	UNIVERSIDAD DON BOSCO **FACULTAD DE INGENIERIA** **ESCUELA DE COMPUTACIÓN**	
Ciclo II		**PRACTICA DE LABORATORIO No. 1**
	Nombre de la practica:	"Introducción a ASP.NET MVC 5 Parte 1"
	Lugar de ejecución	**Centro de cómputo**
	Tiempo estimado:	**2 horas**
	Materia	**Desarrollo de software empresarial**

Objetivos

- Que el estudiante aprenda el uso del IDE Visual Studio para el diseño de aplicaciones web.
- Que el estudiante obtenga conocimientos básicos sobre la utilización del patrón MVC.
- Adquirir conocimientos y habilidades en la construcción de páginas web con ASP.NET MVC 5.

Descripción

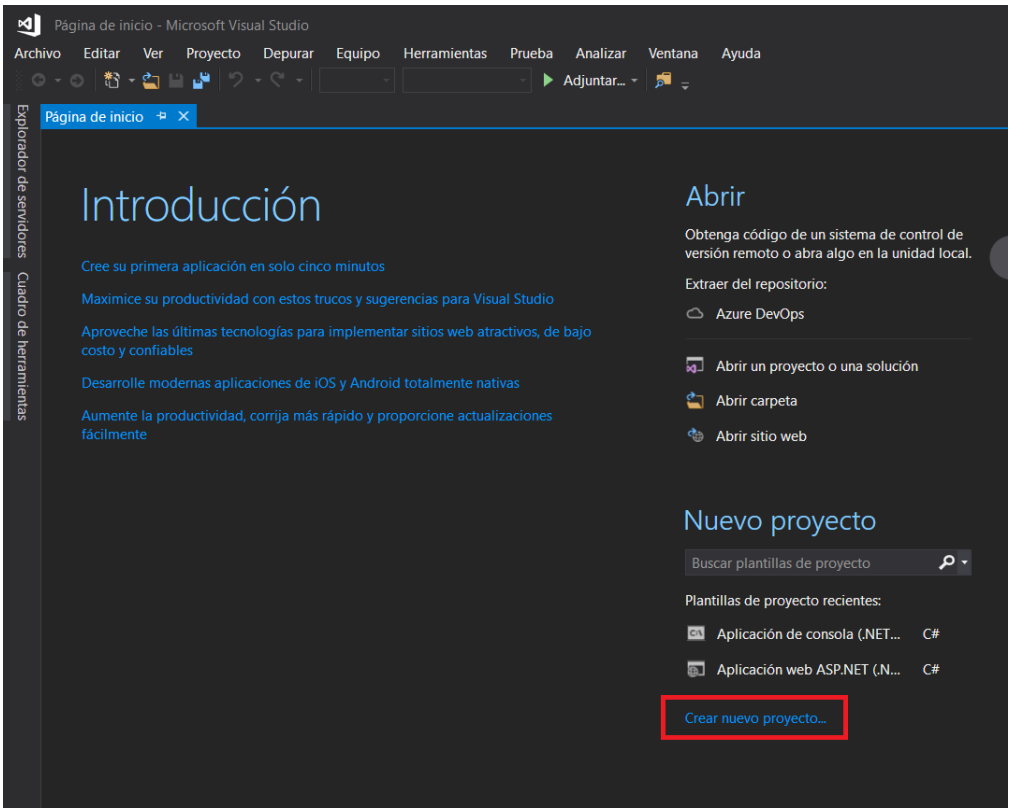
El Laboratorio está dividido en cuatro partes. La primera parte veremos cómo crear un proyecto ASP.NET MVC con Visual Studio 2017, en la segunda parte aprenderemos a crear controladores que son clases que manejan las solicitudes entrantes del navegador, recuperan datos del modelo y luego especificar plantillas de vista que devuelvan una respuesta al navegador., en la tercera parte veremos cómo crear vistas que son archivos de plantilla que nuestra aplicación utiliza para generar dinámicamente respuestas HTML, y en la cuarta parte crearemos clases de Modelo, clases que representan los datos de la aplicación y que utilizan la lógica de validación para hacer cumplir las reglas de negocio para esos datos. Durante el laboratorio se harán una serie de ejercicios para desarrollar las habilidades y destrezas del alumno en la construcción de páginas web en servidor usando la tecnología ASP.NET MVC 5, empleando el lenguaje de programación C#.

Materiales y Recursos Didácticos

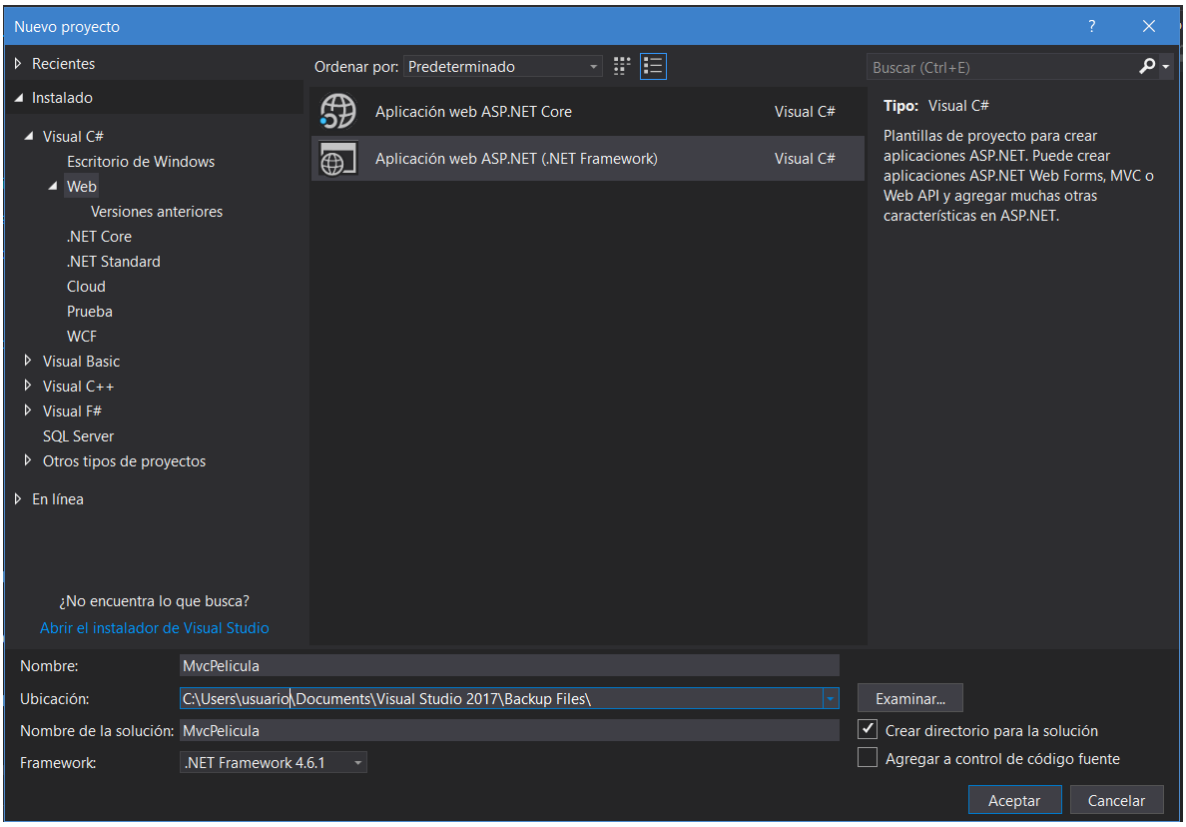
- ➢ Guía de Laboratorio.
- ➢ PC con internet y Navegador Web.
- ➢ PC con IDE Visual Studio 2017 o superior.

I. Creación del proyecto ASP.NET MVC.

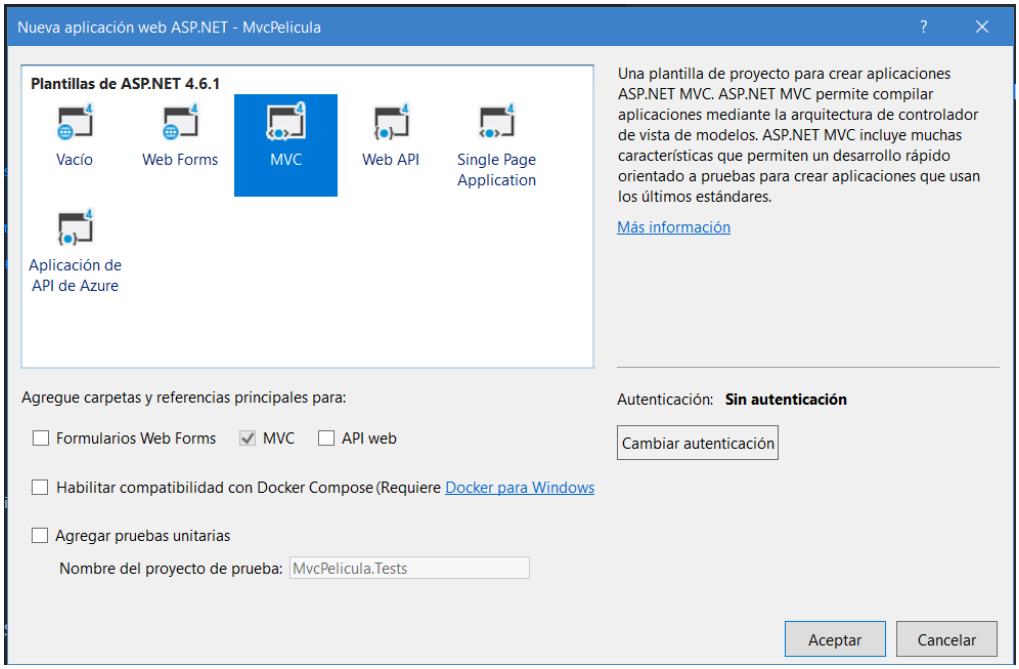
Visual Studio es un IDE (Entorno de Desarrollo Integrado). Al igual que se usa Microsoft Word para escribir documentos, usaremos un IDE para crear aplicaciones. En Visual Studio, hay una lista en la parte inferior que muestra varias opciones disponibles. También hay un menú que proporciona otra forma de realizar tareas en el IDE. Por ejemplo, en lugar de seleccionar Nuevo proyecto en la página de Inicio, puede usar la barra de menú y seleccionar Archivo > Nuevo proyecto.



