Haga clic en un **Edit** link. En el navegador, ver el código fuente de la página. El HTML de la página es similar al ejemplo siguiente. (El marcado menú fue excluido por razones de claridad.)

```

<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8" />
  <title>Edit</title>
  <link href="/Content/Site.css" rel="stylesheet" type="text/css" />
  <script src="/Scripts/jquery-1.5.1.min.js" type="text/javascript"></script>
  <script src="/Scripts/modernizr-1.7.min.js" type="text/javascript"></script>
</head>
<body>
  <div class="page">
    <header>
      <div id="title">
        <h1>MVC Movie App</h1>
      </div>
      ...
    </header>
    <section id="main">

<h2>Edit</h2>

<script src="/Scripts/jquery.validate.min.js" type="text/javascript"></script>
<script src="/Scripts/jquery.validate.unobtrusive.min.js"
type="text/javascript"></script>

<form action="/Movies/Edit/4" method="post">    <fieldset>
  <legend>Movie</legend>

  <input data-val="true" data-val-number="The field ID must be a number."
        data-val-required="The ID field is required." id="ID" name="ID"
type="hidden" value="4" />

  <div class="editor-label">
    <label for="Title">Title</label>
  </div>
  <div class="editor-field">
    <input class="text-box single-line" id="Title" name="Title" type="text"
value="Rio Bravo" />
    <span class="field-validation-valid" data-valmsg-for="Title" data-valmsg-
replace="true"></span>
  </div>

  <div class="editor-label">
    <label for="ReleaseDate">ReleaseDate</label>
  </div>
  <div class="editor-field">
    <input class="text-box single-line" data-val="true" data-val-required="The
ReleaseDate field is required."
        id="ReleaseDate" name="ReleaseDate" type="text" value="4/15/1959

```

```

12:00:00 AM" />
    <span class="field-validation-valid" data-valmsg-for="ReleaseDate" data-
valmsg-replace="true"></span>
</div>

<div class="editor-label">
    <label for="Genre">Genre</label>
</div>
<div class="editor-field">
    <input class="text-box single-line" id="Genre" name="Genre" type="text"
value="Western" />
    <span class="field-validation-valid" data-valmsg-for="Genre" data-valmsg-
replace="true"></span>
</div>

<div class="editor-label">
    <label for="Price">Price</label>
</div>
<div class="editor-field">
    <input class="text-box single-line" data-val="true" data-val-number="The
field Price must be a number."
        data-val-required="The Price field is required." id="Price" name="Price"
type="text" value="9.99" />
    <span class="field-validation-valid" data-valmsg-for="Price" data-valmsg-
replace="true"></span>
</div>

<p>
    <input type="submit" value="Save" />
</p>
</fieldset>
</form>
<div>
    <a href="/Movies">Back to List</a>
</div>

</section>
<footer>
</footer>
</div>
</body>
</html>

```

El marco ASP.NET modelo aglutinante toma los valores del formulario publicado y crea una **película** objeto que se pasa como la **película** parámetro. El **ModelState.IsValid** de verificación en el código verifica que los datos facilitados en el formulario puede ser utilizado para modificar una **película** objeto. Si los datos son válidos, el código guarda los datos de la película en el **Cine** de la colección **MovieDbContext** ejemplo. Después, el código guarda los datos de la nueva película de la base de datos mediante una llamada al **SaveChanges** método de **MovieDbContext**, que persiste cambios a la base de datos. Después de guardar los datos, el código redirige al usuario a la **Índice** método de acción de la **MoviesController** clase, que hace que la película actualizada que se muestra en la lista de películas.

Si los valores publicados no son válidos, se vuelve a mostrar el formulario. Los `Html.ValidationMessageFor` ayudan en el `Edit.cshtml` plantilla de vista cuidar de mostrar mensajes de error apropiados.

Movie

Title

Ghostbusters

ReleaseDate

abc

The value 'abc' is not valid for ReleaseDate.

