# Índice

Contenido

[Índice 1](#_Toc146198058)

[Modelo de datos 2](#_Toc146198059)

[Arquitectura del proyecto 2](#_Toc146198060)

[Crear la base de datos con EF 8](#_Toc146198061)

[Clase DataContext 9](#_Toc146198062)

[Appsettings 10](#_Toc146198063)

[Inyección de dependencias Servicio SQlServer 11](#_Toc146198064)

[Ejecutar proyectos de forma múltiple 11](#_Toc146198065)

[Migración de entidades hacia la base de datos SQL SERVER 12](#_Toc146198066)

[Crear un nuevo Branch en GIT para publicar la nueva versión del proyecto en el repositorio GIT. 12](#_Toc146198067)

[Creando los primeros métodos en el primer controlador 15](#_Toc146198068)

[Creando nuestros primeros componentes en Blazor 18](#_Toc146198069)

[Completando las acciones de crear, editar y borrar países 30](#_Toc146198070)

# Modelo de datos

Vamos a crear un sencillo sistema de ventas que va a utilizar el siguiente modelo de datos:Diagrama

Descripción generada automáticamente

# Arquitectura del proyecto

SQL Server on Azure

.NET Core 7 API

Blazor WEB Application

MAUI Mobile Application

Windows

Android

iOS

Mac



Crear proyecto

Vamos a crear esta estructura en Visual Studio (asegúrese de poner todos los proyectos en el mismo directorio C://Projects

Crear un nuevo repositorio GITHUB, usar gitignore, copiar ruta, repositorio público

Clonar proyecto git desde Visual Studio C://Projects/Market

* Nuevo proyecto Solution Blank llamado **Market**. Dentro de C://Projects al final la .sln queda el ícono dentro de Projects🡪(Market.sln)

Interfaz de usuario gráfica, Tabla

Descripción generada automáticamente

* Sobre el ícono Market.sln del explorador de soluciones oprimimos click derecho y presionamos Open
* Click derecho sobre la solución y agregamos un nuevo proyecto tipo: **Class Library**, llamado **Market.Shared**

Ubicación: C:\Projects/Market (Borramos la class1.cs que se genera)

* Click derecho sobre la solución y agregamos un nuevo proyecto tipo: **ASP.NET Core Web API**, llamado **Market.API** Ubicación: C:\Projects/Market
* Click derecho sobre la solución y agregamos un nuevo proyecto tipo: **Blazor WebAssembly App**, llamado **Market.WEB** Ubicación: C:\Projects/Market

* Click derecho sobre la solución y agregamos un nuevo proyecto tipo: **.NET MAUI App**, llamado **Market.Mobile**. Ubicación: C:\Projects/Market

Después de la creación del proyecto Market.Mobile , debemos aceptar la licencia del SDK de Android:

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Así debe verse al final , la estructura de los proyectos en el Solution Explorer:

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

Hacemos el primer commit en nuestro repositorio. Pestaña Git Changes

\*(Si en Git Changes no se visualiza el árbol de carpetas de los proyectos, será necesario cerrar la solución, y abrirla de nuevamente) Commit All and Sync

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

# Crear la base de datos con EF

Diagrama

Descripción generada automáticamente

Code First y Database First. En este curso trabajaremos con EF Code First,

Documentación: <https://docs.microsoft.com/en-us/ef/core/get-started/aspnetcore/existing-db>

1. Empecemos creando en el proyecto **Market.Shared** la carpeta **Entities** y dentro de esta carpeta la entidad **Country**:

using System.ComponentModel.DataAnnotations;

namespace Market.Shared.Entities

{

public class Country

{

public int Id { get; set; }

[Display(Name = "País")]

[MaxLength(100, ErrorMessage = "El campo {0} debe tener máximo {1} caractéres.")]

