TI-Capital Humano

Desarrollador .Net

Consumir Web API desde Aplicación Web MVC - Express

Contenido

1. Introducción

El objetivo de esta guía es que se pueda construir una aplicación CRUD - Web MVC consumiendo una Web API con ASP .Net Core 6, de una manera sencilla, apoyándose en lo más posible con la generación de código, es decir utilizando lo más posible las plantillas y el scarfolding.

En el ejemplo, las acciones CRUD serán para la tabla Estados de la base de datos InstitutoTich.

La URL de Web API se colocará en el archivo appsetting.json para mayor flexibilidad en el mantenimiento.

Al ser un servicio web de tipo Rest el que se va a consumir, esto se hará por medio de la clase HttpClient.

Se creará una clase que contenga los métodos CRUD y que consuma la Web API, y el controlador utilizará esta clase para proporcionarle el modelo a las Vistas o bien del modelos recibidos enviárselo a esta clase. Los métodos se definirán asíncronos para estar en concordancia con el Controlador.

Si necesita conocer más acerca de Programación Asincrona, revisar el material “Guia Rápida de Programación Asincrona”.

2. Crear Web API Core

Crear la Web API de manera express como se indica en la guía rápida **“Crear Web API con .Net Core y EntityFramework”**

3. Crear Aplicación Web Core MVC

Crear la Aplicación Web MVC de manera express como se indica en la guía rápida **“Crear Aplicación Web con .Net Core 6 y EntityFramework”**

4. Adaptar la Aplicación Web Core MVC

4.1. Agregar la URL en appsettings.json

Agregar un parámetro en el archivo appsettings.json que contenga la url de la web api. Además como ya no se accesará la base de datos directamente, ya que esto lo

hace la web API , eliminar la cadena de conexión de este archivo, quedando como sigue:

```
{
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft.AspNetCore": "Warning"
    }
  },
  "AllowedHosts": "*",
  "urlWebAPI": "http://localhost:5175/api/Estados"
}
```

4.2. Crear Clase de Negocio para Consumir Web API

Aunque la web api se puede consumir desde el controlador, sería una violación al patrón de diseño MVC, por lo que es mejor crear una clase con los métodos necesarios para consumir la web API y satisfacer las necesidades del controlador. En este caso, vamos a crear métodos asíncronos para que estén en concordancia con el controlador.

4.2.1. Crear la Clase de Negocio

Crear la Clase de Negocio en la carpeta Models, en este caso NEstado.

La clase tendrá algunas variables globales que serán necesarias en todos los métodos, su método constructor para cargar la url de la web api, y los métodos asíncronos necesarios para un crud.

```
public class NEstados
{
    private static readonly HttpClient client;
    private string _urlWebAPI;
    1 referencia
    public NEstados()...
    1 referencia
    public async Task<List<Estados>> Consultar()...
    4 referencias
    public async Task<Estados> Consultar(int? id)...
    1 referencia
    public async Task<Estados> Agregar(Estados estado)...
    1 referencia
    public async Task Actualizar(Estados estado)...
    1 referencia
    public async Task Eliminar(int id)...

}
```

4.2.2. Definir Variables Globales

Como la Web Api se va a consumir utilizando un objeto de tipo HttpClient, definir una variable global, así como una variable que contendrá la URL de la web API.

```
public class NEstados
{
    private static readonly HttpClient client;
    private string _urlWebAPI;
```

4.2.3. Crear el constructor

En el método constructor se cargará la url de la web api, que se especificó en el archivo appsettings.json. Para ello se utilizar un objeto `ConfigurationBuilder.`

Una vez cargado el appsettings.json, se extraerá la url de la api desde la sección correspondiente:

```
public NEstados()
{
    var builder = new ConfigurationBuilder().SetBasePath(Directory.GetCurrentDirectory()).AddJsonFile("appsettings.json").Build();
    _urlWebAPI = builder.GetSection("urlWebAPI").Value;
}
```

4.2.4. Crear el método Consultar()

Este método no recibe parámetros y regresará una lista de objetos, en este caso de Estados. Para ello se consumirá la web api con el verbo http Get, que es como está definida en la web api.