Genre

Comedy

Price

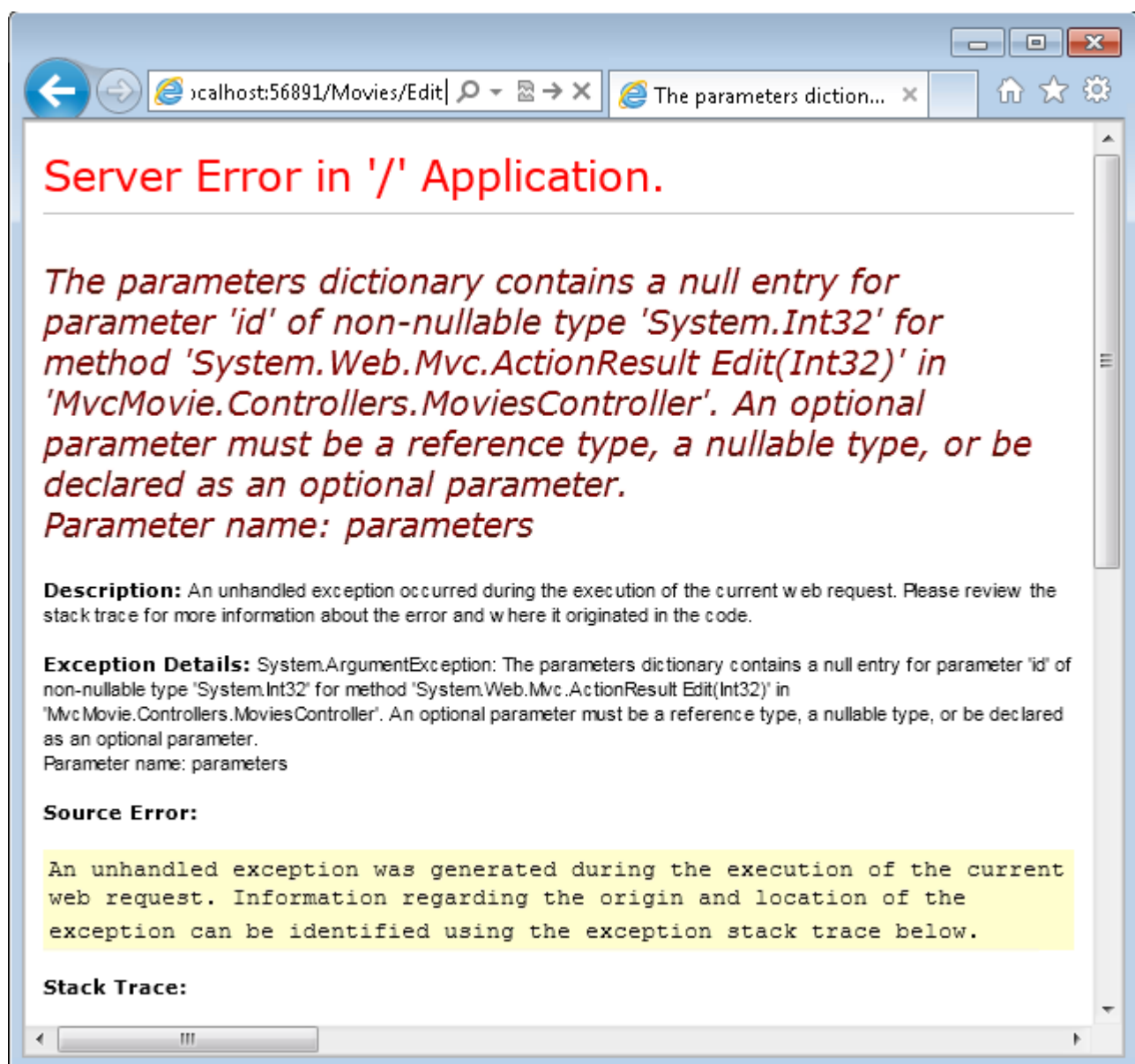
9.99

Save

Nota acerca de lugares Si normalmente trabaja con una configuración regional distinta de Inglés, ver [Apoyar ASP.NET MVC 3 Validación con entornos nacionales no ingleses](#).

Hacer que el método de edición más robusta

El `HttpGet Editar` método generado por el sistema de andamios no comprueba que el identificador que se pasa a su validez. Si un usuario elimina el segmento de ID en el URL (`http://localhost:xxxxx/Movies/Edit`), el error siguiente aparece:



Un usuario también podría pasar un ID que no existe en la base de datos, tales como `http://localhost:xxxxx/Movies/Edit/1234`. Usted puede hacer dos cambios en el `HttpGet Editar` método de acción para hacer frente a esta limitación. En primer lugar, cambiar el **ID** de parámetro tiene un valor predeterminado de cero cuando un ID no esté explícitamente aprobado. También se puede comprobar que la **Búsqueda** método en realidad encontramos una película antes de devolver el objeto de película a la plantilla de vista. La actualización `Edit` método se muestra a continuación.

```
public ActionResult Edit(int id = 0)
{
    Movie movie = db.Movies.Find(id);
    if (movie == null)
    {
        return HttpNotFound();
    }
    return View(movie);
}
```

Si no hay ninguna película se encuentra, el `HttpNotFound` método se llama.

