CI-0126 Ingeniería de Software
Grupo 01 - I-2023

Prof. Dr. Allan Berrocal Rojas
allan.berrocal@ucr.ac.cr

Laboratorio 6[1]
Formularios html y Razor

Contenidos

[1] Basado en material previamente desarrollado en cursos de Gustavo López y Rebeca Obando en la Universidad de Costa Rica

Prof. Dr. Allan Berrocal Rojas
allan.berrocal@ucr.ac.cr

1. Requisitos

1. Haber completado el laboratorio anterior. Para este laboratorio puede utilizar la misma base de datos, y como código base el laboratorio 5.
2. Para ello haga un directorio llamado **Laboratorio6** en su repositorio de control de versiones y copie ahí el proyecto del **Laboratorio5**.
3. **Opcionalmente**, para mayor orden, haga el reemplazo de nombres donde sea necesario para que todo lo que tenga nombre **laboratorio5** pase a llamarse **laboratorio6**. Como sugerencia utilice la funcionalidad de **Find/Replace**.

2. Objetivos

El objetivo de este laboratorio es aprender cómo funcionan los formularios en HTML y cómo utilizarlos para la creación, edición y eliminación de datos en la base de datos. Específicamente:

1. Comprender y utilizar formularios en html.
2. Comprender y aplicar el uso de formularios en páginas razor.

3. Parte - Concepto de formularios html

Esta parte consiste en un repaso y estudio de HTML especialmente elementos relacionados con el uso de formularios. Para ello debe estudiar el siguiente material:

- W3Schools (ver toda la sección de ***HTML Forms***)
- MDN web docs (una pequeña introducción a ***HTML Forms***)
- Video de Youtube (Kiko Palomares) Antes del minuto 20 puede ser introductorio. Posterior al minuto 20 puede ser que encuentren temas nuevos quienes no están familiarizados con HTML.

En el entregable del laboratorio debe incluir una explicación breve en sus propias palabras sobre **dos elementos** que más le llamaron la atención **para cada una** de las siguientes 4 secciones.

- Form elements
- Input types
- Input attributes
- Input form attributes

Es decir, se espera que en total explique al menos 8 elementos.

4. Parte - Formularios en páginas Razor

Para este laboratorio se va utilizar el concepto de ***Model Binding*** el cual es un mecanismo que permite la comunicación y el paso de parámetros entre un cliente web (el navegador) y un servidor. El ***Model Binding*** crea enlaces entre el modelo y las vistas, para detallar esto de una manera más específica, lo que se hará es recuperar información desde los formularios que se implementan en las vistas y pasarlos a la lógica de la aplicación de manera que se puedan tomar estos datos y almacenarlos en una base de datos o en su efecto hacer modificaciones a datos ya existentes. Ustedes quizás lograron ver algunos ejemplos de código en lo que respecta a los métodos get y post de los formularios, y la manera de pasar los datos (dependiendo de la tecnología que se utilice) puede ser complicada.
El ***Model Binding*** de **ASP.NET Core** provee la ventaja que usted puede recibir la información enviada de las vistas en el controlador de una manera compacta y sencilla, al tiempo que usted adquiere una manera de representar un modelo en las vistas.
El ***Model Binding*** consiste entonces en dos aspectos básicos fundamentales, presentar la información de un modelo en las vistas y "postear" los datos del formulario en la lógica de la aplicación a través de un modelo, esto evita que en los **scripts** o clases donde reciben los valores del formulario se tengan que procesar hileras de datos, lo que en otras tecnologías o inclusive en otras versiones de **ASP.NET Core** se debería de hacer. En este caso, con el ***Model Binding*** se establece una relación entre los valores del formulario y las propiedades del modelo. Puede consultar más información sobre el ***model binding*** aquí.
Nota: Antes de continuar, verifique que tiene su conexión abierta a la base de datos **Películas** que usó **laboratorio5**, lo puede hacer desde el ***Server Explorer***. En caso que esté cerrada, abra la conexión.

