

Mi primera aplicación ASP.NET MVC 2 paso a paso – parte 4 « afelipelc Blog

Continuando con el desarrollo de nuestra aplicación ASP.NET MVC 2... del tutorial original : Build your First ASP.NET MVC Application :: <http://www.asp.net/mvc/tutorials/getting-started-with-mvc-part1>

En esta parte (4), crearemos el controlador **AdministrarPelículas** que será el módulo de administración, donde se podrá agregar nuevos registros, editar y eliminar, posteriormente se definirá un rol de acceso a este controlador.

Agregar la clase controlador AdministrarPelículas

Img. 1.- Agregar el controlador AdministrarPelículas

Ya creado el controlador, agregamos el using Películas.Models; Dentro de la clase, declaramos una instancia de nuestro EntityModel que se llama PelículasEntities.

```
PelículasEntities DB = new PelículasEntities();
```

En este caso, lo que mostrara la vista Index.aspx de este controlador, será toda la lista de películas que tenemos en la colección.

Entonces el código de la acción Index() del controlador AdministrarPelículas quedara así:

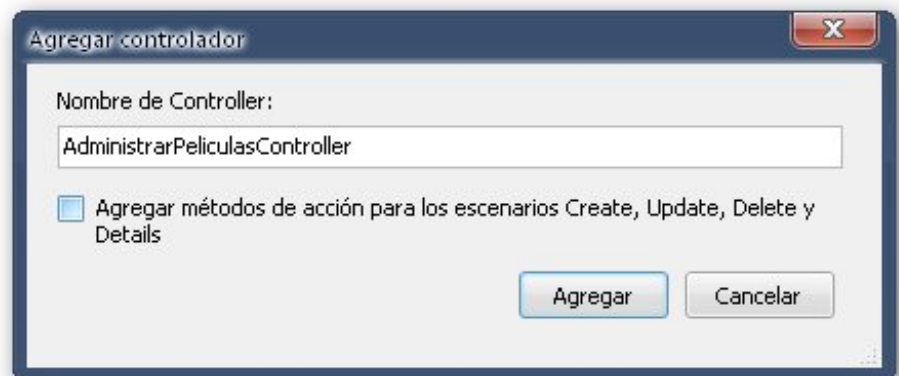
```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Mvc;

//Agregar el using de los modelos
using Películas.Models;

namespace Películas.Controllers
{
    public class AdministrarPelículasController : Controller
    {
        //crear instancia del entity model (Base de datos)
        PelículasEntities DB = new PelículasEntities();

        //
        // GET: /AdministrarPelículas/
        public ActionResult Index()
        {
            //Cargamos todo el contenido de la entidad Película
            var películas = DB.Película;

            //Devolver el modelo cargado (películas) a la vista en forma de lista
            return View(películas.ToList());
            //crear una vista del tipo Movies.Models.Película, con contenido List
        }
    }
}
```

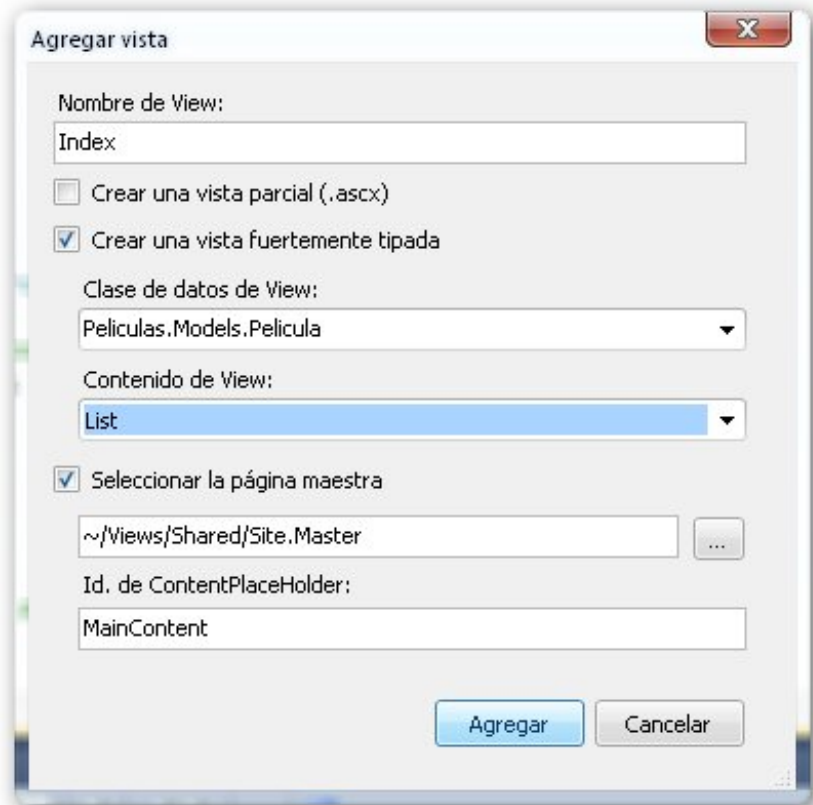


}

Creamos la vista Index.aspx de la clase Peliculas.Models.Pelicula con contenido List.

Img. 2.- Agregar el controlador
AdminsitrarPeliculas

Agregamos el código que genere el link



<http://localhost:7406/AdministrarPeliculas> en la lista id="menu" del Site.Master.

Que quedaría así:

```
<ul id="menu">
<li><%: Html.ActionLink("Página principal", "Index", "Home")%></li>
<li><%: Html.ActionLink("Peliculas", "Index", "Peliculas")%></li>
<li><%: Html.ActionLink("Administrar", "Index", "AdministrarPeliculas")%></li>
<li><%: Html.ActionLink("Acerca de", "About", "Home")%></li>
</ul>
```

Ejecutando el proyecto e ingresando a <http://localhost:7406/AdministrarPeliculas>



Img 3.- Vista Vista Index del controlador AdministrarPeliculas

Solo tenemos que personalizar un poco el código generado, en este caso solo dejamos los links para Eliminar y Editar (Puedes dejar Detalles y agregar la acción y generar la vista como en la parte 3).

```
<%@ Page Title="" Language="C#" MasterPageFile="~/Views/Shared/Site.Master" Inherits='
<asp:Content ID="Content1" ContentPlaceHolderID="TitleContent" runat="server">
  Administrar
</asp:Content>

<asp:Content ID="Content2" ContentPlaceHolderID="MainContent" runat="server">

  <h2>Administrar Peliculas</h2>

  <table>
    <tr>
      <th></th>
      <th>
        ID
      </th>
      <th>
        Titulo
      </th>
      <th>
        Fecha de Lanzamiento
      </th>
      <th>
        Genero
      </th>
      <th>
        Precio
      </th>
      <th>
        Disponible
      </th>
    </tr>

    <% foreach (var item in Model) { %>

      <tr>
        <td>
          <% //Generamos los links, cambie los nombres de las acciones (Edit,Details
            <%= Html.ActionLink("Editar", "Editar", new { id=item.PeliculaId }) %>
            <%= Html.ActionLink("Detalles", "Detalles", new { id=item.PeliculaId }
            <%= Html.ActionLink("Eliminar", "Eliminar", new { id=item.PeliculaId }
          </td>
          <td>
            <%= item.PeliculaId %>
          </td>
          <td>
            <%= item.Titulo %>
          </td>
          <td>
            <%= String.Format("{0:d}", item.FechaLanzamiento) %>
          </td>
          <td>
            <% //Mostrar el nombre del genero del elemento en lugar de GeneroId %>
            <%= item.Genero.Nombre %>
          </td>
          <td>
```

```

        <%: String.Format("$ {0:F}", item.Precio) %>
    </td>
    <td>
        <% //item.Disponible muestra True o False (es de tipo Boleano), deben
        //que el estado Disponible muestre (Si/No) %>
        <% //Aquí posteriormente en vez de mostrar SI o NO, mostraremos un cf
        //a una acción que actualize el estado de la película.
        %>
        <% try
        {
            if ((bool) (item.Disponible))
            { %>
                <% //Aquí posteriormente en vez de mostrar SI, mostraremos
                //a una acción que actualice el estado de la película %>
                Si
                <%
            }
            else
            { %>
                No
                <% }
            }
            catch
            {
                %>
                No <% //Si Disponible es NULL, se producirá un error, entonces lo
            <%
            } %>
        } %>
    </td>
</tr>

<% } %>

</table>

<p>
    <%: Html.ActionLink("Registrar nueva película", "Create") %>
</p>

</asp:Content>

```

Ejecutamos y este es el resultado.

