Desarrollo de Software Empresarial

CICLO 02 -2024

Guía 2: Introducción a ASP.NET MVC - Parte #3

Competencias

- Que el estudiante aprenda a configurar un archivo semilla para inicializar la base de datos. Que el estudiante aprenda a adicionar método de búsqueda.
- Que el estudiante aprenda a agregar reglas de validación al modelo.

Introducción Teórica

Data Seeding

Data Seeding es la práctica de insertar datos iniciales en una base de datos al momento de iniciar una aplicación. En ASP.NET Core MVC 8, esto se puede lograr de varias maneras, siendo una de las más comunes el uso de Entity Framework Core.

Pasos para realizar Data Seeding con Entity Framework Core:

1. *Definir el Modelo:* Define las entidades que representarán tus datos en la base de datos.
2. *Configurar el Contexto:* Configura el contexto de la base de datos y sobrescribe el método OnModelCreating para especificar los datos iniciales.
3. *Migraciones:* Usa migraciones para aplicar los cambios en la base de datos.

Ventajas de la Siembra de Datos:

- *Pre-carga de datos necesarios:* Útil para datos de referencia que son constantes.
- *Pruebas consistentes:* Facilita la creación de entornos de pruebas con datos consistentes.
- *Desarrollo acelerado:* Ayuda a desarrolladores a tener datos con los cuales trabajar desde el inicio.

Atributos de Validación

Los atributos de validación en ASP.NET Core MVC 8 permiten definir reglas para validar las propiedades de un modelo. Esto asegura que los datos que ingresan a la aplicación cumplen con ciertos requisitos antes de ser procesados.

Ejemplos de Atributos de Validación:

Required: Indica que una propiedad debe tener un valor.

```
public class User
{
    [Required(ErrorMessage = "Nombre es requerido")]
    0 referencias
    public string Name { get; set; }
}
```

MinimumLength: Establece la longitud mínima de una propiedad de cadena.

```
0 referencias
public class User
{
    [MinLength(5, ErrorMessage = "El nombre debe tener al menos 5 caracteres de longitud")]
    0 referencias
    public string Name { get; set; }
}
```

RegularExpression: Restringe los caracteres que se pueden ingresar mediante una expresión regular.

```
0 referencias
public class User
{
    [RegularExpression(@"^[a-zA-Z''-'\s]{1,40}$", ErrorMessage = "Caracteres no válidos")]
    0 referencias
    public string Name { get; set; }
}
```

Range: Restringe el valor de una propiedad a un rango específico.

```
0 referencias
public class Product
{
    [Range(1, 100, ErrorMessage = "El precio debe encontrarse entre 1 y 100")]
    0 referencias
    public decimal Price { get; set; }
}
```

StringLength: Define la longitud máxima y opcionalmente la mínima de una cadena.

```
0 referencias
public class User
{
    [StringLength(50, MinimumLength = 5, ErrorMessage = "El nombre debe tener entre 5 y 50 caracteres")]
    0 referencias
    public string Name { get; set; }
}
```

Materiales y Equipos

1. Guía No 3.
2. Computadora con programa Microsoft Visual Studio 2022.

Procedimiento

Data Seeding

Para esta práctica, utilizararemos el proyecto que hemos venido trabajando en la práctica #1 y #2.

1. Dentro de la carpeta **Models**, cree una nueva carpata llamada **Seeds**.

2. Cree una nueva clase llamada **GeneroSeed**, la cual debe exteneder de la interfaz **IEntityTypeConfiguration** y agregue lo siguiente:

```
using Microsoft.EntityFrameworkCore;
using Microsoft.EntityFrameworkCore.Metadata.Builders;

namespace MVCPelicula.Models.Seeds
{
    0 referencias
    public class GeneroSeed: IEntityTypeConfiguration<Genero>
    {
        0 referencias
        public void Configure(EntityTypeBuilder<Genero> builder)
        {
            builder.HasData(
                new Genero { Id = 1, Nombre = "Comedia" },
                new Genero { Id = 2, Nombre = "Romance" },
                new Genero { Id = 3, Nombre = "Aventura" },
                new Genero { Id = 4, Nombre = "Acción" },
                new Genero { Id = 5, Nombre = "Ficción" },
                new Genero { Id = 6, Nombre = "Drama" }
            );
        }
    }
}
```

