UNIVERSIDAD DE COSTA RICA

ECCI Escuela de Ciencias de la Computación e Informática

Curso: CI-0126
Grupo: 01

I ciclo 2022

Ingeniería de software

Laboratorio 6: Formularios

Resumen

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.

Prerrequisitos

Debe haber completado el laboratorio #5 y utilizar el mismo proyecto y base de datos para realizar este laboratorio. Todos los commits a este laboratorio deben utilizar en la descripción el tag: Laboratorio #6

Primera parte - Comprendiendo formularios en HTML

Para comprender qué son los formularios en HTML les recomiendo ver los primeros 18min del video TODO lo que NECESITAS saber de los FORMULARIOS en HTML [Curso de HTML desde cero]. Estos primeros minutos son una introducción a los formularios. Sin embargo se les recomienda también ver hasta el minuto 34 para poder comprender mejor los diferentes tipos de inputs que se pueden utilizar. Pero pueden ver el video completo para complementar estos temas con la sección de atributos en los formularios.
Nota: Este video habla sobre formularios en HTML 5

Segunda parte - Formularios en Razor pages

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 Países que usó laboratorio 5, esto lo puede hacer desde el Server Explorer. En caso que esté cerrada, abra la conexión.

Modificando el modelo

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

```
namespace Laboratorio5.Models
{
    7 references
    public class PaisModel
    {
        2 references
        public int Id { get; set; }
        2 references
        public string Nombre { get; set; }
        2 references
        public string Continente { get; set; }
        2 references
        public string Idioma { 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;
using System.ComponentModel.DataAnnotations;

namespace Laboratorio5.Models
{
    12 references
    public class PaisModel
    {
        2 references
        public int Id { get; set; }

        [Required(ErrorMessage = "Debe proporcionar un nombre")]
        [DisplayName("Nombre del país:")]
        7 references
        public string Nombre { get; set; }

        [Required(ErrorMessage = "Debe seleccionar un continente")]
        [DisplayName("Cotinente:")]
        6 references
        public string Continente { get; set; }

        [Required(ErrorMessage = "Debe ingresar el idioma")]
        [DisplayName("Idioma:")]
        [RegularExpression("^[a-zA-Z]+$", ErrorMessage = "No puede ingresar numeros")]
        6 references
        public string Idioma { get; set; }
    }
}
```

Observaciones del código:

- System.ComponentModel.DataAnnotations incluye las librerías que permiten utilizar los elementos *Requiered*, *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.
- 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 nombre del país, entonces automáticamente se restringe la acción y se muestra el mensaje especificado en *ErrorMessage*.
- 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.
- El elemento *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.

- 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.

Inserción de un país datos en la base de datos

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

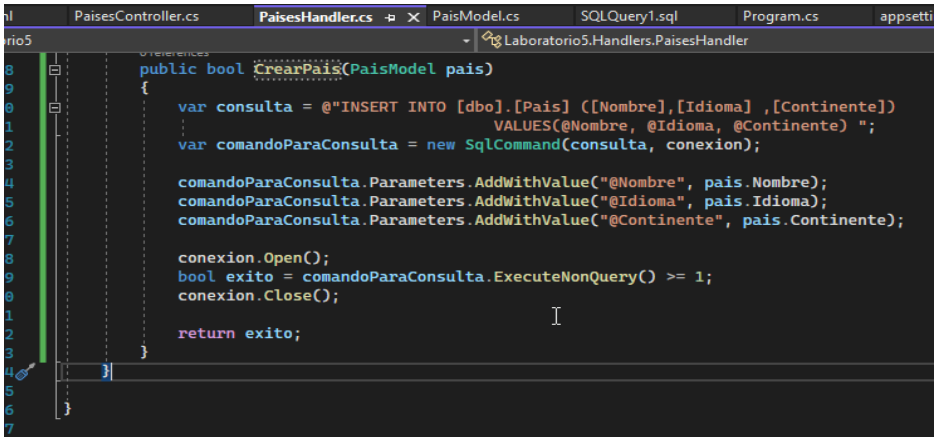
```
public bool CrearPais(PaisModel pais)
    {
        var consulta = @"INSERT INTO [dbo].[Pais] ([Nombre],[Idioma] ,[Continente])
                                VALUES(@Nombre, @Idioma, @Continente) ";
        var comandoParaConsulta = new SqlCommand(consulta, conexion);

        comandoParaConsulta.Parameters.AddWithValue("@Nombre", pais.Nombre);
        comandoParaConsulta.Parameters.AddWithValue("@Idioma", pais.Idioma);
        comandoParaConsulta.Parameters.AddWithValue("@Continente", pais.Continente);

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

        return exito;
    }
```