El procedimiento básicamente sigue los siguientes pasos:

a) Instanciar la clase `HttpClient`
b) Invocar el método `GetAsync` del objeto `HttpClient`, el cual envía una solicitud GET al URI especificado como parámetro, como una operación asincrónica, obteniendo el objeto `HttpResponseMessage`
c) Validar el valor de la propiedad IsSuccessStatusCode del `HttpResponseMessage`, para verificar que la operación haya sido ejecutada con éxito, y si es el caso leer el contenido de `HttpContent`
d) Invocar al método `ReadAsStringAsync` del objeto `HttpContent` el cual serializa el contenido HTTP en una cadena como una operación asincrónica.
e) Deserealizar el objeto recibido, en este caso una lista de estados

```
public async Task<List<Estados>> Consultar()
{
    var estados = new List<Estados>();
    try
    {
        //Instancia el objeto HttpClient
        using (var client = new HttpClient())
        {
            //Invocar el método GetAsync del objeto HttpClient, el cual envía una solicitud GET al
            //URI especificado como parámetro, como una operación asincrónica
            var responseTask = await client.GetAsync(_urlWebAPI);

            //Se Obtiene el objeto HttpResponseMessage en la variable responseTask

            // Validar el valor de la propiedad IsSuccessStatusCode del HttpResponseMessage
            // para verificar que la operación haya sido ejecutada con éxito
            if (responseTask.IsSuccessStatusCode)
            {
                //Invocar al método ReadAsStringAsync del objeto HttpContent el cual serializa
                //el contenido HTTP en una cadena como una operación asincrónica.
                var respuestaJson = await responseTask.Content.ReadAsStringAsync();

                //Deserealizar el objeto recibido, en este caso una lista de estados
                estados = JsonConvert.DeserializeObject<List<Estados>>(respuestaJson);
            }
            else //web api envió error de respuesta
            {
                throw new Exception($"WebAPI. Respondio con error.{responseTask.StatusCode}");
            }
        }

    }
    catch (Exception ex)
    {
        throw new Exception($"WebAPI. Respondio con error.{ex.Message}");
    }
    return estados;
}
```

4.2.5. Crear el método Consultar(int id)

Este método recibe como parámetro el id del Objeto que se quiera consultar, en este caso el id del estado. y regresará el objeto solicitado, en este caso un Estados. Para ello se consumirá la web api también con el verbo http Get pero se le concatenará el id a la uri, que es como está definida en la web api.

Por lo anterior, el procedimiento, sigue básicamente los mismos pasos que para el método Consultar().

```
public async Task<Estados> Consultar(int? id)
{
    var estado = new Estados();
    try
    {
        //Instancia el objeto HttpClient
        using (var client = new HttpClient())
        {
            //Invocar el método GetAsync del objeto HttpClient, el cual envía una solicitud GET al
            //URI especificado como parámetro, como una operación asincrónica.
            //En este caso se concatena el id recibido como parámetro
            var responseTask = await client.GetAsync(_urlWebAPI + $"/{id}");

            //Obtenemos el objeto HttpResponseMessage en la variable responseTask

            // Validar el valor de la propiedad IsSuccessStatusCode del HttpResponseMessage
            // para verificar que la operación haya sido ejecutada con éxito
            if (responseTask.IsSuccessStatusCode)
            {
                //Invocar al método ReadAsStringAsync del objeto HttpContent el cual serializa
                //el contenido HTTP en una cadena como una operación asincrónica.
                var respuestaJson = await responseTask.Content.ReadAsStringAsync();

                //Deserealizar el objeto recibido, en este caso un estado
                estado = JsonConvert.DeserializeObject<Estados>(respuestaJson);
            }
            else //web api envió error de respuesta
            {
                throw new Exception($"WebAPI. Respondio con error.{responseTask.StatusCode}");
            }

        }
    }
    catch (Exception ex)
    {
        throw new Exception($"WebAPI. Respondio con error.{ex.Message}");
    }
    return estado;
}
```

4.2.6. Crear el método Eliminar(int id)

Este método recibe como parámetro el id del Objeto que se quiera eliminar, en este caso el id del estado. Para ello se consumirá la web api con el verbo http Delete pero se le concatenará el id al uri, que es como está definida en la web api.