[[Iniciar sesión](#)]

Películas

Página principal
Películas
Administrar
Acerca de

Administrar Películas

	ID	Título	Fecha de Lanzamiento	Genero	Precio	Disponible
Editar Detalles Eliminar	1	Identidad Sustituta	25/09/2009	Ficcion	\$ 80.00	Si
Editar Detalles Eliminar	2	Bourne Ultimatum	15/06/2007	Ficcion	\$ 80.00	Si
Editar Detalles Eliminar	3	Dejavu	22/02/2004	Accion	\$ 70.00	Si

[Registrar nueva película](#)

Creando mi aplicación en ASP.NET MVC 2, 2010

Img 4.-Vista Index del controlador AdministrarPeliculas modificado

Ahora viene lo más interesante.

Vamos a crear el formulario para crear un nuevo registro.
Normalmente mi acción Crear() quedaría así:

```
// GET: /AdministrarPeliculas/Crear
public ActionResult Crear()
{
    return View();
}
```

Agrego la vista de la acción Crear() del tipo de la clase Peliculas.Models.Pelicula con contenido Create.
Si ejecuto, el resultado sería:

Img 5.-Vista Crear predeterminada

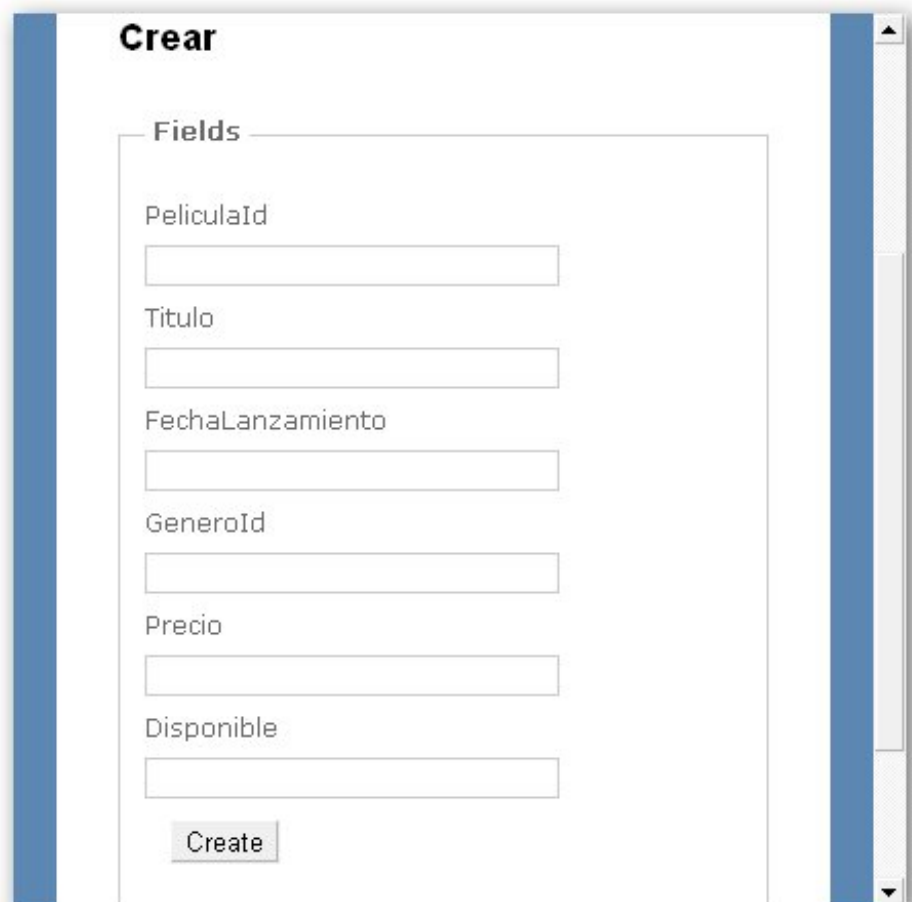
En este caso, me pide PeliculaId (Este campo es auto incrementable), también me pide GeneroId (tengo que especificar el Id del genero de la película a registrar).

Lo que quiero es que NO me muestre el TXT de PeliculaId y que en lugar de GeneroId, me muestre la lista de géneros que tengo disponibles para poder seleccionar el genero de dicha película.

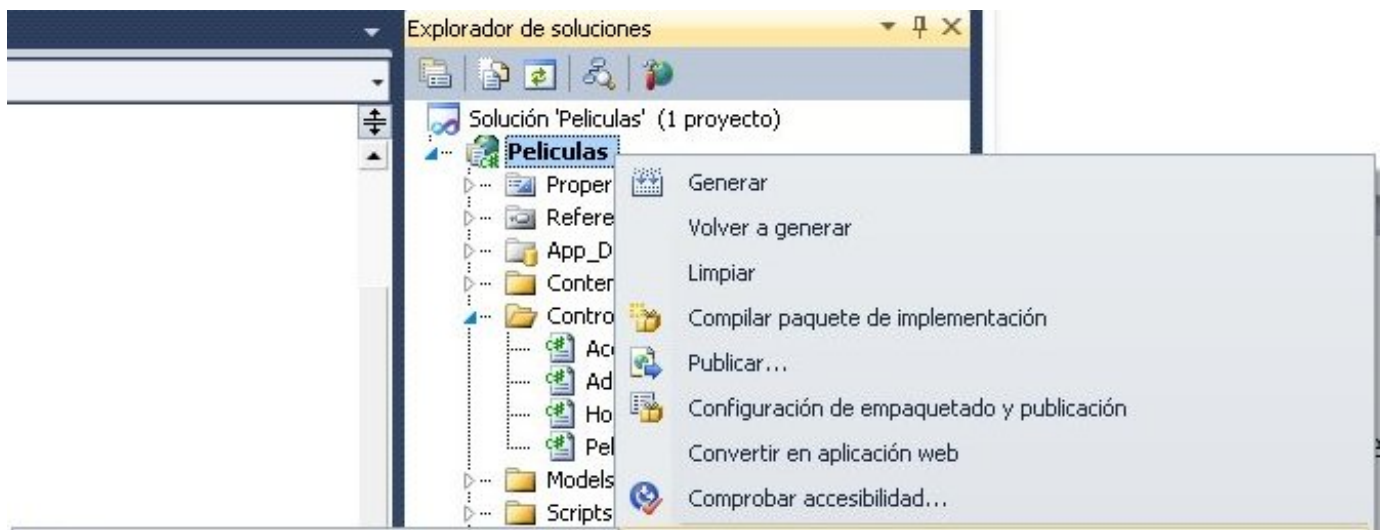
¿Cómo hago esto?