Se verá de la siguiente manera:



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 un país 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.

Método para crear países 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 el país 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 países, agregue los siguientes métodos:

Método 1: método para enlazar la vista del formulario

Como vimos en formularios anteriores vamos a crear un método que nos permita obtener la vista:

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

Método 2: 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:

PaisesController.cs | PaisesHandler.cs | PaisModel.cs | SQLQuery1.sql | Program.cs | appsettings.json

Laboratorio5.Controllers.PaisesController | CrearPais(PaisModel pais)

```
        [HttpPost]
        0 references
        public ActionResult CrearPais(PaisModel pais)
        {
            ViewBag.ExitoAlCrear = false;
            try
            {
                if (ModelState.IsValid)
                {
                    PaisesHandler paisesHandler = new PaisesHandler();
                    ViewBag.ExitoAlCrear = paisesHandler.CrearPais(pais);

                    if (ViewBag.ExitoAlCrear)
                    {
                        ViewBag.Message = "El pais" + " " + pais.Nombre + " fue creado con éxito" ;
                        ModelState.Clear();
                    }
                }
                return View();
            }
            catch
            {
                ViewBag.Message = "Algo salió mal y no fue posible crear el  país";
                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 que común de los laboratorios anteriores. No obstante, el segundo método también desplegará la misma vista que el primero, pero únicamente después que a través de un formulario haga post direccionado 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 del país.

Formulario con auxiliares de HTML en Razor

Es hora de crear el formulario para esto cree en el folder: Views/Paises la nueva vista llamada CrearPais, en este use el siguiente codigo pero analice qué está haciendo:

```
@model Laboratorio5.Models.PaisModel
@{
    ViewBag.Title = "Crear País";
    List<SelectListItem> listaContinentes = new List<SelectListItem>() { new
    SelectListItem { Text = "América" }, new SelectListItem { Text = "Asia" }, new
    SelectListItem { Text = "Europa" }, new SelectListItem { Text = "Oceanía" }, new
    SelectListItem { Text = "Antártida" }, new SelectListItem { Text = "África" }};
}
<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("CrearPais", "paises", FormMethod.Post, new { enctype =
"multipart/form-data" }))
        {
            @Html.AntiForgeryToken()
            <h1>Formulario de creación de países</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>
```

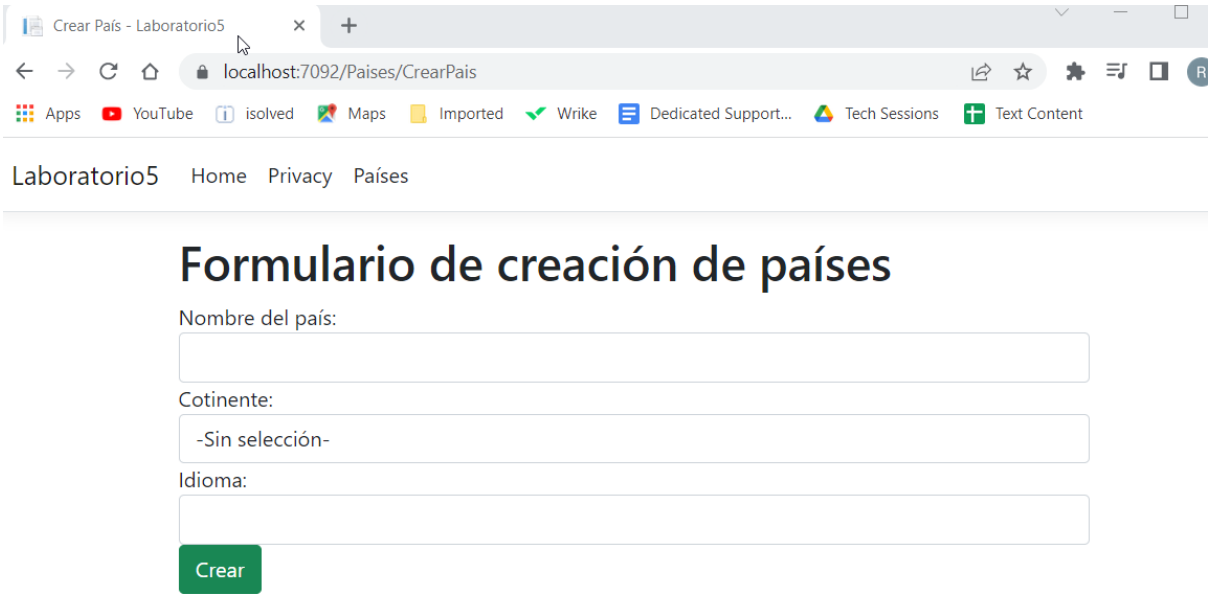
```
            <div class="form-group">
                @Html.LabelFor(model => model.Continente)
                @Html.DropDownListFor(model => model.Continente, listaContinentes, "-Sin
selección-", new { @class = "form-control" })
                @Html.ValidationMessageFor(model => model.Continente, "", new { @class =
"text-danger" })
            </div>
            <div class="form-group">
                @Html.LabelFor(model => model.Idioma)
                @Html.TextBoxFor(model => model.Idioma, new { @class =  "form-control" })
                @Html.ValidationMessageFor(model => model.Idioma, "", 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:

1. @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
2. 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.
3. Los métodos restantes son análogos, lo que cambia es el tipo de auxiliar. Existen otros como, por ejemplo: radio botones, checkbox o los drop down como el que se muestra en este mismo ejemplo. 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 **paises/crearpais**, por ejemplo https://localhost:7092/paises/crearpais . La vista debe verse similar a esta:



Cree un país, y luego de click en el menú países y asegúrese que puede ver el nuevo país. **Tome un screenshot del país creado y adjúntelo en la documentación que entregará del laboratorio.**

Edición un país

Para iniciar con lo correspondiente a lograr editar países, es necesario crear los elementos necesarios que lo permitan. En este caso, se hará una nueva vista para acceder a editar un país en específico. En resumen, se requiere; un método para invocar la consulta tipo UPDATE en SQL; los métodos del controlador; la vista respectiva.

Agregar método en el handler

Agregue el siguiente método en su paisHandler para editar un país:

```
1 reference
public bool EditarPais(PaisModel pais)
{
    var consulta = @"UPDATE [dbo].[Pais] SET Nombre = @Nombre,
                        Idioma =@Idioma,
                        Continente = @Continente
                        WHERE Id=@Id ";
    var comandoParaConsulta = new SqlCommand(consulta, conexion);
    comandoParaConsulta.Parameters.AddWithValue("@Nombre", pais.Nombre);
    comandoParaConsulta.Parameters.AddWithValue("@Idioma", pais.Idioma);
    comandoParaConsulta.Parameters.AddWithValue("@Continente", pais.Continente);
    comandoParaConsulta.Parameters.AddWithValue("@Id", pais.Id);

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

    return exito;
}
```

Metodos edición en el controlador

Agregue los siguientes métodos a su controlador

```
[HttpGet]
0 references
public ActionResult EditarPais(int? identificador)
{
    ActionResult vista;
    try
    {
        var paisesHandler = new PaisesHandler();
        var pais = paisesHandler.ObtenerPaises().Find(model => model.Id == identificador);
        if (pais == null)
        {
            vista = RedirectToAction("Index");
        }
        else
        {
            vista = View(pais);
        }
    }
    catch
    {
        vista = RedirectToAction("Index");
    }
    return vista;
}

[HttpPost]
0 references
public ActionResult EditarPais(PaisModel pais)
{
    try
    {
        var paisesHandler = new PaisesHandler();
        paisesHandler.EditarPais(pais);
        return RedirectToAction("Index", "Paises");
    }
    catch
    {
        return View();
    }
}
```

Primer método

Segundo método

El primer método es análogo al *get* del crearpaís. Esta acción del controlador será la encargada de ejecutar la vista que contiene el formulario para editar un país. 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 país es el que se va editar (coincide con el id del modelo).