El procedimiento es todavía más sencillo que el de consultar:

a) Instanciar la clase `HttpClient`
b) Invocar el método `DeleteAsync` del objeto `HttpClient`, el cual envía una solicitud DELETE al URI especificado como parámetro, como una operación asincrónica, obteniendo el objeto `HttpResponseMessage`
c) Validar el valor de la propiedad IsSuccessStatusCode del `HttpResponseMessage`, para verificar que la operación haya sido ejecutada con éxito, y en caso contrario mandar una excepción.

```
public async Task Eliminar(int id)
{
    try
    {
        //Instancia el objeto HttpClient
        using (var client = new HttpClient())
        {
            //Invocar el método DeleteAsync del objeto HttpClient, el cual envía una solicitud DELETE al
            //URI especificado como parámetro, incluyéndo el id del estado, como una operación asincrónica,
            var responseTask = await client.DeleteAsync(_urlWebAPI + $"/{id}");

            // Verificamos que si la operación no fue exitosa se levanta una excepción
            if (!responseTask.IsSuccessStatusCode)
            {
                throw new Exception($"WebAPI. Respondio con error.{responseTask.StatusCode}");
            }
        }
    }
    catch (Exception ex)
    {
        throw new Exception($"WebAPI. Respondio con error.{ex.Message}");
    }

}
```

4.2.7. Crear el método Agregar(Objeto objeto)

Este método recibe como parámetro el Objeto que se quiere agregar, en este caso un Estado. Para ello se consumirá la web api con el verbo http Post pero con el objeto a agregar en el httpContent, que es como está definida en la web api.

El procedimiento básicamente sigue los siguientes pasos:

a) Instanciar la clase `HttpClient`
b) Crear un objeto `HttpContent` instanciando la clase `StringContent`, la cual es una clase que proporciona contenido HTTP basado en una cadena. Este contenido se crea con el objeto Estado que se está recibiendo
c) Asignar el tipo `application/json` a la propiedad `ContentType` del encabezado de HttpContent
d) Invocar el método `PostAsync` del objeto `HttpClient`, el cual envía una solicitud POST al URI especificado como parámetro, como una operación asincrónica, asimismo le envía el contenido (objeto estado) dentro del `httpContent`
e) Verificar que la operación haya sido ejecutada con éxito, para proceder a obtener el resultado recibido desde la web api, en caso contrario enviar una excepción
f) Invocar al método `ReadAsStringAsync` del objeto `HttpContent` el cual serializa el contenido HTTP en una cadena como una operación asincrónica. Hay que tener en cuenta que el Web API regresa un objeto tipo Estado
g) Deserealizar el objeto recibido, en este caso un estado

```
public async Task<Estados> Agregar(Estados estado)
{
    try
    {
        //Instanciar la clase HttpClient
        using (var client = new HttpClient())
        {
            //Crear un objeto HttpContent instanciando la clase StringContent,
            //la cual es una clase que proporciona contenido HTTP basado en una cadena.
            //Este contenido se crea con el objeto Estado que se está recibiendo
            HttpContent httpContent = new StringContent(JsonConvert.SerializeObject(estado), Encoding.UTF8);

            //Asignar el tipo application/json a la propiedad ContentType del encabezado de HttpContent
            httpContent.Headers.ContentType = new MediaTypeHeaderValue("application/json");

            //Invocar el método PostAsync del objeto HttpClient, el cual envía una solicitud POST al
            //URI especificado como parámetro, como una operación asincrónica,
            //asimismo le envía el contenido (objeto estado) dentro del httpContent
            var responseTask = await client.PostAsync(_urlWebAPI, httpContent);

            // Verificar que la operación haya sido ejecutada con éxito,
            // para proceder a obtener el resultado enviado
            // desde la web api, en caso contrario enviamos una excepción
            if (responseTask.IsSuccessStatusCode)
            {
                //Invocar al método ReadAsStringAsync del objeto HttpContent el cual serializa
                //el contenido HTTP en una cadena como una operación asincrónica.
                //Hay que tener en cuenta que el Web API regresa un objeto tipo Estado
                var respuestaJson = await responseTask.Content.ReadAsStringAsync();
                //Deserealizar el objeto recibido, en este caso un estado
                estado = JsonConvert.DeserializeObject<Estados>(respuestaJson);
            }
            else //web api envió error de respuesta
            {
                throw new Exception($"WebAPI. Respondio con error.{responseTask.StatusCode}");
            }
        }
    }
    catch (Exception ex)
    {
        throw new Exception($"WebAPI. Respondio con error.{ex.Message}");
    }
    return estado;
}
```