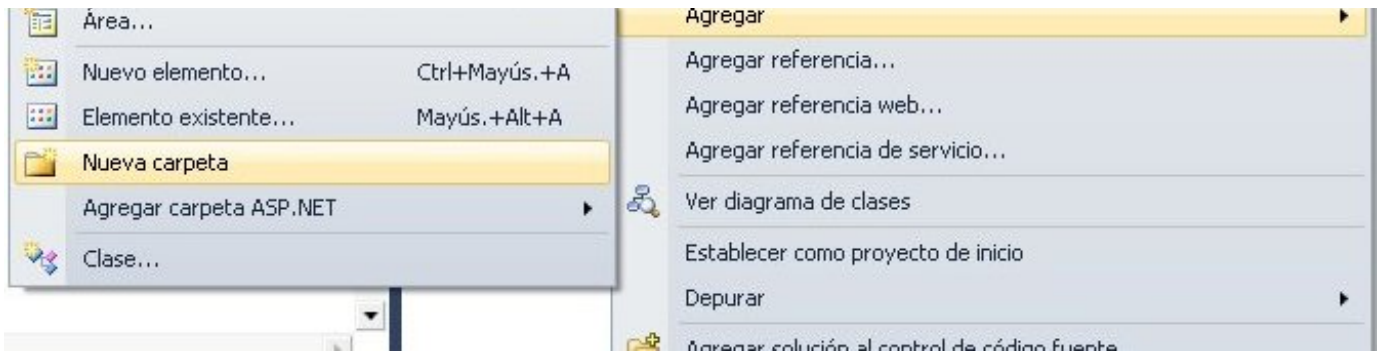
Tengo que crear un modelo especial para mi vista Crear y Editar porque ambas utilizaran el mismo formulario.

Entonces agrego una carpeta llamada ViewModel a mi proyecto.



The screenshot shows a web browser window displaying a form titled "Crear". The form is enclosed in a box labeled "Fields" and contains several input fields: "PeliculaId", "Titulo", "FechaLanzamiento", "GeneroId", "Precio", and "Disponible". Each field is represented by a simple text input box. At the bottom of the form, there is a button labeled "Create". The browser's address bar and other UI elements are not visible.



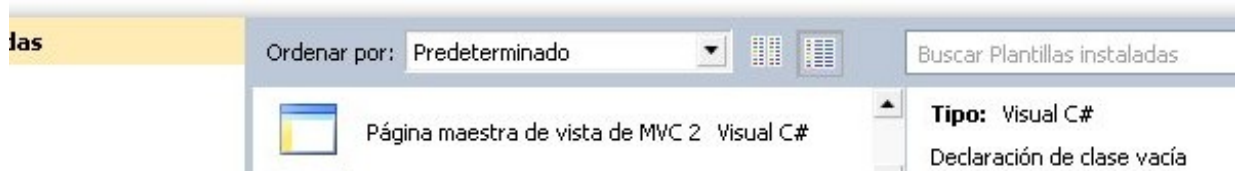
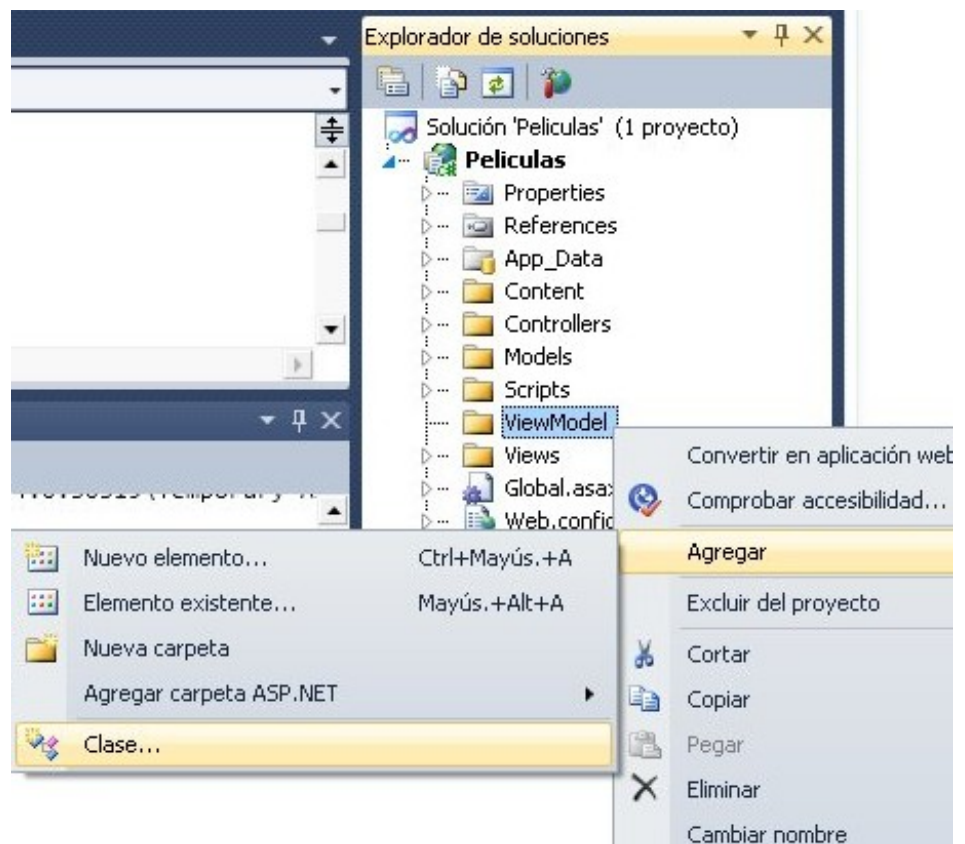
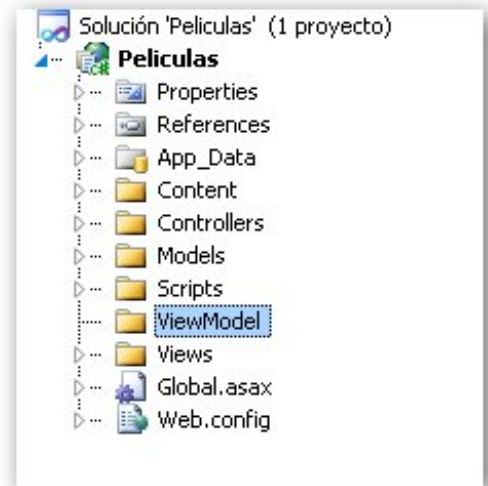