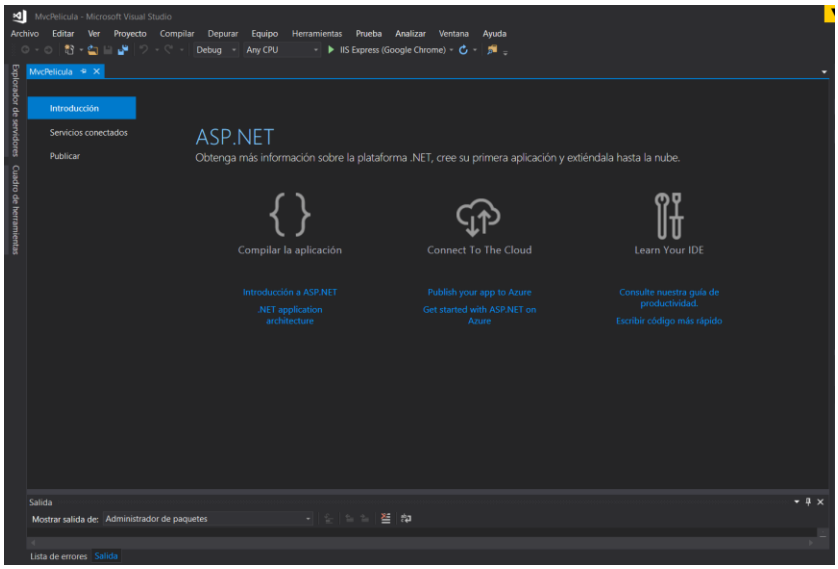
En la página de inicio, seleccione Nuevo proyecto. En el cuadro de diálogo Nuevo proyecto, seleccione la categoría Visual C # a la izquierda, luego Web y luego seleccione la plantilla de proyecto de la Aplicación Web ASP .NET (.NET Framework). Nombre su proyecto "MvcPelicula" y luego elija **Aceptar**.



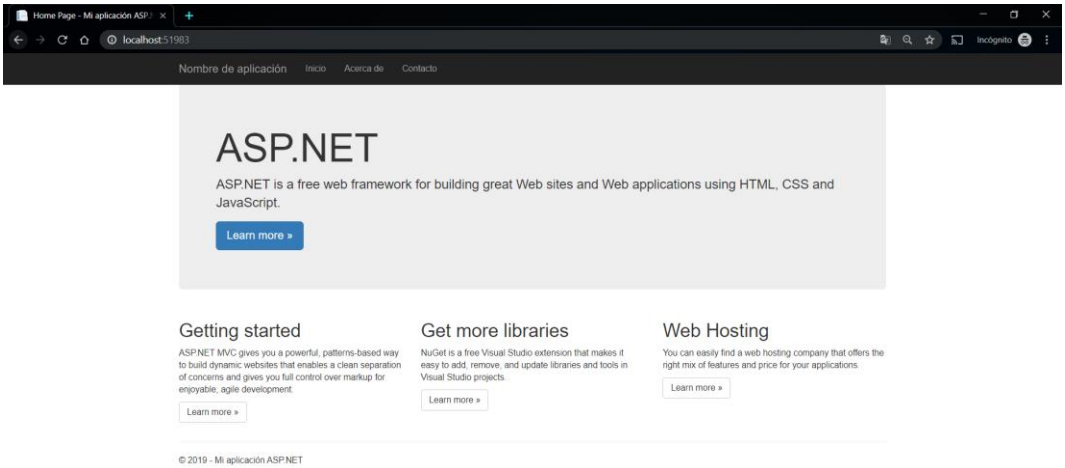
En el cuadro de diálogo **Nueva aplicación web ASP.NET**, elija **MVC** y luego elija **Aceptar**.



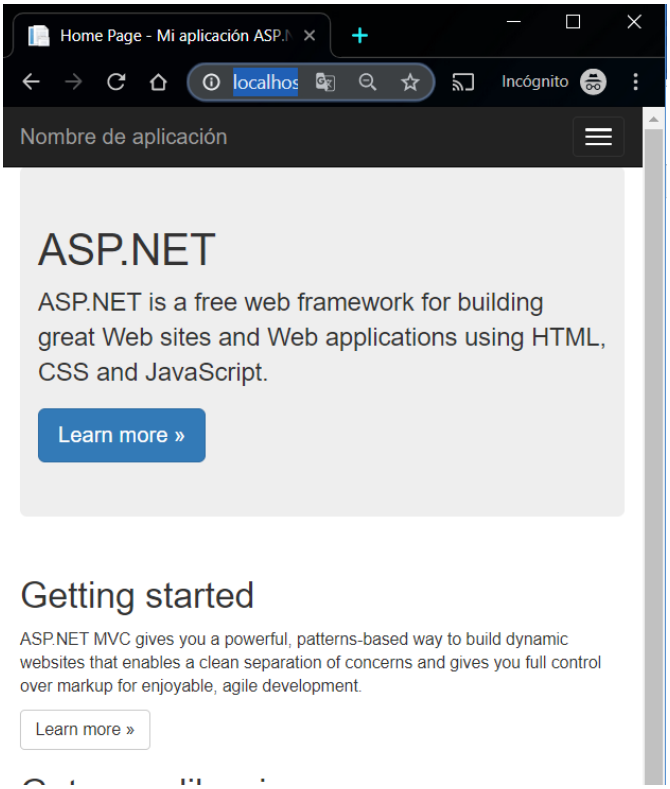
Visual Studio usó una plantilla predeterminada para el proyecto ASP.NET MVC que acaba de crear, por lo que ahora tiene una aplicación que funciona sin hacer nada. Este es un simple proyecto “¡Hello World!", y es un buen lugar para comenzar su aplicación.



Presione F5 para iniciar la depuración. Cuando presiona F5, Visual Studio inicia IIS Express (IIS Express es una versión reducida del servidor de aplicaciones web de Microsoft, Internet Information Server. Nos proporciona una experiencia mucho más real a la hora de probar las aplicaciones, frente al habitual "Cassini" (o Web Development Server, como se le llama oficialmente) que viene con Visual Studio) y ejecuta su aplicación web. Visual Studio inicia un navegador y abre la página de inicio de la aplicación. Tenga en cuenta que la barra de direcciones del navegador dice localhost:port# y no algo como example.com. Eso es porque localhost es el que el servidor utiliza para su computadora local, que en este caso está ejecutando la aplicación que acaba de crear. Cuando Visual Studio ejecuta un proyecto web, se utiliza un puerto aleatorio para el servidor web. En la imagen a continuación, el número de puerto es 51983. Cuando ejecute la aplicación, verá un número de puerto diferente.



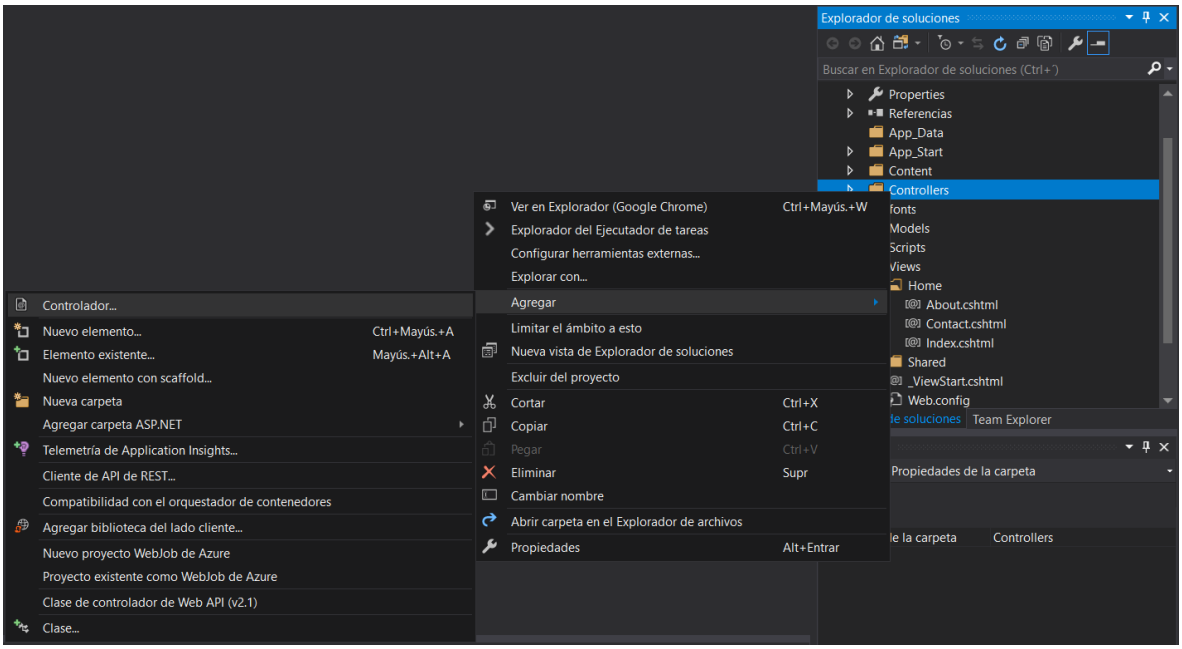
Nada más crear el proyecto tendrá esta plantilla por defecto con las páginas: Inicio, Acerca de, Contacto. La imagen a continuación no muestra los enlaces Inicio, Acerca de y Contacto. Dependiendo del tamaño de la ventana de su navegador, es posible que deba hacer clic en el icono de navegación para ver estos enlaces.



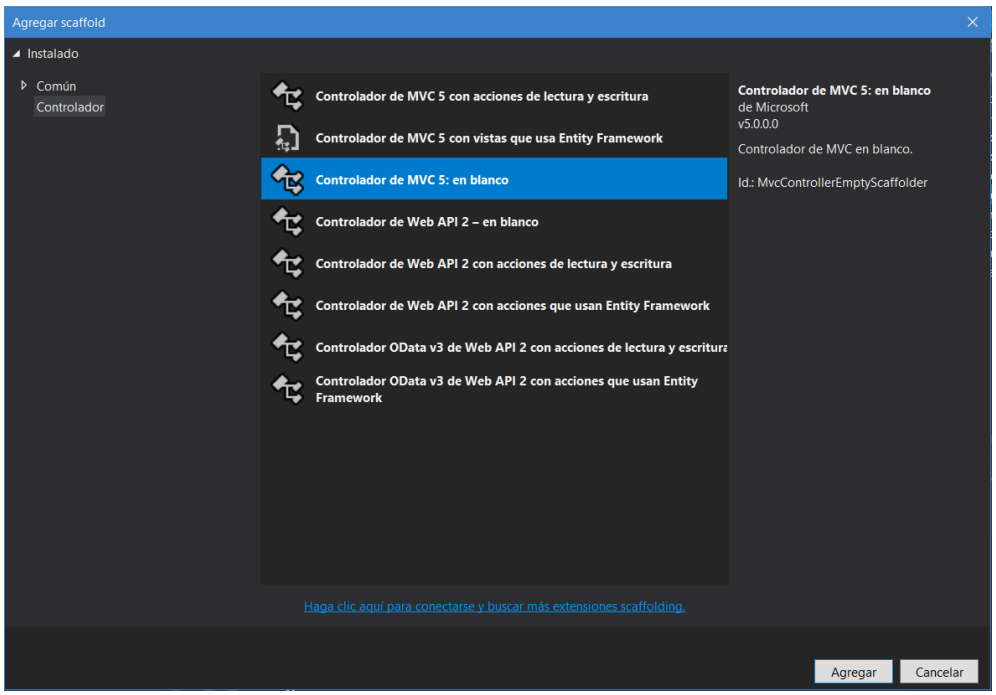
El siguiente paso es cambiar el funcionamiento de esta aplicación y aprender un poco sobre ASP.NET MVC. Cierre la aplicación ASP.NET MVC y cambiemos algo de código.