4.2.8. Crear el método Actualizar(Objeto objeto)

Este método recibe como parámetro el Objeto(con los datos nuevos) que se quiere actualizar, en este caso un Estado. Para ello se consumirá la web api con el verbo http Put, pero con el objeto que se actualizará dentro del httpContent, que es como está definida en la web api.

El procedimiento básicamente es el mismo del método Agregar():

a) Instanciar la clase `HttpClient`
b) Crear un objeto `HttpContent` instanciando la clase `StringContent`, la cual es una clase que proporciona contenido HTTP basado en una cadena. Este contenido se crea con el objeto Estado que se está recibiendo
c) Asignar el tipo application/json a la propiedad ContentType del encabezado de `HttpContent`
d) Invocar el método `PutAsync` del objeto `HttpClient`, el cual envía una solicitud PUT al URI especificado como parámetro, como una operación

asincrónica, asimismo le envía el contenido (objeto estado) dentro del `httpContent`

e) Verificar que la operación haya sido ejecutada con éxito, en caso que no sea así enviar una excepción

```
public async Task Actualizar(Estados estado)
{
    try
    {
        //Instanciar la clase HttpClient
        using (var client = new HttpClient())
        {
            //Crear un objeto HttpContent instanciando la clase StringContent,
            //la cual es una clase que proporciona contenido HTTP basado en una cadena.
            //Este contenido se crea con el objeto Estado que se está recibiendo
            HttpContent httpContent = new StringContent(JsonConvert.SerializeObject(estado), Encoding.UTF8);

            //Asignar el tipo application/json a la propiedad ContentType del encabezado de HttpContent
            httpContent.Headers.ContentType = new MediaTypeHeaderValue("application/json");

            //Invocar el método PutAsync del objeto HttpClient, el cual envia una solicitud PUT al
            //URI especificado como parámetro, incluyéndo el id del estado, como una operación asincrónica,
            //asimismo le envía el contenido (objeto estado) dentro del httpContect
            var responseTask = await client.PutAsync(_urlWebAPI + $"/{estado.Id}", httpContent);

            // Verificamos que si la operación no fue exitosa se levanta una excepción
            if (!responseTask.IsSuccessStatusCode)
            {
                throw new Exception($"WebAPI. Respondio con error.{responseTask.StatusCode}");
            }
        }

    }
    catch (Exception ex)
    {
        throw new Exception($"WebAPI. Respondio con error.{ex.Message}");
    }
}
```

4.3. Actualizar el Controlador

Actualizar el controlador para que en lugar de obtener los datos desde `dbcontext`, hacerlo desde la clase de negocio creada que a su vez es el que consume el web api.