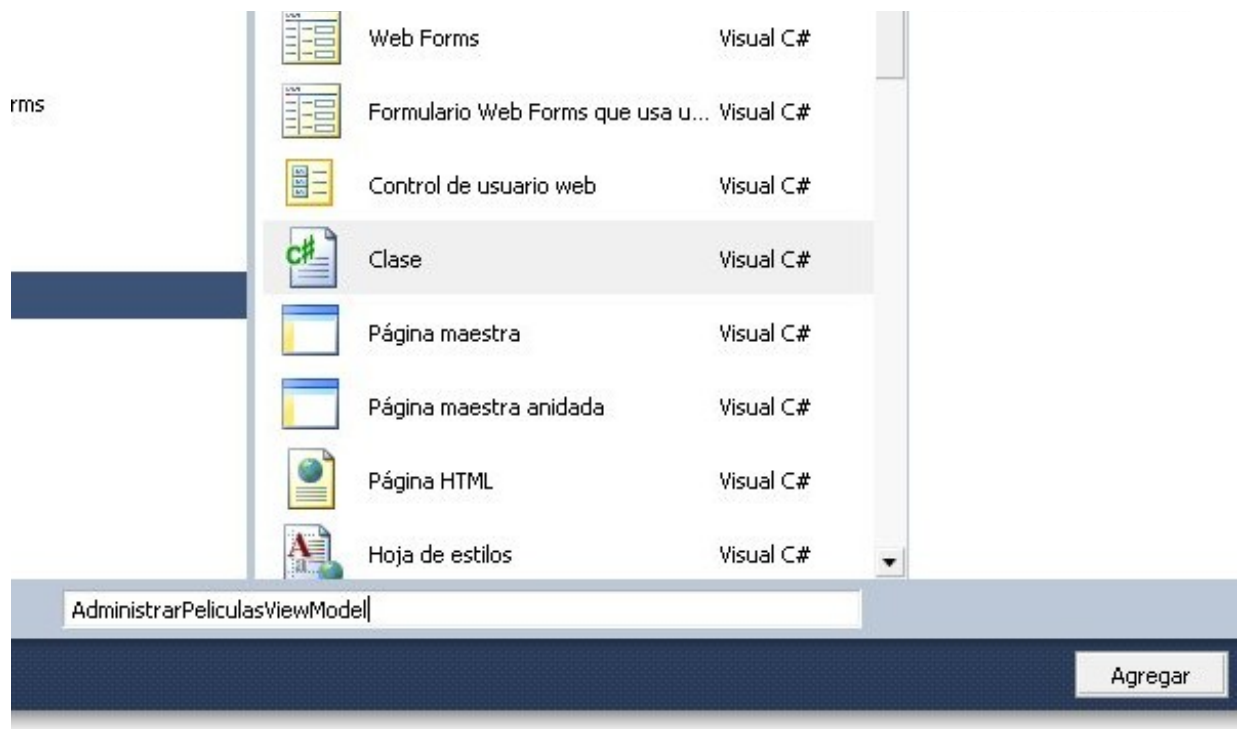
Img 6.-Agregar la carpeta ViewModel al proyecto

Img 7.-Agregar la carpeta ViewModel al proyecto

Para que lo comentado anteriormente funcione, debemos crear un nuevo modelo que sera una clase llama AdministrarPelículasViewModel, la creamos en la carpeta ViewModel.

Img 8.-Agregar nueva clase a la carpeta ViewModel





Img 9.-Agregar clase AdministrarPeliculasViewModel

La clase **AdministrarPeliculasViewModel** quedara así:

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

using Peliculas.Models;
namespace Peliculas.ViewModel
{
    public class AdministrarPeliculasViewModel
    {
        //Esta clase es un modelo personalizado que:
        //como propiedades de esta clase

        //tendrá una propiedad llamada Película de la clase película la cual recibira
        //o proporcionara un objeto Película
        //ya que se estará creando o editando UN SOLO Objeto de dicha clase
        public Pelicula Pelicula { get; set; }

        //Tendra otra propiedad llamada Generos que sera el contenido de la entidad
        //Genero pero en una Lista, a partir de esta lista se crea el ListBox en la
        public List<Genero> Generos { get; set; }
    }
}
```

Después de crear la clase AdministrarPeliculasViewModel, debemos compilar el proyecto, para que podamos usar la nueva clase como otro MODELO más.

Agregar el espacio de nombres Peliculas.ViewModel a la clase controlador AdministrarPeliculas

```
using Peliculas.ViewModel;
```

Posteriormente la Acción Crear() deberá quedar así;

```
// GET: /AdministrarPeliculas/Crear
public ActionResult Crear()
{
    //Creamos un objeto de nuestro nuevo modelo
```