II. AGREGAR UN CONTROLADOR.

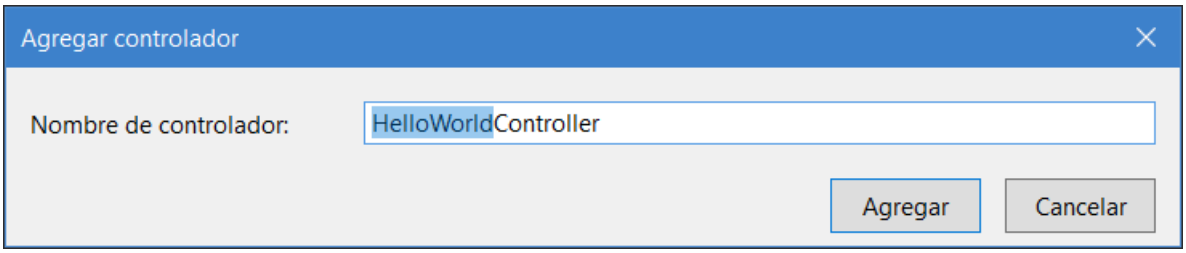
Comencemos creando una clase de controlador. En el Explorador de soluciones, haga clic con el botón derecho en la carpeta Controllers y luego haga clic en Agregar, luego en Controlador.



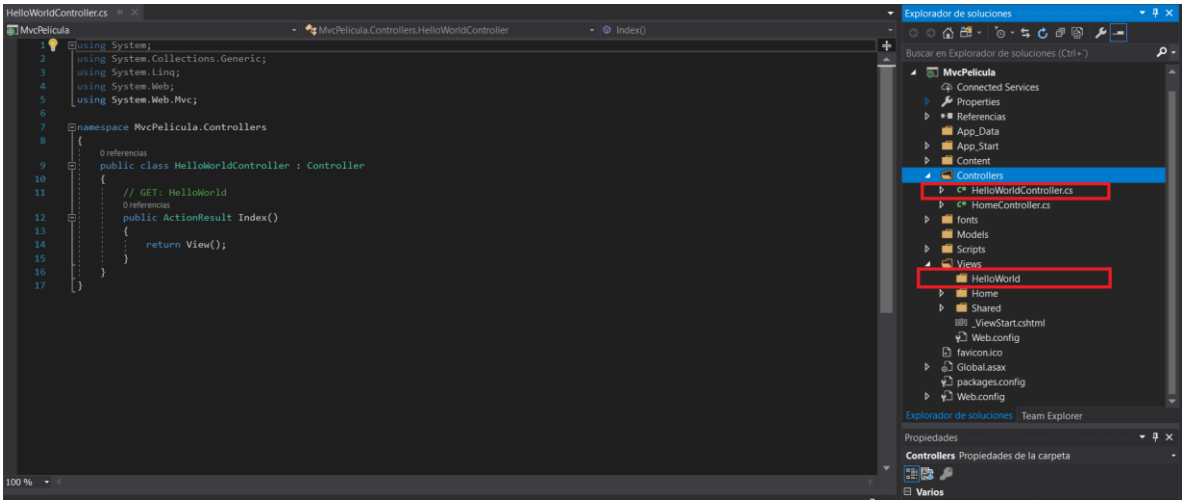
En el cuadro de diálogo **Agregar scaffold**, haga clic en Controlador MVC 5: en blanco, y luego haga clic en Agregar.



Nombre su nuevo controlador "HelloWorldController" y haga clic en **Agregar**.



Observe en el Explorador de soluciones que se ha creado un nuevo archivo llamado HelloWorldController.cs y una nueva carpeta Views\HelloWorld.

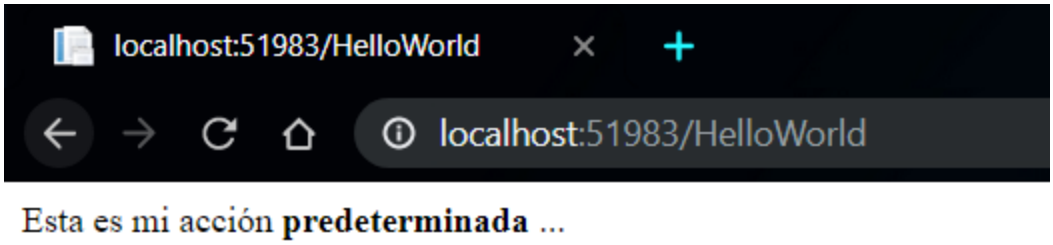


Reemplace el contenido del controlador con el siguiente código.

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

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

        public string Index()
        {
            return "Esta es mi acción <b> predeterminada </b> ...";
        }

        //
        // GET: /HelloWorld/Welcome/

        public string Welcome ()
```

```
    {
        return "Este es el método de acción Bienvenido ...";
    }
  }
}
```

Los métodos del controlador devolverán una cadena de HTML como ejemplo. Se nombra el controlador HelloWorldController y se nombra el primer método Index. Invoquémoslo desde un navegador. Ejecute la aplicación (presione F5 o Ctrl + F5). En el navegador, agregue "HelloWorld" a la ruta en la barra de direcciones. (Por ejemplo, en la siguiente ilustración, es http://localhost:51983/HelloWorld.) La página en el navegador se verá como la siguiente captura de pantalla. En el método anterior, el código devolvió una cadena directamente. Le dijimos al sistema que solo devolviera algo de HTML, ¡y lo hizo!



ASP.NET MVC invoca diferentes clases de controlador (y diferentes métodos de acción dentro de ellos) dependiendo de la URL entrante. La lógica de enrutamiento de URL predeterminada utilizada por ASP.NET MVC, utiliza un formato como este, para determinar qué código invocar:

/[Controller]/[ActionName]/[Parameters]

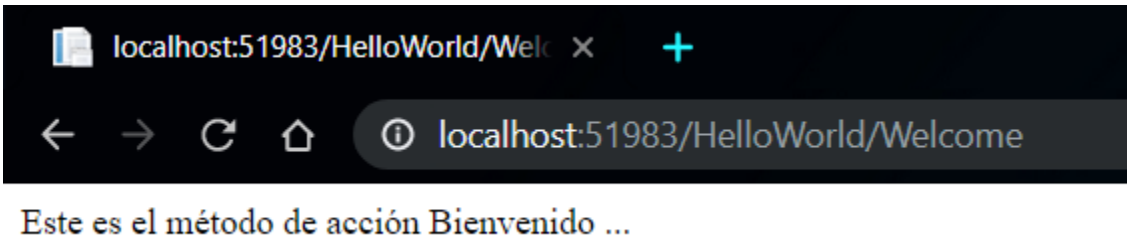
Estableciendo el formato para el enrutamiento en el archivo App_Start/RouteConfig.cs.

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

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

Cuando se ejecuta la aplicación y no se proporciona ningún segmento de URL, el controlador predeterminado es "Inicio" y el método de acción "Index" especificado en la sección de valores predeterminados del código anterior.

La primera parte de la URL, determina la clase de controlador a ejecutar. Entonces /HelloWorld se asigna a la clase HelloWorldController. La segunda parte de la URL determina el método de acción en la clase a ejecutar. Entonces /HelloWorld/Index haría que el método Index de la clase HelloWorldController se ejecute. Tenga en cuenta que solo tuvimos que buscar /HelloWorld y el método Index se usó de forma predeterminada. Esto se debe a que un método llamado Index es el método predeterminado que se invocará en un controlador si no se especifica uno explícitamente. La tercera parte del segmento URL (Parameters) es para datos de ruta. Veremos los datos de la ruta más adelante en esta guía.

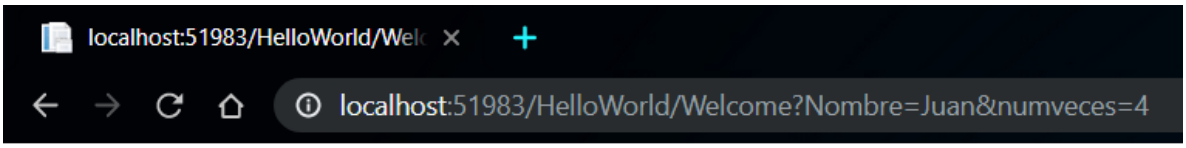
Vaya a http://localhost:xxxx/HelloWorld/Welcome. El método Welcome se ejecuta y devuelve la cadena "Este es el método de acción Bienvenido ...". La asignación de MVC predeterminada es /[Controller]/[ActionName]/[Parameters]. Para esta URL, el controlador es HelloWorld y Welcome es el método de acción. Aún no se ha utilizado la parte de la URL para [Parameters].



Modifiquemos ligeramente el ejemplo para que pueda pasar información de parámetros desde la URL al controlador (por ejemplo, /HelloWorld/Welcome?nombre=Juan&numVeces=4). Cambie su método Welcome para incluir dos parámetros como se muestra a continuación. Tenga en cuenta que el código usa la función de parámetro opcional C# para indicar que el parámetro numVeces debe tener un valor predeterminado de 1, si no se pasa ningún valor para ese parámetro.