4.3.1. Cambiar el DBContext por la clase de negocio

Cambiar la variable global donde se almacena el objeto dbcontext por la clase de negocio, así mismo modificar el método constructor para que ahí se instancia la clase de negocio.

```
public class EstadosController : Controller
{
    private readonly NEstados _nEstados;

    0 referencias
    public EstadosController()
    {
        _nEstados = new NEstados();
    }
```

4.3.2. Modificar el método Index

Modificar el método Index para que la lista de objetos a mostrar se obtenga por medio del método Consultar() del objeto de negocio.

```
public async Task<IActionResult> Index()
{
    var estados =  await _nEstados.Consultar();
    return View(estados);
}
```

4.3.3. Modificar el método Details

Modificar el método Details para que el objeto a mostrar se obtenga por medio del método Consultar(int id) del objeto de negocio.

```
public async Task<IActionResult> Details(int? id)
{
    if (id == null)
    {
        return NotFound();
    }

    var estados = await _nEstados.Consultar(id);
    if (estados == null)
    {
        return NotFound();
    }

    return View(estados);
}
```

4.3.4. Modificar el método Delete

Modificar el método Delete para la solicitud GET para que el objeto a mostrar se obtenga por medio del método Consultar(int id) del objeto de negocio.

```
// GET: Estados/Delete/5
0 referencias
public async Task<IActionResult> Delete(int? id)
{
    if (id == null)
    {
        return NotFound();
    }
    var estados = await _nEstados.Consultar(id);
    if (estados == null)
    {
        return NotFound();
    }
    return View(estados);
}
```

Modificar el método Delete para la solicitud POST para que se utilice el método Eliminar(int id) del objeto de negocio.

```
// POST: Estados/Delete/5
[HttpPost, ActionName("Delete")]
[ValidateAntiForgeryToken]
0 referencias
public async Task<IActionResult> DeleteConfirmed(int id)
{
    await _nEstados.Eliminar(id);

    return RedirectToAction(nameof(Index));
}
```

4.3.5. Modificar el método Create

Modificar el método Create para la solicitud POST, a fin de que se utilice el método Agregar(Objeto objeto) del objeto de negocio.

```
// POST: Estados/Create
// To protect from overposting attacks, enable the specific properties you want to bind to.
// For more details, see http://go.microsoft.com/fwlink/?LinkId=317598.
[HttpPost]
[ValidateAntiForgeryToken]
0 referencias
public async Task<IActionResult> Create([Bind("Id,Nombre")] Estados estados)
{
    if (ModelState.IsValid)
    {
        var edo = await _nEstados.Agregar(estados);
        return RedirectToAction(nameof(Index));
    }
    return View(estados);
}
```

4.3.6. Modificar el método Edit

Modificar el método Edit para la solicitud GET para que el objeto a mostrar se obtenga por medio del método Consultar(int id) del objeto de negocio.

```
public async Task<IActionResult> Edit(int? id)
{
    if (id == null)
    {
        return NotFound();
    }
    var estados = await _nEstados.Consultar(id);
    if (estados == null)
    {
        return NotFound();
    }
    return View(estados);
}
```

Modificar el método Edit para la solicitud POST para que se utilice el método Actualizar(Objet objeto) del objeto de negocio.

```
// POST: Estados/Edit/5
// To protect from overposting attacks, enable the specific properties you want to bind to.
// For more details, see http://go.microsoft.com/fwlink/?LinkId=317598.
[HttpPost]
[ValidateAntiForgeryToken]
0 referencias
public async Task<IActionResult> Edit(int id, [Bind("Id,Nombre")] Estados estados)
{
    if (id != estados.Id)
    {
        return NotFound();
    }
    if (ModelState.IsValid)
    {
        try
        {
            await _nEstados.Actualizar(estados);
        }
        catch (DbUpdateConcurrencyException)
        {
            if (!EstadosExists(estados.Id))
            {
                return NotFound();
            }
            else
            {
                throw;
            }
        }
        return RedirectToAction(nameof(Index));
    }
    return View(estados);
}
```

De paso es necesario modificar el método privado ObjetoExists para que utilice el método Consultar(int id) del objeto de Negocio.