[Required(ErrorMessage = "El campo {0} es obligatorio.")]

public string Name { get; set; } = null!;

}

}

# Clase DataContext

1. En el proyecto **API** creamos la carpeta **Data** y dentro de esta la clase **DataContext**:

using Microsoft.EntityFrameworkCore;

using Market.Shared.Entities;

namespace Market.API.Data

{

public class DataContext : DbContext

{

public DataContext(DbContextOptions<DataContext> options) : base(options)

{

}

public DbSet<Country> Countries { get; set; }

protected override void OnModelCreating(ModelBuilder modelBuilder)

{

base.OnModelCreating(modelBuilder);

modelBuilder.Entity<Country>().HasIndex(c => c.Name).IsUnique();

}

}

}

# Appsettings

1. Configurar el string de conexión en el **appsettings.Development.json** y en **appsettings.json** del proyecto **API**:

{

"ConnectionStrings": {

"DefaultConnection": "Server= MyServer;Database=Market;Encrypt=False;User Id=dba;Password=Abcd1234\*;"

},

"Logging": {

"LogLevel": {

"Default": "Information",

"Microsoft.AspNetCore": "Warning"

}

},

"AllowedHosts": "\*"

}

1. Agregar/verificar los paquetes al proyecto **API**:

Microsoft.EntityFrameworkCore.SqlServer

Microsoft.EntityFrameworkCore.Tools

# Inyección de dependencias Servicio SQlServer

1. Configurar la inyección del DataContext en la clase **Program** del proyecto **API**:

builder.Services.AddSwaggerGen();

builder.Services.AddDbContext<DataContext>(x => x.UseSqlServer("name=DefaultConnection"));

var app = builder.Build();

# Ejecutar proyectos de forma múltiple

1. En el desplegable Startup Projects seleccionar Market.API como proyecto de inicio, abrir Package Manager Console(Tool) , e igualmente elegir Market.API ,como Default Project

# Migración de entidades hacia la base de datos SQL SERVER

1. Correr los siguientes comandos en Package Manager Console:

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

add-migration InitialDb

update-database

# Crear un nuevo Branch en GIT para publicar la nueva versión del proyecto en el repositorio GIT.

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Tabla

Descripción generada automáticamente

Una captura de pantalla de una computadora

Descripción generada automáticamente

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

1. Hacemos nuestro segundo **Commit**. All and Sync y luego el Push en una nuevo branch(Si en Git Changes no se visualiza el árbol de carpetas de los proyectos, será necesario cerrar la solución, y abrirla de nuevamente)

Creando los primeros métodos en el primer controlador

1. En el proyecto **API** en la carpeta **Controllers** creamos la clase **CountriesController**:

using Microsoft.AspNetCore.Mvc;

using Microsoft.EntityFrameworkCore;

using Market.API.Data;

using Market.Shared.Entities;

namespace Market.API.Controllers

{

[ApiController]

[Route("/api/countries")]

public class CountriesController : ControllerBase

{

private readonly DataContext \_context.

public CountriesController(DataContext context)

{

\_context = context;

}

[HttpGet]

public async Task<ActionResult> Get()

{

return Ok(await \_context.Countries.ToListAsync());

}

}

}

[HttpGet("{id:int}")]

public async Task<ActionResult> Get(int id)

{

var country = await \_context.Countries.FirstOrDefaultAsync(x => x.Id == id);

if (country is null)

{

return NotFound();

}

return Ok(country);

}

[HttpPost]

public async Task<ActionResult> Post(Country country)

{

\_context.Add(country);

await \_context.SaveChangesAsync();

return Ok(country);

}

1. Agregamos estas líneas al **Program** del proyecto **API** para habilitar su consumo:

app.MapControllers();

app.UseCors(x => x

.AllowAnyMethod()

.AllowAnyHeader()

.SetIsOriginAllowed(origin => true)

.AllowCredentials());

app.Run();

1. Borramos las clases de **WeatherForecast**.
2. Probamos la creación y listado de paises por el **swagger** y por **Postman**.
3. Hacemos el **commit** de lo que llevamos.