Todos los **HttpGet** métodos siguen un patrón similar. Consiguen un objeto de película (o lista de objetos, en el caso de **Index**), y pasar el modelo a la vista. La **Create** método pasa un objeto de película vacía a la vista **Create**. Todos los métodos que crear, editar, borrar o modificar los datos de hacerlo en el **HttpPost** sobrecarga del método. Modificar datos en un método HTTP GET es un riesgo de seguridad, como se describe en el blog de entrada [ASP.NET MVC Consejo # 46 - No utilice Delete Links porque crean agujeros de seguridad](#). Modificar datos en un método GET también viola las mejores prácticas HTTP y el patrón arquitectónico REST, que especifica que las solicitudes GET no debe cambiar el estado de su solicitud. En otras palabras, la realización de una operación GET debería ser una operación segura que no tiene efectos secundarios.

Agregar un método de búsqueda y de búsqueda Ver

En esta sección vamos a añadir un **SearchIndex** método de acción que te permite buscar películas por género o nombre. Esto estará disponible con los `/Cine/SearchIndex` URL. La solicitud se mostrará un formulario HTML que contiene los elementos de entrada que un usuario puede completar con el fin de buscar una película. Cuando un usuario envía el formulario, el método de acción obtendrá los valores de búsqueda enviados por el usuario y utilizar los valores para buscar en la base de datos.

Viendo el Formulario SearchIndex

Comience agregando un **SearchIndex** método de acción para el actual **MoviesController** clase. El método devolverá una vista que contiene un formulario HTML. Aquí está el código:

```
public ActionResult SearchIndex(string searchString)
{
    var movies = from m in db.Movies
                 select m;

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

    return View(movies);
}
```

La primera línea de la **SearchIndex** método crea el siguiente [LINQ](#) consulta para seleccionar las películas:

```
var películas = de m en db . Películas
                seleccionar m ;
```

La consulta se define en este punto, pero aún no ha sido dirigido contra el almacén de datos.

Si el **SearchString** parámetro contiene una cadena, la consulta películas se modifica para filtrar el valor de la cadena de búsqueda, usando el siguiente código:

```
si (! cuerdas . IsNullOrEmpty ( SearchString ))
{
    cine = películas . Donde ( s => s . Título . Contiene ( SearchString ));
}
```

Consultas LINQ no se ejecutan cuando se definen o cuando son modificados por llamar a un método tal como el **case de** o **OrderBy**. En cambio, la ejecución de la consulta es diferido, lo que significa que la evaluación de una expresión se retrasa hasta que se dio cuenta de su valor es en realidad la iteración o el **ToList** se llama al método. En el **SearchIndex** muestra, se ejecuta la consulta en la vista **SearchIndex**. Para obtener más información acerca de la ejecución de consultas diferido, vea [Ejecución de la consulta](#).

Ahora puede implementar el **SearchIndex** vista que va a mostrar el formulario al usuario. Haga clic dentro del **SearchIndex** método y luego haga clic en **Añadir vista**. En la **Vista Añadir** cuadro de diálogo,

especifique que va a pasar una **película** objeto a la plantilla de la vista como su clase del modelo. En la **plantilla del andamio** lista, elija **Lista** y, a continuación, haga clic en **Agregar**.

The 'Add View' dialog box is shown with the following settings:

- View name: SearchIndex
- View engine: Razor (CSHTML)
- ☒ Create a strongly-typed view
- Model class: Movie (MvcMovie.Models)
- Scaffold template: List
- ☒ Reference script libraries
- ☐ Create as a partial view
- ☒ Use a layout or master page:
- ContentPlaceHolder ID: MainContent

The 'Add' button is highlighted in blue.