```
public string Welcome(string nombre, int numVeces = 1)
    {
        return HttpUtility.HtmlEncode("Hola " + nombre + ", NumVeces es: " + numVeces);
    }
```

Ejecute su aplicación y busque la URL de ejemplo (http://localhost:xxxxx/HelloWorld/Welcome?nombre=Scott&numveces=4). Puede probar diferentes valores para nombre y numVeces en la URL. El sistema de enlace del modelo ASP.NET MVC asigna automáticamente los parámetros con nombre, de la cadena de consulta en la barra de direcciones, a los parámetros de su método.



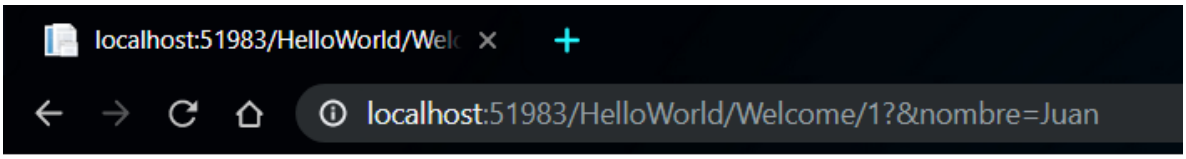
Hola Juan, NumVeces es: 4

En el ejemplo anterior, el segmento URL (Parameters) no se usa, los parámetros nombre y numVeces se pasan como cadenas de consulta. Los ? (signo de interrogación) en la URL anterior son un separador, y siguen las cadenas de consulta. El carácter & separa las cadenas de consulta.

Reemplace el método de bienvenida con el siguiente código:

```
public string Welcome(string nombre, int ID = 1)
    {
        return HttpUtility.HtmlEncode("Hola " + nombre + ", ID: " + ID);
    }
```

Ejecute la aplicación e ingrese la siguiente URL: http://localhost:xxx/HelloWorld/Welcome/1?name=Juan



Hola Juan, ID: 1

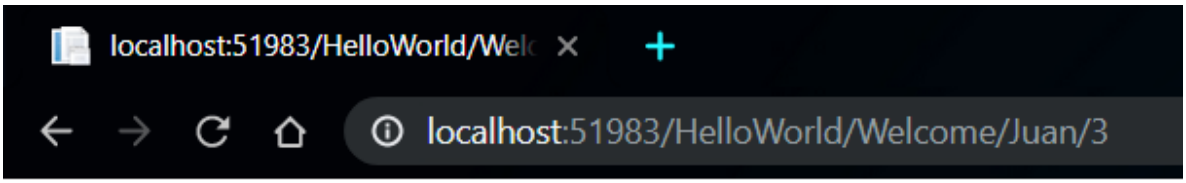
Esta vez, el tercer segmento de URL coincidió con el parámetro ID de la ruta. El método de acción Welcome contiene un parámetro (ID) que coincide con la especificación de URL en el método RegisterRoutes.

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

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

En las aplicaciones ASP.NET MVC, es más típico pasar parámetros como datos de ruta (como hicimos con la ID anterior) que pasarlos como cadenas de consulta. También puede agregar una ruta para pasar los parámetros nombre y numVeces en los datos de ruta en la URL. Para esto en el archivo App_Start\RouteConfig.cs, agregue la ruta "Hola":

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

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

}
```

Ejecute la aplicación y busque localhost:XXXX/HelloWorld/Welcome/Juan/3.



Hola Juan, ID: 3

Para muchas aplicaciones MVC, la ruta predeterminada funciona bien. Aprenderá más adelante en esta guía a pasar datos utilizando el modelo de carpeta, y no tendrá que modificar la ruta predeterminada para eso.

En estos ejemplos, el controlador ha estado haciendo la parte "VC" de MVC, es decir, la vista y el trabajo del controlador. El controlador está devolviendo HTML directamente. Por lo general, no deseamos que los controladores devuelvan HTML directamente, ya que el código se vuelve muy engorroso. En su lugar, usaremos un archivo de plantilla de vista separado para ayudar a generar la respuesta HTML. Veamos a continuación cómo podemos hacer esto.

III. AGREGAR UNA VISTA.

En esta sección, modificaremos la clase HelloWorldController para usar los archivos de plantilla de vista, para encapsular limpiamente el proceso de generación de respuestas HTML a un cliente.

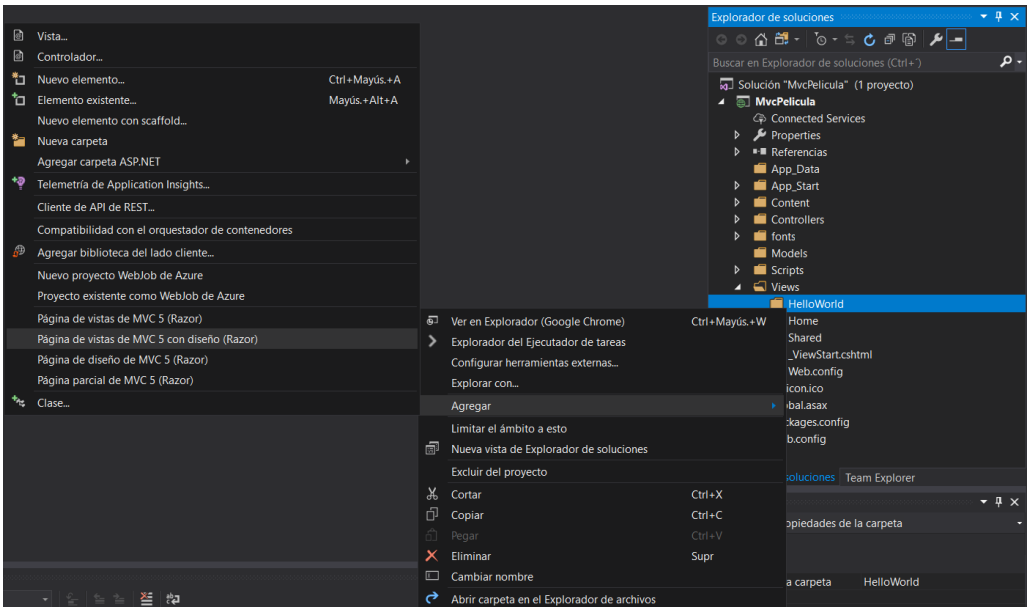
Crearemos un archivo de plantilla de vista, utilizando el motor de vista Razor. Las plantillas de vista basadas en Razor tienen una extensión de archivo .cshtml y proporcionan una manera elegante de crear resultados HTML usando C#. Razor minimiza la cantidad de caracteres y pulsaciones de teclas necesarias al escribir una plantilla de vista, y permite un flujo de trabajo de codificación rápido y fluido.

Actualmente, el método Index devuelve una cadena con un mensaje codificado en la clase de Controlador. Cambie el método Index para llamar a los métodos controladores **View**, como se muestra en el siguiente código:

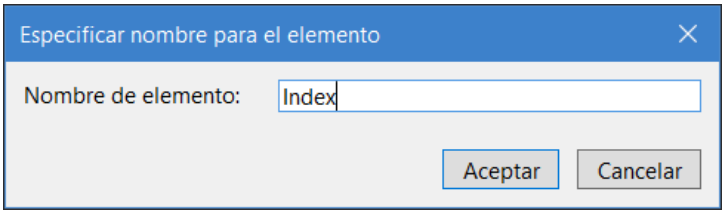
```
public class HelloWorldController : Controller
{
    //
    // GET: /HelloWorld/

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

    //
    // GET: /HelloWorld/Welcome/

    public string Welcome(string nombre, int ID = 1)
    {
        return HttpUtility.HtmlEncode("Hola " + nombre + ", ID: " + ID);
    }
}
```

En el método Index anterior se usa una plantilla de vista para generar una respuesta HTML al navegador. Los métodos de controlador (también conocidos como **métodos de acción**), como el método Index anterior, generalmente devuelven un ActionResult (o una clase derivada de ActionResult), no tipos primitivos como string.

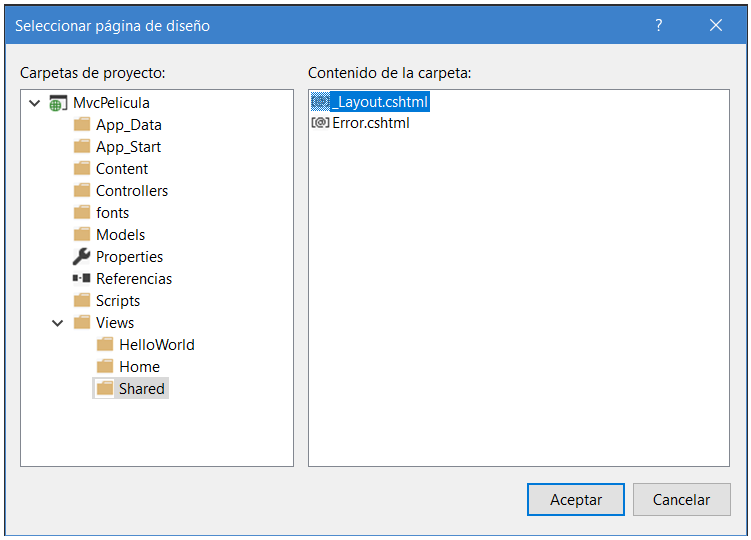
Haga clic con el botón derecho en la carpeta Views\HelloWorld y haga clic en Agregar, luego haga clic en Pagina de Vistas MVC 5 con Diseño (Razor).



En el cuadro de diálogo Especificar nombre para elemento, ingrese Index y luego haga clic en Aceptar.

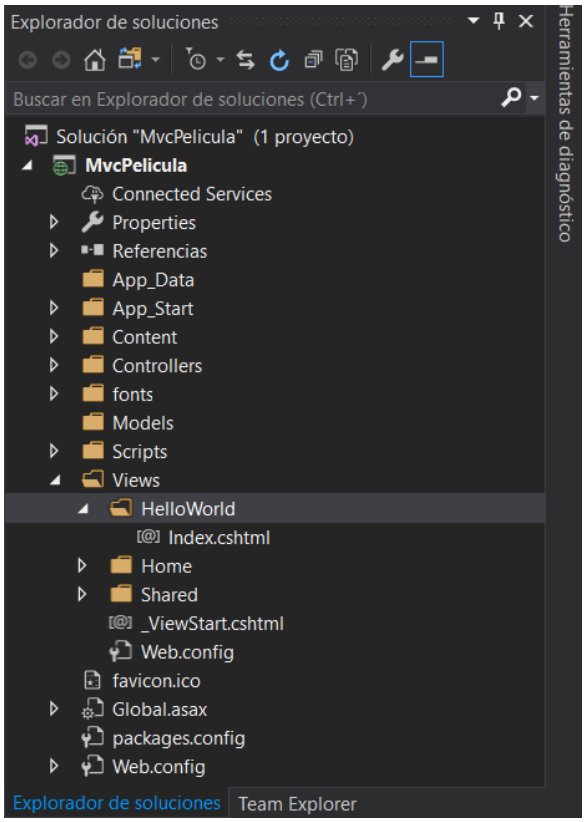


En el cuadro de diálogo Seleccionar una página de diseño, acepte el _Layout.cshtml que aparece de forma predeterminada y haga clic en Aceptar.



En el cuadro de diálogo anterior, la carpeta Views\Shared se selecciona en el panel izquierdo. Si tenía un archivo de diseño personalizado en otra carpeta, podría seleccionarlo. Hablaremos sobre el archivo de diseño más adelante en el guía.