# Creando nuestros primeros componentes en Blazor

1. Ahora vamos listar y crear países por la interfaz WEB.

Primero configuramos en el proyecto **WEB** la dirección por la cual sale nuestra **API**.

Verificar en cada proyecto el puerto por el cual se despliega, pues el puerto cambia en cada solución.

En mi caso la uri es: https://localhost:7000

**Market.WEB🡪 Program**

builder.Services.AddScoped(sp => new HttpClient { BaseAddress = new Uri("https://localhost:7000/") });

1. En el proyecto **WEB** creamos a carpeta **Repositories** y dentro de esta creamos la clase **HttpResponseWrapper** con el siguiente código:

using System.Net;

namespace Market.WEB.Repositories {

public class HttpResponseWrapper<T>

{

public HttpResponseWrapper(T? response, bool error, HttpResponseMessage httpResponseMessage)

{

Error = error;

Response = response;

HttpResponseMessage = httpResponseMessage;

}

public bool Error { get; set; }

public T? Response { get; set; }

public HttpResponseMessage HttpResponseMessage { get; set; }

public async Task<string?> GetErrorMessage()

{

if (!Error)

{

return null;

}

var codigoEstatus = HttpResponseMessage.StatusCode;

if (codigoEstatus == HttpStatusCode.NotFound)

{

return "Recurso no encontrado";

}

else if (codigoEstatus == HttpStatusCode.BadRequest)

{

return await HttpResponseMessage.Content.ReadAsStringAsync();

}

else if (codigoEstatus == HttpStatusCode.Unauthorized)

{

return " Debes loguearte para realizar esta acción";

}

else if (codigoEstatus == HttpStatusCode.Forbidden)

{

return " No tienes permisos para ejecutar esta acción";

}

return "Ha ocurrido un error inesperado";

}

}

}

1. En la misma carpeta creamos la interfaz **IRepository**:

namespace Market.WEB.Repositories{

public interface IRepository

{

Task<HttpResponseWrapper<T>> Get<T>(string url);

Task<HttpResponseWrapper<object>> Post<T>(string url, T model);

Task<HttpResponseWrapper<TResponse>> Post<T, TResponse>(string url, T model);

}

}

1. En la misma carpeta creamos la clase **Repository**:

using System.Text;

using System.Text.Json;

namespace Market.WEB.Repositories

{

public class Repository : IRepository

{

private readonly HttpClient \_httpClient;

private JsonSerializerOptions \_jsonDefaultOptions => new JsonSerializerOptions

{

PropertyNameCaseInsensitive = true,

};

public Repository(HttpClient httpClient)

{

\_httpClient = httpClient;

}

public async Task<HttpResponseWrapper<T>> Get<T>(string url)

{

var responseHttp = await \_httpClient.GetAsync(url);

if (responseHttp.IsSuccessStatusCode)

{

var response = await UnserializeAnswer<T>(responseHttp, \_jsonDefaultOptions);

return new HttpResponseWrapper<T>(response, false, responseHttp);

}

return new HttpResponseWrapper<T>(default, true, responseHttp);

}

public async Task<HttpResponseWrapper<object>> Post<T>(string url, T model)

{

var mesageJSON = JsonSerializer.Serialize(model);

var messageContet = new StringContent(mesageJSON, Encoding.UTF8, "application/json");

var responseHttp = await \_httpClient.PostAsync(url, messageContet);

return new HttpResponseWrapper<object>(null, !responseHttp.IsSuccessStatusCode, responseHttp);

}

public async Task<HttpResponseWrapper<TResponse>> Post<T, TResponse>(string url, T model)

{

var messageJSON = JsonSerializer.Serialize(model);

var messageContet = new StringContent(messageJSON, Encoding.UTF8, "application/json");

var responseHttp = await \_httpClient.PostAsync(url, messageContet);

if (responseHttp.IsSuccessStatusCode)