Al hacer clic en **Añadir** botón, la *Views \ Movies \ SearchIndex.cshtml* plantilla de vista es creada. Dado que seleccionó **lista** en la **plantilla Andamio** lista, Visual Web Developer genera automáticamente (andamiaje) algún contenido por defecto en la vista. El andamiaje creado un formulario HTML. Examinó la **película** de clase y código creado para hacer **<label>** elementos para cada propiedad de la clase. La lista a continuación muestra la vista de creación que se ha generado:

```
@ Modelo IEnumerable <MvcMovie . Modelos . Película >
```

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

```
<h2> SearchIndex </ h2>
```

```
<p>  
    @ Html.ActionLink ("Crear nuevo",
```

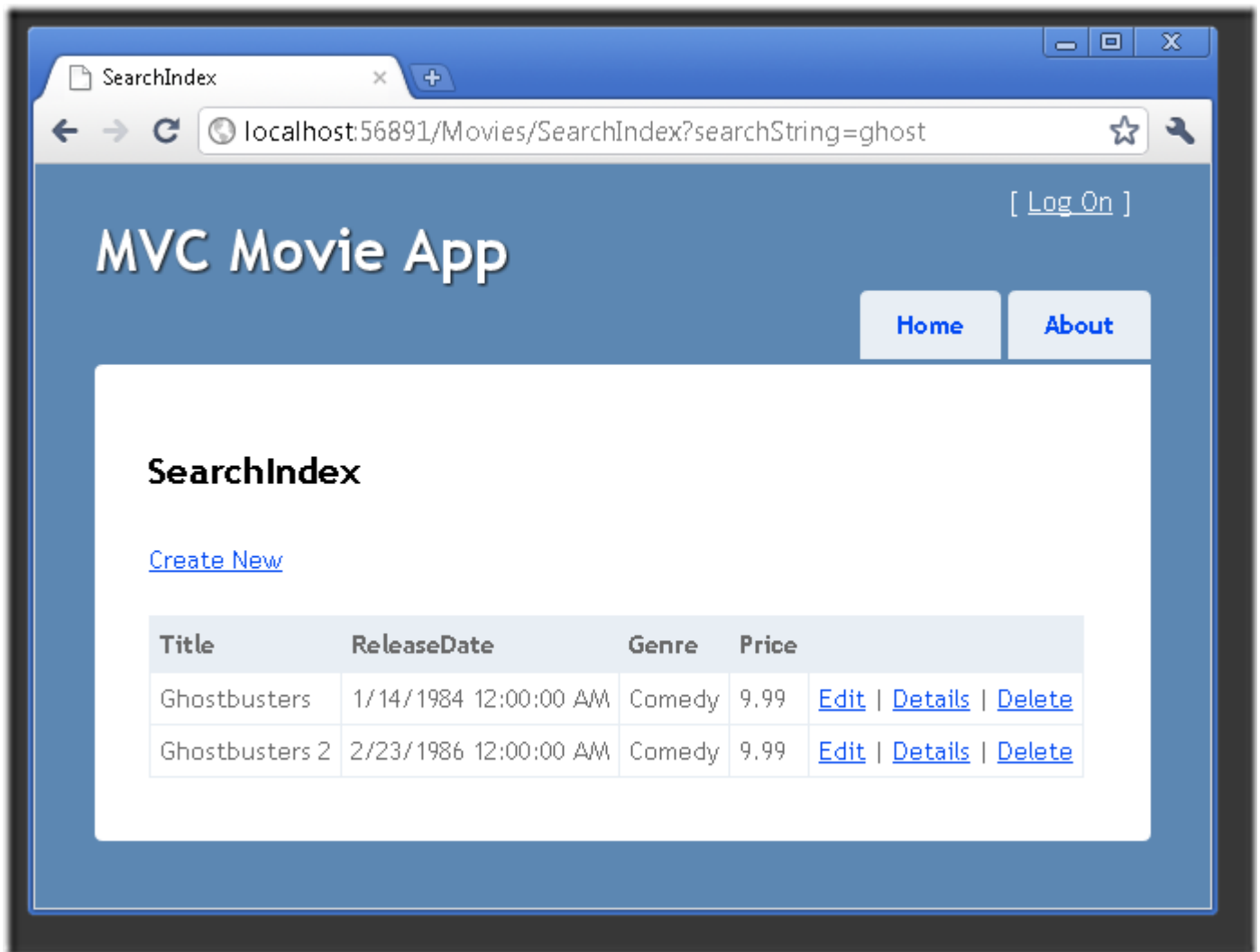
```

(Punto en el Modelo var) {
    <tr>
        <td>
            @ Html.DisplayFor (ModelItem => item.Title)
        </ td>
        <td>
            @ Html.DisplayFor (ModelItem => item.ReleaseDate)
        </ td>
        <td >
            @ Html.DisplayFor (ModelItem => item.Genre)
        </ td>
        <td>
            @ Html.DisplayFor (ModelItem => item.Price)
        </ td>
        <td>
            @ Html.ActionLink ("Edit", "Editar ", el nuevo id = {} item.ID) |
            @ Html.ActionLink ("Detalles" y "Detalles", nuevo {id = item.ID}) |
            @ Html.ActionLink ("Borrar", "Borrar", los nuevos { id = item.ID})
        </ td>
    </ tr>
}

</ table>

```

Ejecute la aplicación y vaya a */ Cine / SearchIndex* . Anexar una cadena de consulta como **? SearchString fantasma =** a la URL. Las películas filtrados se muestran.



