Blazor & Maui

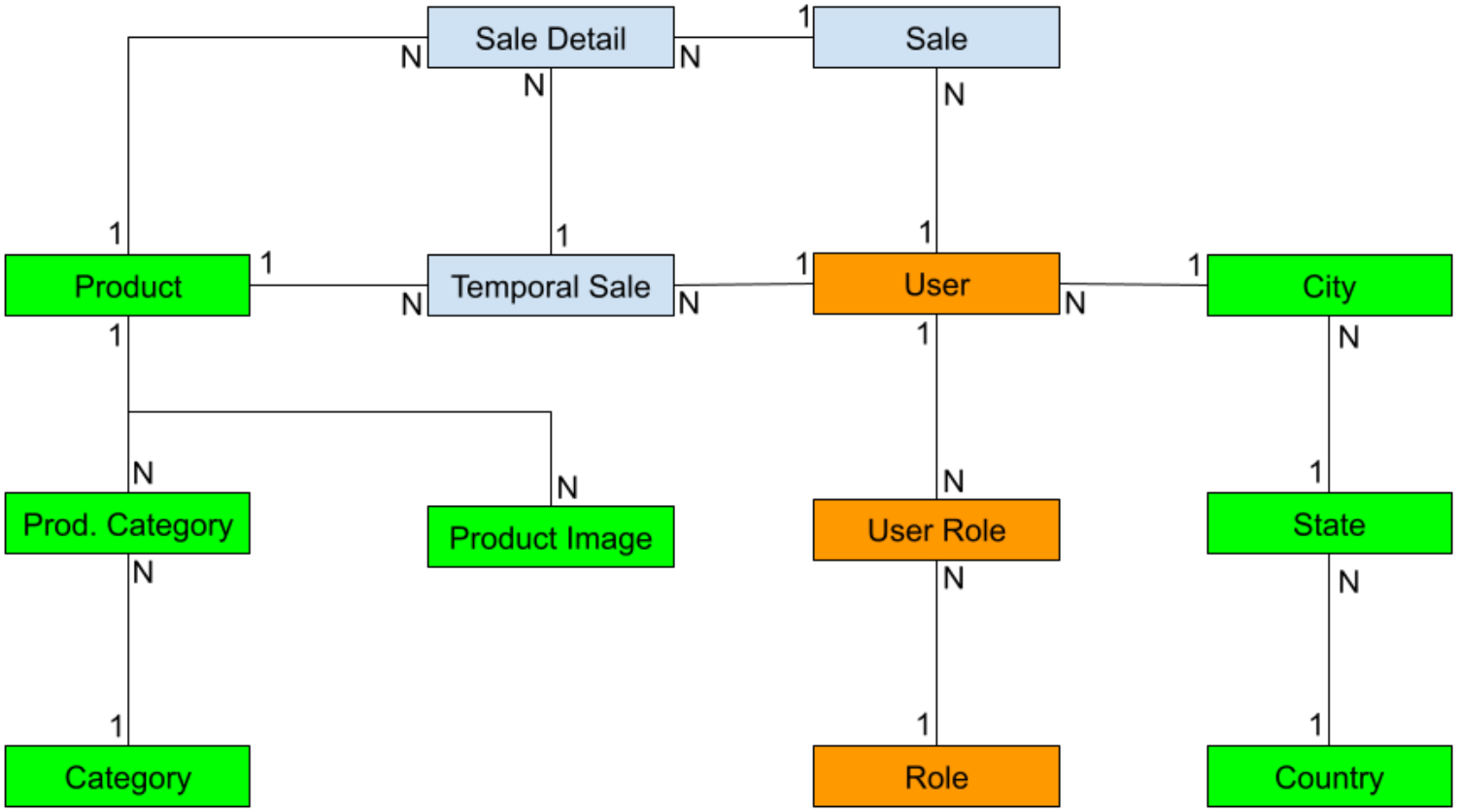
Juan Carlos Zuluaga

2023, Semestre 1

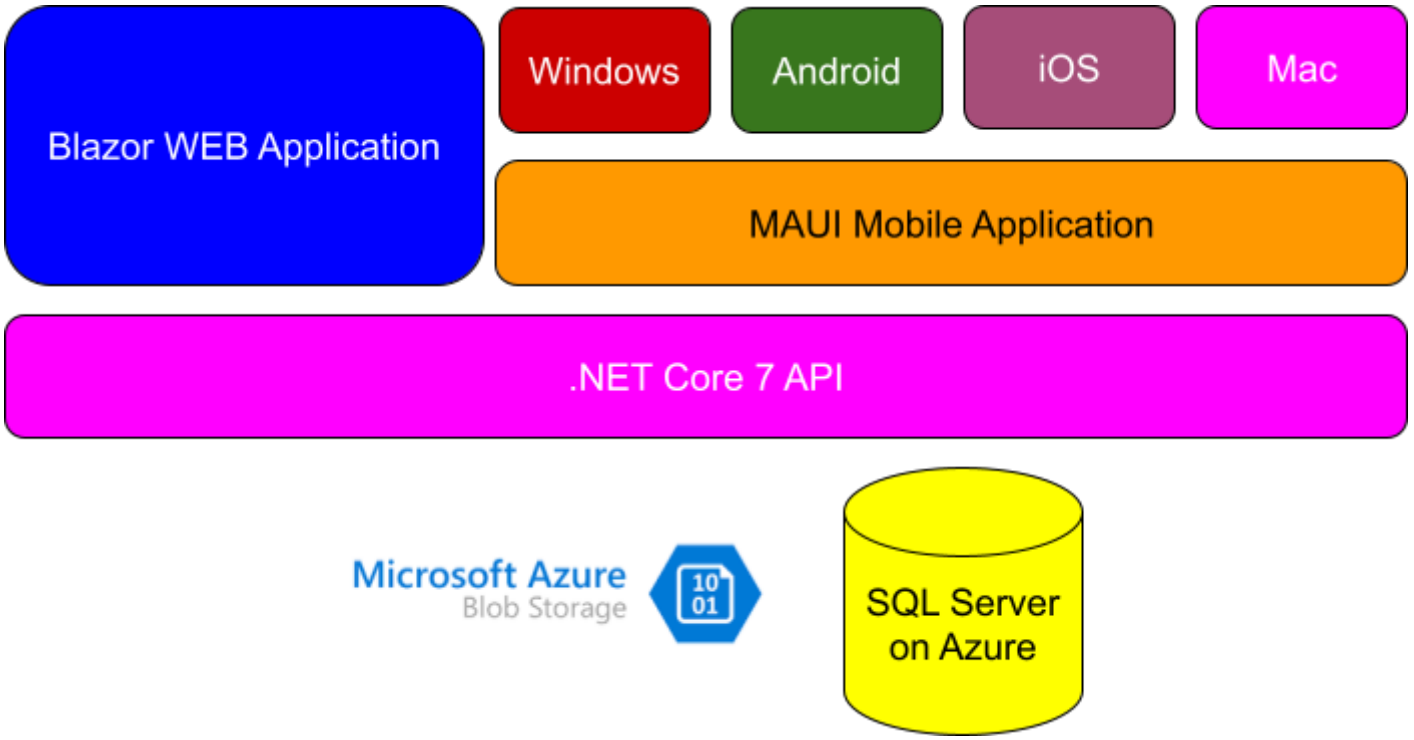
Indice

Diagrama Entidad Relación

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



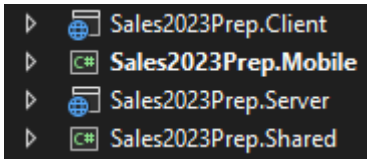
Estructura básica de proyecto



Vamos a crear esta estructura en Visual Studio (asegurese de poner todos los proyectos rn :

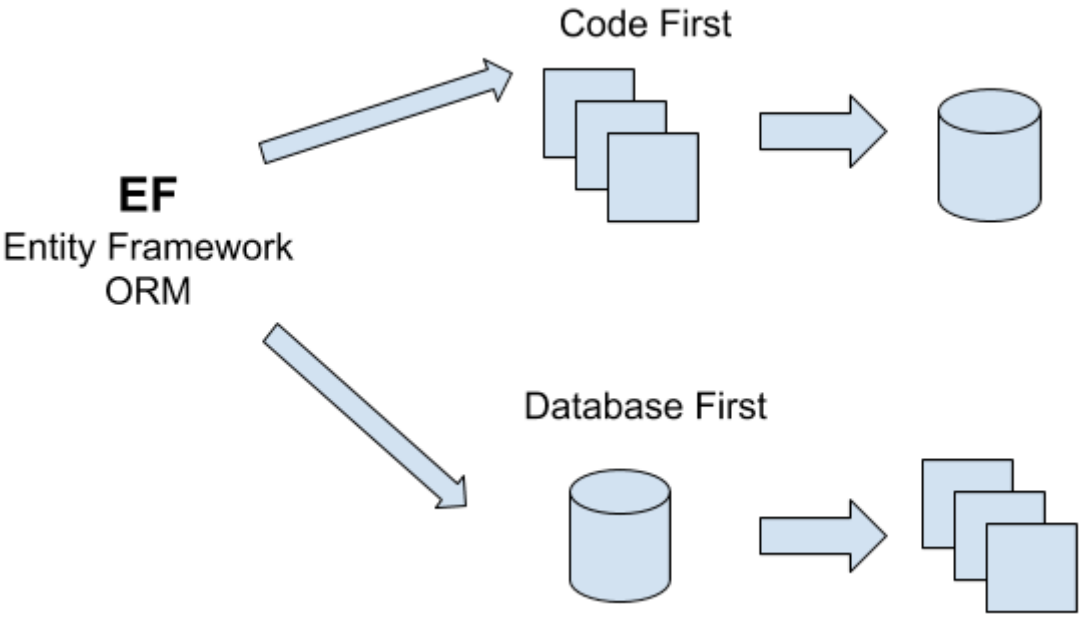
- Una solución en blanco llamada **Sales**.
- A la solución le agregamos un proyecto tipo: **Class Library**, llamado **Sales.Shared**.
- A la solución le agregamos un proyecto tipo: **ASP.NET Core Web API**, llamado **Sales.API**.
- A la solución le agregamos un proyecto tipo: **Blazor WebAssembly App**, llamado **Sales.WEB**.
- A la solución le agregamos un proyecto tipo: **.NET MAUI App**, llamado **Sales.Mobile**.

Debe quedar algo como esto:



Hacemos el primer commit en nuestro repositorio.

Crear la BD con EF



Recomiendo buscar y leer documentación sobre Code First y Database First. En este curso trabajaremos con EF Code First, si están interesados en conocer más sobre EF Database First acá les dejo un enlace: https://docs.microsoft.com/en-us/ef/core/get-started/aspnetcore/existing-db

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

```