3. Cree una nueva clase llamada **PeliculaSeed** y agregue lo siguiente:

```
using Microsoft.EntityFrameworkCore;
using Microsoft.EntityFrameworkCore.Metadata.Builders;

namespace MVCPelicula.Models.Seeds
{
    1 referencia
    public class PeliculaSeed : IEntityTypeConfiguration<Pelicula>
    {
        0 referencias
        public void Configure(EntityTypeBuilder<Pelicula> builder)
        {
            builder.HasData(
                new Pelicula {
                    ID = 1,
                    Titulo = "Matrix recargado",
                    FechaLanzamiento = DateTime.Parse("2003-11-13"),
                    GeneroId = 4,
                    Precio = 10.99M,
                    Director = "Hermanas Wachowski"
                },
                new Pelicula
                {
                    ID = 2,
                    Titulo = "El señor de los anillos: Las dos torres",
                    FechaLanzamiento = DateTime.Parse("2002-12-18"),
                    GeneroId = 3,
                    Precio = 11.99M,
                    Director = "Peter Jackson"
                }, new Pelicula
                {
                    ID = 3,
                    Titulo = "Harry Potter y la cámara secreta",
                    FechaLanzamiento = DateTime.Parse("2002-11-15"),
                    GeneroId = 5,
                    Precio = 9.99M,
                    Director = "Chris Columbus"
                }
            );
        }
    }
}
```

4. Realice la siguientes modificación en la clase **PeliculasDBContext**

```
using Microsoft.EntityFrameworkCore;
using MVCPelicula.Models.Seeds;

namespace MVCPelicula.Models
{
    4 referencias
    public class PeliculasDBContext: DbContext
    {
        0 referencias
        public PeliculasDBContext(DbContextOptions options) : base(options)
        {
        }

        7 referencias
        public DbSet<Pelicula> Peliculas { get; set; }
        4 referencias
        public DbSet<Genero> Generos { get; set; }

        0 referencias
        protected override void OnModelCreating(ModelBuilder modelBuilder)
        {
            modelBuilder.ApplyConfiguration(new GeneroSeed());
            modelBuilder.ApplyConfiguration(new PeliculaSeed());
        }
    }
}
```

5. Antes de continuar, elimine la base de datos **Peliculas** y tambien la carpeta **Migrations**.

6. Abra la consola del administrador de paquete NuGet y escribir el siguiente comando **Add-Migration "Migracion_Inicial".**