Si cambia la firma del **SearchIndex** método para tener un parámetro llamado **id**, el **id** coincide con el parámetro **{id}** marcador de posición para las rutas por defecto establecidos en el *Global.asax* archivo.

```
{ controlador } / { acción } / { id }
```

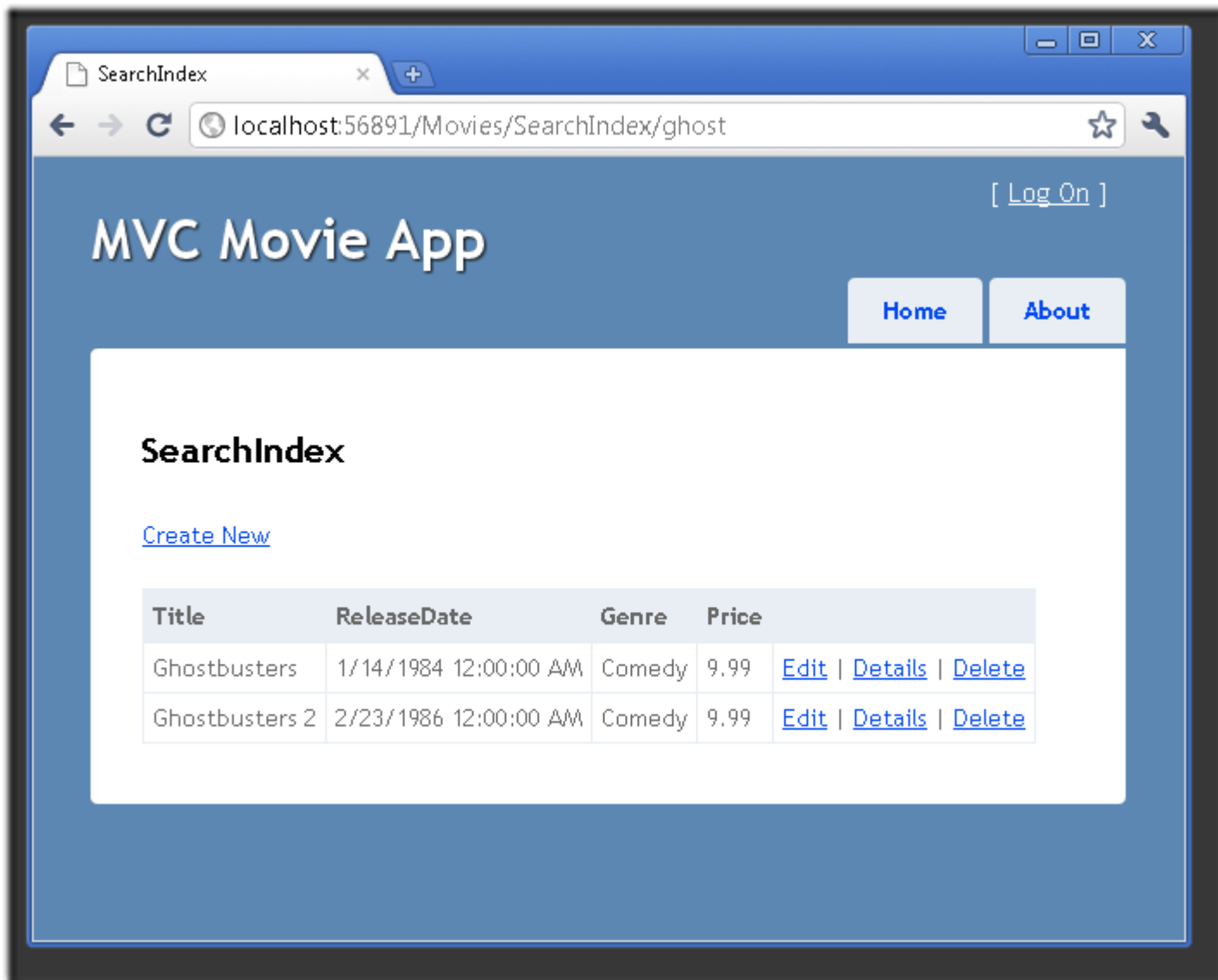
La modificación **SearchIndex** método tendría el siguiente aspecto:

```
public ActionResult SearchIndex(string id)
{
    string searchString = id;
    var movies = from m in db.Movies
                 select m;

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

    return View(movies);
}
```

Ahora puede pasar el título de búsqueda como datos de la ruta (un segmento URL) en lugar de como un valor de cadena de consulta.



