	<p align="center">UNIVERSIDAD DON BOSCO FACULTAD DE INGENIERIA ESCUELA DE COMPUTACIÓN</p>
<p align="center">Ciclo II</p>	<p>PRACTICA DE LABORATORIO No.4 Nombre de la practica: "Introducción a ASP.NET MVC 5 Parte 3" Lugar de ejecución: Centro de cómputo Tiempo estimado: 2 horas Materia: Desarrollo de software empresarial</p>

Objetivo General

Adquirir conocimientos, habilidades y destrezas en la creación de aplicaciones web básicas utilizando ASP.NET MVC 5 con Visual Studio 2017 y utilizando la base de datos localDB de SQLServer proporcionada por el IDE.

Objetivos Específicos

- Que el estudiante aprenda a configurar migraciones de Code First para cambios de modelo.
- Que el estudiante aprenda a agregar validaciones a la aplicación.

Descripción

El Laboratorio está dividido en dos partes. En la primera parte veremos cómo usar Migraciones de Entity Framework Code First para migrar algunos cambios a las clases de modelo para que el cambio se aplique a la base de datos. En la tercera parte aprenderemos a agregar reglas de validación al modelo de película, cómo se produce la validación en el método Crear vista y Crear acción y el uso de atributos DataType.

Materiales y Recursos Didácticos

- Guía de Laboratorio.
- PC con internet y Navegador Web.
- PC con IDE Visual Studio 2017.
- Haber completado la guía de laboratorio anterior.

I. Configuración de Migraciones de Code First para cambios de modelo.

De forma predeterminada, cuando se usa Entity Framework Code First para crear automáticamente una base de datos, como hizo anteriormente en la guía 2, Code First agrega una tabla a la base de datos para ayudar a realizar un seguimiento de si el esquema de la base de datos está sincronizado con las clases de modelo a las que se generó. Si no están sincronizados, Entity Framework produce un error. Esto facilita el seguimiento de los problemas en tiempo de desarrollo que, de lo contrario, solo se pueden encontrar (por errores ocultos) en tiempo de ejecución.

En el Explorador de soluciones haga clic con el botón derecho en el archivo Peliculas.mdf y seleccione Eliminar para quitar la base de datos peliculas. Si no ve el archivo Peliculas.mdf haga clic



en el icono Mostrar todos los archivos.

En el menú Herramientas, haga clic en Administrador de paquetes NuGet y, a continuación, en Consola del Administrador de paquetes. En la ventana Consola del Administrador de paquetes en el símbolo del PM> sistema, escriba

Enable-Migrations -ContextTypeName MvcPelicula.Models.PeliculaDBContext

```
PM> Enable-Migrations -ContextTypeName MvcPelicula.Models.PeliculaDBContext
Checking if the context targets an existing database...
Detected database created with a database initializer. Scaffolded migration '202308230200086_InitialCreate'
Migrations folder and re-run Enable-Migrations specifying the -EnableAutomaticMigrations parameter.
Code First Migrations enabled for project MvcPelicula.
```

El comando Enable-Migrations (mostrado anteriormente) crea un archivo Configuration.cs en una nueva carpeta Migrations. Abre el archivo Configuration.cs. Reemplace el método Seed en el archivo Configuration.cs por el código siguiente:

```
protected override void Seed(MvcPelicula.Models.PeliculaDBContext context)
{
    context.Peliculas.AddOrUpdate(i => i.Titulo,
        new Pelicula
        {
            Titulo = "Harry Potter y las reliquias de la muerte 2",
            FechaLanzamiento = DateTime.Parse("2011-3-15"),
            Genero = "Ficción",
            Precio = 8.99M
        },

        new Pelicula
        {
            Titulo = "Harry Potter y la piedra filosofal",
            FechaLanzamiento = DateTime.Parse("2001-11-16"),
            Genero = "Ficción",
            Precio = 10.99M
        },

        new Pelicula
        {
            Titulo = "Salvando al soldado Ryan",
            FechaLanzamiento = DateTime.Parse("1998-07-24"),
            Genero = "Guerra",
            Precio = 3.99M
        }
    );
}
```

```

    },
    new Pelicula
    {
        Titulo = "ET",
        FechaLanzamiento = DateTime.Parse("1982-6-11"),
        Genero = "Fantasia",
        Precio = 2.99M
    }
);
}

```

Agregue la referencia: `using MvcPelicula.Models;`

Al ejecutar Migraciones de Code First este llama al método Seed después de cada migración (es decir, llamando a update-database en la consola del Administrador de paquetes) y este método actualiza las filas que ya se han insertado o las inserta si aún no existen.