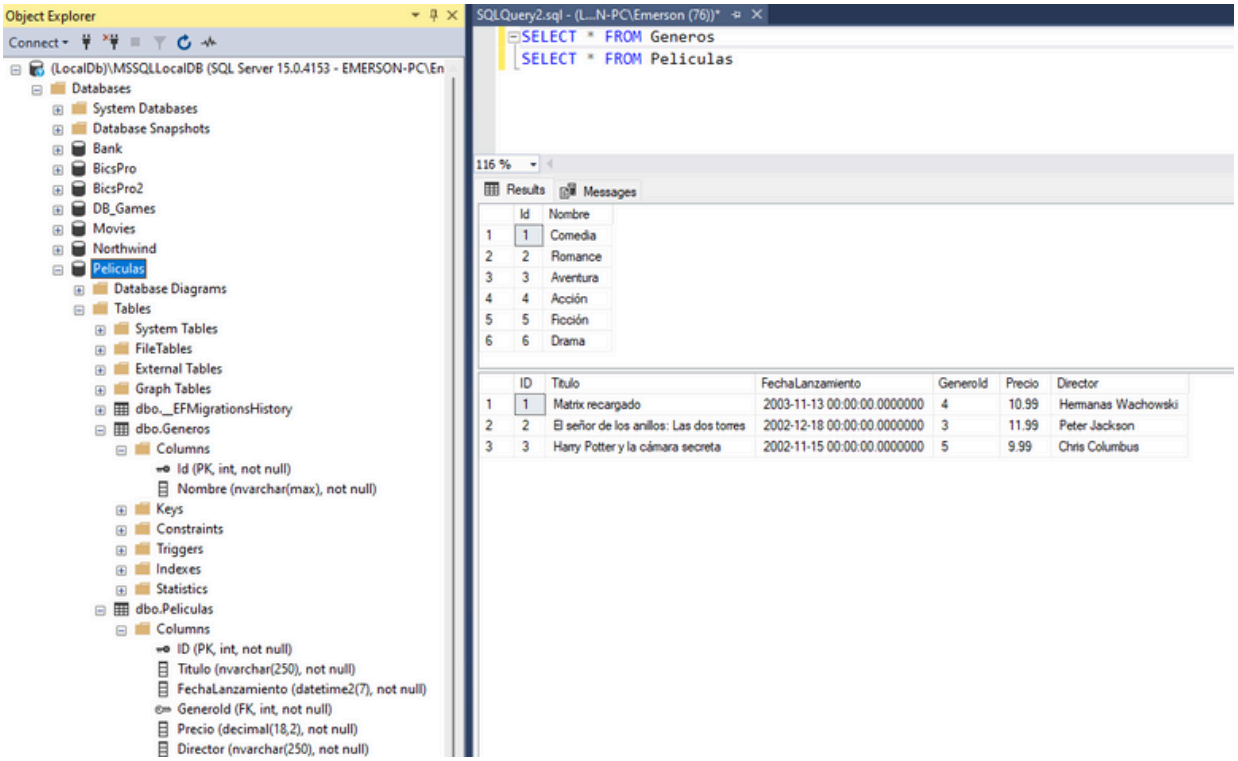
```
PM> Add-Migration "Migracion_Inicial"
Build started...
Build succeeded.
Microsoft.EntityFrameworkCore.Model.Validation[30000]
      No store type was specified for the decimal property 'Precio' on entity type 'Pelicula'. This will cause values to be silently
 truncated if they do not fit in the default precision and scale. Explicitly specify the SQL server column type that can accommodate
 all the values in 'OnModelCreating' using 'HasColumnType', specify precision and scale using 'HasPrecision', or configure a value
converter using 'HasConversion'.
Microsoft.EntityFrameworkCore.Model.Validation[30000]
      No store type was specified for the decimal property 'Precio' on entity type 'Pelicula'. This will cause values to be silently
 truncated if they do not fit in the default precision and scale. Explicitly specify the SQL server column type that can accommodate
 all the values in 'OnModelCreating' using 'HasColumnType', specify precision and scale using 'HasPrecision', or configure a value
converter using 'HasConversion'.
No store type was specified for the decimal property 'Precio' on entity type 'Pelicula'. This will cause values to be silently
truncated if they do not fit in the default precision and scale. Explicitly specify the SQL server column type that can accommodate
all the values in 'OnModelCreating' using 'HasColumnType', specify precision and scale using 'HasPrecision', or configure a value
converter using 'HasConversion'.
To undo this action, use Remove-Migration.
```

7. Abra el archivo de migración generado. Como podrá observar a parte de generar la estructura de tablas de la base de datos, tambien se han agregado las consultas para agregar registros a la base de datos.

```
migrationBuilder.InsertData(
    table: "Generos",
    columns: new[] { "Id", "Nombre" },
    values: new object[,]
    {
        { 1, "Comedia" },
        { 2, "Romance" },
        { 3, "Aventura" },
        { 4, "Acción" },
        { 5, "Ficción" },
        { 6, "Drama" }
    });

migrationBuilder.InsertData(
    table: "Peliculas",
    columns: new[] { "ID", "Director", "FechaLanzamiento", "GeneroId", "Precio", "Titulo" },
    values: new object[,]
    {
        { 1, "Hermanas Wachowski", new DateTime(2003, 11, 13, 0, 0, 0, 0, DateTimeKind.Unspecified), 4, 10.99m, "Matrix recargado" },
        { 2, "Peter Jackson", new DateTime(2002, 12, 18, 0, 0, 0, 0, DateTimeKind.Unspecified), 3, 11.99m, "El señor de los anillos: Las dos torres" },
        { 3, "Chris Columbus", new DateTime(2002, 11, 15, 0, 0, 0, 0, DateTimeKind.Unspecified), 5, 9.99m, "Harry Potter y la cámara secreta" }
    });

migrationBuilder.CreateIndex(
    name: "IX_Peliculas_GeneroId",
    table: "Peliculas",
    column: "GeneroId");
```