Sin embargo, no se puede esperar que los usuarios modifiquen la dirección URL cada vez que quieren buscar una película. Así que ahora vamos a añadir la interfaz de usuario para ayudarles a películas de filtro. Si ha cambiado la firma del **SearchIndex** método para probar cómo pasar el parámetro ID ruta de ruedas, volver a cambiarlo para que su **SearchIndex** método toma un parámetro de cadena denominada **SearchString** :

```
public ActionResult SearchIndex(string searchString)
{
    var movies = from m in db.Movies
                  select m;

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

    return View(movies);
}
```

Abra las *Reproducciones \ Movies \ SearchIndex.cshtml* archivo, y justo después de `@Html.ActionLink ("Crear nuevo", "Crear")`, añada lo siguiente:

```
@ Usando (Html.BeginForm ()) {
    <p> Título: @ Html.TextBox ("SearchString")
    <input tipo = "submit" valor = "Filtro" /> </ p>
}
```

El siguiente ejemplo muestra una parte de las *Vistas \ Movies \ SearchIndex.cshtml* archivo con el formato agregó filtrado.

```
@ Modelo IEnumerable <MvcMovie . Modelos . Película >

@ {
    ViewBag.Title = "SearchIndex";
}

<h2> SearchIndex </ h2>

<p>
    @ Html.ActionLink ("Crear nuevo", "Crear")      @ usando (Html . BeginForm ()) {

        <p> Título: @ Html.TextBox ("SearchString") <br />
        <input tipo = "submit" valor = "Filtro" /> </ p>
    }
</ p>
```

El **Html.BeginForm** ayudante crea una apertura **<form>** etiqueta. El **Html.BeginForm** ayudante hace que el formulario para enviar a sí mismo cuando el usuario envía el formulario haciendo clic en el **filtro** botón. Ejecute la aplicación y tratar de buscar una película.

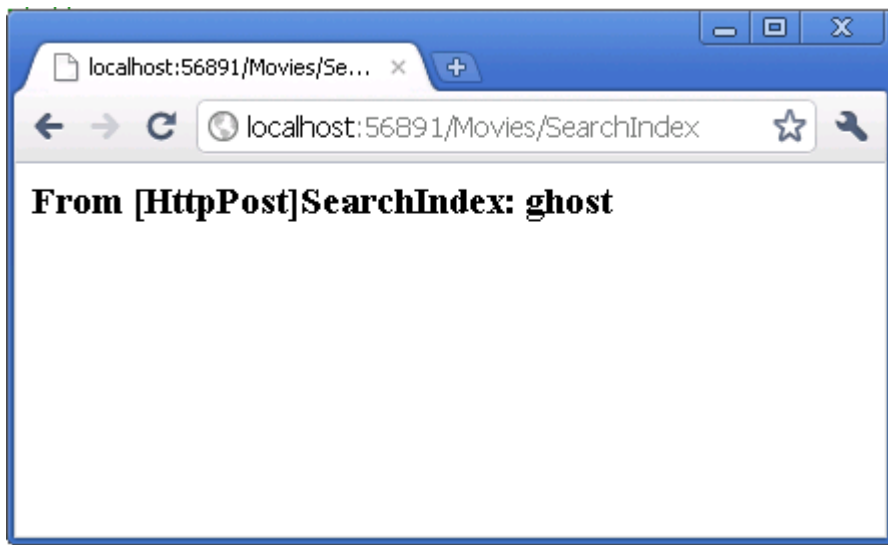
[Create New](#)

Filter

Title	ReleaseDate	Genre	Price	
Ghostbusters	1/14/1984 12:00:00 AM	Comedy	9.99	Edit Details Delete
Ghostbusters 2	2/23/1986 12:00:00 AM	Comedy	9.99	Edit Details Delete

Se podría añadir lo siguiente `HttpPost SearchIndex` método. En ese caso, el invocador de acción se correspondería con el `HttpPost SearchIndex` método, y el `HttpPost SearchIndex` método sería como se muestra en la imagen a continuación.

```
[ HttpPost ]
público cuerda SearchIndex ( FormCollection fc , cadena SearchString )
{
    volver "<h3> Desde [HttpPost] SearchIndex:" + SearchString + "</ h3>" ;
}
```