4.1. Modificar el modelo

Recordemos el modelo que tenemos, a este modelo se le van a hacer algunas modificaciones para aprovechar las ventajas del ***model binding***.

```
public class PeliculaModel
{
    public int ID {get; set; }

    public string Nombre {get; set; }

    public int Año {get; set; }
}
```

Vamos a utilizar algunas propiedades que nos permitirán decirle a nuestro modelo si el atributo es requerido (propiedad ***required***) y que valor va a tener el ***label*** del ***input*** que representará cada atributo en el formulario (propiedad ***displayName***). Modifique el modelo para que se vea de la siguiente manera:

```
using System.ComponentModel;

namespace laboratorio6.Models
{
    7 references
    public class PeliculaModel
    {
        1 reference
        public int ID { get; set; }

        [Required(ErrorMessage = "Debe ingresar un nombre")]
        [DisplayName("Nombre de la pelicula:")]
        2 references
        public string? Nombre { get; set; }

        [Required(ErrorMessage = "Debe ingresar un año")]
        [DisplayName("Año:")]
        [RegularExpression("^(18|19|20)[0-9]{2}$", ErrorMessage = "Ingrese un año válido")]
        2 references
        public int Año { get; set; }
    }
}
```

Observaciones del código:

1. **System.ComponentModel.DataAnnotations** incluye las librerías que permiten utilizar los elementos ***Required***, ***DisplayName*** y ***Regular Expression***. Estos elementos sirven para hacer el ***binding*** de las propiedades del modelo en la vista, la sintaxis es así como se muestra en la imagen.
2. **Required**: indica que el atributo será requerido al llenar el formulario. Esto se logra en conjunto con el auxiliar de **C#** para crear formularios, como se mostrará más adelante. En caso que un usuario intente llenar un formulario sin indicar el atributo, entonces automáticamente se restringe la acción y se muestra el mensaje especificado en ***ErrorMessage***.
3. **DisplayName**: se utiliza a través una etiqueta (***label***) para especificar el mensaje que corresponde a la propiedad. Message será el mensaje mostrado. De igual manera, más adelante esto se comprenderá mejor con el desarrollo del laboratorio.
4. **RegularExpression**, sirve para validar que los ***inputs*** cumplan con ciertos criterios utilizando expresiones regulares (ejemplo que solo ingresen números, o solo letras). Si no tiene conocimiento acerca de las expresiones regulares y desea conocer un poco más, para saber más haga click aquí. En este caso la expresión regular utilizada sirve para prohibir el uso de números en el nombre del idioma.
5. Los mensajes de errores, algunos de los elementos de ***DataAnnotations*** se les puede enviar otros parámetros para indicar, por ejemplo, que una hilera debe tener un mínimo

y/o máximo de caracteres. Además, existen otras funciones para realizar validación de datos que se incluyen con esta librería e incluso es posible "personalizar" el tipo de validación, si desea ver más información relacionada a este tema, puede ver más aquí.

4.2. Insertar datos de una película en la BD

Ahora vamos a agregar el siguiente código para poder insertar países en nuestra base de datos, este código debe estar en el **PeliculasHandlers.cs**, clase creada en el laboratorio pasada, que es la encargada de conectar con la base de datos.

```
public bool CrearPelicula(PeliculaModel pelicula)
{
    var consulta = @"INSERT INTO [dbo].[Pelicula] ([Nombre],[Año]) VALUES(@Nombre, @Año) ";
    var comandoParaConsulta = new SqlCommand(consulta, conexion);
    comandoParaConsulta.Parameters.AddWithValue("@Nombre", pelicula.Nombre);
    comandoParaConsulta.Parameters.AddWithValue("@Año", pelicula.Año);

    conexion.Open();
    bool exito = comandoParaConsulta.ExecuteNonQuery() >= 1;
    conexion.Close();

    return exito;
}
```

Observe que el **query** que se creó para realizar la inserción de los datos tiene algunos elementos precedidos de un **@**. Esos elementos son parámetros del **query**. También se pueden agregar estos parámetros en consultas con **SELECT** y otras según sea necesario, en este caso se necesita tener la consulta parametrizada porque se reciben datos de una película que fueron ingresados desde la interfaz los cuales son desconocidos y se requieren agregar a la base de datos.