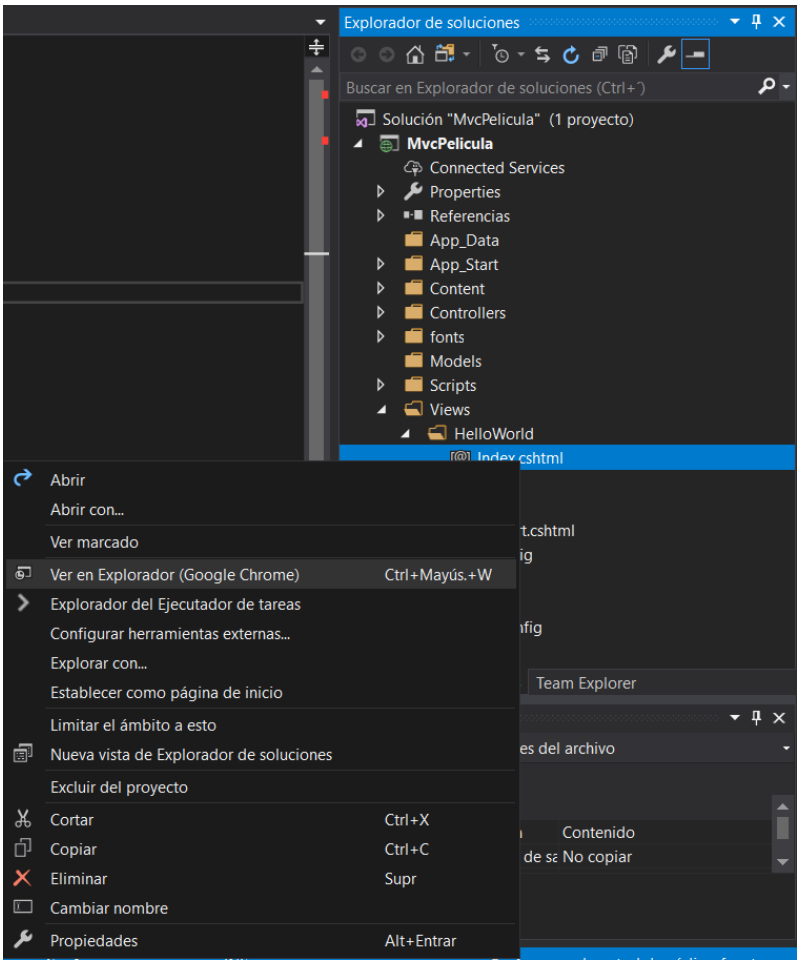
Se crea el archivo MvcPelicula\Views\HelloWorld\Index.cshtml.



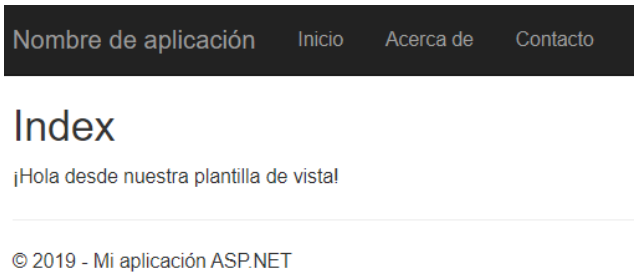
Agregue el siguiente código resaltado al archivo Index.

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

<h2>Index</h2>

<p>¡Hola desde nuestra plantilla de vista!</p>
```

Haga clic derecho en el archivo Index.cshtml y seleccione Ver en el explorador.



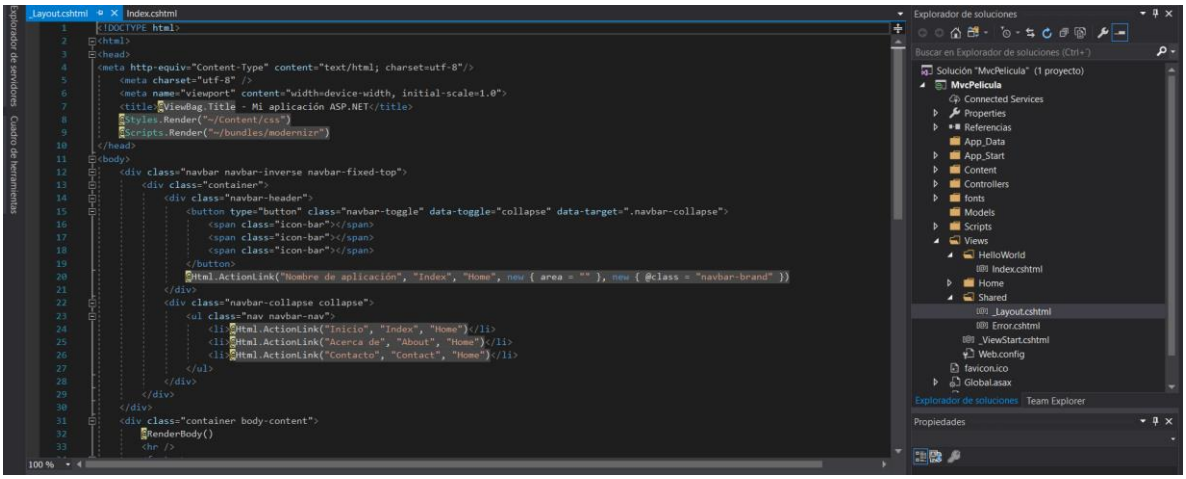
Alternativamente, ejecute la aplicación y busque el controlador HelloWorld (http://localhost:xxxx/HelloWorld). El método Index en su controlador no hizo mucho trabajo; simplemente ejecutó la declaración return View (), que especificaba que el método debería usar un archivo de plantilla de vista para responder al navegador. Debido a que no especificó explícitamente el nombre del archivo de plantilla de vista a usar, ASP.NET MVC de manera predeterminada utilizó el archivo de vista Index.cshtml en la carpeta \Views\HelloWorld. La siguiente imagen muestra la cadena codificada en la vista "¡Hola desde nuestra plantilla de vista!".



Se ve bastante bien. Sin embargo, observe que la barra de título del navegador muestra "Índice - Mi aplicación ASP.NET", y el enlace grande en la parte superior de la página dice "Nombre de la aplicación". Dependiendo de cuán pequeña sea la ventana de su navegador, es posible que deba hacer clic en las tres barras en la esquina superior derecha para ver los enlaces Inicio, Acerca de, Contacto.

Cambio de vistas y páginas de diseño.

Primero, deseamos cambiar el enlace "Nombre de la aplicación" en la parte superior de la página. Ese texto es común a todas las páginas. En realidad, se implementa en un solo lugar en el proyecto, a pesar de que aparece en cada página de la aplicación. Vaya a la carpeta /Views/Shared en el Explorador de soluciones y abra el archivo _Layout.cshtml. Este archivo se denomina página de diseño y está en la carpeta compartida que utilizan todas las demás páginas.



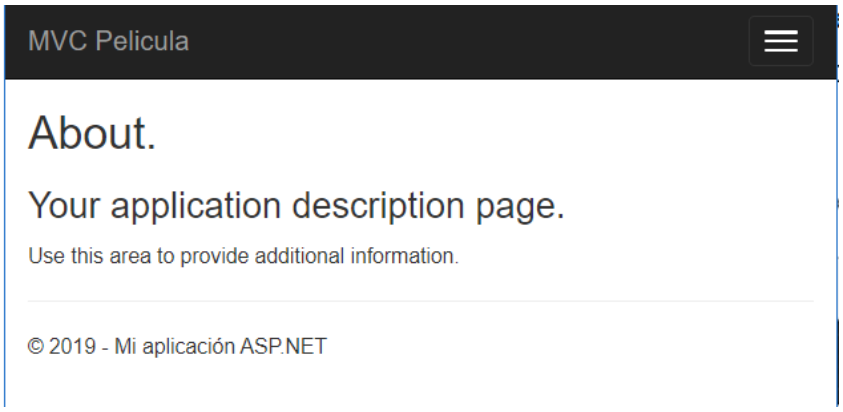
Las plantillas de diseño le permiten especificar el diseño del contenedor HTML de su sitio en un lugar y luego aplicarlo en varias páginas de su sitio. Encuentra la línea @RenderBody (). RenderBody es un marcador de posición donde aparecen todas las páginas específicas de vista que crea " wrapped" en la página de diseño. Por ejemplo, si selecciona el enlace Acerca de, la vista Views\Home\About.cshtml se representa dentro del método RenderBody.

Cambie el contenido del elemento del título. Luego cambie el ActionLink en la plantilla de diseño de "Nombre de la aplicación" a "MVC Película" y el controlador de Home a Películas. El archivo de diseño completo se muestra a continuación:

```
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8"/>
   <meta charset="utf-8" />
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
   <title>@ViewBag.Title – Aplicacion de Pelicula</title>
   @Styles.Render("~/Content/css")
   @Scripts.Render("~/bundles/modernizr")
</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="collapse" data-target=".navbar-collapse">
                    <span class="icon-bar"></span>
                    <span class="icon-bar"></span>
                    <span class="icon-bar"></span>
                </button>
                @Html.ActionLink("MVC Pelicula", "Index", "Peliculas", new { area = "" }, new { @class = "navbar-brand" })
            </div>
            <div class="navbar-collapse collapse">
                <ul class="nav navbar-nav">
                    <li>@Html.ActionLink("Inicio", "Index", "Home")</li>
                    <li>@Html.ActionLink("Acerca de", "About", "Home")</li>
                    <li>@Html.ActionLink("Contacto", "Contact", "Home")</li>
                </ul>
            </div>
        </div>
    </div>
    <div class="container body-content">
        @RenderBody()
        <hr />
        <footer>
            <p>&copy; @DateTime.Now.Year - Mi aplicación ASP.NET</p>
        </footer>
    </div>

    @Scripts.Render("~/bundles/jquery")
    @Scripts.Render("~/bundles/bootstrap")
    @RenderSection("scripts", required: false)
</body>
</html>
```

Ejecute la aplicación y observe que ahora dice "MVC Pelicula". Haga clic en el enlace Acerca de y verá cómo esa página muestra "MVC Pelicula" también. Pudimos hacer el cambio una vez en la plantilla de diseño y hacer que todas las páginas del sitio reflejen el nuevo título.



Cuando creamos por primera vez el archivo Views\HelloWorld\Index.cshtml, contenía el siguiente código:

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

El código Razor anterior está configurando explícitamente la página de diseño. Examine el archivo Views_ViewStart.cshtml, que contiene exactamente el mismo marcado de Razor. El archivo Views_ViewStart.cshtml define el diseño común que utilizarán todas las vistas, por lo tanto, puede comentar o eliminar ese código del archivo Views\HelloWorld\Index.cshtml.

```
@*@{
    Layout = "~/Views/Shared/_Layout.cshtml";
}*@
@{
    ViewBag.Title = "Index";
}

<h2>Index</h2>

<p>¡Hola desde nuestra plantilla de vista!</p>
```

Puede usar la propiedad Layout para establecer una vista de diseño diferente, o configurarla para que no se use ningún archivo de diseño (null).

Ahora, cambiemos el título de la vista de índice.

Abra MvcPelicula\Views\HelloWorld\Index.cshtml. Hay dos lugares para hacer un cambio: primero, el texto que aparece en el título del navegador, y luego en el encabezado secundario (el elemento <h2>). Los hará ligeramente diferentes para que pueda ver qué parte del código cambia qué parte de la aplicación.