El primer parámetro pasado al método AddOrUpdate especifica la propiedad que se va a usar para comprobar si ya existe una fila. Para los datos de la tabla película, se usa el campo Titulo para este propósito, ya que cada título de la lista es único.

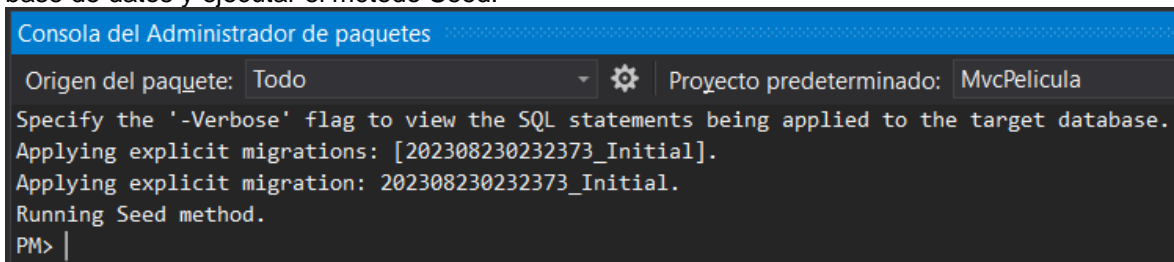
COMPILE EL PROYECTO

Ahora vamos a crear una clase DbMigration para la migración inicial. Esta migración crea una nueva base de datos, por lo que se eliminó el archivo Peliculas.mdf anteriormente.

En la ventana Consola del Administrador de paquetes, escriba el comando **add-migration Initial** para crear la migración inicial. El nombre "Initial" es arbitrario y se usa para asignar un nombre al archivo de migración creado.

Migraciones de Code First crea otro archivo de clase en la carpeta Migrations (con el nombre {FechaActual}_Initial.cs), y esta clase contiene el código que crea el esquema de la base de datos. El nombre de archivo de migración se ha corregido previamente con una marca de tiempo que ayuda con el orden cronológico. Abra el archivo {FechaActual}_Initial.cs, verificará que contiene las instrucciones para crear la tabla Películas para la base de datos Pelicula. Al actualizar la base de datos en las instrucciones siguientes, este archivo {FechaActual}_Initial.cs se ejecutará y creará el esquema de base de datos. A continuación, se ejecutará el método Seed para rellenar la base de datos con datos de prueba.

En la consola del Administrador de paquetes, escriba el comando **update-database** para crear la base de datos y ejecutar el método Seed.



The screenshot shows the Package Manager Console with the following text:

```

Consola del Administrador de paquetes
Origen del paquete: Todo
Proyecto predeterminado: MvcPelicula
Specify the '-Verbose' flag to view the SQL statements being applied to the target database.
Applying explicit migrations: [202308230232373_Initial].
Applying explicit migration: 202308230232373_Initial.
Running Seed method.
PM>

```

Si devuelve un error indicando que ya existe una tabla y no se puede crear, verifique que ha eliminado la base de datos, ejecute la aplicación y después de ejecute update-database. Si sigue

recibiendo un error, elimine la carpeta de migraciones y su contenido, asegúrese de haber eliminado la base de datos, ejecute nuevamente el comando **Enable-Migrations**. Si sigue recibiendo un error, abra "SQL Server Explorador de objetos" y elimine la base de datos de la lista. Si recibe un error que indica "No se puede adjuntar el archivo .mdf como base de datos", quite la propiedad Initial Catalog como parte de la cadena de conexión en el archivo web.config.

Ejecute la aplicación añadiendo a la URL /Películas

Index

[Create New](#)

Titulo	FechaLanzamiento	Genero	Precio	
Harry Potter y las reliquias de la muerte 2	15/03/2011 0:00:00	Ficción	8,99	Edit Details Delete
Harry Potter y la piedra filosofal	16/11/2001 0:00:00	Ficción	10,99	Edit Details Delete
Salvando al soldado Ryan	24/07/1998 0:00:00	Guerra	3,99	Edit Details Delete
ET	11/06/1982 0:00:00	Fantasia	2,99	Edit Details Delete

© 2023 - Mi aplicación ASP.NET

Agregar una propiedad de clasificación al modelo Pelicula.