Los métodos del comando para la consulta, **Parameters.AddWithValue** justamente son los encargados de transferir los valores que se indiquen al respectivo parámetro (observe la sintaxis) para que el analizador de consultas de SQL ejecuta el query con los datos correspondientes. Finalmente, observe que se abre la conexión, se ejecuta el query mediante la sentencia **ExecuteNonQuery()** y se cierra nuevamente la conexión, el método **ExecuteNonQuery()** retorna un 0 cuando algo a nivel de SQL al ejecutar la consulta falla y un número mayor que cero cuando se modificó una tupla correctamente, en este caso es una inserción correcta.

4.3. Método para crear películas en el controlador

Ahora que se tiene un modelo con sus propiedades definidas y los métodos del Handler para poder insertar elementos en la base de datos, entonces se va crear una vista que tendrá un formulario para crear la película con los datos respectivos. Para esto se hará lo que usualmente se ha hecho en laboratorios anteriores, crear un método en el controlador con su respectiva vista y uno adicional que se explica más adelante. En el controlador de películas, agregue los siguientes métodos:

- Método para enlazar la vista del formulario

```
[HttpGet]
0 references
public ActionResult CrearPelicula()
{
    return View();
}
```

- Método para la creación

El formulario que se creará tendrá un método **post**, como el que usted estudió en la primera parte de este laboratorio. Entonces, resulta necesario recibir los datos enviados por el formulario en la lógica de la aplicación, para ello se agregará este método:

```
[HttpPost]
0 references
public ActionResult CrearPelicula(PeliculaModel pelicula)
{
    ViewBag.ExitoAlCrear = false;
    try
    {
        if (ModelState.IsValid)
        {
            PeliculasHandler peliculasHandler = new PeliculasHandler();
            ViewBag.ExitoAlCrear = peliculasHandler.CrearPelicula(pelicula);

            if (ViewBag.ExitoAlCrear)
            {
                ViewBag.Message = "La película " + pelicula.Nombre + " fue creada con éxito.";
                ModelState.Clear();
            }
        }
        return View();
    }
    catch
    {
        ViewBag.Message = "Algo salió mal y no fue posible crear la película";
        return View();
    }
}
```

Observe que los dos métodos tienen el mismo nombre (para mantener una consistencia semántica), pero son diferentes en la carga de parámetros. El método encargado de “llamar” o “ejecutar” la vista es el que no recibe ningún parámetro, este método es común de los laboratorios anteriores. No obstante, el segundo método también desplegará la misma vista

Prof. Dr. Allan Berrocal Rojas
allan.berrocal@ucr.ac.cr

que el primero, pero únicamente después de que a través de un formulario haga post direccionando los datos de este método.
De hecho, usted puede observar que el parámetro que recibe el segundo método es un modelo, este modelo cuando el formulario se envía, va cargado de información. Es decir, el modelo es el paquete de datos. De hecho, el **ModelState** funciona para revisar que el paquete entrante no tenga "defectos". Se puede decir que, en general, usualmente cuando se crea un formulario se hacen dos métodos de controlador: el que ejecuta la vista para llegar al formulario y el que recibe los datos del formulario.
Una observación más, en este caso se tiene dos **return View()** y ambos por ser de esta manera van a volver a la vista del formulario, pero esto no necesariamente debe ser así. En el **return** usted podría colocar un **RedirectToAction** o alguno de los tantos posibles métodos que soporta el **ActionResult**. En este caso se hace de esta manera para enviar un mensaje en la misma vista del formulario, el mensaje va acusar lo ocurrido en la creación de la película.