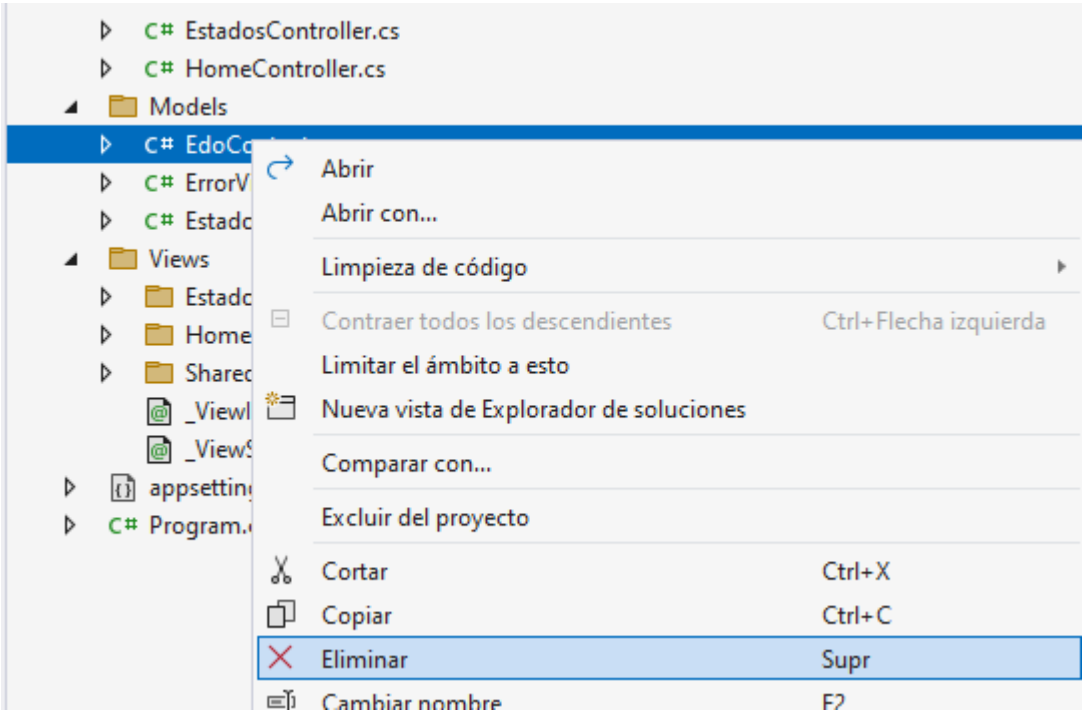
```
private bool EstadosExists(int id)
{
    return (_nEstados.Consultar(id) != null ? true : false);
}
```

Como se podrá observar el los métodos Consutar(int id), Edit(int id), Delete(int id) para la solicitud Get son los mismos, por lo que se puede crear un método privado con este código para que se invoque en estos tres métodos en lugar de estar repitiendo código:

```
private async Task<IActionResult> Consutar(int? id)
{
    if (id == null)
    {
        return NotFound();
    }
    var estados = await _nEstados.Consultar(id);
    if (estados == null)
    {
        return NotFound();
    }
    return View(estados);
}
// GET: Estados/Details/5
0 referencias
public async Task<IActionResult> Details(int? id)
{
   return await Consutar(id);
}
// GET: Estados/Edit/5
0 referencias
public async Task<IActionResult> Edit(int? id)
{
    return await Consutar(id);
}
// GET: Estados/Delete/5
0 referencias
public async Task<IActionResult> Delete(int? id)
{
    return await Consutar(id);
}
```

4.4. Eliminar la Clase DBContext

Como se mencionó anteriormente, como ya no se accesará la base de datos directamente desde esta aplicación, ya no es necesario la clase de contexto, por lo que hay que eliminarla.