```

var viewModel = new AdministrarPeliculasViewModel
{
    //Asignamos un nuevo objeto Pelicula a la propiedad Pelicula de nuestro nuevo n
    //Y en la propiedad Géneros, le pasamos la colección (lista) de géneros de la e
    Pelicula = new Pelicula(),
    Generos = DB.Genero.ToList()
};
//Devolvemos a la vista nuestro modelo personalizado.
return View(viewModel);
}

```

Nota: Si creaste la vista Crear.aspx, elimina ese archivo.

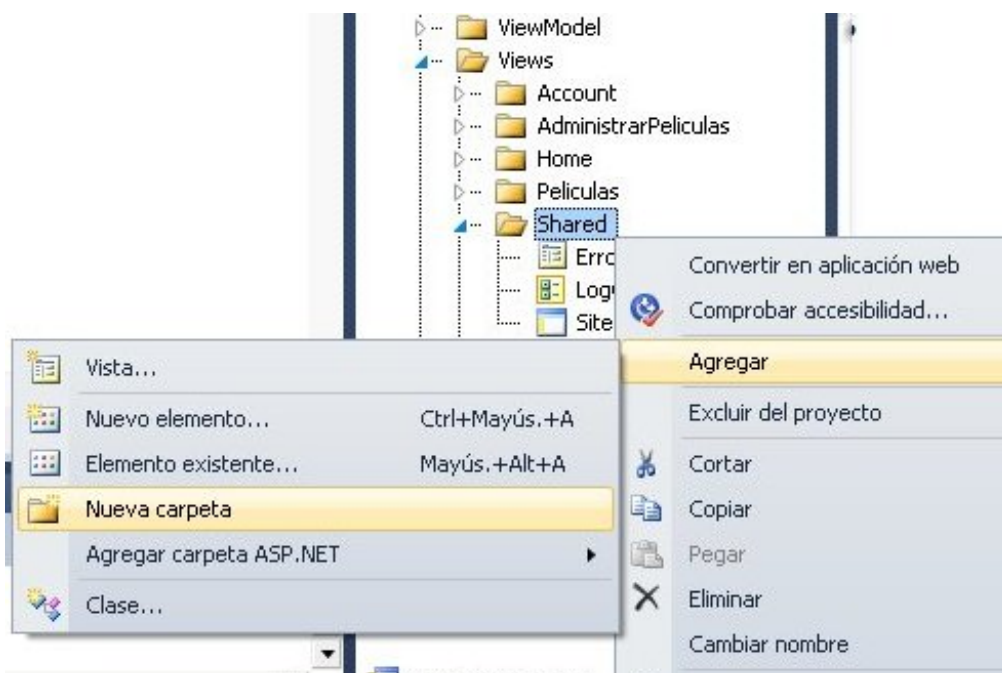
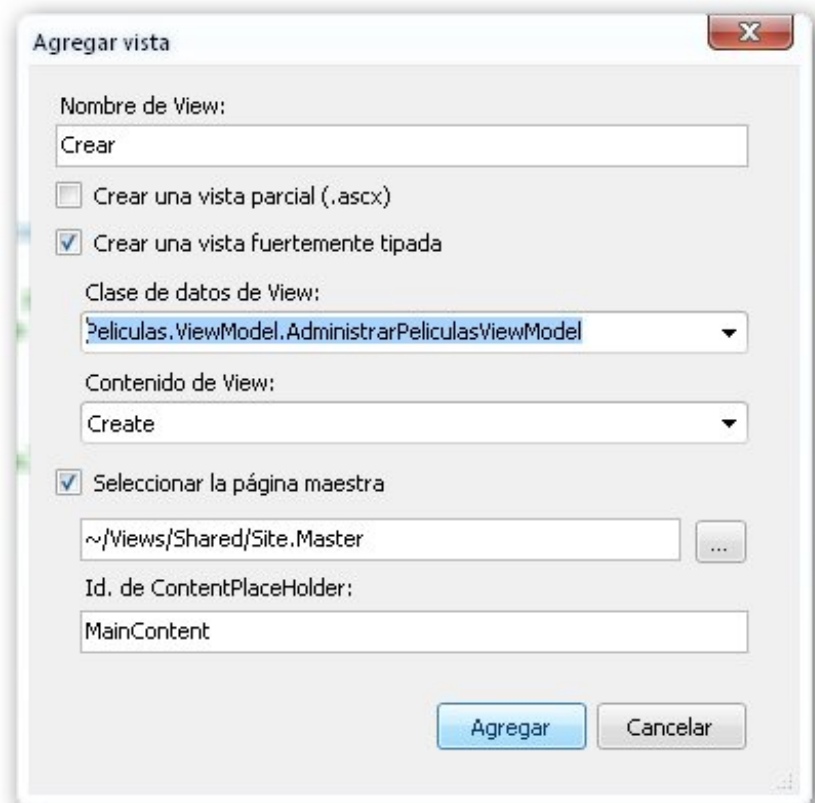
Creamos la vista para la acción Crear() de la clase Peliculas.ViewModel. AdministrarPeliculasViewModel con contenido Create, esto para indicar que el formulario sera para creará un nuevo registro.

Img 10.-Crear vista Crear.asp de la clase AdministrarPeliculasViewModel

Si observas el código que se genera, solo declara el formulario y el botón SUBMIT, si lo ejecutas, no mostrara los cuadros de texto donde se ingresaran los datos.

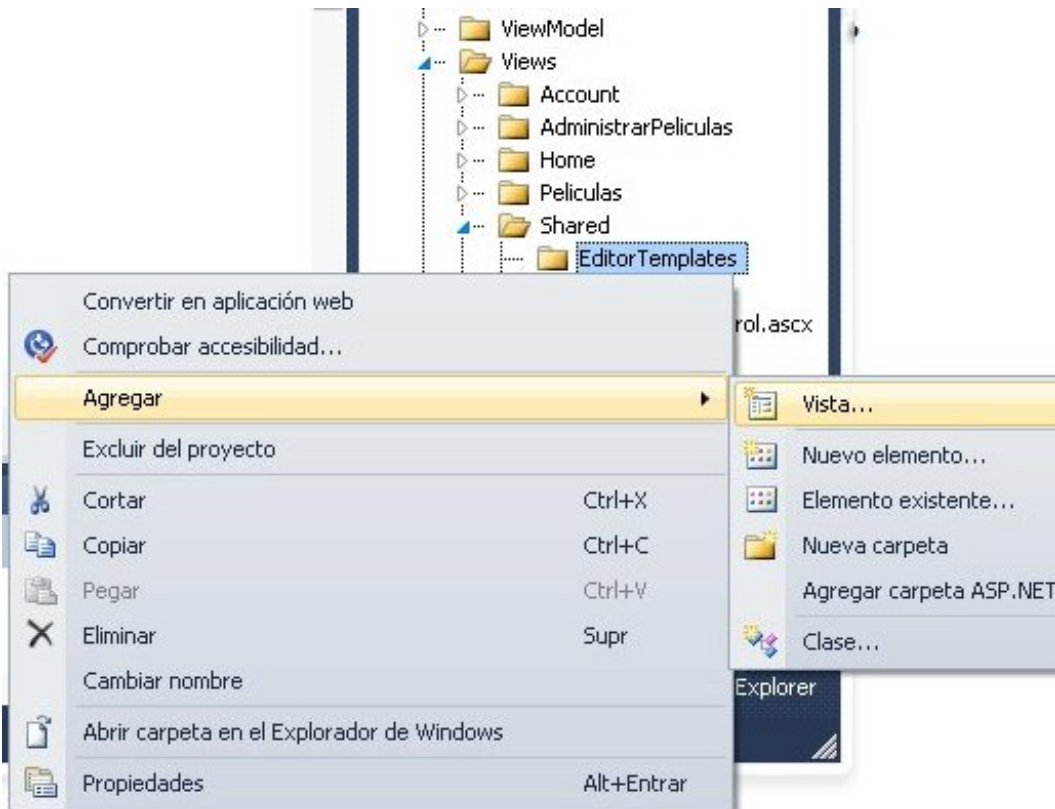
Necesitamos crear nuestra propia plantilla para editar o crear un objeto de la clase Pelicula.

Para ello, creamos una carpeta llamada EditorTemplates en Views/Shared donde se encuentra Site.Master.



Img 11.-Crear carpeta donde guardar nuestra plantilla.

En esa carpeta, creamos una vista llamada Pelicula (Mismo nombre de la entidad).



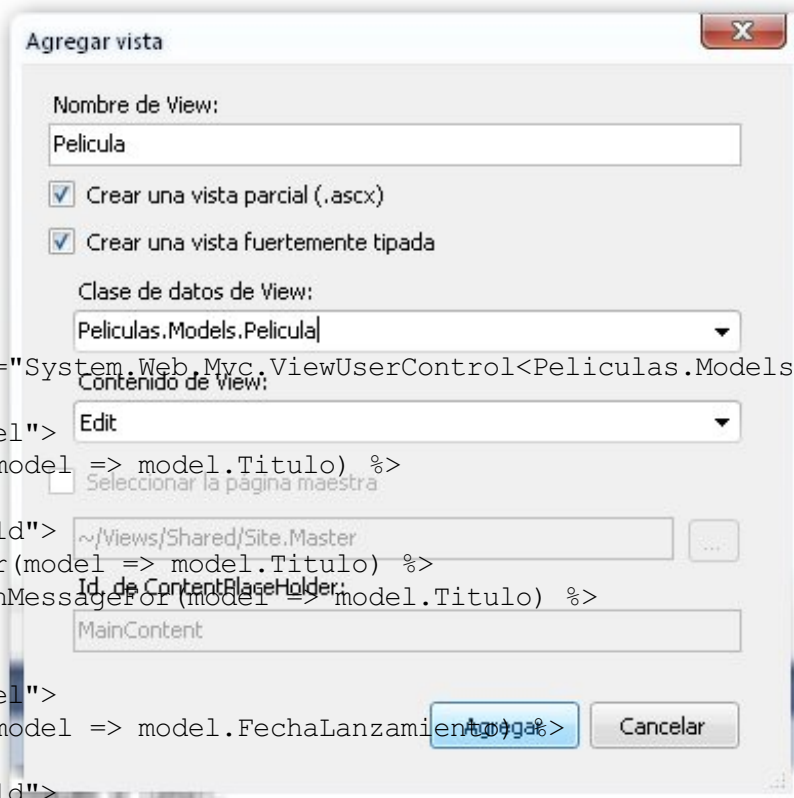
Img 12.-Agregar una vista a la carpeta EditorTemplates.

En crear vista, seleccionamos Crear una vista Parcial (.ascx), clase de datos Peliculas.Models.Pelicula, contenido de view Edit.

Img 13.-Crear vista Pelicula.ascx con contenido Edit.

Al código generado le eliminamos algunas partes ya que solo necesitamos que nos genere los cuadros de textos necesarios, eliminando el Txt para PeliculaId, y el txt Disponible y generamos el ListBox con la lista de géneros, entonces lo dejamos así:

```
<%@ Control Language="C#" Inherits="System.Web.Mvc.ViewUserControl<Peliculas.Models.Pelicula>" ScaffoldingPath="Views/Shared/EditorTemplates" %>
<div class="editor-label">
  <%= Html.LabelFor(model => model.Titulo) %>
</div>
<div class="editor-field">
  <%= Html.TextBoxFor(model => model.Titulo) %>
  <%= Html.ValidationMessageFor(model => model.Titulo) %>
</div>
<div class="editor-label">
  <%= Html.LabelFor(model => model.FechaLanzamiento) %>
</div>
<div class="editor-field">
  <%= Html.TextBoxFor(model => model.FechaLanzamiento, String.Format("{0:dd/MM/yyyy}")) %>
  <%= Html.ValidationMessageFor(model => model.FechaLanzamiento) %>
</div>
```



```

<div class="editor-label">
    <%: Html.LabelFor(model => model.GeneroId) %>
</div>
<div class="editor-field">
    <% //Reemplazamos Html.ValidationMessageFor(model => model.GeneroId) p
    <%: Html.DropDownList("GeneroId", new SelectList(ViewData["Generos"] a
    <%: Html.ValidationMessageFor(model => model.GeneroId) %>
</div>

<div class="editor-label">
    <%: Html.LabelFor(model => model.Precio) %>
</div>
<div class="editor-field">
    <%: Html.TextBoxFor(model => model.Precio, String.Format("{0:F}", Mode
    <%: Html.ValidationMessageFor(model => model.Precio) %>
</div>

```

Entonces el código de la vista Crear.aspx, quedara de esta forma:

```

<asp:Content ID="Content2" ContentPlaceHolderID="MainContent" runat="server">

    <h2>Crear nuevo registro</h2>

    <% using (Html.BeginForm()) {%>
        <%: Html.ValidationSummary(true) %>

        <fieldset>
            <legend>Datos de la pelicula</legend>
            <%//Llamamos al metodo EditFor indicandole por medio del parametro que est
            //un objeto de la entidad Pelicula (de esa clase generamos la vista Pelicu
            //donde tambien le pasamos como parametro la lista de generos que debera n

            <%: Html.EditorFor(model => model.Pelicula, new { Generos = Model.Generos
            <p>
                <input type="submit" value="Crear registro" />
            </p>
        </fieldset>

    <% } %>

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

</asp:Content>

```

Ejecutamos, vamos a Administrar, elegimos Registrar nueva película y obtenemos.

Img 14.-Vista de nuestro nuevo formulario creado...

Hasta aquí tenemos ya nuestro formulario para crear (lo utilizaremos también para editar), pero si intentamos guardar el registro, no lo hará, ya que no hay una acción que tome los datos enviados por el form.

Pero necesitamos validar los datos ingresados por el usuario donde los campos no pueden ser null o validar el tipo de dato ingresado.

The screenshot shows a web browser window with a form titled "Crear nuevo registro". The form is enclosed in a box with a legend "Datos de la pelicula". There are two input fields: "Titulo" and "FechaLanzamiento". The "FechaLanzamiento" field contains the text "01/01/0001 12:00:00 a.m.". Below the "FechaLanzamiento" field, there is a partially visible label "GeneroId".

La validación deberá realizarse cuando el usuario ingrese los datos.

Vamos a crear una parte de la clase Pelicula (partial class Pelicula). En la carpeta Models, Agregamos la clase llamada Pelicula.

Img 15.-Agregar la Clase Pelicula a la carpeta Models

La clase creada quedaría:

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

//Agregamos los espacios de nombres
//Para usar DataAnnotations (Investigar que son los DataAnnotations)

using System.ComponentModel;
using System.ComponentModel.DataAnnotations;
using System.Web.Mvc;

namespace Peliculas.Models
{
    //Indicamos que la validación será realizada por medio de la clase PeliculaMetadata
    [MetadataType(typeof(PeliculaMetadata))]
    public partial class Pelicula
    {
        // clase para la validación de los datos de la entidad Pelicula

        //Como PeliculaId es autoincremento, lo excluimos de la validación
        [Bind(Exclude = "PeliculaId")]
        public class PeliculaMetadata
        {
            //Algo sobre las clases de System.ComponentModel

            //Required - Mostrar mensaje que indique que la propiedad es un campo req
            // DisplayName - Define el texto que queremos mostrar como etiqueta de un
            // StringLength - Define la longitud máxima de la cadena de un campo
            // Range - Especifica el valor Mínimo y Máximo de un campo numérico
            // ScaffoldColumn - Permite ocultar campos en el formulario

            //Entonces podemos realizar la validación de los datos de la entidad Pelic
            //Con el siguiente código

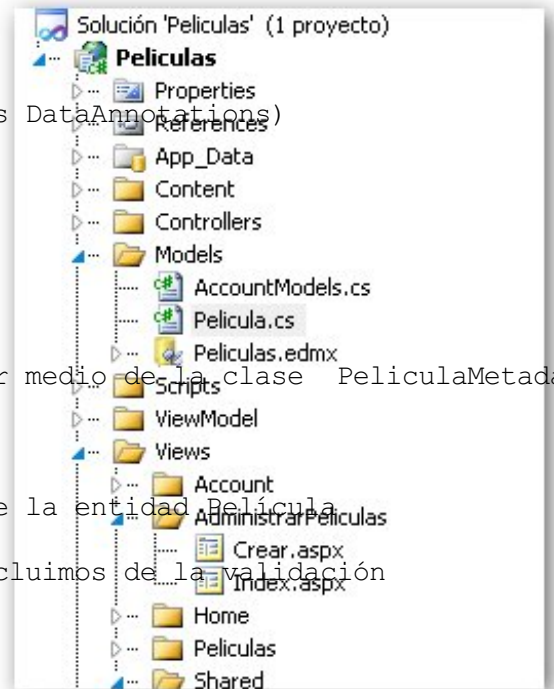
            [ScaffoldColumn(false)]
            public int PeliculaId { get; set; }

            [DisplayName("Genero")]
            public int GeneroId { get; set; }

            [Required(ErrorMessage = "El Titulo de la pelicula es obligatorio")]
            public string Titulo { get; set; }

            [Required(ErrorMessage = "La fecha de lanzamiento es obligatoria")]
            [DataType(DataType.Date, ErrorMessage = "La fecha debe ser dia/mes/año --<
            public DateTime FechaLanzamiento { get; set; }

            [Required(ErrorMessage = "El precio es obligatorio")]
            [Range(50, 300, ErrorMessage = "Las peliculas deben costar entre $50.00 y
            public decimal Precio { get; set; }
        }
    }
}
```



```

    }
}
}

```

Y para que la validación se lleve a cabo en nuestro formulario, agregamos el siguiente código a la vista Pelicula.ascx que se encuentra en Views / Shared / EditorTemplates/Pelicula.ascx.

```
<%@ Import Namespace="Películas"%>
```

```
<%@ Control Language="C#" Inherits="System.Web.Mvc.ViewUserControl<Películas.Models.Pe
```

```
<script src="/Scripts/MicrosoftAjax.js" type="text/javascript"></script>
```

```
<script src="/Scripts/MicrosoftMvcAjax.js" type="text/javascript"></script>
```

```
<script src="/Scripts/MicrosoftMvcValidation.js" type="text/javascript"></script>
```

En la vista Crear.aspx reemplazamos el código que valida el formulario de:

```

<% using (Html.BeginForm()) {%>
    <%: Html.ValidationSummary(true) %>

```

Por el código que habilita la validación por el lado del cliente (navegador)

```

<% Html.EnableClientValidation(); %>
<% using (Html.BeginForm()) {%>

```

Si ejecutamos nuestro proyecto, nos situamos en <http://localhost:7406/AdministrarPelículas/Crear> e ingresamos un título, lo eliminamos y pasamos a otro txt, podremos ver que la validación sí funciona.