using System.ComponentModel.DataAnnotations;

namespace Sales.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!;
    }
}
```

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

```
using Microsoft.EntityFrameworkCore;
using Sales.Shared.Entities;

namespace Sales.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();
        }
    }
}
```

3. Configurar el string de conexión en el **appsettings.json** del proyecto **API**:

```
{
  "ConnectionStrings": {
    "DockerConnection": "Data Source=.;Initial Catalog=SalesPrep;User ID=sa;Password=Roger1974.;Connect
Timeout=30;Encrypt=False;TrustServerCertificate=False;ApplicationIntent=ReadWrite;MultiSubnetFailover=False",
    "AzureConnection": "Server=tcp:ventasdbserver.database.windows.net,1433;Initial Catalog=Sales2023;Persist
Security Info=False;User
ID=Zulu;Password=Roger1974.;MultipleActiveResultSets=False;Encrypt=True;TrustServerCertificate=False;Connection
Timeout=30;",
    "LocalConnection":
"Server=(localdb)\\MSSQLLocalDB;Database=Sales2023;Trusted_Connection=True;MultipleActiveResultSets=true"
  },
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft.AspNetCore": "Warning"
    }
  },
  "AllowedHosts": "*"
}
```

Nota: dejo los 3 string de conexión para que use el que más le convenga en el vídeo de clase explico mejor cual utilizar en cada caso.