8. Utilice el comando **Update-Database**. Ahora verifique si se ha creado la base de datos **Peliculas** con las tablas de **Generos** y **Peliculas** con sus respectivos registros iniciales.

```
PM> Update-Database
Build started...
Build succeeded.
Microsoft.EntityFrameworkCore.Model.Validation[30000]
      No store type was specified for the decimal property 'Precio' on entity type 'Pelicula'. This will cause values to be silently
 truncated if they do not fit in the default precision and scale. Explicitly specify the SQL server column type that can accommodate
 all the values in 'OnModelCreating' using 'HasColumnType', specify precision and scale using 'HasPrecision', or configure a value
converter using 'HasConversion'.
Microsoft.EntityFrameworkCore.Model.Validation[30000]
      No store type was specified for the decimal property 'Precio' on entity type 'Pelicula'. This will cause values to be silently
 truncated if they do not fit in the default precision and scale. Explicitly specify the SQL server column type that can accommodate
 all the values in 'OnModelCreating' using 'HasColumnType', specify precision and scale using 'HasPrecision', or configure a value
converter using 'HasConversion'.
No store type was specified for the decimal property 'Precio' on entity type 'Pelicula'. This will cause values to be silently
truncated if they do not fit in the default precision and scale. Explicitly specify the SQL server column type that can accommodate
all the values in 'OnModelCreating' using 'HasColumnType', specify precision and scale using 'HasPrecision', or configure a value
converter using 'HasConversion'.
Microsoft.EntityFrameworkCore.Database.Command[20101]
      Executed DbCommand (195ms) [Parameters=[], CommandType='Text', CommandTimeout='60']
      CREATE DATABASE [Peliculas];
```



9. Ejecute la aplicación y verifique que la información se muestra correctamente.

10. Como podemos ver, el la vista Index de peliculas, nos aparece el Id de Genero, pero esta información es irrelevante para el usuario, por lo que realizaremos las siguientes modificaciones para mostrar el nombre en vez del ID. Vaya al **archivo Index.cshtml** el cual esta dentron de la carpeta **Views/Peliculas** y realice las siguientes modificaciones.

```
@model IEnumerable<MVCPelicula.Models.Pelicula>
@{
    ViewData["Title"] = "Index";
}
<h1>Index</h1>
<p>
    <a asp-action="Create">Create New</a>
</p>
<table class="table">
    <thead>
        <tr>
            <th>@Html.DisplayNameFor(model => model.Titulo)</th>
            <th>@Html.DisplayNameFor(model => model.FechaLanzamiento)</th>
            <th>@Html.DisplayNameFor(model => model.Genero)</th>
            <th>@Html.DisplayNameFor(model => model.Precio)</th>
            <th>@Html.DisplayNameFor(model => model.Director)</th>
            <th></th>
        </tr>
    </thead>
    <tbody>
@foreach (var item in Model) {
        <tr>
            <td>@Html.DisplayFor(modelItem => item.Titulo)</td>
            <td>@Html.DisplayFor(modelItem => item.FechaLanzamiento)</td>
            <td>@Html.DisplayFor(modelItem => item.Genero.Nombre)</td>
            <td>@Html.DisplayFor(modelItem => item.Precio)</td>
            <td>@Html.DisplayFor(modelItem => item.Director)</td>
            <td>
                <a asp-action="Edit" asp-route-id="@item.ID">Edit</a> |
                <a asp-action="Details" asp-route-id="@item.ID">Details</a> |
                <a asp-action="Delete" asp-route-id="@item.ID">Delete</a>
            </td>
        </tr>
}
    </tbody>
</table>
```

11. Ejecute de nuevo la aplicación y observe los cambios.

Modificación de las vistas

La aplicación está casi lista, pero hay algunos textos que no son ideales, Por ejemplo, **FechaDeLanzamiento** debería escribirse en tres palabras, y algunas palabras les faltan tildes.