{

var response = await UnserializeAnswer<TResponse>(responseHttp, \_jsonDefaultOptions);

return new HttpResponseWrapper<TResponse>(response, false, responseHttp);

}

return new HttpResponseWrapper<TResponse>(default, !responseHttp.IsSuccessStatusCode, responseHttp);

}

private async Task<T> UnserializeAnswer<T>(HttpResponseMessage httpResponse, JsonSerializerOptions jsonSerializerOptions)

{

var respuestaString = await httpResponse.Content.ReadAsStringAsync();

return JsonSerializer.Deserialize<T>(respuestaString, jsonSerializerOptions)!;

}

}

}

8

1. **En el Program del proyecto WEB** configuramos la inyección del **Repository**:

builder.Services.AddScoped(sp => new HttpClient { BaseAddress = new Uri("https://localhost:7000/") });

builder.Services.AddScoped<IRepository, Repository>();

await builder.Build().RunAsync();

1. En la carpeta **Shared del proyecto WEB** creamos el componente razor **GenericList**:

@typeparam Titem

@if(MyList is null)

{

@if(Loading is null)

{

<div class="align-items-center">

<img src=" https://img.pikbest.com/png-images/20190918/cartoon-snail-loading-loading-gif-animation\_2734139.png!bw700 " />

</div>

}

else

{

@Loading

}

}

else if(MyList.Count == 0)

{

@if(NoRecords is null)

{

<p>No hay registros para mostrar...</p>

}

else

{

@NoRecords

}

}

else

{

@Body

}

@code {

[Parameter]

public RenderFragment? Loading { get; set; }

[Parameter]

public RenderFragment? NoRecords { get; set; }

[Parameter]

[EditorRequired]

public RenderFragment Body { get; set; } = null!;

[Parameter]

[EditorRequired]

public List<Titem> MyList { get; set; } = null!;

}

1. En el proyecto **WEB** Dentro de **Pages** creamos la carpeta **Countries** y dentro de esta carpeta creamos la página **CountriesIndex**:

@page "/countries"

@inject IRepository repository

<h3>Paises</h3>

<div class="mb-3">

<a class="btn btn-primary" href="/countries/create">Nuevo País</a>

</div>

<**GenericList** **MyList**="Countries">

<**Body**>

<table class="table table-striped">

<thead>

<tr>

<th>País</th>

<th></th>

</tr>

</thead>

<tbody>

@foreach (var country in Countries!)

{

<tr>

<td>

@country.Name

</td>

<td>

<a class="btn btn-warning">Editar</a>

<button class="btn btn-danger">Borrar</button>

</td>

</tr>

}

</tbody>

</table>

</**Body**>

</**GenericList**>

@code {

public List<Country>? Countries { get; set; }

protected async override Task OnInitializedAsync()

{

var responseHppt = await repository.Get<List<Country>>("/api/countries");

Countries = responseHppt.Response!;

}

}

\*Importante:

Agregamos una referencia al proyecto Web para que obtenga comunicación con el proyecto Shared (Click derecho sobre el proyecto Market.WEB Add reference>

Market.Shares

1. Agregamos los problemas de los usings y luego movemos esos using al **\_Imports.razor**:

@using Market.WEB.Shared

@using Market.Shared.Entities

@using Market.WEB.Repositories

1. Cambiamos el menú en el **NavMenu.razor**:

<div class="nav-item px-3">

<NavLink class="nav-link" href="counter">

<span class="oi oi-plus" aria-hidden="true"></span> Counter

</NavLink>

</div>

<div class="nav-item px-3">

<NavLink class="nav-link" href="countries">

<span class="oi oi-list-rich" aria-hidden="true"></span> Países

</NavLink>

</div>

1. Configuramos nuestra solución para que inicie al mismo tiempo el proyecto **API** y el proyecto **WEB**:

Vamos a las Solution Market, click derecho properties:

Interfaz de usuario gráfica

Descripción generada automáticamente

1. Probamos y hacemos nuestro commit.

# Completando las acciones de crear, editar y borrar países

1. En el proyecto **API** vamos adicionar estos métodos al **CountriesController**:

[HttpPut]

public async Task<ActionResult> Put(Country country)

{

\_context.Update(country);

await \_context.SaveChangesAsync();

return Ok(country);

}

[HttpDelete("{id:int}")]

public async Task<ActionResult> Delete(int id)

{

var afectedRows = await \_context.Countries

.Where(x => x.Id == id)

.ExecuteDeleteAsync();

if (afectedRows == 0)

{

return NotFound();

}

return NoContent();

}

1. Probamos estos métodos por **Swagger** o por **Postman**.
2. Agregamos estos métodos a la interfaz **IRepository**.

Task<HttpResponseWrapper<object>> Delete(string url);

Task<HttpResponseWrapper<object>> Put<T>(string url, T model);

Task<HttpResponseWrapper<TResponse>> Put<T, TResponse>(string url, T model);

1. Luego los implementamos en el **Repository**.

public async Task<HttpResponseWrapper<object>> Delete(string url)

{

var responseHTTP = await \_httpClient.DeleteAsync(url);

return new HttpResponseWrapper<object>(null, !responseHTTP.IsSuccessStatusCode, responseHTTP);

}

public async Task<HttpResponseWrapper<object>> Put<T>(string url, T model)

{

var messageJSON = JsonSerializer.Serialize(model);

var messageContent = new StringContent(messageJSON, Encoding.UTF8, "application/json");

var responseHttp = await \_httpClient.PutAsync(url, messageContent);

return new HttpResponseWrapper<object>(null, !responseHttp.IsSuccessStatusCode, responseHttp);

}

public async Task<HttpResponseWrapper<TResponse>> Put<T, TResponse>(string url, T model)

{

var messageJSON = JsonSerializer.Serialize(model);

var messageContent = new StringContent(messageJSON, Encoding.UTF8, "application/json");

var responseHttp = await \_httpClient.PutAsync(url, messageContent);

if (responseHttp.IsSuccessStatusCode)

{

var response = await UnserializeAnswer<TResponse>(responseHttp, \_jsonDefaultOptions);

return new HttpResponseWrapper<TResponse>(response, false, responseHttp);

}

return new HttpResponseWrapper<TResponse>(default, !responseHttp.IsSuccessStatusCode, responseHttp);

}

1. Vamos agregarle al proyecto **WEB** el nugget **CurrieTechnologies.Razor.SweetAlert2**, que nos va a servir para mostrar alertas muy bonitas.
2. Vamos a la página de Sweet Alert 2 ([Basaingeal/Razor.SweetAlert2: A Razor class library for interacting with SweetAlert2 (github.com)](https://github.com/Basaingeal/Razor.SweetAlert2) y copiamos el script que debemos de agregar al **index.html** que está en el **wwwroot** de nuestro proyecto **WEB**.

<script src="\_framework/blazor.webassembly.js"></script>

<script src="\_content/CurrieTechnologies.Razor.SweetAlert2/sweetAlert2.min.js"></script>

</body>

1. En el program del proyecto **WEB** configuramos la inyección del servicio de alertas:

builder.Services.AddScoped<IRepository, Repository>();

builder.Services.AddSweetAlert2();

Insertar lo siguiente en el archivo imports del proyecto Web

@using CurrieTechnologies.Razor.SweetAlert2;

1. En la carpeta **Countries** agregar el componente **CountryForm**:

<EditForm Model="Country" OnValidSubmit="OnSubmit">

<DataAnnotationsValidator />

<div class="mb-3">

<label>País:</label>

<div>

<InputText class="form-control" @bind-Value="@Country.Name" />

<ValidationMessage For="@(() => Country.Name)" />

</div>

</div>

<button class="btn btn-primary" type="submit">Guardar Cambios</button>

<button class="btn btn-success" @onclick="ReturnAction">Regresar</button>

</EditForm>

@code {

[EditorRequired]