4.4. Formulario con auxiliares de HTML en Razor

Es hora de crear el formulario para esto cree en el folder: **Views/Peliculas** la nueva vista llamada **CrearPelicula**, en este use el siguiente código pero analice qué está haciendo:

```
@model laboratorio6.Models.PeliculaModel
@{
    ViewBag.Title = "Crear Película";
}
<html>
<head> </head>
<body>
    @if (ViewBag.Message != null)
    {
        if (ViewBag.ExitoAlCrear)
        {
            <div class="alert-success"> <h3> @ViewBag.Message </h3> </div>
        }
        else
        {
            <div class="alert-danger"> <h3> @ViewBag.Message </h3> </div>
        }
    }
    @using (Html.BeginForm("CrearPelicula", "peliculas", FormMethod.Post, new { enctype = "multipart/form-data" }))
    {
        @Html.AntiForgeryToken()
        <h1>Formulario de creación de películas</h1>
        <div class="form-horizontal">
            <div class="form-group">
                @Html.LabelFor(model => model.Nombre)
                @Html.TextBoxFor(model => model.Nombre, new { @class = "form-control" })
                @Html.ValidationMessageFor(model => model.Nombre, "", new { @class = "text-danger" })
            </div>
            <div class="form-group">
                @Html.LabelFor(model => model.Año)
                @Html.TextBoxFor(model => model.Año, new { @class = "form-control" })
                @Html.ValidationMessageFor(model => model.Año, "", new { @class = "text-danger" })
            </div>
            <input type="submit" class="btn btn-success" value="Crear" />
        </div>
    }
</body>
</html>
```

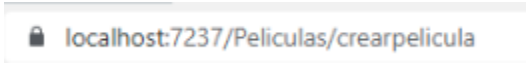
Observe que es bastante código y revise los siguientes elementos:

- **@using (Html.BeginForm()){ }** es el auxiliar que crea el cuerpo del formulario, básicamente hace el equivalente la etiqueta. Los auxiliares en general lo que hacen es crear el código de **HTML** por usted. Observe que la función **BeginForm** recibe algunos parámetros. Los **String** iniciales sirven para para construir la ruta **url** que dirige los datos, en este caso, al método de **post** que se agregó al controlador, además, con un parámetro se indica que es formulario de método post
- Preste atención al contenido del primer contenedor, en este caso el class es solamente una clase pre cargada de **Bootstrap** para mejorar el aspecto visual. Lo que es realmente importante son los elementos del auxiliar; **Html.LabelFor(model => model.Nombre)** hace el **bind** del **display** de la propiedad nombre que se indicó en el modelo; **@Html.TextBoxFor()** esto genera una **input** de tipo **texto** y almacena lo que se ingrese en la variable del modelo que se específica **(model=>model.Nombre** en

este caso); **Html.ValidationMessageFor()** valida que los datos que se ingresen tengan el formato esperado, en caso contrario, muestra los mensajes de error.

- El método restante es análogo, en este caso se mantiene el mismo tipo de auxiliar. Existen otros como, por ejemplo: ***radio buttons***, ***checkbox*** o ***drop down***. Estos métodos reciben algunos parámetros que le permiten personalizar las etiquetas generadas, en este caso se observa que asigna una clase a algunos elementos. Usted aquí podría agregar, por ejemplo, parámetros para que el input sea de tipo “number”, el máximo, mínimo y otros como lo vio en el video.

Muy bien es hora de revisar el resultado obtenido, para eso corra la solución y diríjase a la ruta creada `peliculas/crearpelicula` por ejemplo:



La vista debe verse similar a esta:

Cree una película, y luego haga clic en el menú Películas y asegúrese que puede ver el nuevo registro de película. **Tome un screenshot de la película creada y adjúntelo en la documentación que entregará del laboratorio.**

Prof. Dr. Allan Berrocal Rojas
allan.berrocal@ucr.ac.cr

5. Parte - Edición

5.1. Edición de una película

Para lograr editar películas, es necesario crear algunos elementos. En este caso, se hará una nueva vista para acceder a editar una película. En resumen, se requiere de un método para invocar la consulta tipo **UPDATE** en **SQL**, los métodos del controlador, y la vista respectiva.