Agregaremos una nueva propiedad a la clase existente Pelicula. Abra el archivo Models\Movie.cs y agregue la propiedad "Clasificación":

```
public string Clasificacion { get; set; }
```

COMPILE LA APLICACION

Dado que se ha agregado un nuevo campo a la clase Pelicula, también se debe actualizar la lista de enlaces permitidos para que se incluya esta nueva propiedad. Actualice el controlador en el atributo bind para el método Create los métodos de acción y Edit para incluir la propiedad Clasificación:

```

12 {
13     return HttpNotFound();
14 }
15 return View(pelicula);
16 }
17
18 // GET: Peliculas/Create
19 // Definimos
20 public ActionResult Create()
21 {
22     return View();
23 }
24
25 // POST: Peliculas/Create
26 // Para protegerse de ataques de publicación excesiva, habilite las propiedades específicas a las que quiere enlazarse. Para
27 // más detalles, vea https://go.microsoft.com/fwlink/?LinkID=317598.
28 [HttpPost]
29 [ValidateAntiForgeryToken]
30 // Definimos
31 public ActionResult Create([Bind(Include = "ID,Titulo,FechaLanzamiento,Genero,Precio,Clasificacion")] Pelicula pelicula)

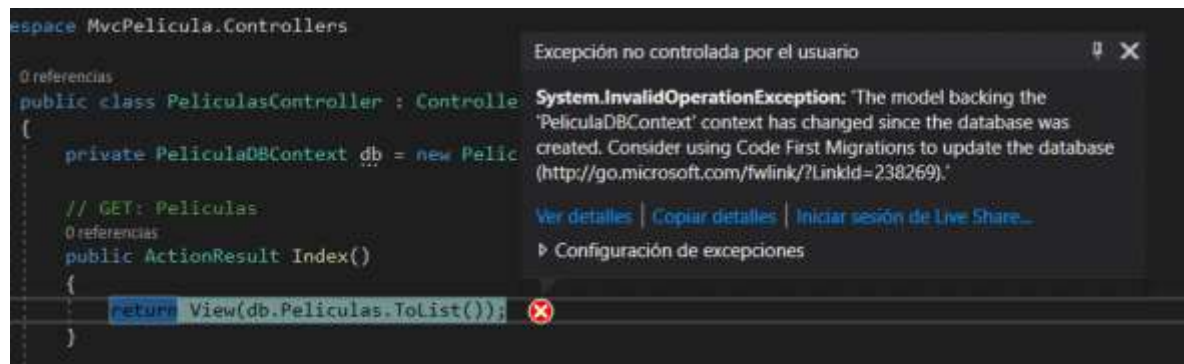
```

También se debe actualizar la nueva propiedad Clasificación en las plantillas de vista *mostrar*, *crear* y *editar*. Abra el archivo \Views\Peliculas\Index.cshtml y agregue una columna de encabezado: `<th>Clasificación</th>` justo después de la columna Precio. A continuación, agregue una etiqueta `<td>` después de la columna para precio, para representar el `@item.Clasificacion`.

Luego, abra el archivo `\Views\Películas\Create.cshtml` y agregue el campo Clasificación, después del campo Precio:

```
<div class="form-group">
    @Html.LabelFor(model => model.Rating, new { @class = "control-label col-md-2" })
    <div class="col-md-10">
        @Html.EditorFor(model => model.Rating, new {
            htmlAttributes = new { @class = "form-control" } })
        @Html.ValidationMessageFor(model => model.Rating, "", new
        { @class = "text-danger" })
    </div>
</div>
```

Ejecute la aplicación y añada a la URL /Películas
Recibirá el error:



Aparece este error porque se ha actualizado la clase Película del modelo y ahora es diferente del esquema de la tabla de la base de datos existente. (No hay ninguna columna Clasificación en la tabla de la base de datos).