Sin embargo, incluso si se agrega este **HttpPost** versión del **SearchIndex** método, hay una limitación en la forma en que todo esto ha sido implementado. Imagínese que usted desea marcar una búsqueda particular o si desea enviar un enlace a tus amigos que pueden hacer clic para ver la misma lista filtrada de películas. Observe que la dirección URL de la solicitud HTTP POST es la misma que la dirección URL de la solicitud GET (localhost: xxxx / Películas / SearchIndex) - no hay información de búsqueda en la propia URL. En este momento, la información de la cadena de búsqueda se envía al servidor como un valor de campo de formulario. Esto significa que no puede capturar la información de búsqueda para marcar o enviar a tus amigos en una URL.

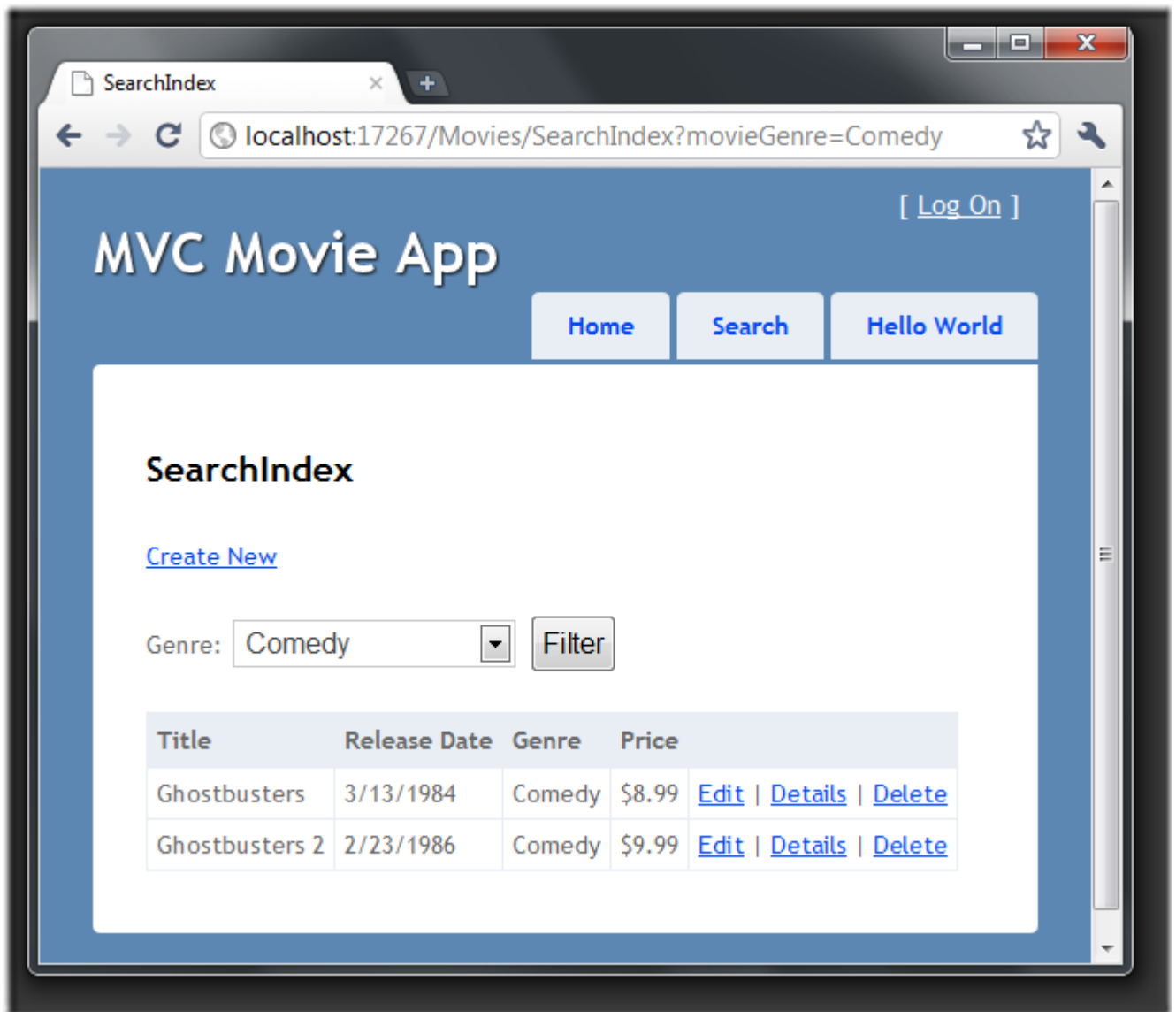
La solución es utilizar una sobrecarga del **BeginForm** que especifica que la solicitud POST debe añadir la información de búsqueda para la URL y que se deben colocar a la versión de la **HttpGet SearchIndex** método. Vuelva a colocar la existente sin parámetros **BeginForm** método con la siguiente:

```
@ Usando (Html.BeginForm ("SearchIndex", "Películas", FormMethod.Get))
```

```
@Html.ActionLink("Create New", "Create")
@using (Html.BeginForm("SearchIndex", "Movies", FormMethod.Get))
{
    <p>
    @*
    <in
    </in>
}
```