1. Abra el archivo **Models/Pelicula.cs** y agregue las líneas resaltadas que se muestran a continuación:

```
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;

namespace MVCPelicula.Models
{
    24 referencias
    public class Pelicula
    {
        14 referencias
        public int ID { get; set; }

        [StringLength(250)]
        [Display(Name = "Título")]
        16 referencias
        public string Titulo { get; set; }

        [Display(Name = "Fecha de Lanzamiento")]
        [DataType(DataType.Date)]
        13 referencias
        public DateTime FechaLanzamiento { get; set; }

        //Propiedad para la llave foránea
        [ForeignKey("Genero")]
        [Display(Name = "Género")]
        11 referencias
        public int? GeneroId { get; set; }

        //Propiedad de navegación
        8 referencias
        public Genero Genero { get; set; }

        [Column(TypeName = "decimal(18,2)")]
        15 referencias
        public decimal Precio { get; set; }

        [StringLength(250)]
        15 referencias
        public string Director { get; set; }

    }
}
```

El atributo Display especifica qué se muestra como nombre de un campo (en este caso, "Fecha de lanzamiento" en lugar de "FechaLanzamiento"). El atributo DataType especifica el tipo de los datos (Date), así que la información de hora almacenada en el campo no se muestra.

La anotación de datos [Column(TypeName = "decimal(18, 2)")] es necesaria para que Entity Framework Core asigne correctamente al campo precio a la moneda en la base de datos.

2. Los vínculos Edit (Editar), Details (Detalles) y Delete (Eliminar) se generan mediante el asistente de etiquetas de delimitador de MVC Core en el archivo Views/Peliculas/Index.cshtml. Entre al archivo y traducirlos al español:

```
<td>
    <a asp-action="Edit" asp-route-id="@item.ID">Editar</a> |
    <a asp-action="Details" asp-route-id="@item.ID">Detalles</a> |
    <a asp-action="Delete" asp-route-id="@item.ID">Eliminar</a>
</td>
```

3. Traslade al español el texto de las etiquetas y mensajes del resto de vistas. Luego ejecute la aplicación y compruebe los cambios realizados.

Método de búsqueda

Vamos a agregar la capacidad de búsqueda en Index que permitirá buscar películas por género o nombre.

1. Busque el método Index, que está en **Controllers/PeliculasController.cs**, y agregue el siguiente código al método:

```
3 referencias
public async Task<IActionResult> Index(string textoABuscar)
{
    if(_context.Peliculas == null)
    {
        return Problem("No se ha inicializado el contexto");
    }

    var peliculas = from p in _context.Peliculas
                    select p;

    if (!String.IsNullOrEmpty(textoABuscar))
    {
        peliculas = peliculas.Where(p => p.Titulo.Contains(textoABuscar));
    }

    return View(await peliculas.ToListAsync());
}
```

En la línea resaltada en verde en la imagen del método se crea una consulta **LINQ** para seleccionar las películas, en este momento, solo se define la consulta y no se ejecuta en la base de datos.

Mientras que en la línea resaltada en rojo se verifica si el parámetro **textoABuscar** contiene una cadena, la consulta de películas se modifica para filtrar según el valor de la cadena de búsqueda.

El código **s => s.Titulo!.Contains(textoABuscar)** es una expresión Lambda. Las lambdas se usan en consultas LINQ basadas en métodos como argumentos para métodos de operador de consulta estándar, como el método Where o Contains (que se usa en el código anterior). Las consultas LINQ no se ejecutan cuando se definen ni cuando se modifican mediante una llamada a un método como Where, Contains u OrderBy. En su lugar, se aplaza la ejecución de la consulta. Esto significa que la evaluación de una expresión se aplaza hasta que su valor realizado se repita realmente o se llame al método **ToListAsync**.

El método **Contains** se ejecuta en la base de datos, no en el código de c# que se muestra arriba. La distinción entre mayúsculas y minúsculas en la consulta depende de la base de datos y la intercalación. En SQL Server, **Contains** se asigna a **SQL LIKE**, que distingue entre mayúsculas y minúsculas.