[Parameter]

public Country Country { get; set; } = null!;

[EditorRequired]

[Parameter]

public EventCallback OnSubmit { get; set; }

[EditorRequired]

[Parameter]

public EventCallback ReturnAction { get; set; }

}

1. En la carpeta **Countries** agregar el componente **CountryCreate**:

@page "/countries/create"

@inject IRepository repository

@inject NavigationManager navigationManager

@inject SweetAlertService sweetAlertService

<h3>Crear País</h3>

<CountryForm Country="country" OnSubmit="Create" ReturnAction="Return"/>

@code {

private Country country = new();

private async Task Create()

{

var responseHttp = await repository.Post("/api/countries", country);

if (responseHttp.Error)

{

var message = await responseHttp.GetErrorMessageAsync();

await sweetAlertService.FireAsync("Error", message);

return;

}

navigationManager.NavigateTo("/countries");

}

private void Return()

{

navigationManager.NavigateTo("/countries");

}

}

1. Agregamos el boton de crear país en **CountriesIndex**:

<h3>Países</h3>

<a class="btn btn-primary" href="/countries/create">Nuevo País</a>

<GenericList MyList="Countries">

1. Probamos la creación de países por interfaz.
2. Mejoremos el formulario previniendo que el usuario salga y deje el formulario incompleto, modificamos nuestro componente **CountryForm**:

@inject SweetAlertService swal

<NavigationLock OnBeforeInternalNavigation="OnBeforeInternalNavigation"></NavigationLock>

<EditForm EditContext="editContext" OnSubmit="OnSubmit">

<DataAnnotationsValidator />

<div class="mb-3">

<label>País:</label>

<div>

<InputText class="form-control" @bind-Value="@Country.Name" />

<ValidationMessage For="@(() => Country.Name)" />

</div>

</div>

<button class="btn btn-primary" type="submit">Guardar Cambios</button>

<button class="btn btn-success" @onclick="ReturnAction">Regresar</button>

</EditForm>

@code {

private EditContext editContext = null!;

protected override void OnInitialized()

{

editContext = new(Country);

}

[EditorRequired]

[Parameter]

public Country Country { get; set; } = null!;

[EditorRequired]

[Parameter]

public EventCallback OnValidSubmit { get; set; }

[EditorRequired]

[Parameter]

public EventCallback ReturnAction { get; set; }

public bool FormPostedSuccessfully { get; set; } = false;

private async Task OnBeforeInternalNavigation(LocationChangingContext context)

{

var formWasEdited = editContext.IsModified();

if (!formWasEdited)

{

return;

}

if (FormPostedSuccessfully)

{

return;

}

var result = await swal.FireAsync(new SweetAlertOptions

{

Title = "Confirmación",

Text = "¿Deseas abandonar la página y perder los cambios?",

Icon = SweetAlertIcon.Warning,

ShowCancelButton = true

});

var confirm = !string.IsNullOrEmpty(result.Value);

if (confirm)

{

return;

}

context.PreventNavigation();

}

}

1. Y hacemos este cambio en **CountryCreate**:

@page "/countries/create"

@inject NavigationManager navigationManager

@inject IRepository repository

@inject SweetAlertService swal

<h3>Crear País</h3>

<CountryForm @ref="countryForm" Country="country" OnSubmit="Create" ReturnAction="Return" />

@code {

private Country country = new();

private CountryForm? countryForm;

private async Task Create()

{

var httpResponse = await repository.Post("api/countries", country);

if (httpResponse.Error)

{

var mensajeError = await httpResponse.GetErrorMessageAsync();

await swal.FireAsync("Error", mensajeError, SweetAlertIcon.Error);