Agregue el siguiente método en su **peliculaHandler** para editar un país:

```
0 references
public bool EditarPelicula(PeliculaModel pelicula)
{
    var consulta = @"UPDATE [dbo].[Pelicula] SET
            Nombre = @Nombre,
            Año = @Año
            WHERE Id=@Id ";

    var cmdParaConsulta = new SqlCommand(consulta, conexion);
    cmdParaConsulta.Parameters.AddWithValue("@Nombre", pelicula.Nombre);
    cmdParaConsulta.Parameters.AddWithValue("@Año", pelicula.Año);
    cmdParaConsulta.Parameters.AddWithValue("@Id", pelicula.ID);

    conexion.Open();
    bool exito = cmdParaConsulta.ExecuteNonQuery() >= 1;
    conexion.Close();

    return exito;
}
```

Agregue los siguientes métodos a su controlador:

```
0 references
public ActionResult EditarPelicula(int? identificador)
{
    ActionResult vista;
    try
    {
        var peliculasHandler = new PeliculasHandler();
        var pelicula = peliculasHandler.ObtenerPeliculas().Find(model => model.ID == identificador);
        if (pelicula == null)
        {
            vista = RedirectToAction("Index");
        }
        else
        {
            vista = View(pelicula);
        }
    }
    catch
    {
        vista = RedirectToAction("Index");
    }
    return vista;
}
```

```
0 references
public ActionResult EditarPelicula(PeliculaModel pelicula)
{
    try
    {
        var peliculasHandler = new PeliculasHandler();
        peliculasHandler.EditarPelicula(pelicula);
        return RedirectToAction("Index", "Peliculas");
    }
    catch
    {
        return View();
    }
}
```

El primer método es análogo al **get** del **crearPelícula**. Esta acción del controlador será la encargada de ejecutar la vista que contiene el formulario para editar. Note que tiene pocas diferencias con respecto al método de crear, usted debe analizarlas para comprenderlas. En resumen, se recibe un parámetro identificador que servirá para indicarle al formulario cuál película es la que se va editar (coincide con el id del modelo).

Además, en este caso se obtiene de la lista de todas las películas, la película en específico con el identificador que recibe por parámetro. Esto se puede mejorar (bastante) en términos de eficiencia, usted debería analizar porque esta solución no es eficiente y cuál es una manera de mejorarla (no es necesario que modifique el código, solo analice las posibles respuestas).

Con respecto al segundo método, este es el método encargado de llamar al **handler** para crear las modificaciones en la base de datos. Note que en este caso no se “valida” ni notifica al usuario que ocurrió con la edición de la película. Si usted quiere puede agregar el código necesario para lograrlo, nuevamente, es opcional hacerlo. Una manera de lograrlo podría ser con **ViewBag TempData**. Si usted lo desea, puede investigar un poco al respecto. Lo demás, es exactamente igual que el **post** de crear película.

5.2. Vista para el formulario de edición

Cree una nueva vista parcial y vacía, de la misma manera que lo ha hecho en ocasiones anteriores, esta es para la acción del controlador que se acaba de crear (**EditarPelicula**). En el contenido de vista, copie exactamente el mismo código de crear película y solo debe hacer 5 cambios:

a. Borre las líneas de código que son para mostrar el mensaje de éxito o error al crear la película (el `if-else` del inicio).
b. En `Html.BeginForm("crearPelicula" ...)` cambie “**crearPelicula**” por el nombre de la acción del controlador de editar (“**editarPelicula**”).
c. Cambie la etiqueta **H1** para que diga algo relacionado a editar películas.
d. Cambie el texto del botón para que diga algo relacionado a aplicar los cambios en lugar de “crear”.
e. Agregue en algún lugar del **form**, el siguiente elemento auxiliar: `@Html.HiddenFor(model=>model.Id)`. Lo puede agregar, por ejemplo, justo después de la etiqueta **H1**.