Para resolver este error primero hay que añadir en el método Seed de la clase Migrations.cs el campo Clasificación. Agregue este valor para cada objeto creado en el método, por ejemplo:

```
context.Películas.AddOrUpdate(i => i.Titulo,
    new Película
    {
        Titulo = "Harry Potter y las reliquias de la muerte 2",
        FechaLanzamiento = DateTime.Parse("2011-3-15"),
        Genero = "Ficción",
        Precio = 8.99M,
        Clasificacion = "B"
    },
```

Compile la solución y, a continuación, abra la ventana Consola del Administrador de paquetes y escriba el siguiente comando: **add-migration Clasificacion**

El comando add-migration indica al marco de migración que examine el modelo de película actual con el esquema de base de datos de película actual y cree el código necesario para migrar la base

de datos al nuevo modelo. El nombre Clasificacion es arbitrario y se usa para asignar un nombre al archivo de migración. Resulta útil usar un nombre descriptivo para saber que cambios se han realizado.

Cuando finaliza este comando, Visual Studio abre el archivo de clase que define la nueva clase derivada DbMigration y, en el método Up puede ver el código que crea la nueva columna.

Compile la solución y escriba el comando **update-database** en la ventana Consola del Administrador de paquetes.

Ejecute la aplicación y compruebe que ya se agregó correctamente el campo Clasificación.

Index

Create New

Titulo	FechaLanzamiento	Genero	Precio	Clasificacion	
Harry Potter y las reliquias de la muerte 2	15/03/2011 0:00:00	Ficción	8,99	B	Edit Details Delete
Harry Potter y la piedra filosofal	16/11/2001 0:00:00	Ficción	10,99	TP	Edit Details Delete
Salvando al soldado Ryan	24/07/1998 0:00:00	Guerra	3,99	B	Edit Details Delete
ET	11/06/1982 0:00:00	Fantasia	2,99	TP	Edit Details Delete

Ahora que ya sabe usar migraciones, no tendrá que quitar la base de datos al agregar un nuevo campo o actualizar el esquema de otro modo. En la sección siguiente, realizaremos más cambios de esquema y usaremos migraciones para actualizar la base de datos.

II. Agregar validaciones.

En esta sección, se agregará lógica de validación al modelo Película para asegurarnos que las reglas de validación se apliquen cada vez que un usuario intente crear o editar una película mediante la aplicación.

Uno de los principios principales de diseño de ASP.NET MVC es DRY (del inglés don't repeat yourself o "no te repitas"). ASP.NET MVC permite especificar la funcionalidad o el comportamiento una sola vez y, a continuación, que se refleje en todas partes de una aplicación. Esto reduce la cantidad de código que necesita escribir y hace que el código que se escriba sea menos propenso a errores y más fácil de mantener.

La compatibilidad de validación proporcionada por ASP.NET MVC y Entity Framework Code First es un excelente ejemplo del principio DRY en acción. Se puede especificar mediante declaración reglas de validación en un solo lugar (en la clase de modelo) y las reglas se aplican en todas partes de la aplicación.

Agregar reglas de validación al modelo de película

Abra el archivo Pelucula.cs. Agregue el espacio de nombres:

System.ComponentModel.DataAnnotations, este proporciona un conjunto integrado de atributos de validación que se pueden aplicar declarativamente a cualquier clase o propiedad. (También contiene atributos de formato como DataType que ayudan con el formato y no proporcionan ninguna validación).