}

else

{

countryForm!.FormPostedSuccessfully = true;

navigationManager.NavigateTo("countries");

}

}

private void Return()

{

navigationManager.NavigateTo("countries");

}

}

1. Probamos la creación de países por interfaz y luego hacemos nuestro **commit**. **Asegúrate de presionar Ctrl + F5, para que te tome los cambios**.
2. Ahora creamos el componente **CountryEdit**:

@page "/countries/edit/{Id:int}"

@inject NavigationManager navigationManager

@inject IRepository repository

@inject SweetAlertService swal

<h3>Editar País</h3>

@if (country is null)

{

<p>Cargando...</p>

}

else

{

<CountryForm @ref="countryForm" Country="country" OnValidSubmit="Edit" ReturnAction="Return" />

}

@code {

private Country? country;

private CountryForm? countryForm;

[Parameter]

public int Id { get; set; }

protected override async Task OnInitializedAsync()

{

var responseHTTP = await repository.Get<Country>($"api/countries/{Id}");

if (responseHTTP.Error)

{

if (responseHTTP.HttpResponseMessage.StatusCode == System.Net.HttpStatusCode.NotFound)

{

navigationManager.NavigateTo("countries");

}

else

{

var messageError = await responseHTTP.GetErrorMessage();

await swal.FireAsync("Error", messageError, SweetAlertIcon.Error);

}

}

else

{

country = responseHTTP.Response;

}

}

private async Task Edit()

{

var responseHTTP = await repository.Put("api/countries", country);

if (responseHTTP.Error)

{

var mensajeError = await responseHTTP.GetErrorMessage();

await swal.FireAsync("Error", mensajeError, SweetAlertIcon.Error);

}

else

{

countryForm!.FormPostedSuccessfully = true;

navigationManager.NavigateTo("countries");

}

}

private void Return()

{

navigationManager.NavigateTo("countries");

}

}

1. Luego modificamos el componente **CountriesIndex**:

@page "/countries"

@inject IRepository repository

@inject NavigationManager navigationManager

@inject SweetAlertService swal

<h3>Paises</h3>

<div class="mb-3">

<a class="btn btn-primary" href="/countries/create">Nuevo País</a>

</div>

<GenericList MyList="Countries">

<Body>

<table class="table table-striped">

<thead>

<tr>

<th>País</th>

<th></th>

</tr>

</thead>

<tbody>

@foreach (var country in Countries!)

{

<tr>

<td>

@country.Name

</td>

<td>

<a href="/countries/edit/@country.Id" class="btn btn-warning">Editar</a>

<button class="btn btn-danger" @onclick=@(() => Delete(country))>Borrar</button>

</td>

</tr>

}

</tbody>

</table>

</Body>

</GenericList>

@code {

public List<Country>? Countries { get; set; }

protected async override Task OnInitializedAsync()

{

await Load();

}

private async Task Load()

{

var responseHppt = await repository.Get<List<Country>>("api/countries");

Countries = responseHppt.Response!;

}

private async Task Delete(Country country)

{

var result = await swal.FireAsync(new SweetAlertOptions

{

Title = "Confirmación",

Text = "¿Esta seguro que quieres borrar el registro?",

Icon = SweetAlertIcon.Question,

ShowCancelButton = true

});

var confirm = string.IsNullOrEmpty(result.Value);

if (confirm)

{

return;

}

var responseHTTP = await repository.Delete($"api/countries/{country.Id}");

if (responseHTTP.Error)

{

if (responseHTTP.HttpResponseMessage.StatusCode == System.Net.HttpStatusCode.NotFound)

{

navigationManager.NavigateTo("/");

}

else

{

var mensajeError = await responseHTTP.GetErrorMessageAsync();

await swal.FireAsync("Error", mensajeError, SweetAlertIcon.Error);

}

}

else

{

await Load();

}

}

}

1. Y probamos la edición y eliminación de países por interfaz. No olvides hacer el **commit**.