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

```
Microsoft.EntityFrameworkCore.SqlServer
Microsoft.EntityFrameworkCore.Tools
```

5. Configurar la inyección del data context en el **Program** del proyecto **API**:

```
builder.Services.AddSwaggerGen();
builder.Services.AddDbContext<DataContext>(x => x.UseSqlServer("name=AzureConnection"));


var app = builder.Build();
```

6. Correr los comandos:

```
add-migration InitialDb
update-database
```

7. Hacemos nuestro segundo **Commit**.

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 Sales.API.Data;
using Sales.Shared.Entities;

namespace Sales.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());
        }

        [HttpPost]
        public async Task<ActionResult> Post(Country country)
        {
            _context.Add(country);
            await _context.SaveChangesAsync();
            return Ok(country);
```

```
        }
    }
}
```

2. 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();
```

3. Borramos las clases de **WeatherForecast**.

4. Probamos la creación y listado de paises por el **swagger** y por **Postman**.

5. 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**:

```
builder.Services.AddScoped(sp => new HttpClient { BaseAddress = new Uri("https://localhost:7201//") });
```

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

```
using System.Net;

namespace 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 "Tienes que logearte para hacer esta operación";
            }
            else if (codigoEstatus == HttpStatusCode.Forbidden)
            {
                return "No tienes permisos para hacer esta operación";
            }

            return "Ha ocurrido un error inesperado";
        }
    }
}
```

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

```
namespace 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);
    }
}
```

4. En la misma carpeta creamos la case **Repository**:

```
using System.Text;
using System.Text.Json;

namespace Sales.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)!;
    }
  }
}
```

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

```
builder.Services.AddScoped(sp => new HttpClient { BaseAddress = new Uri("https://localhost:7230/") });
builder.Services.AddScoped<IRepository, Repository>();

await builder.Build().RunAsync();
```

6. En la carpeta **Shared** creamos el componente genérico **GenericList**:

```
@typeparam Titem

@if (MyList is null)
{
    @if (Loading is null)
    {
        <img alt="loading..." src="https://upload.wikimedia.org/wikipedia/commons/b/b1/Loading_icon.gif?20151024034921"
/>
    }
    else
    {
        @Loading
    }
}
else if (MyList.Count == 0)
{
    @if (NoRecords is null)
    {
        <p>No hay registros para mostrar.</p>
    }
    else
    {
        @NoRecords
    }
}
else
{
    if (Records is not null)
    {
        @foreach (var item in MyList)
        {
            @Records(item)
        }
    }
    else
    {
        @RecordsComplete
    }

}

@code {
    [Parameter]
    public RenderFragment Loading { get; set; } = null!;
```

```
    [Parameter]
    public RenderFragment NoRecords { get; set; } = null!;

    [Parameter]
    public RenderFragment<Titem> Records { get; set; } = null!;

    [Parameter]
    public RenderFragment RecordsComplete { get; set; } = null!;

    [Parameter]
    [EditorRequired]
    public List<Titem> MyList { get; set; } = null!;
}
```

7. 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">
    <RecordsComplete>
        <table class="table table-striped">
            <thead>
                <tr>
                    <th></th>
                    <th>País</th>
                </tr>
            </thead>
            <tbody>
                @foreach (var country in Countries!)
                {
                    <tr>
                        <td>
                            <a class="btn btn-warning">Editar</a>
                            <button class="btn btn-danger">Borrar</button>
                        </td>
                        <td>
                            @country.Name
                        </td>
                    </tr>
                }
            </tbody>
        </table>
    </RecordsComplete>
</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!;
    }
}
```

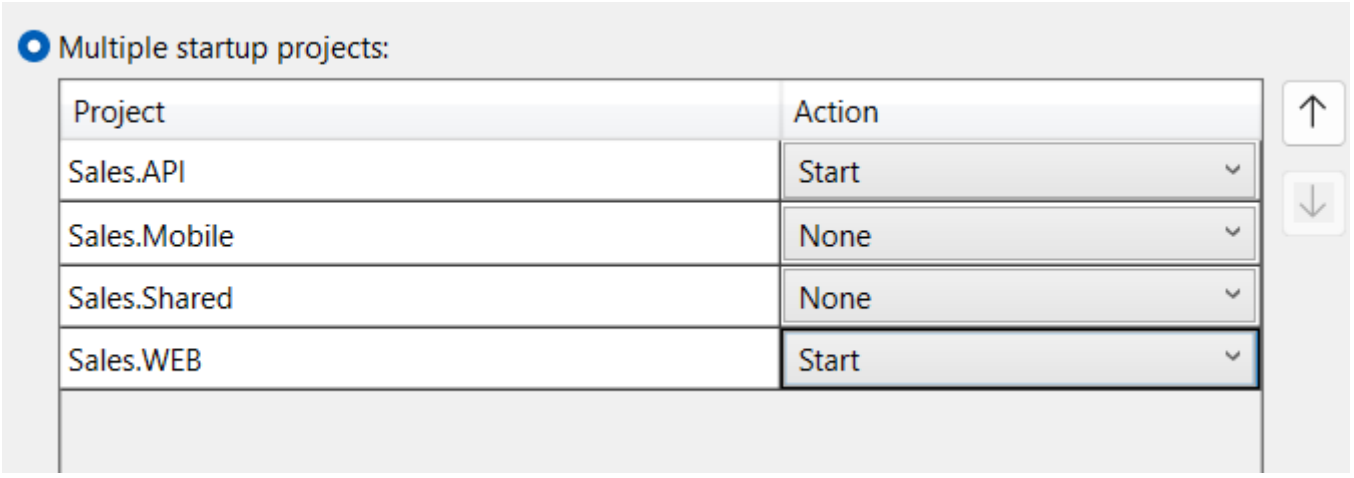
8. Agregamos los problemas de los using y luego movemos esos using al **_Imports.razor**:

```
@using Sales.WEB.Shared
@using Sales.Shared.Entities
@using Sales.WEB.Repositories
```

9. 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>
```

10. Configuramos nuestro proyecto para que inicie al mismo tiempo el proyecto **API** y el proyecto **WEB**:



11. 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**:

```
[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);
}

[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();
}
```

2. Probamos estos métodos por **Swagger** o por **Postman**.

3. Agregamos estos métodos a la interfaz **IRepository**.

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

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

4. 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);
}
```

5. Vamos agregarle al proyecto **WEB** el paquete **CurrieTechnologies.Razor.SweetAlert2**, que nos va a servir para mostrar modeles de alertas muy bonitos.

6. Vamos a la página de Sweet Alert 2 (Basaingeal/Razor.SweetAlert2: A Razor class library for interacting with SweetAlert2 (github.com) 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>
```

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

```
builder.Services.AddScoped<IRepository, Repository>();
builder.Services.AddSweetAlert2();
```

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

```
@inject SweetAlertService swal

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

<EditForm EditContext="editContext" OnValidSubmit="OnValidSubmit">
    <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();
    }
}
```

9. En la carpeta **Countries** agregar el componente **CreateCountry**:

```
@page "/countries/create"
@inject NavigationManager navigationManager
@inject IRepository repository
@inject SweetAlertService swal

<h3>Crear País</h3>
```

```
<CountryForm @ref="countryForm" Country="country" OnValidSubmit="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.GetErrorMessage();
            await swal.FireAsync("Error", mensajeError, SweetAlertIcon.Error);
        }
        else
        {
            countryForm!.FormPostedSuccessfully = true;
            navigationManager.NavigateTo("countries");
        }
    }

    private void Return()
    {
        navigationManager.NavigateTo("countries");
    }
}
```

10. Probamos la creación de países por interfaz y luego hacemos nuestro **commit**. Asegurate de presionar Ctrl + F5, para que te tome los cambios.

11. Ahora creamos el componente **EditCountry**:

```
@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");
    }
}
```

12. 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">
    <RecordsComplete>
        <table class="table table-striped">
            <thead>
                <tr>
                    <th></th>
                    <th>País</th>
                </tr>
            </thead>
            <tbody>
                @foreach (var country in Countries!)
                {
                    <tr>
                        <td>
                            <a href="/countries/edit/@country.Id" class="btn btn-warning">Editar</a>
                            <button class="btn btn-danger" @onclick=@(() => Delete(country))>Borrar</button>
                        </td>
                        <td>
                            @country.Name
                        </td>
                    </tr>
                }
            </tbody>
        </table>
    </RecordsComplete>
</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.GetErrorMessage();
            await swal.FireAsync("Error", mensajeError, SweetAlertIcon.Error);
        }
    }
    else
    {
        await Load();
    }
  }
}
```

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

Solucionando el problema de países con el mismo nombre y adicionando un Seeder a la base de datos

1. Si intentamos crear un país con el mismo nombre, sale un error no muy claro para el cliente. Vamos a solucionar esto, lo primero que vamos hacer es corregir el **Post** y el **Put** en el controlador de países:

```
[HttpPost]
public async Task<ActionResult> Post(Country country)
{
  _context.Add(country);
  try
  {
    await _context.SaveChangesAsync();
    return Ok(country);
  }
  catch (DbUpdateException dbUpdateException)
```

```
    {
        if (dbUpdateException.InnerException!.Message.Contains("duplicate"))
        {
            return BadRequest("Ya existe un país con el mismo nombre.");
        }
        else
        {
            return BadRequest(dbUpdateException.InnerException.Message);
        }
    }
    catch (Exception exception)
    {
        return BadRequest(exception.Message);
    }
}

[HttpPut]
public async Task<ActionResult> Put(Country country)
{
    _context.Update(country);
    try
    {
        await _context.SaveChangesAsync();
        return Ok(country);
    }
    catch (DbUpdateException dbUpdateException)
    {
        if (dbUpdateException.InnerException!.Message.Contains("duplicate"))
        {
            return BadRequest("Ya existe un registro con el mismo nombre.");
        }
        else
        {
            return BadRequest(dbUpdateException.InnerException.Message);
        }
    }
    catch (Exception exception)
    {
        return BadRequest(exception.Message);
    }
}
```

2. Probamos. Ahora vamos a adicionar un alimentador de la base de datos. Para esto primero creamos en el proyecto **API** dentro de la carpeta **Data** la clase **SeedDb**:

```
using Sales.Shared.Entities;

namespace Sales.API.Data
{
    public class SeedDb
    {
        private readonly DataContext _context;
```

```
        public SeedDb(DataContext context)
        {
            _context = context;
        }

        public async Task SeedAsync()
        {
            await _context.Database.EnsureCreatedAsync();
            await CheckCountriesAsync();
        }

        private async Task CheckCountriesAsync()
        {
            if (!_context.Countries.Any())
            {
                _context.Countries.Add(new Country { Name = "Colombia" });
                _context.Countries.Add(new Country { Name = "Estados Unidos" });
            }

            await _context.SaveChangesAsync();
        }
    }
}
```

3. Luego modificamos el **Program** del proyecto **API** para llamar el alimentador de la BD:

```
builder.Services.AddDbContext<DataContext>(x => x.UseSqlServer("name=DockerConnection"));
builder.Services.AddTransient<SeedDb>();

var app = builder.Build();
SeedData(app);

void SeedData(WebApplication app)
{
    IServiceScopeFactory? scopedFactory = app.Services.GetService<IServiceScopeFactory>();

    using (IServiceScope? scope = scopedFactory!.CreateScope())
    {
        SeedDb? service = scope.ServiceProvider.GetService<SeedDb>();
        service!.SeedAsync().Wait();
    }
}
```

4. Borramos la base de datos con el comando **drop-database**.

5. Probamos y hacemos el **commit**.

Actividad #1

Con el conocimiento adquirido hasta el momento hacer lo mismo para las categorías. El modelo categoría es muy sencillo, solo tiene el Id y el Name (igual a país). Cree todo lo necesario para que

haya un CRUD de categorías, y modifique el alimentador de base de datos para que adicione algunas categorías por defecto.

Relación uno a muchos e índice compuesto

1. Creamos la entidad **State**:

```
using System.ComponentModel.DataAnnotations;

namespace Sales.Shared.Entities
{
    public class State
    {
        public int Id { get; set; }

        [Display(Name = "Departamento/Estado")]
        [MaxLength(100, ErrorMessage = "El campo {0} debe tener máximo {1} caractéres.")]
        [Required(ErrorMessage = "El campo {0} es obligatorio.")]
        public string Name { get; set; } = null!;

        public Country? Country { get; set; }
    }
}
```

2. Modificamos la entidad **Country**:

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

public ICollection<State>? States { get; set; }

[Display(Name = "Estados/Departamentos")]
public int StatesNumber => States == null ? 0 : States.Count;
```

3. Creamos la entidad **City**:

```
using System.ComponentModel.DataAnnotations;

namespace Sales.Shared.Entities
{
    public class City
    {
        public int Id { get; set; }

        [Display(Name = "Ciudad")]
        [MaxLength(100, ErrorMessage = "El campo {0} debe tener máximo {1} caractéres.")]
        [Required(ErrorMessage = "El campo {0} es obligatorio.")]
        public string Name { get; set; } = null!;

        public State? State { get; set; }
    }
}
```

4. Modificamos la entidad **State**:

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

public ICollection<City>? Cities { get; set; }

[Display(Name = "Ciudades")]
public int CitiesNumber => Cities == null ? 0 : Cities.Count;
```

5. Modificamos el **DataContext**:

```
public DataContext(DbContextOptions<DataContext> options) : base(options)
{
}

public DbSet<City> Cities { get; set; }

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

public DbSet<State> States { get; set; }

protected override void OnModelCreating(ModelBuilder modelBuilder)
{
    base.OnModelCreating(modelBuilder);

    modelBuilder.Entity<Country>().HasIndex(c => c.Name).IsUnique();
    modelBuilder.Entity<State>().HasIndex("Name", "CountryId").IsUnique();
    modelBuilder.Entity<City>().HasIndex("Name", "StateId").IsUnique();
}
```

6. Para evitar la redundancia ciclica en la respuesta de los JSON vamos a agregar la siguiente configuración, modificamos el **Program** del **API**:

```
builder.Services.AddControllers()
    .AddJsonOptions(x => x.JsonSerializerOptions.ReferenceHandler = ReferenceHandler.IgnoreCycles);
```

7. Modificamos el Seeder:

```
private async Task CheckCountriesAsync()
{
    if (!_context.Countries.Any())
    {
        _context.Countries.Add(new Country
        {
            Name = "Colombia",
            States = new List<State>()
            {
                new State()
                {
                    Name = "Antioquia",
                    Cities = new List<City>() {
                        new City() { Name = "Medellín" },
                        new City() { Name = "Itagüí" },
```

```
                    new City() { Name = "Envigado" },
                    new City() { Name = "Bello" },
                    new City() { Name = "Rionegro" },
                }
            },
            new State()
            {
                Name = "Bogotá",
                Cities = new List<City>() {
                    new City() { Name = "Usaquen" },
                    new City() { Name = "Champinero" },
                    new City() { Name = "Santa fe" },
                    new City() { Name = "Useme" },
                    new City() { Name = "Bosa" },
                }
            },
        }
    });
    _context.Countries.Add(new Country
    {
        Name = "Estados Unidos",
        States = new List<State>()
        {
            new State()
            {
                Name = "Florida",
                Cities = new List<City>() {
                    new City() { Name = "Orlando" },
                    new City() { Name = "Miami" },
                    new City() { Name = "Tampa" },
                    new City() { Name = "Fort Lauderdale" },
                    new City() { Name = "Key West" },
                }
            },
            new State()
            {
                Name = "Texas",
                Cities = new List<City>() {
                    new City() { Name = "Houston" },
                    new City() { Name = "San Antonio" },
                    new City() { Name = "Dallas" },
                    new City() { Name = "Austin" },
                    new City() { Name = "El Paso" },
                }
            },
        }
    });
  }

  await _context.SaveChangesAsync();
}
```

8. Modificamos los **Get** del controlador de países:

```
[HttpGet]
public async Task<ActionResult> Get()
{
    return Ok(await _context.Countries
        .Include(x => x.States)
        .ToListAsync());
}

[HttpGet("full")]
public async Task<ActionResult> GetFull()
{
    return Ok(await _context.Countries
        .Include(x => x.States!)
        .ThenInclude(x => x.Cities)
        .ToListAsync());
}
```

9. Borramos la base de datos con el comando **drop-database** para que el Seeder vueva a ser ejecutado.

10. Adicionamos la nueva migración de la base de datos con el comando: **add-migration AddStatesAndCities** y aunque el Seeder corre automáticamente el Update Database, prefiero correrlo manualmente para asegurarme que no genere ningun error: **update-databse**.

11. Cambiemos el **Index** de países para ver el número de departamentos/estados de cada país y adicionar el botón de detalles:

```
<GenericList MyList="Countries">
    <RecordsComplete>
        <table class="table table-striped">
            <thead>
                <tr>
                    <th></th>
                    <th>País</th>
                    <th>Departamentos/Estados</th>
                </tr>
            </thead>
            <tbody>
                @foreach (var country in Countries!)
                {
                    <tr>
                        <td>
                            <a href="/countries/details/@country.Id" class="btn btn-info">Detalles</a>
                            <a href="/countries/edit/@country.Id" class="btn btn-warning">Editar</a>
                            <button class="btn btn-danger" @onclick=@(() => Delete(country))>Borrar</button>
                        </td>
                        <td>
                            @country.Name
                        </td>
                        <td>
                            @country.StatesNumber
                        </td>
                    </tr>
```