```
@{
    ViewBag.Title = "Lista de Peliculas";
}
<h2>Mi lista de Peliculas</h2>

<p>¡Hola desde nuestra plantilla de vista!</p>
```

Para indicar el título HTML para mostrar, el código anterior establece una propiedad Title del objeto ViewBag (que se encuentra en la plantilla de vista Index.cshtml). Tenga en cuenta que la plantilla de diseño (Views\Shared_Layout.cshtml) usa este valor en el elemento <title> como parte de la sección <head> del HTML que modificamos anteriormente.

```
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8"/>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>@ViewBag.Title - Aplicación de Película </title>
    @Styles.Render("~/Content/css")
    @Scripts.Render("~/bundles/modernizr")
</head>
```

Con este enfoque ViewBag, puede pasar fácilmente otros parámetros entre su plantilla de vista y su archivo de diseño.

Ejecute la aplicación. Observe que el título del navegador, el encabezado principal y los encabezados secundarios han cambiado. (Si no ve cambios en el navegador, es posible que esté viendo contenido en caché. Presione Ctrl + F5 en su navegador para forzar la carga del servidor). El título del navegador se crea con el ViewBag.Title que configuramos en la plantilla de la vista index.cshtml y la "Aplicación de Película" adicional agregada en el archivo de diseño.

Observe también cómo el contenido de la plantilla de vista Index.cshtml se fusionó con la plantilla de vista _Layout.cshtml y se envió una única respuesta HTML al navegador. Las plantillas de diseño hacen que sea realmente fácil realizar cambios que se apliquen en todas las páginas de su aplicación.

Sin embargo, nuestro pequeño "dato" (en este caso, el mensaje "¡Hola desde nuestra plantilla de vista!") está codificado. La aplicación MVC tiene una "V" (vista) y usted tiene un "C" (controlador), pero todavía no tiene una "M" (modelo). En la siguiente guía, veremos cómo crear una base de datos y recuperar datos del modelo.

Pasar datos del controlador a la vista

Sin embargo, antes de ir a una base de datos y hablar sobre modelos, primero hablemos sobre cómo pasar información del controlador a una vista. Las clases de controlador se invocan en respuesta a una solicitud de URL entrante. Una clase de controlador es donde se escribe el código que maneja las solicitudes entrantes del navegador, recupera los datos de una base de datos y, en última

instancia, decide qué tipo de respuesta enviar al navegador. Las plantillas de vista se pueden usar desde un controlador para generar y formatear (dar formato) una respuesta HTML al navegador.

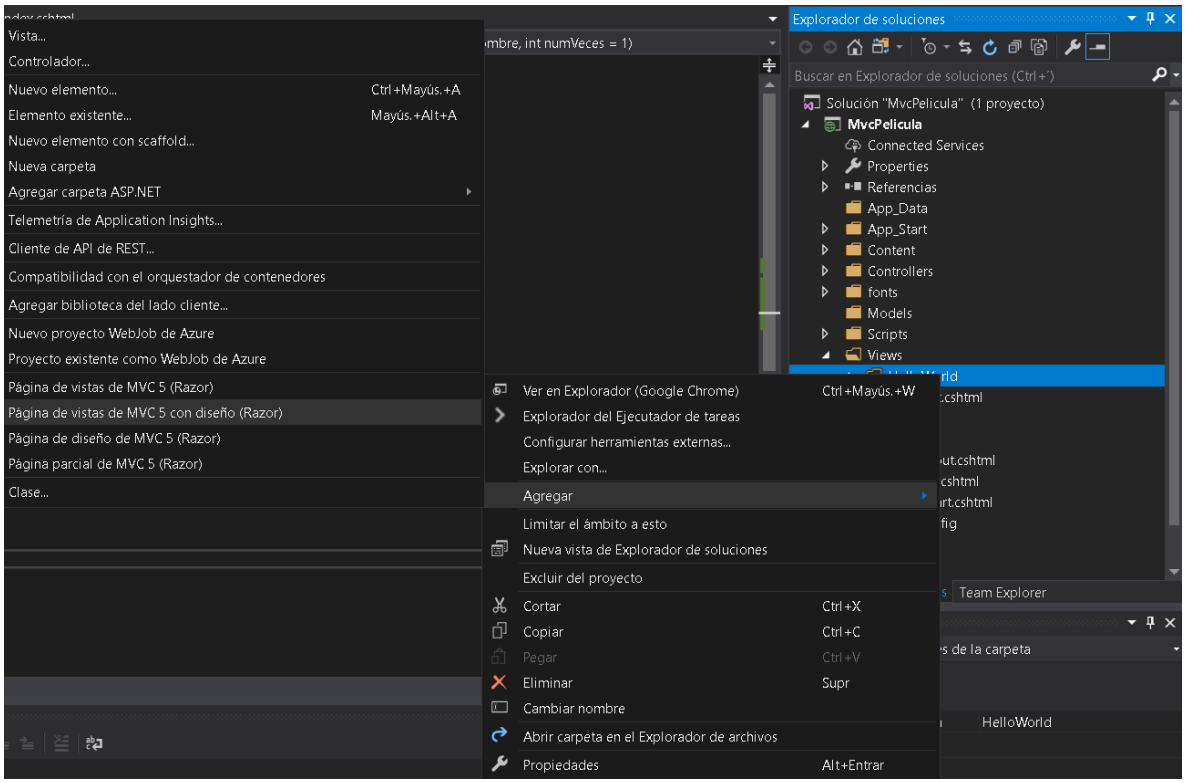
Los controladores son responsables de proporcionar los datos u objetos necesarios para que una plantilla de vista responda al navegador. Una práctica recomendada es que: una plantilla de vista nunca debe realizar la lógica de negocio o interactuar con una base de datos directamente. En cambio, una plantilla de vista debería funcionar solo con los datos que le proporciona el controlador. Mantener esta "separación de responsabilidades" ayuda a mantener su código limpio, comprobable y más fácil de mantener.

Actualmente, el método de acción Welcome en la clase HelloWorldController toma los parámetros nombre y numVeces y luego envía los valores directamente al navegador. En lugar de que el controlador presente esta respuesta como una cadena, cambiemos el controlador para que use una plantilla de vista. La plantilla de vista generará una respuesta dinámica, lo que significa que debe pasar los bits de datos apropiados del controlador a la vista para generar la respuesta. Puede hacer esto haciendo que el controlador ponga los datos dinámicos (parámetros) que la plantilla de vista necesita en un objeto ViewBag al que la plantilla de vista puede acceder.

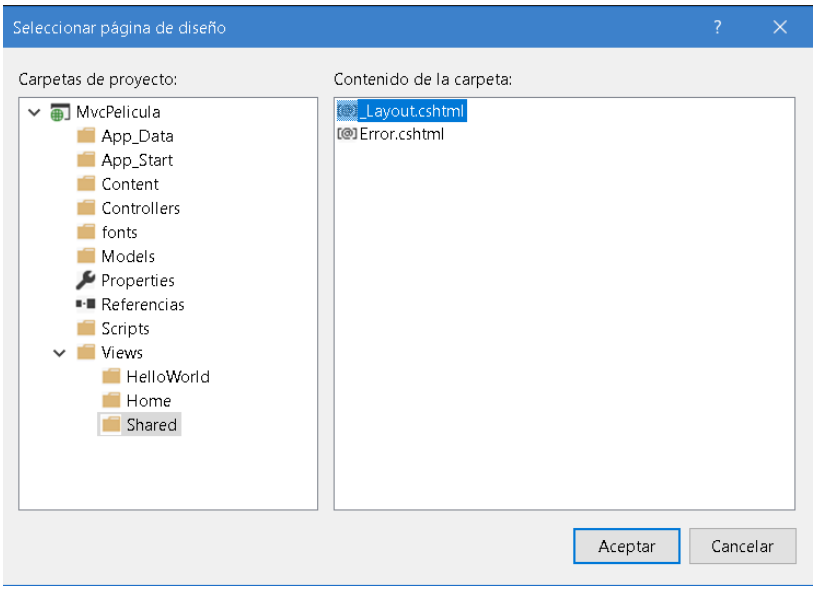
Volveremos al archivo HelloWorldController.cs y cambiaremos el método Welcome para agregar un valor de Mensaje y NumVeces al objeto ViewBag. ViewBag es un objeto dinámico, lo que significa que puede poner lo que quiera en él; el objeto ViewBag no tiene propiedades definidas hasta que coloque algo dentro de él. El sistema de enlace del modelo ASP.NET MVC asigna automáticamente los parámetros con nombre (nombre y numVeces) de la cadena de consulta en la barra de direcciones, a los parámetros de su método. El archivo completo HelloWorldController.cs tiene este aspecto:

```
public class HelloWorldController : Controller
   {
      //
      // GET: /HelloWorld/

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

      //
      // GET: /HelloWorld/Welcome/

      public ActionResult Welcome(string nombre, int numVeces = 1)
      {
         ViewBag.Mensaje= "Hola "+nombre;
         ViewBag.NumVeces = numVeces;

         return View();