Img 16.-Resultado de haber creado la validación por parte del cliente.

Ya que tenemos la validación, crearemos otra función en el controlador que reciba los datos por medio de POST (similar a GET), la acción quedará así (debajo de la acción Crear()):

```

//Acción que recibirá los datos del formulario (datos de nueva Película --> Instanc
[HttpPost]
public ActionResult Crear(Pelicula pelicula)
{
    //Como se agregó la clase que realice la validación, solo recogemos el ok
    //Si se genera el error ExceptionConstraint, agregar un try catch en Movie
    //donde se genera el error, para
    //que pueda retornar los campos obligatorios al usuario

    try
    {
        //poner en TRUE la propiedad Disponible
        //ya que por default pondrá False
        pelicula.Disponible = true;

        //agregar al modelo el objeto película recibido
        DB.AddToPelicula(pelicula);
        //guardar cambios en el modelo (DB)
        DB.SaveChanges();

        //regresar a la página principal de administración
        return Redirect("Index");
    }
    catch
    {
        //si se produce un error, regresará al form informado nuestro mensaje

```

```

//Con ViewData["Message"] = Pasamos datos a la vista.
ViewData["Message"] = "Error: Ingresa los datos de la película";

//Volvemos a crear un objeto de nuestro ViewModel personalizado
var viewModel = new AdministrarPeliculasViewModel
{
    Pelicula = pelicula,
    Generos = DB.Genero.ToList()
};

return View(viewModel);
}
}

```

Ejecutamos el proyecto y creamos un nuevo registro, Listo, ya podemos crear nuevos registros con validación de datos.



Administrar Peliculas

	ID	Titulo	Fecha de Lanzamiento	Genero	Precio	Disp
Editar Detalles Eliminar	1	Identidad Sustituta	25/09/2009	Ficcion	\$ 80.00	Si
Editar Detalles Eliminar	2	Bourne Ultimatum	15/06/2007	Ficcion	\$ 80.00	Si
Editar Detalles Eliminar	3	Dejavu	22/02/2004	Accion	\$ 70.00	Si
Editar Detalles Eliminar	5	El principe de Persia	12/05/2010	Ficcion	\$ 87.00	No
Editar Detalles Eliminar	6	Y tu cuanto cuestas?	01/05/2007	Accion	\$ 90.00	No
Editar Detalles Eliminar	7	El Plan Perfecto	28/08/2006	Accion	\$ 90.00	Si

[Registrar nueva pelicula](#)

Creando mi aplicacion en ASP.NET MVC 2, 2010

Img 17.-Realizando registros en la coleccion.

Solo nos falta la acción de Editar y eliminar, dicha acción utiliza el mismo formulario y la validación creados.

Implementamos la acción Editar junto con la acción que recibirá los datos por medio de POST similar a la accion Crear.

```

//
//AdministrarPeliculas/Editar/#
public ActionResult Editar(int id)
{
    //Creamos un objeto de nuestro nuevo modelo
    var vistaModelo = new AdministrarPeliculasViewModel
    {
        //Recuperamos el objeto Pelicula de PeliculasEntities
        //y lo asignamos a la propiedad de nuestro nuevo modelo
        Pelicula = DB.Pelicula.Single(a => a.PeliculaId == id),
        Generos = DB.Genero.ToList()
    };

    //Devolvemos a la vista nuestro modelo personalizado

```

```

        return View(vistaModelo);
    }

    //POST
    //Recibe PeliculaId y los valores del formulario
    [HttpPost]
    public ActionResult Editar(int id, FormCollection formValues)
    {
        //Recupera el objeto de PeliculasEntities
        var pelicula = DB.Pelicula.Single(p => p.PeliculaId == id);
        try
        {
            //Actualiza los datos del objeto recuperado por los valores del form
            UpdateModel(pelicula, "Pelicula");
            //Guardar los cambios
            DB.SaveChanges();
            //regresar al index
            return RedirectToAction("Index");
        }
        catch
        {
            //si se produce un error, regresara nuevamente el registro para editar
            var vistaModelo = new AdministrarPeliculasViewModel
            {
                Pelicula = DB.Pelicula.Single(a => a.PeliculaId == id),
                Generos = DB.Genero.ToList()
            };

            return View(vistaModelo);
        }
    }
}

```

Img 18.- Crear la vista Editar.

La cual su código será similar al código de la vista Crear, ya que usa el mismo formulario Pelicula.ascx

```

<%@ Page Title="" Language="C#" MasterPageFile="~/Views/Shared/Site.Master" Inherits="
<asp:Content ID="Content1" ContentPlaceHolderID="TitleContent" runat="server">
    Editar pelicula <%= Model.Pelicula.Titulo %>
</asp:Content>

<asp:Content ID="Content2" ContentPlaceHolderID="MainContent" runat="server">

    <h2>Editar: <%= Model.Pelicula.Titulo %></h2>

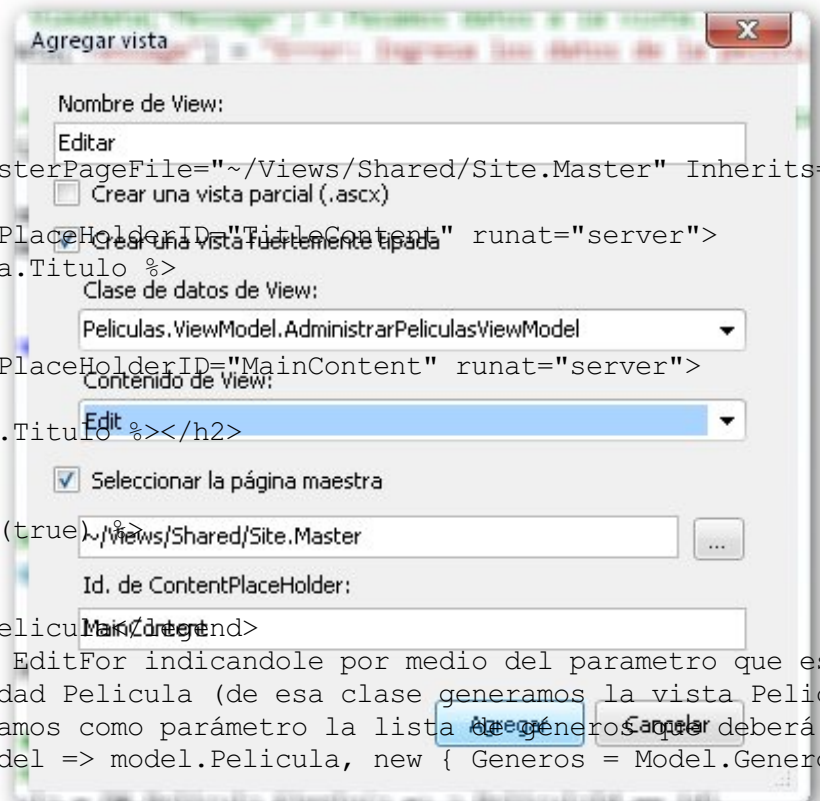
    <% using (Html.BeginForm()) {%>
        <%= Html.ValidationSummary(true) %>

        <fieldset>
            <legend>Datos de la pelicula</legend>
            <%=//Llamamos al metodo EditFor indicandole por medio del parametro que est
            //un objeto de la entidad Pelicula (de esa clase generamos la vista Pelicu
            //donde también le pasamos como parámetro la lista de generos que deberá n
            <%= Html.EditorFor(model => model.Pelicula, new { Generos = Model.Generos

        <p>
            <input type="submit" value="Guardar cambios" />
        </p>
        </fieldset>

    <% } %>

```



```
<div>
    <%: Html.ActionLink("Regresar al índice", "Index") %>
</div>
```

```
</asp:Content>
```

Si ejecutamos, vamos a Administrar y elegimos editar algún registro, ya debe funcionar...