```
            }
        </tbody>
    </table>
</RecordsComplete>
</GenericList>
```

12. Probamos y hacemos el **commit**.

Creando un CRUD multinivel

13. Vamos ahora a tener la posibilidad de crear, editar, borrar estados y ciudades. Empecemos creando el **StatesController**:

```
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
using Sales.API.Data;
using Sales.Shared.Entities;

namespace Sales.API.Controllers
{
    [ApiController]
    [Route("/api/states")]
    public class StatesController : ControllerBase
    {
        private readonly DataContext _context;

        public StatesController(DataContext context)
        {
            _context = context;
        }

        [HttpGet]
        public async Task<ActionResult> Get()
        {
            return Ok(await _context.States
                .Include(x => x.Cities)
                .ToListAsync());
        }

        [HttpGet("{id:int}")]
        public async Task<ActionResult> Get(int id)
        {
            var state = await _context.States
                .Include(x => x.Cities)
                .FirstOrDefaultAsync(x => x.Id == id);
            if (state is null)
            {
                return NotFound();
            }

            return Ok(state);
        }
```

```
    [HttpPost]
    public async Task<ActionResult> Post(State state)
    {
        _context.Add(state);
        try
        {
            await _context.SaveChangesAsync();
            return Ok(state);
        }
        catch (DbUpdateException dbUpdateException)
        {
            if (dbUpdateException.InnerException!.Message.Contains("duplicate"))
            {
                return BadRequest("Ya existe un registro con el mismo nombre.");
            }
            else
            {
                return BadRequest(dbUpdateException.InnerException.Message);
            }
        }
        catch (Exception exception)
        {
            return BadRequest(exception.Message);
        }
    }

    [HttpPut]
    public async Task<ActionResult> Put(State state)
    {
        try
        {
            _context.Update(state);
            await _context.SaveChangesAsync();
            return Ok(state);
        }
        catch (DbUpdateException dbUpdateException)
        {
            if (dbUpdateException.InnerException!.Message.Contains("duplicate"))
            {
                return BadRequest("Ya existe un registro con el mismo nombre.");
            }
            else
            {
                return BadRequest(dbUpdateException.InnerException.Message);
            }
        }
        catch (Exception exception)
        {
            return BadRequest(exception.Message);
        }
    }
```

```
        [HttpDelete("{id:int}")]
        public async Task<ActionResult> Delete(int id)
        {
            var afectedRows = await _context.States
                .Where(x => x.Id == id)
                .ExecuteDeleteAsync();

            if (afectedRows == 0)
            {
                return NotFound();
            }

            return NoContent();
        }
    }
}
```

14. Luego creamos el **CitiesController**:

```
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
using Sales.API.Data;
using Sales.Shared.Entities;

namespace Sales.API.Controllers
{
    [ApiController]
    [Route("/api/cities")]
    public class CitiesController : ControllerBase
    {
        private readonly DataContext _context;

        public CitiesController(DataContext context)
        {
            _context = context;
        }

        [HttpGet]
        public async Task<ActionResult> Get()
        {
            return Ok(await _context.Cities
                .ToListAsync());
        }

        [HttpGet("{id:int}")]
        public async Task<ActionResult> Get(int id)
        {
            var city = await _context.Cities
                .FirstOrDefaultAsync(x => x.Id == id);
            if (city is null)
            {
                return NotFound();
            }
```

```
            return Ok(city);
        }

        [HttpPost]
        public async Task<ActionResult> Post(City city)
        {
            _context.Add(city);
            try
            {
                await _context.SaveChangesAsync();
                return Ok(city);
            }
            catch (DbUpdateException dbUpdateException)
            {
                if (dbUpdateException.InnerException!.Message.Contains("duplicate"))
                {
                    return BadRequest("Ya existe un registro con el mismo nombre.");
                }
                else
                {
                    return BadRequest(dbUpdateException.InnerException.Message);
                }
            }
            catch (Exception exception)
            {
                return BadRequest(exception.Message);
            }
        }

        [HttpPut]
        public async Task<ActionResult> Put(City city)
        {
            _context.Update(city);
            try
            {
                await _context.SaveChangesAsync();
                return Ok(city);
            }
            catch (DbUpdateException dbUpdateException)
            {
                if (dbUpdateException.InnerException!.Message.Contains("duplicate"))
                {
                    return BadRequest("Ya existe un registro con el mismo nombre.");
                }
                else
                {
                    return BadRequest(dbUpdateException.InnerException.Message);
                }
            }
            catch (Exception exception)
            {
                return BadRequest(exception.Message);
```

```
        }
    }

    [HttpDelete("{id:int}")]
    public async Task<ActionResult> Delete(int id)
    {
        var afectedRows = await _context.Cities
            .Where(x => x.Id == id)
            .ExecuteDeleteAsync();

        if (afectedRows == 0)
        {
            return NotFound();
        }

        return NoContent();
    }
  }
}
```

15. En el proyecto **WEB** en la carpeta **Pages/Countries** vamos a crear la págima **CountryDetails**:

```
@page "/countries/details/{Id:int}"
@inject IRepository repository
@inject NavigationManager navigationManager
@inject SweetAlertService swal

@if (country is null)
{
  <p>Cargando...</p>
}
else
{
  <h3>@country.Name</h3>
  <div class="mb-3">
    <a class="btn btn-primary" href="/states/create/@country.Id">Nuevo Estado/Departamento</a>
    <a class="btn btn-success" href="/countries">Regresar</a>
  </div>

  <GenericList MyList="states">
    <RecordsComplete>
      <table class="table table-striped">
        <thead>
          <tr>
            <th></th>
            <th>Departamento</th>
            <th>Ciudades</th>
          </tr>
        </thead>
        <tbody>
          @foreach (var state in country.States!)
          {
            <tr>
```