Además, en este caso se obtiene de la lista de todos los países, el país 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). No se brinda una “mejor” solución solo para que sea más sencillo de entender para usted y también la ejecución del laboratorio sea más rápida, y para que usted analice las alternativas.

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 del país. Si usted quiere puede agregar el código necesario para lograrlo, nuevamente, es opcional hacerlo. Una manera de lograrlo podría ser la tradicional con ViewBag TempData. Si usted lo desea, puede investigar un poco al respecto. Lo demás, es exactamente igual que el post de crear país

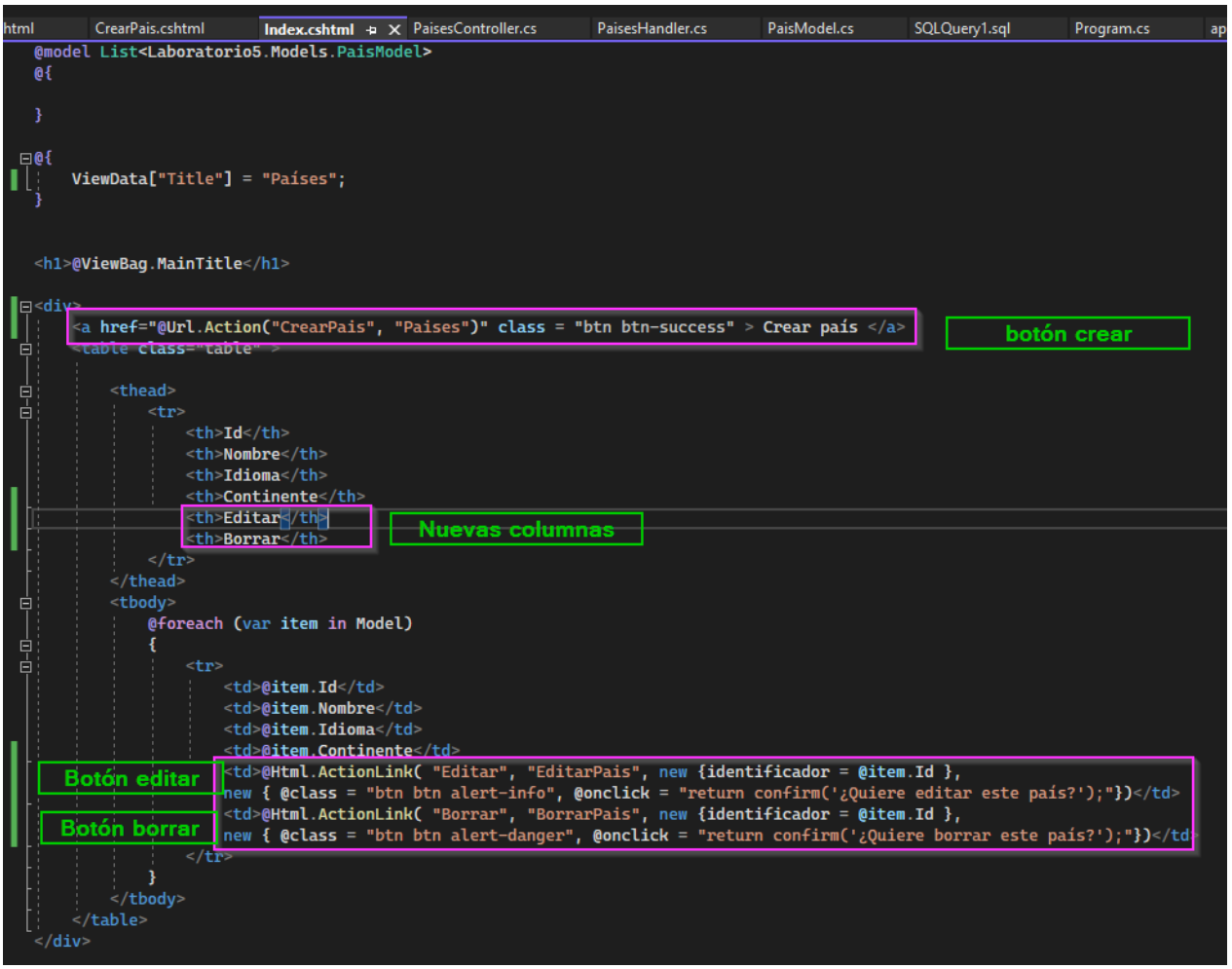
Vista para el formulario de edición

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 (EditarPais). En el contenido de vista, copie exactamente el mismo código de crear país y solo debe hacer 5 cambios:

1. Borre las líneas de código que son para mostrar el mensaje de éxito o error al crear el país (el if-else del inicio).
2. En Html.BeginForm("crearPais" …) cambie “crearPais” por el nombre de la acción del controlador de editar (“editarPais”)
3. Cambie la etiqueta H1 para que no diga algo relacionado a editar países.
4. Cambie el texto del botón para que diga algo relacionado a aplicar los cambios en lugar de “crear”.
5. 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.

Modificando la vista index

Para evitar estar escribiendo los URL completos, vamos a agregar botones a la vista Index creada en el laboratorio 5. Vamos a crear 3 botones: Crear país, Editar, Borrar. Para hacer esto siga el siguiente código:

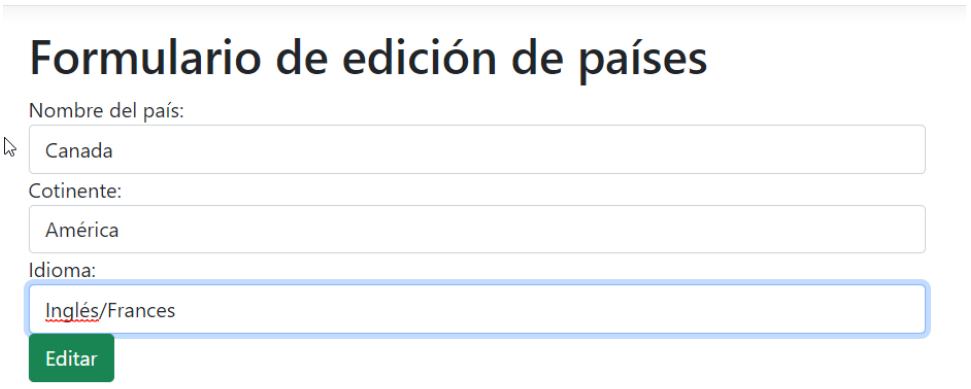


Observe que básicamente, se agregó un enlace donde se invoca a la acción del controlador y se le pasa el id del país respectivo. Este auxiliar genera una etiqueta HTML tipo con los elementos que se le indique (el class hace que el link tenga una apariencia de botón), usted puede ver esto en el navegador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

Comprobando Resultados

Ahora su página index debe lucir similar a esta:

Luego intente editar un país, tome screenshots del antes la edición (index view), el cambio que está haciendo en la página editarPais y el resultado obtenido en la vista index de países. Ejemplo



Borrar un país

En este momento la vista index se creó el botón para borrar países, cree el método tipo ***post*** para borrar países en el controlador. Además cree el método para borrar países en el handler, el script de bases de datos a utilizar es el siguiente
Delete [dbo].[pais] where Id = @id

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

Entregable

Para este laboratorio se debe compartir el repositorio usado con el asistente y la profesora a los siguientes usuarios de github:

- rebeca-ov
- ChristianRojasRios

Cree un pequeño documento con diferentes screenshots solicitados en las instrucciones del laboratorio.

Referencias

- Laboratorio de Ingeniería de software, realizado por Edwin Brenes
- MVC .NET Core turorials https://docs.microsoft.com/en-us/aspnet/core/tutorials/first-mvc-app/adding-model?view=aspnetcore-6.0&tabs=visual-studio