2. Abra el archivo **Views/Peliculas/Index.cshtml** y agregue el marcado resaltado a continuación:

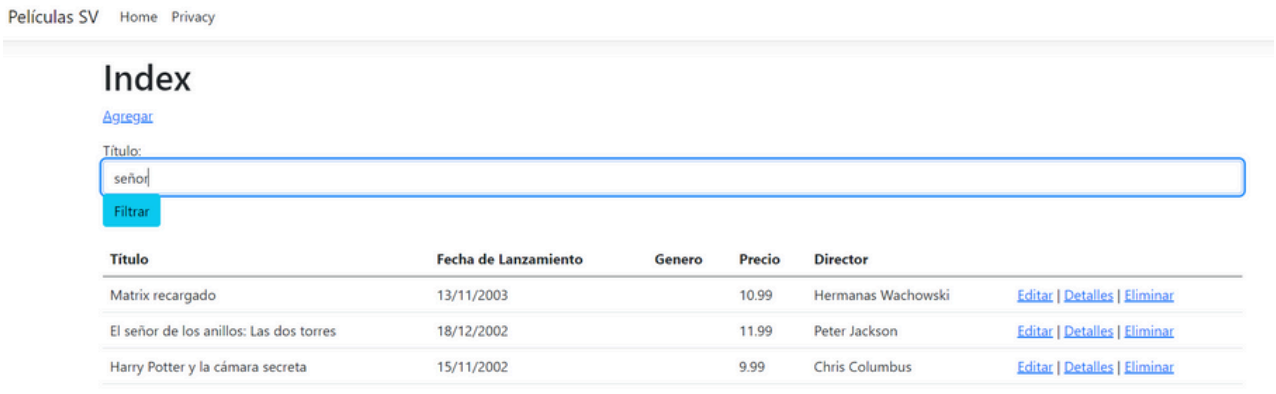
```
@model IEnumerable<MVCPelicula.Models.Pelicula>
@{
    ViewData["Title"] = "Index";
}
<h1>Index</h1>
<p>
    <a asp-action="Create">Agregar</a>
</p>
<form asp-controller="Peliculas" asp-action="Index">
    <p>
        Título: <input type="text" name="textoABuscar" class="form-control">
        <input type="submit" value="Filtrar" class="btn btn-info" />
    </p>
</form>
<table class="table">
```

3. Adicionalmente agregue en el archivo **Controllers/PeliculasController.cs** el método [HttpPost] Index con el código siguiente:

```
[HttpPost]
3 referencias
public string Index(string textoABuscar, bool notUsed)
{
    return "From [HttpPost]Index: filter on " + textoABuscar;
}
```

El parámetro **notUsed** se usa para crear una sobrecarga para el método Index. Al agregar este método, el invocador de acción coincidiría con el método [HttpPost] Index, mientras que este nuevo método [HttpPost] Index se ejecuta en respuesta al botón Filtrar.

Ahora ejecute la aplicación y utilice el botón filtrar para hacer una búsqueda por Título, notará que el resultado es el siguiente:




```
From [HttpPost]Index: filter on señor
```

El parámetro de búsqueda se encuentra en el cuerpo de solicitud y no en la dirección **URL**. Por eso no se puede capturar dicha información para marcarla o compartirla con otros usuarios. Para corregir este problema, se debe especificar que la solicitud debe ser **HTTP GET**, modifique la etiqueta que anteriormente agregó al archivo archivo **Views/Peliculas/Index.cshtml** y agregue el atributo method con el valor **GET**.