      }
   }
```

Ahora el objeto ViewBag contiene datos que se pasarán a la vista automáticamente. A continuación, necesitamos una plantilla de vista para el método Welcome. En el menú Compilar, seleccione Compilar Solución (o Ctrl + Shift + B) para asegurarse de que el proyecto se compila. Haga clic con el botón derecho en la carpeta Views\HelloWorld y haga clic en Agregar, luego haga clic en Pagina de vistas de MVC 5 con diseño (Razor).



En el cuadro de diálogo Especificamos el nombre para el elemento, ingrese el nombre Welcome y luego haga clic en Aceptar.



Se crea el archivo MvcPelicula\Views\HelloWorld\Welcome.cshtml.

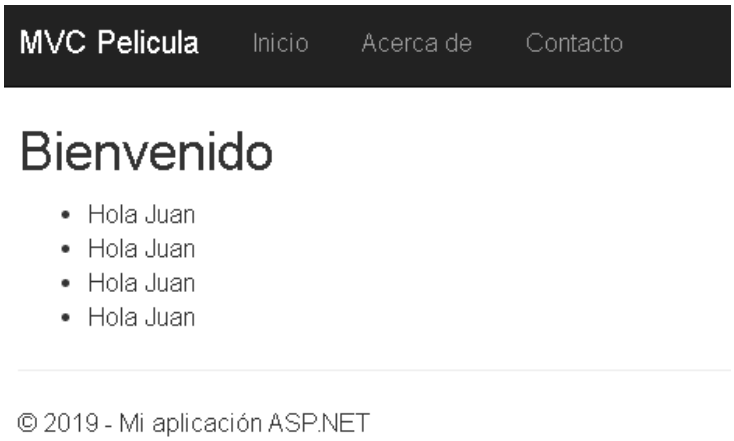
Reemplace el código en el archivo Welcome.cshtml. Esto creará un bucle que diga "Hola" tantas veces como el usuario diga. El archivo completo Welcome.cshtml se muestra a continuación.

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

<h2>Bienvenido</h2>

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

Ejecute la aplicación y busque la siguiente URL:

http://localhost:xxxxx/HelloWorld/Welcome?nombre=Juan&numVeces=4

Ahora los datos se toman de la URL y se pasan al controlador utilizando el modelo de carpeta. El controlador empaqueta los datos en un objeto ViewBag y pasa ese objeto a la vista. La vista luego muestra los datos como HTML para el usuario.



En el ejemplo anterior, usamos un objeto ViewBag para pasar datos del controlador a una vista. Más adelante en la segunda parte, utilizaremos un modelo de vista para pasar datos de un controlador a una vista. El enfoque del modelo de vista para pasar datos, es generalmente preferido sobre el enfoque de la bolsa de vista.

Lo anterior era una especie de "M" para el modelo, pero no del tipo de base de datos. Tomemos lo que hemos aprendido y creemos una base de datos de películas.

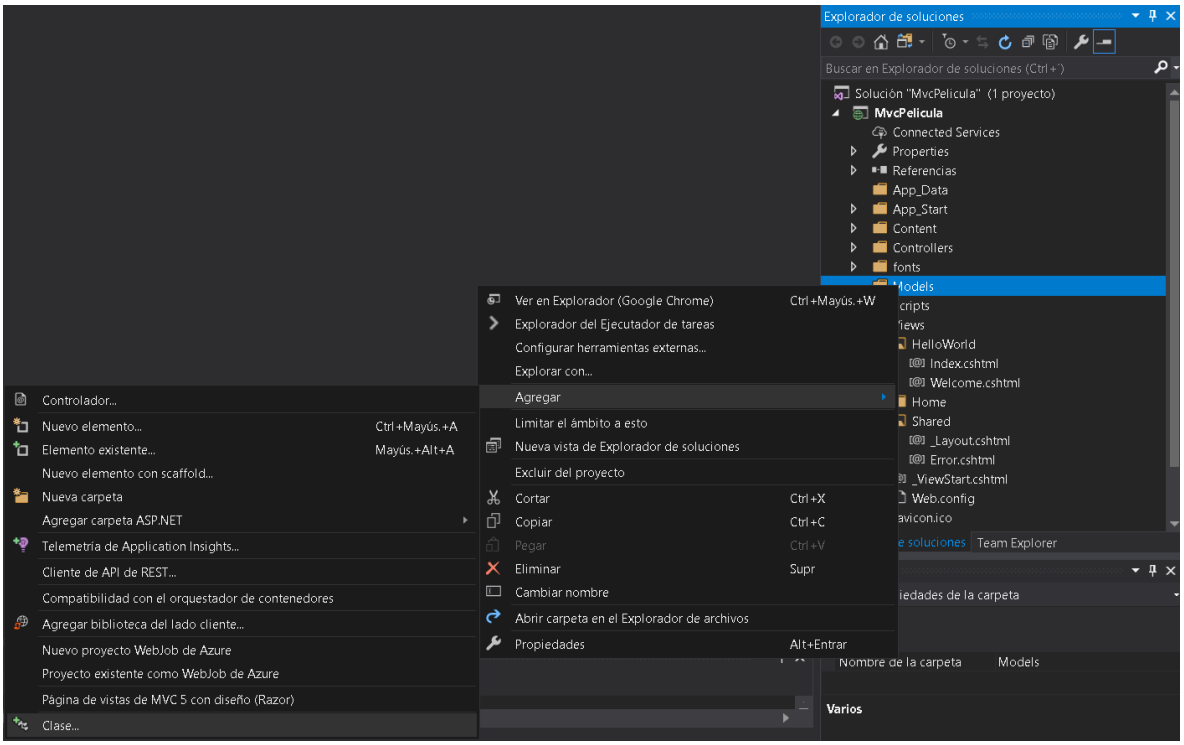
IV. AGREGAR UN MODELO

En esta sección, agregará algunas clases para administrar películas en una base de datos. Estas clases serán la parte "Modelo" de la aplicación ASP.NET MVC.

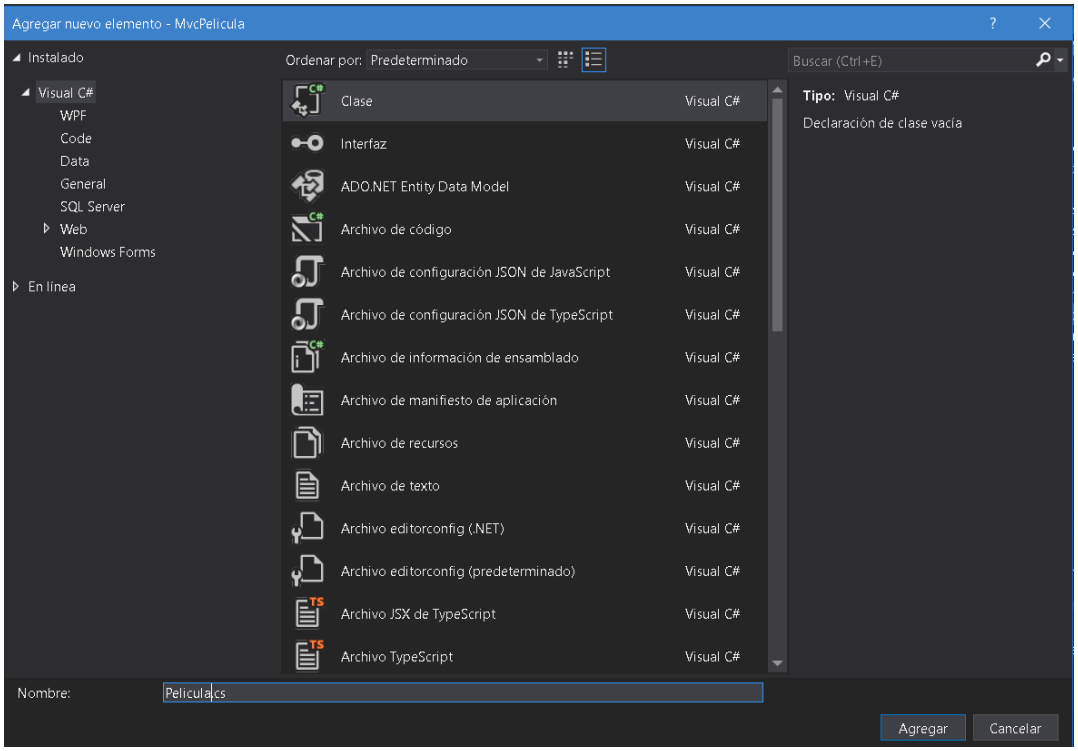
Utilizaremos una tecnología de acceso a datos de .NET Framework conocida como Entity Framework, para, definir y trabajar con estas clases de modelos. El Entity Framework (a menudo denominado EF) admite un paradigma de desarrollo llamado Code First. Code First le permite crear modelos escribiendo clases simples. (También se conocen como clases POCO, de "objetos CLR simples"). Luego, puede crear la base de datos sobre la marcha desde sus clases a través de migraciones, lo que permite un flujo de trabajo de desarrollo muy limpio y rápido.

Agregar clases de modelos

En el Explorador de soluciones, haga clic con el botón derecho en la carpeta Models, seleccione Agregar y luego seleccione Clase.



Ingrese el nombre de la clase "Película" y presione el botón agregar.

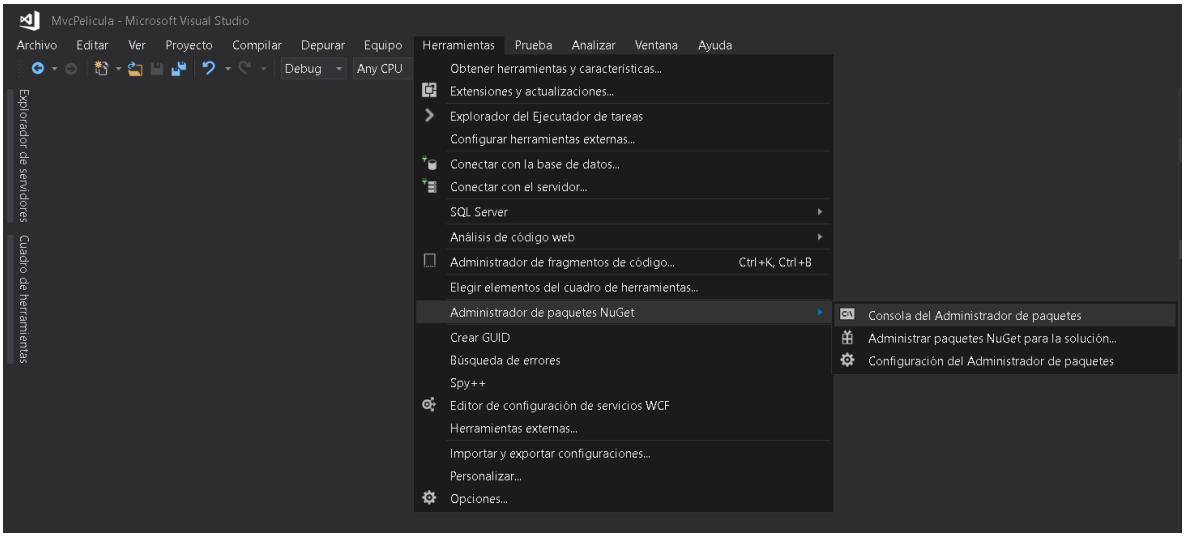


Agregue las siguientes cinco propiedades a la clase Película:

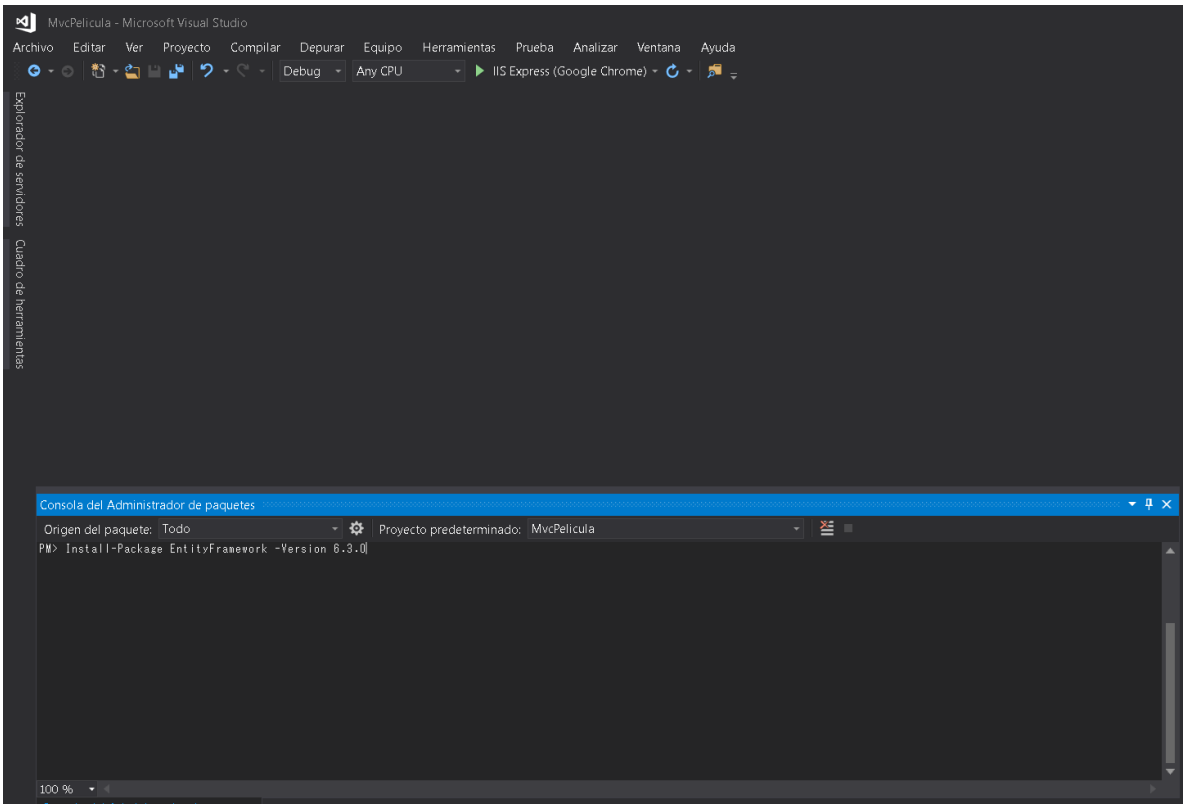
```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;