5.3. Modificando la vista index

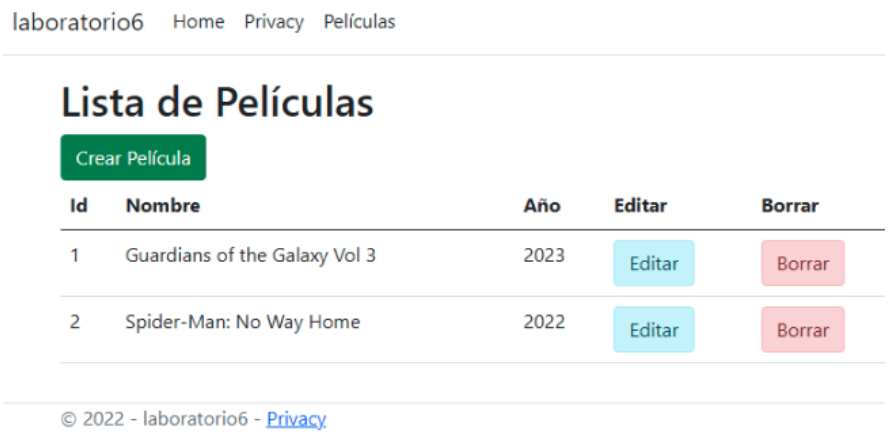
Para evitar estar escribiendo los **URL** completos, vamos a agregar botones a la vista **Index.cshtml** creada en el laboratorio anterior. Vamos a crear 3 botones: Crear, Editar, y Borrar Película. Para hacer esto siga el siguiente código:

```
@model List<laboratorio6.Models.PeliculaModel>;
@{
    ViewData["Title"] = "Peliculas";
}
<h1>@ViewBag.MainTitle</h1>
<div>
    <a href="@Url.Action("CrearPelicula","Peliculas")"
        class="btn btn-success">Creal Película</a>
    <table class="table">
        <thread>
            <tr>
                <th>Id</th>
                <th>Nombre</th>
                <th>Año</th>
                <th>Editar</th>
                <th>Borrar</th>
            </tr>
        </thread>
        <tbody>
            @foreach(var item in Model)
            {
                <tr>
                    <tr>@item.ID</tr>
                    <td>@item.Nombre</td>
                    <td>@item.Año</td>
                    <td>
                        @Html.ActionLink("Editar", "EditarPelicula",
                        new {identificador = @item.ID},
                        new {@class = "btn btn alert-info",
                            @onclick = "return confirm('¿Desea editar esta pelicula?')"})
                    </td>
                    <td>
                        @Html.ActionLink("Borrar", "BorrarPelicula",
                        new {identificador = @item.ID},
                        new {@class = "btn btn alert-danger",
                            @onclick = "return confirm('¿Desea borrar esta película?')"})
                    </td>
                </tr>
            }
        </tbody>
    </table>
</div>
```

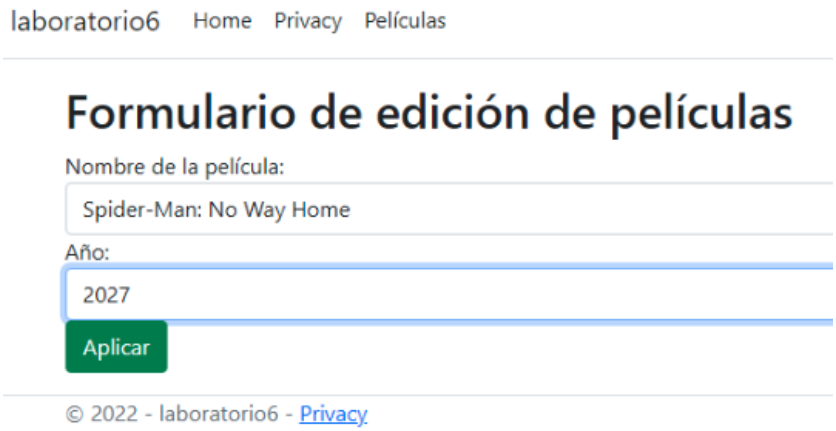
Observe que se agregó un enlace donde se invoca a la acción del controlador y se le pasa el id de la película respectiva. Este auxiliar genera una etiqueta **HTML** tipo con los elementos que se le indique (el **class** hace que el **link** tenga una apariencia de botón), usted puede ver esto en el navegador a través del código fuente de la página.