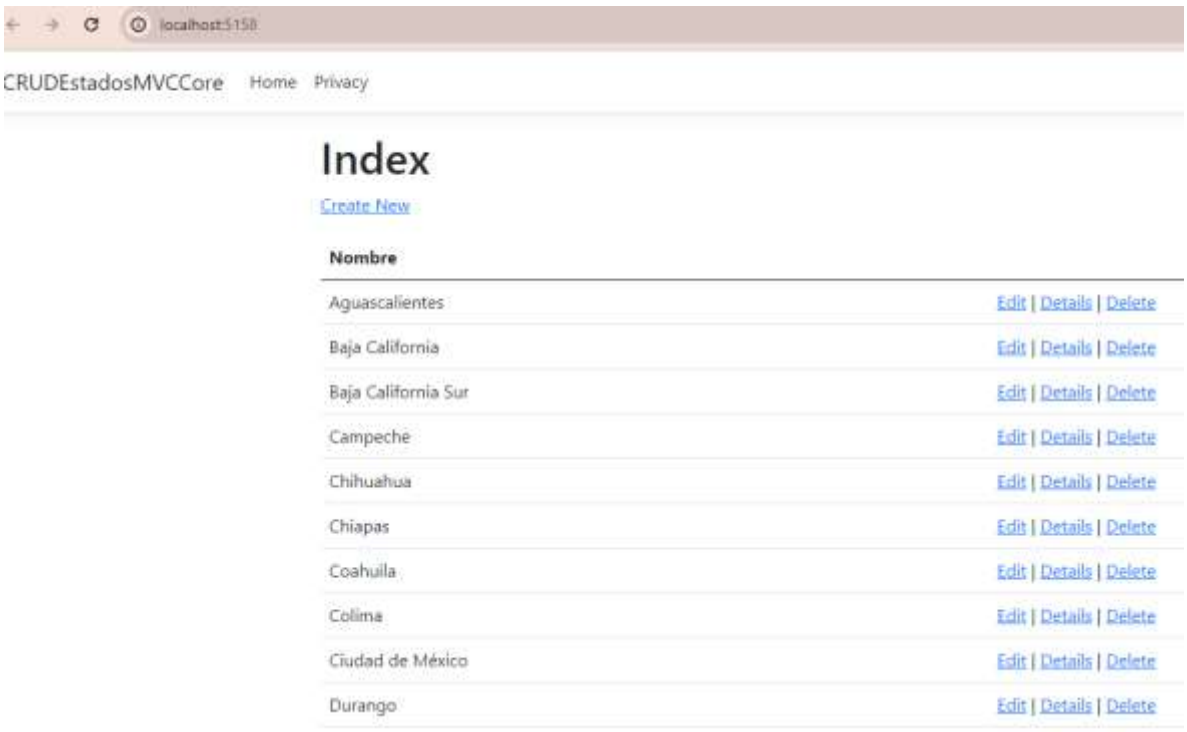


Además en el archivo **program.cs** también habrá que eliminar el servicio DBContext y los usings correspondientes.

```
using Microsoft.EntityFrameworkCore;
using CRUDEstadosMVCCore.Models;

var builder = WebApplication.CreateBuilder(args);

// Add services to the container.
builder.Services.AddControllersWithViews();

builder.Services.AddDbContext<EdoContext>(opcion => {
    opcion.UseSqlServer(builder.Configuration.GetConnectionString("InstitutoTich"));
});
var app = builder.Build();
```

4.5. Ejecutar la Aplicación Web MVC

Con lo hecho anteriormente esta lista la Aplicación Web con acciones y vistas de CRUD


localhost:5158
CRUDEstadosMVCCore
Home
Privacy
Index
Create New
Nombre
Aguascalientes
Baja California
Baja California Sur
Campeche
Chihuahua
Chiapas
Coahuila
Colima
Ciudad de México
Durango
Edit | Details | Delete