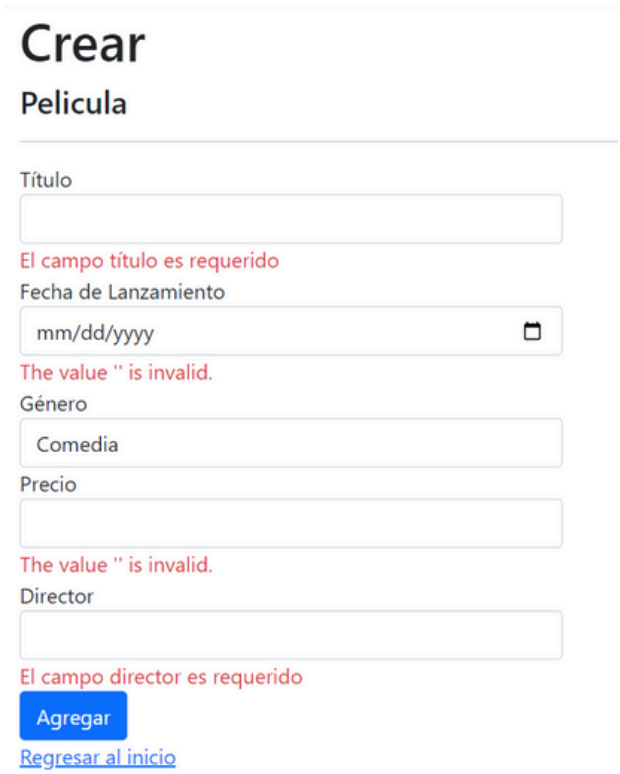
Ejecute nuevamente la aplicación y pruebe filtrar por título, verificará que cuando se envía una búsqueda, la URL contiene la cadena de consulta de búsqueda. La búsqueda también será dirigida al método de acción HttpGet Index, aunque tenga un método HttpPost Index.

Validación del modelo

Actualice la clase Pelicula para agregar atributos de validación integrados como: Required, StringLength, RegularExpression y Range y el atributo de formato DataType.

```
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;

namespace MVCPelicula.Models
{
    24 referencias
    public class Pelicula
    {
        14 referencias
        public int ID { get; set; }

        [StringLength(60, MinimumLength =3)]
        [Required(ErrorMessage = "El campo título es requerido")]
        [Display(Name = "Título")]
        16 referencias
        public string Titulo { get; set; }

        [Display(Name = "Fecha de Lanzamiento")]
        [DataType(DataType.Date)]
        [Required(ErrorMessage = "El campo fecha de lanzamiento es requerido")]
        15 referencias
        public DateTime FechaLanzamiento { get; set; }

        //Propiedad para la llave foránea
        [Required]
        [ForeignKey("Genero")]
        [Display(Name = "Género")]
        11 referencias
        public int? GeneroId { get; set; }

        //Propiedad de navegación
        8 referencias
        public Genero Genero { get; set; }

        [Range(1, 100)]
        [DataType(DataType.Currency)]
        [Column(TypeName = "decimal(18,2)")]
        15 referencias
        public decimal Precio { get; set; }

        [StringLength(250)]
        [Required(ErrorMessage = "El campo director es requerido")]
        15 referencias
        public string Director { get; set; }

    }
}
```

Ejecute la aplicación e intente agregar una nueva película ingresando datos incorrectos, la validación del lado cliente de jQuery detecta el problema, muestra un mensaje de error.



Observe cómo el formulario presenta automáticamente un mensaje de error de validación adecuado en cada campo que contiene un valor no válido. Los errores se aplican en el lado cliente (con JavaScript y jQuery) y en el lado servidor (cuando un usuario tiene JavaScript deshabilitado).

Una ventaja importante es que no fue necesario cambiar ni una sola línea de código en la clase **PeliculasController** o en la vista **Create.cshtml** para habilitar esta interfaz de usuario de validación. El controlador y las vistas que creó en pasos las guías anteriores seleccionaron automáticamente las reglas de validación que se especificaron mediante atributos de validación en las propiedades de la clase del modelo Pelicula. Pruebe la aplicación mediante el método de acción Editar y se aplicará la misma validación.

Los datos del formulario no se enviarán al servidor hasta que dejen de producirse errores de validación de cliente.

Desarrollo de Habilidades

Ejercicio 1

Agregue métodos de búsqueda y validaciones al ejercicio 2 de la guía 2 (proyecto para gestionar empleados, proyectos y las asignaciones de proyectos a empleados.

Bibliografía

BRADFORD, M.; VALVERDE, R. & TALLA, M. (2010). Modern ERP: Select, Implement & Use Todays Advanced Business Systems. North Carolina, State University: Universidad de Carolina del Norte.

VALVERDE, R. (2012). Information Systems Reengineering for Modern Business Systems. IGI Global, USA.

SHIELDS, M. (2001). E-Business and ERP Rapid Implementation and Project.