```
                        <td>
                            <a href="/states/details/@state.Id" class="btn btn-info">Detalles</a>
                            <a href="/states/edit/@Id/@state.Id" class="btn btn-warning">Editar</a>
                            <button class="btn btn-danger" @onclick=@(() => Delete(state))>Borrar</button>
                        </td>
                        <td>@state.Name</td>
                        <td>@state.CitiesNumber</td>
                    </tr>
                }
            </tbody>
        </table>
    </RecordsComplete>
</GenericList>
}


@code {
    private Country? country;
    private List<State>? states;

    [Parameter]
    public int Id { get; set; }

    protected async override Task OnInitializedAsync()
    {
        await Load();
    }

    private async Task Load()
    {
        var responseHppt = await repository.Get<Country>($"api/countries/{Id}");
        country = responseHppt.Response!;
        states = country.States!.ToList();
    }

    private async Task Delete(State state)
    {
        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/states/{state.Id}");
```

```
        if (responseHTTP.Error)
        {
            if (responseHTTP.HttpResponseMessage.StatusCode == System.Net.HttpStatusCode.NotFound)
            {
                navigationManager.NavigateTo("/");
            }
            else
            {
                var mensajeError = await responseHTTP.GetErrorMessage();
                await swal.FireAsync("Error", mensajeError, SweetAlertIcon.Error);
            }
        }
        else
        {
            await Load();
        }
    }
}
```

16. Probamos lo que llevamos hasta el momento.

17. Ahora vamos a implementar la creación de estados. En el proyecto **WEB** en la carpeta **Pages** la carpeta **States** y dentro de esta creamos el componente **StateForm**:

```
@inject SweetAlertService swal

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

<EditForm EditContext="editContext" OnValidSubmit="OnValidSubmit">
    <DataAnnotationsValidator />
    <div class="mb-3">
        <label>Estado/Departamento:</label>
        <div>
            <InputText class="form-control" @bind-Value="@State.Name" />
            <ValidationMessage For="@(() => State.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(State);
    }

    [EditorRequired]
    [Parameter]
```

```
    public State State { 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();
    }
}
```

18. En el proyecto **WEB** en la carpeta **Pages** la carpeta **States** y dentro de esta creamos el componente **CreateState**:

```
@page "/states/create/{Id:int}"
@inject NavigationManager navigationManager
@inject IRepository repository
@inject SweetAlertService swal
```

```
<h3>Crear Estado/Departamento</h3>

<StateForm @ref="stateForm" State="state" OnValidSubmit="Create" ReturnAction="Return" />

@code {
    private State state = new();
    private StateForm? stateForm;

    [Parameter]
    public int Id { get; set; }

    private async Task Create()
    {
        Country country = new();
        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!;
        }

        if (country.States == null)
        {
            country.States = new List<State>();
        }

        country.States.Add(state);
        var httpResponse = await repository.Put("api/countries", country);

        if (httpResponse.Error)
        {
            var mensajeError = await httpResponse.GetErrorMessage();
            await swal.FireAsync("Error", mensajeError, SweetAlertIcon.Error);
        }
        else
        {
            stateForm!.FormPostedSuccessfully = true;
            navigationManager.NavigateTo($"/countries/details/{Id}");
        }
    }
```

```
    private void Return()
    {
        navigationManager.NavigateTo($"/countries/details/{Id}");
    }
}
```

19. En el proyecto **WEB** en la carpeta **Pages** la carpeta **States** y dentro de esta creamos el componente **EditState**:

```
@page "/states/edit/{CountryId:int}/{StateId:int}"
@inject NavigationManager navigationManager
@inject IRepository repository
@inject SweetAlertService swal

<h3>Editar Estado/Departamento</h3>

@if (state is null)
{
    <p>Cargando...</p>
}
else
{
    <StateForm @ref="stateForm" State="state" OnValidSubmit="Edit" ReturnAction="Return" />
}


@code {
    private State? state;
    private StateForm? stateForm;

    [Parameter]
    public int CountryId { get; set; }

    [Parameter]
    public int StateId { get; set; }

    protected override async Task OnInitializedAsync()
    {
        var responseHTTP = await repository.Get<State>($"api/states/{StateId}");

        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
        {
```

```
            state = responseHTTP.Response;
        }
    }

    private async Task Edit()
    {
        var responseHTTP = await repository.Put("api/states", state);

        if (responseHTTP.Error)
        {
            var mensajeError = await responseHTTP.GetErrorMessage();
            await swal.FireAsync("Error", mensajeError, SweetAlertIcon.Error);
        }
        else
        {
            stateForm!.FormPostedSuccessfully = true;
            navigationManager.NavigateTo($"/countries/details/{CountryId}");
        }
    }

    private void Return()
    {
        navigationManager.NavigateTo($"/countries/details/{CountryId}");
    }
}
```

20. En el proyecto **WEB** en la carpeta **Pages** la carpeta **States** y dentro de esta creamos el componente **StateDetails**:

```
@page "/states/details/{Id:int}"
@inject IRepository repository
@inject NavigationManager navigationManager
@inject SweetAlertService swal

@if (state is null)
{
    <p>Cargando...</p>
}
else
{
    <h3>@state.Name</h3>
    <div class="mb-3">
        <a class="btn btn-primary" href="/cities/create/@state.Id">Nueva Ciudad</a>
        <a class="btn btn-success" href="/states/detailes/@Id">Regresar</a>
    </div>

    <GenericList MyList="cities">
        <RecordsComplete>
            <table class="table table-striped">
                <thead>
                    <tr>
                        <th></th>
                        <th>Ciudades</th>
```

```
                </tr>
            </thead>
            <tbody>
                @foreach (var city in state.Cities!)
                {
                    <tr>
                        <td>
                            <a href="/cities/edit/@Id/@city.Id" class="btn btn-warning">Editar</a>
                            <button class="btn btn-danger" @onclick=@(() => Delete(city))>Borrar</button>
                        </td>
                        <td>@city.Name</td>
                    </tr>
                }
            </tbody>
        </table>
    </RecordsComplete>
</GenericList>
}