Ahora actualice la clase Pelicula para aprovechar las ventajas de los atributos de validación integrados Required, StringLength, RegularExpression y Range. Reemplace la clase Pelicula por lo siguiente:

```
public int ID { get; set; }
[StringLength(60, MinimumLength = 3)]
public string Titulo { get; set; }

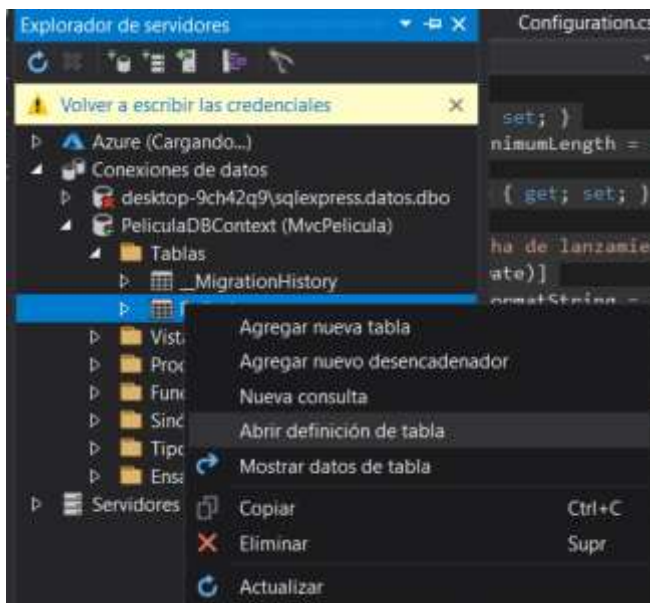
[Display(Name = "Fecha de lanzamiento")]
[DisplayFormat(DataFormatString = "{0:yyyy-MM-dd}", ApplyFormatInEditMode
= true)]
public DateTime FechaLanzamiento { get; set; }

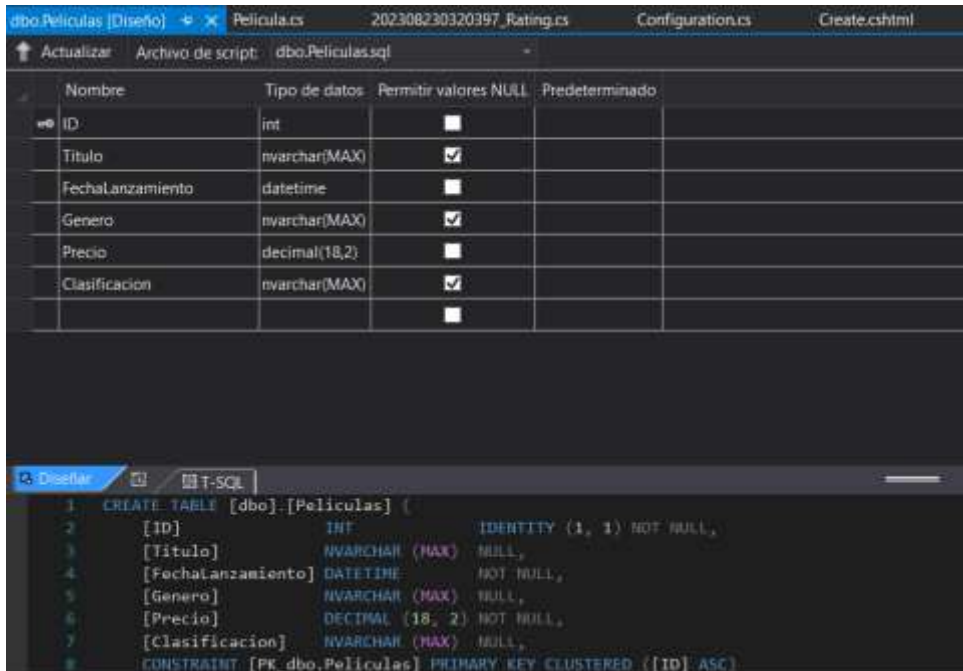
[RegularExpression(@"^[A-Z]+[a-zA-Z]*$")]
[Required]
[StringLength(30)]
public string Genero { get; set; }

[Range(1, 100)]
public decimal Precio { get; set; }

[RegularExpression(@"^[A-Z]+[a-zA-Z]*$")]
[StringLength(5)]
public string Clasificacion { get; set; }
```

El atributo StringLength establece la longitud máxima de la cadena y establece esta limitación en la base de datos, por lo que el esquema de la base de datos cambiará. Haga clic con el botón derecho en la tabla Peliculas en el Explorador de servidores y haga clic en Abrir definición de tabla:





En la imagen anterior, puede ver que todos los campos de cadena se establecen en NVARCHAR (MAX). Usaremos migraciones para actualizar el esquema. Compile la solución y, a continuación, abra la ventana Consola del Administrador de paquetes y escriba los siguientes comandos:

add-migration DataAnnotations
update-database

Al ejecutar el primer comando se modifica el esquema de la base de datos y muestra el nuevo archivo de migración, puede verificar que el campo Genero ya no acepta valores nulos, el campo Clasificación tiene una longitud máxima de 5 y Titulo tiene una longitud entre 3 a 60.

Los atributos de validación especifican el comportamiento que quiere aplicar en las propiedades del modelo al que se aplican. Los atributos *Required* y *MinimumLength* indican que una propiedad debe tener un valor, pero nada evita que un usuario escriba espacios en blanco para satisfacer esta validación.

El atributo *RegularExpression* se usa para limitar los caracteres que se pueden escribir. En el código anterior, Genero y Clasificacion solamente pueden usar letras (no se permiten espacios en blanco, números ni caracteres especiales). El atributo Clasificacion restringe un valor a un intervalo determinado.