Img 19.- Editando un registro.

Pues bien, solo nos falta la acción Eliminar()

```
//
// GET: /AdministrarPelículas/Eliminar/#
public ActionResult Eliminar(int id)
{
    //Recuperamos el objeto a eliminar
    var pelicula = DB.Pelicula.Single(p => p.PeliculaId == id);
    //lo devolvemos a la vista para la confirmación del usuario
    return View(pelicula);
}

//HTTP Post Eliminar
//Esta acción recibe PeliculaId y la confirmación del usuario
[HttpPost]
public ActionResult Eliminar(int id, string confirmar)
{
    //Recupera el objeto del modelo
    var pelicula = DB.Pelicula.Single(p => p.PeliculaId == id);
    //Eliminamos el objeto del Modelo
    DB.DeleteObject(pelicula);
    //Guardamos los cambios del Modelo
    DB.SaveChanges();

    //Aqui si queremos informar al usuario que se ha eliminado el registro
    //creamos otra acción Eliminado para redirigir al usuario
    return View("Eliminado");
}
```

Creamos la vista Eliminar.aspx de la clase Películas.Models.Pelicula con contenido Delete.

Img 20.- Crear la vista Eliminar.

La vista eliminar la podemos dejar con todos los campos que nos muestra o dejarla de esta forma...

```
<%@ Page Title="" Language="C#" MasterPageFile="~/Views/Shared/Site.Master" Inherits="
<asp:Content ID="Content1" ContentPlaceHolderID="TitleContent" runat="server">
    Eliminar <%: Model.Titulo %>
</asp:Content>

<asp:Content ID="Content2" ContentPlaceHolderID="MainContent" runat="server">

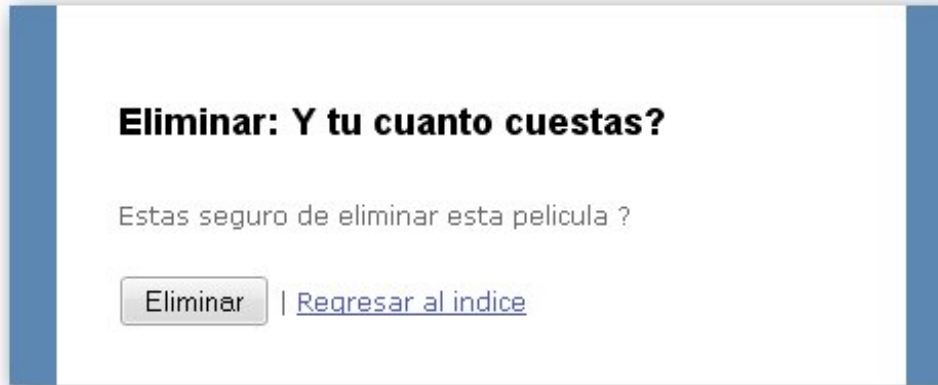
    <h2>Eliminar: <%: Model.Titulo %></h2>

    <% using (Html.BeginForm()) { %>
        <p>
            <input type="submit" value="Eliminar" /> |
            <%: Html.ActionLink("Regresar al índice", "Index") %>
        </p>
```

```
<% } %>
```

```
</asp:Content>
```

Si ejecutamos el proyecto, Administrar y elegimos eliminar algún registro.



Img 21.- Resultado al intentar eliminar un registro.

Como en nuestra acción `HttpPost Eliminar()`, redirigimos al usuario a la acción `Eliminado()`, debemos crear la acción y crear la vista `Eliminado.aspx`.

```
//Esta accion solo devolvera la vista que informa al usuario
public ActionResult Eliminado()
{
    return View();
}
```

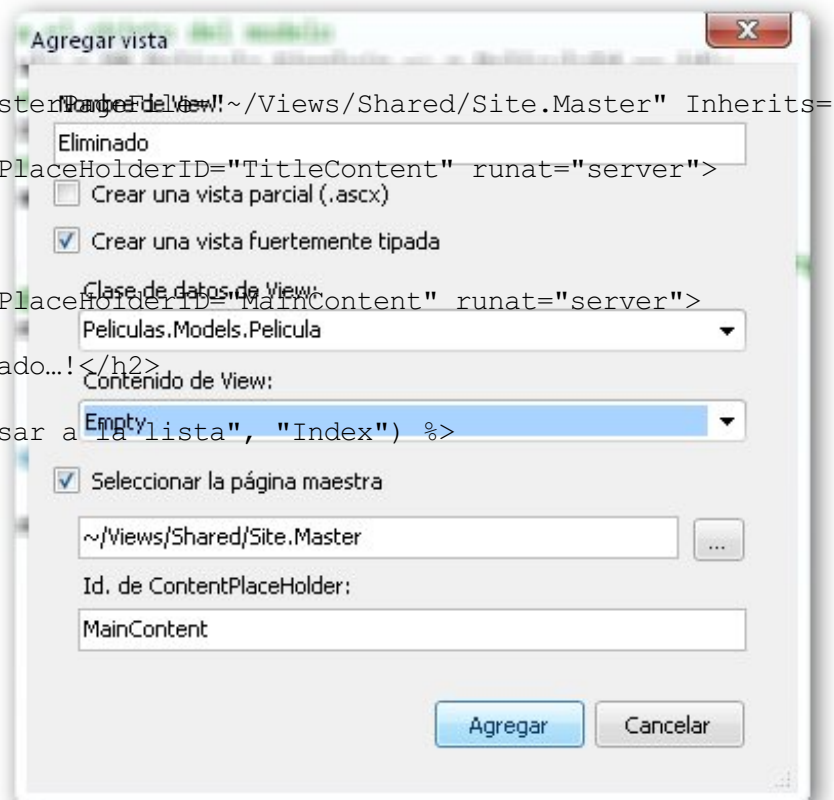
La vista será de la clase `Peliculas.Models.Pelicula` con contenido `Empty` (vacío).

Img 22.- Crear la vista `Eliminado`.

A la que agregaremos el código:

```
<%@ Page Title="" Language="C#" MasterPageFile="~/Views/Shared/Site.Master" Inherits="
<asp:Content ID="Content1" ContentPlaceHolderID="TitleContent" runat="server">
    Registro eliminado.
</asp:Content>
<asp:Content ID="Content2" ContentPlaceHolderID="MainContent" runat="server">
    <h2>El Registro ha sido eliminado...!</h2>
    <div>
        <%: Html.ActionLink("Regresar a la lista", "Index") %>
    </div>
</asp:Content>
```

Entonces ahora si elegimos eliminar un registro y confirmamos la eliminación, el resultado será:



El Registro ha sido eliminado...!

[Regresar a la lista](#)

Creando mi aplicacion en ASP.NET MVC 2, 2010

Img 23.-Registro eliminado.

Pues bien, hasta aquí ha sido todo lo del controlador AdministrarPeliculas

Posteriormente agregare el extra que es crear la lista de CheckBoxes para modificar el estado Disponible, generar los reportes en PDF y al final, el panel de administración de usuarios...

Mi primera aplicación ASP.NET MVC 2 paso a paso – parte 4, 5.0 out of 5 based on 1 rating