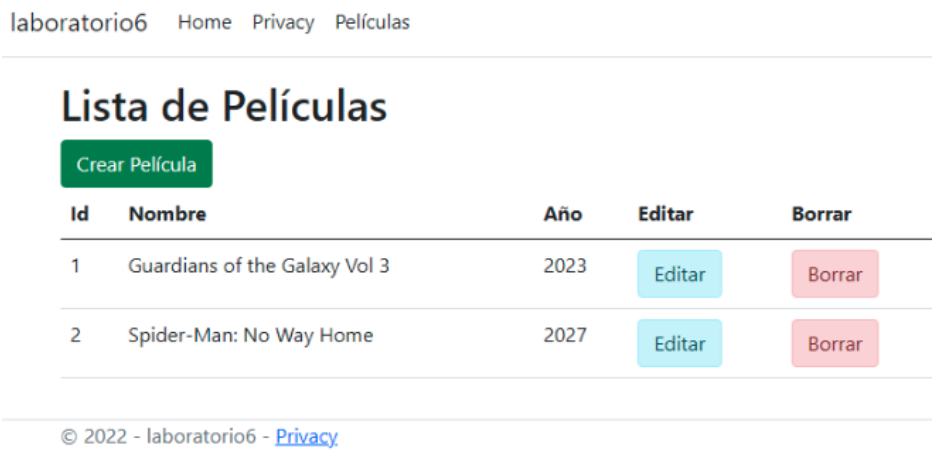
Ahora su página index debe lucir similar a esta:

Prof. Dr. Allan Berrocal Rojas
allan.berrocal@ucr.ac.cr



Luego intente editar una película, tome **screenshots** del antes y después de la edición (**index view**), el cambio que está haciendo en la página **editarPelicula** y el resultado obtenido en la vista **index** de películas. Ejemplo





5.4. Borrar una película

En este momento la vista index se creó el botón para borrar películas. Cree el método para borrar películas en el controlador. Además cree el método para borrar películas en el handler, el script de bases de datos a utilizar es el siguiente

```
Delete [dbo].[pelicula] where Id = @id
```

Tome **screenshots** del **view index** antes y después de borrar una película. Además adjunte **screenshots** de los métodos creados en el controlador y handler.

6. Entregables del laboratorio

0.25%	Todos los entregables de este laboratorio deben estar en su repositorio de GitHub en un directorio con nombre **laboratorio6**.
0.5%	Cree un documento ordenado con su nombre y número de carné. Utilice el nombre **ReporteLab6_CARNE.pdf** (su número de carné). No entregue archivos en Word o en ningún otro formato editable. Utilice PDF.
32%	Explicaciones que se solicitan en la sección 1 sobre 8 elementos de formularios en HTML.
54%	El reporte debe tener una serie de **screenshots** que evidencien la realización de los pasos principales del laboratorio. Los siguientes **screenshots** son requeridos pero puede agregar algunos otros más que agreguen valor al reporte. 1. 2 Screenshots del final de la sección 4.4, formulario para crear películas. Uno con mensajes de error y uno exitoso. 2. 3 Screenshots del final de la sección 5.3, formulario de edición de películas. Uno antes, uno durante la edición y uno después de la edición. 3. 4 Screenshots del final de la sección 5.4, formulario de borrar películas. Uno antes de borrar, uno después de borrar, uno de los métodos del controlador y uno de los métodos del handler.
4%	En los screenshots debe aparecer su nombre de usuario de manera clara e.g. nombre y apellidos, correo electrónico, usuario de windows, o usuario de GitHub que refleje su nombre claramente y no un pseudónimo o nombre críptico.
0.25%	Agregue al reporte el enlace a su repositorio de GitHub.
9%	Agregue un pequeño resumen, no más de 200 palabras indicando claramente 1. Dos cosas que no sabía y aprendió en el laboratorio 2. Una cosa que se le hizo difícil de realizar y explique por qué fue difícil. 3. Una cosa que se le hizo fácil de realizar y explique por qué fue fácil. 4. Indique cuánto tiempo tardó en realizar el laboratorio.