▲ 5 of 13 ▼ (extension) MvcForm HtmlHelper.BeginForm(string actionName, string controllerName, FormMethod method)
Writes an opening <form> tag to the response. When the user submits the form, the request will be processed by an action method.
method: The HTTP method for processing the form, either GET or POST.

Ahora, cuando usted envía una búsqueda, la URL contiene una cadena de consulta de búsqueda. Buscando también irá a la **HttpGet SearchIndex** método de acción, incluso si usted tiene un **HttpPost SearchIndex** método.



Adición Buscar por Género

Si ha agregado el **HttpPost** versión del **SearchIndex** método, borrarlos ahora.

A continuación, vamos a añadir una característica que permite a los usuarios buscar películas por género. Vuelva a colocar la **SearchIndex** método con el siguiente código:

```
public ActionResult SearchIndex(string movieGenre, string searchString)
{
    var GenreLst = new List<string>();

    var GenreQry = from d in db.Movies
                   orderby d.Genre
                   select d.Genre;
    GenreLst.AddRange(GenreQry.Distinct());
    ViewBag.movieGenre = new SelectList(GenreLst);

    var movies = from m in db.Movies
                 select m;

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

```

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

if (string.IsNullOrEmpty(movieGenre))
    return View(movies);
else
{
    return View(movies.Where(x => x.Genre == movieGenre));
}
}

```

Esta versión de la **SearchIndex** método toma un parámetro adicional, a saber **movieGenre** . Las primeras líneas de código crean una **lista** objeto de celebrar géneros cinematográficos desde la base de datos. El código siguiente es una consulta LINQ que recupera todos los géneros, desde la base de datos.

```

var GenreQry = de d en db . Películas
                orderby d . Género
                seleccione d . Género ;

```

El código utiliza la **AddRange** método genérico de la **lista** colección para agregar todos los géneros distintos a la lista. (Sin el **Distinct** modificador, los géneros duplicados se sumaría - por ejemplo, la comedia se añadirían dos veces en nuestra muestra). El código a continuación, almacena la lista de géneros en la **ViewBag** objeto.

El código siguiente muestra cómo comprobar el **movieGenre** parámetro. Si no está vacío el código limita aún más las películas de consulta para limitar las películas seleccionadas para el género especificado.

```

if (string.IsNullOrEmpty(movieGenre))
    return View(movies);
else
{
    return View(movies.Where(x => x.Genre == movieGenre));
}

```

Adición de marcado a la vista SearchIndex de Apoyo a la Búsqueda por Género

Añadir un **Html.DropDownList** ayudante para las *Visitas \ Movies \ SearchIndex.cshtml* archivos, justo antes del **TextBox** ayudante. El marcado completado se muestra a continuación:

```

<p>
    @ Html.ActionLink ("Crear nuevo", "Crear")
    @ usando (Html.BeginForm ()) {
        <p> Género: @ Html.DropDownList ("movieGenre", "All")
        Título: @ Html. TextBox ("SearchString")
        <input tipo = "submit" valor = "Filtro" /> </ p>
    }
</ p>

```

Ejecute la aplicación y vaya a */ Cine / SearchIndex* . Intente una búsqueda por género, por nombre de la película, y por ambos criterios.

En esta sección examina los métodos de acción CRUD y opiniones generadas por el marco. Ha creado un método de acción de búsqueda y opina que permitirá buscar usuarios por título de la película y el género. En

la siguiente sección, veremos cómo añadir un alojamiento a la **película de** modelo y cómo agregar un inicializador que automáticamente creará una base de datos de prueba.