El atributo *StringLength* permite establecer la longitud máxima de una propiedad de cadena y, opcionalmente, su longitud mínima. Los tipos de valor (como decimal, int, float, DateTime) son inherentemente necesarios y no necesitan el atributo *Required*.

Code First garantiza que las reglas de validación que especifique en una clase de modelo se aplican antes de que la aplicación guarde los cambios en la base de datos. Tener reglas de validación aplicadas automáticamente por .NET Framework ayuda a que la aplicación sea más

sólida. También permite asegurarnos de que todo se valida y que no dejamos ningún dato incorrecto en la base de datos accidentalmente.

Interfaz de usuario para error de validación

Ejecute la aplicación y vaya a la dirección URL de /Peliculas.

Haga clic en el vínculo Crear nuevo para agregar una nueva película. Rellene el formulario con algunos valores no válidos. En cuanto la validación del lado cliente de jQuery detecta el problema, muestra un mensaje de error.

Create

Pelicula

Titulo	<input type="text" value="3"/>	El campo Titulo debe ser una cadena con una longitud minima de 3 y una longitud máxima de 60.
Fecha de lanzamiento	<input type="text" value="dd/03/0001"/>	El campo Fecha de lanzamiento es obligatorio.
Genero	<input type="text" value="123123"/>	El campo Genero debe coincidir con la expresión regular <code>^[A-Z]+[a-zA-Z]*\$</code> .
Precio	<input type="text" value="dsad"/>	El campo Precio debe ser un número.
Clasificacion	<input type="text" value="12312"/>	El campo Clasificacion debe coincidir con la expresión regular <code>^[A-Z]+[a-zA-Z]*\$</code> .
<input type="button" value="Create"/>		

Observe cómo el formulario ha emitido un mensaje de error de validación adecuado junto a cada uno de los cuadros que contienen valores no válidos. 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 real es que no era necesario cambiar 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 se crearon previamente 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 `Edit` y se aplicará la misma validación.

Los datos del formulario no se envían al servidor hasta que no hay ningún error de validación del lado cliente.

Uso de atributos `DataType`

Abra el archivo `Pelicula.cs` y agregue en la clase `Película` a las propiedades `FechaLanzamiento` y `Precio` el atributo adecuado con `DataType`.

```
[DataType(DataType.Date)]
public DateTime FechaLanzamiento { get; set; }

[DataType(DataType.Currency)]
public decimal Precio { get; set; }
```

Los atributos *DataType* solo proporcionan sugerencias para que el motor de vista dé formato a los datos (y proporcione atributos como <a> para las direcciones URL y para el correo electrónico). Se puede usar el atributo *RegularExpression* para validar el formato de los datos.

El atributo *DataType* se usa para indicar un tipo de datos más específico que el tipo intrínseco de la base de datos, no son atributos de validación. En este caso solo se quiere realizar el seguimiento de la fecha, no de la fecha y la hora. La enumeración *DataType* proporciona muchos tipos de datos, como Date, Time, PhoneNumber, Currency, EmailAddress y [muchos más](#).

El atributo *DataType* también puede permitir que la aplicación proporcione automáticamente características específicas del tipo. Por ejemplo, en exploradores que admiten HTML5 se puede crear un vínculo tipo mailto: para *DataType.EmailAddress* y se puede proporcionar un selector de fecha para *DataType.Date*.

DataType.Date no especifica el formato de la fecha que se muestra. De forma predeterminada, el campo de datos se muestra según los formatos predeterminados basados en *cultureInfo* del servidor.

III. Ejercicios

1. Agregue el campo Clasificación a las vistas Editar, Detalles y Eliminar.
2. En base al ejercicio 2 de la guía de laboratorio anterior, agregue dos campos adicionales a la clase persona, detalle todas las reglas de validación necesarias al modelo.

IV. 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>
- Archiveddocs, A. R. (2015, 10 junio). Utilizar IntelliSense. Recuperado 2 noviembre, 2019, de [https://docs.microsoft.com/es-es/previous-versions/visualstudio/visual-studio-2013/hcw1s69b\(v=vs.120\)?redirectedfrom=MSDN](https://docs.microsoft.com/es-es/previous-versions/visualstudio/visual-studio-2013/hcw1s69b(v=vs.120)?redirectedfrom=MSDN)