namespace MvcPelicula.Models
{
    public class Pelicula
    {
        public int ID { get; set; }
        public string Titulo { get; set; }
        public DateTime FechaLanzamiento { get; set; }
        public string Genero { get; set; }
        public decimal Precio { get; set; }
    }
}
```

Usaremos la clase Película para representar películas en una base de datos. Cada instancia de un objeto Película corresponderá a una fila dentro de una tabla de base de datos, y cada propiedad de la clase Película se asignará a una columna en la tabla.

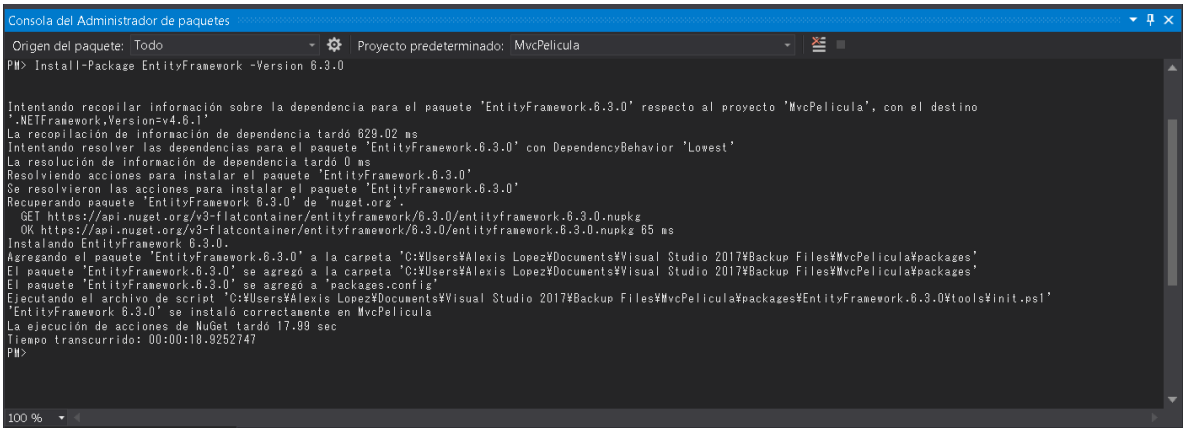
Nota: Para usar System.Data.Entity y la clase relacionada, debe instalar el paquete NuGet de Entity Framework. Para esto ejecutamos la sentencia "*Install-Package EntityFramework -Version 6.3.0*" a través de la consola de Administración de paquetes de NuGet, ubicada en el menú de Herramientas-Administrador de paquetes de NuGet.



Colocamos la línea de código "*Install-Package EntityFramework -Version 6.3.0*" en la consola y presionamos enter.



Cuando el proceso haya finalizado mostrara la siguiente información.



En el mismo archivo del modelo, agregue la siguiente clase MovieDBContext:

```
using System;
using System.Collections.Generic;
using System.Data.Entity;
using System.Linq;
using System.Web;

namespace MvcPelicula.Models
```

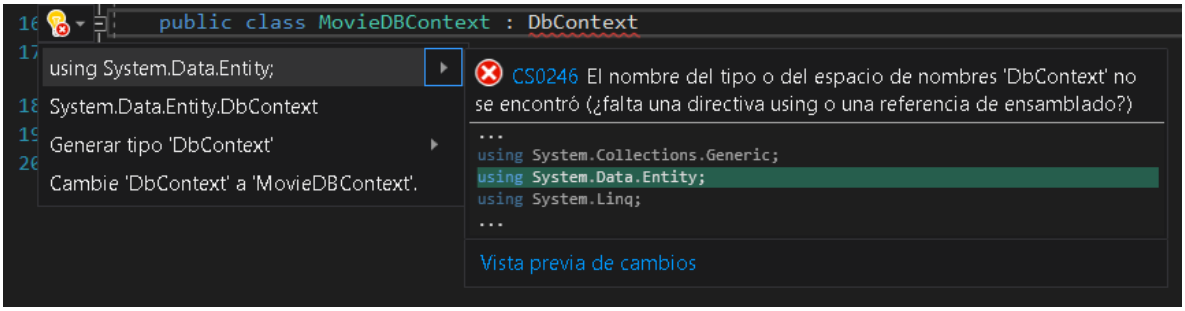
```
{
    public class Pelicula
    {
        public int ID { get; set; }
        public string Titulo { get; set; }
        public DateTime FechaLanzamiento { get; set; }
        public string Genero { get; set; }
        public decimal Precio { get; set; }
    }
    public class PeliculaDBContext : DbContext
    {
        public DbSet<Pelicula> Peliculas { get; set; }
    }
}
```

La clase PeliculaDBContext representa el contexto de la base de datos de películas de Entity Framework, que maneja la obtención, el almacenamiento y la actualización de instancias de clase Película en una base de datos. PeliculaDBContext deriva de la clase DbContext base proporcionada por el marco de la entidad.

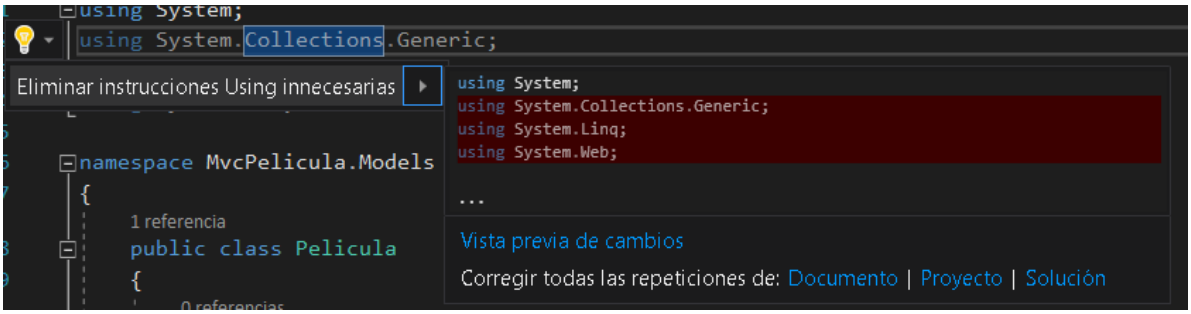
Para poder hacer referencia DbContexty DbSet, debe agregar la siguiente declaración using en la parte superior del archivo:

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

Puede hacer esto agregando manualmente la declaración de uso, o puede pasar el mouse sobre las líneas rojas onduladas, hacer clic Mostrar posibles soluciones y hacer clic using System.Data.Entity;



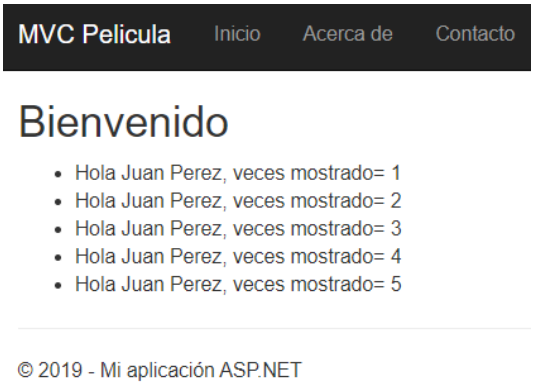
Nota: Podemos eliminar varias declaraciones no utilizadas. Visual Studio mostrará las dependencias no utilizadas en gris. Puede eliminar las dependencias no utilizadas pasando el cursor sobre las dependencias grises, haga clic Mostrar posibles soluciones y haga clic en Eliminar usos no utilizados.



Finalmente hemos agregado un modelo (la M en MVC). En la siguiente guía de laboratorio, trabajaremos con la cadena de conexión de la base de datos.

V. Ejercicios.

1. Modifique la vista HelloWorld/Welcome y el archivo RouteConfig.cs para que además de recibir los parámetros de Nombre y NumVeces en la URL reciba el apellido y muestre en la vista el contador de las veces que se ha mostrado el nombre. Ejemplo:



2. Cree un nuevo Proyecto con el nombre Persona y siga los pasos necesarios para agregar el modelo, los atributos que debe tener la clase persona son DUI, Nombre, Apellido, Fecha de nacimiento, Dirección y correo electrónico, tenga en cuenta que el desarrollo de este ejercicio será necesario para realizar los ejercicios de guías posteriores.

VI. Referencias

- Rick-Anderson, R. A. (2018, 4 octubre). Getting Started with ASP.NET MVC 5. Recuperado 2 noviembre, 2019, de https://docs.microsoft.com/en-us/aspnet/mvc/overview/getting-started/introduction/getting-started
- Vaidy4github, V. A. I. D. Y. 4. (2010, 6 julio). IIS Express Overview. Recuperado 2 noviembre, 2019, de https://docs.microsoft.com/en-us/iis/extensions/introduction-to-iis-express/iis-express-overview