@code {
    private State? state;
    private List<City>? cities;

    [Parameter]
    public int Id { get; set; }

    protected async override Task OnInitializedAsync()
    {
        await Load();
    }

    private async Task Load()
    {
        var responseHppt = await repository.Get<State>($"api/states/{Id}");
        state = responseHppt.Response!;
        cities = state.Cities!.ToList();
    }

    private async Task Delete(City city)
    {
        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/cities/{city.Id}");

        if (responseHTTP.Error)
        {
            if (responseHTTP.HttpResponseMessage.StatusCode == System.Net.HttpStatusCode.NotFound)
            {
                navigationManager.NavigateTo("/");
            }
            else
            {
                var mensajeError = await responseHTTP.GetErrorMessage();
                await swal.FireAsync("Error", mensajeError, SweetAlertIcon.Error);
            }
        }
        else
        {
            await Load();
        }
    }
}
```

21. En el proyecto **WEB** en la carpeta **Pages** creamos la carpeta **Cities** y dentro de esta creamos el componente **CityForm**:

```
@inject SweetAlertService swal

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

<EditForm EditContext="editContext" OnValidSubmit="OnValidSubmit">
    <DataAnnotationsValidator />
    <div class="mb-3">
        <label>Ciudad:</label>
        <div>
            <InputText class="form-control" @bind-Value="@City.Name" />
            <ValidationMessage For="@(() => City.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(City);
    }

    [EditorRequired]
```

```
    [Parameter]
    public City City { 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();
    }
}
```

22. En el proyecto **WEB** en la carpeta **Pages** en la carpeta **Cities** y dentro de esta creamos el componente **CreateCity**:

```
@page "/cities/create/{Id:int}"
@inject NavigationManager navigationManager
@inject IRepository repository
@inject SweetAlertService swal
```

```
<h3>Crear Ciudad</h3>

<CityForm @ref="cityForm" City="city" OnValidSubmit="Create" ReturnAction="Return" />

@code {
    private City city = new();
    private CityForm? cityForm;

    [Parameter]
    public int Id { get; set; }

    private async Task Create()
    {
        State state = new();
        var responseHTTP = await repository.Get<State>($"api/states/{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
        {
            state = responseHTTP.Response!;
        }

        if (state.Cities == null)
        {
            state.Cities = new List<City>();
        }

        state.Cities.Add(city);
        var httpResponse = await repository.Put("api/states", state);

        if (httpResponse.Error)
        {
            var mensajeError = await httpResponse.GetErrorMessage();
            await swal.FireAsync("Error", mensajeError, SweetAlertIcon.Error);
        }
        else
        {
            cityForm!.FormPostedSuccessfully = true;
            navigationManager.NavigateTo($"/states/details/{Id}");
        }
    }
```

```
    private void Return()
    {
        navigationManager.NavigateTo($"/states/details/{Id}");
    }
}
```

23. En el proyecto **WEB** en la carpeta **Pages** en la carpeta **Cities** y dentro de esta creamos el componente **EditCity**:

```
@page "/cities/edit/{StateId:int}/{CityId:int}"
@inject NavigationManager navigationManager
@inject IRepository repository
@inject SweetAlertService swal

<h3>Editar Ciudad</h3>

@if (city is null)
{
    <p>Cargando...</p>
}
else
{
    <CityForm @ref="cityForm" City="city" OnValidSubmit="Edit" ReturnAction="Return" />
}


@code {
    private City? city;
    private CityForm? cityForm;

    [Parameter]
    public int StateId { get; set; }

    [Parameter]
    public int CityId { get; set; }

    protected override async Task OnInitializedAsync()
    {
        var responseHTTP = await repository.Get<City>($"api/cities/{CityId}");

        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
```

```
        {
            city = responseHTTP.Response;
        }
    }

    private async Task Edit()
    {
        var responseHTTP = await repository.Put("api/cities", city);

        if (responseHTTP.Error)
        {
            var mensajeError = await responseHTTP.GetErrorMessage();
            await swal.FireAsync("Error", mensajeError, SweetAlertIcon.Error);
        }
        else
        {
            cityForm!.FormPostedSuccessfully = true;
            navigationManager.NavigateTo($"/states/details/{StateId}");
        }
    }

    private void Return()
    {
        navigationManager.NavigateTo($"/states/details/{StateId}");
    }
}
```

24. Probamos y hacemos el **commit**.

Poblar los Países, Estados y Ciudades con un API externa

1. Para llenar la información de todos, o al menos la mayorìa de paìses, estados y ciudades del mundo. Vamos a utilizar esta API: https://countrystatecity.in/docs/api/all-countries/ Para poderla utilizar vas a necesitar un token, puedes solicitar tu propio token en: https://docs.google.com/forms/d/e/1FAIpQLSciOf_227-3pKGKJok6TM0QF2PZhSgfQwy-F-bQaBj0OUgMmA/viewform llena el formulario y en pocas horas te lo enviarán (la menos eso paso conmigo), luego de tener tu token has los siguientes cambios al proyecto:

2. Al proyecto **API** agrega al **appstettings.json** los siguientes parámetros. No olvides cambiar el valor del **TokenValue** por la obtenida por usted en el paso anterior:

```
{
  "ConnectionStrings": {
    "DockerConnection": "Data Source=.;Initial Catalog=SalesPrep;User ID=sa;Password=Roger1974.;Connect
Timeout=30;Encrypt=False;TrustServerCertificate=False;ApplicationIntent=ReadWrite;MultiSubnetFailover=False"
  },
  "CoutriesAPI": {
    "urlBase": "https://api.countrystatecity.in",
    "tokenName": "X-CSCAPI-KEY",
    "tokenValue": "NUZicm9hR0FUb0oxUU5mck14NEY3cEFkcU9GR3VqdEhVOGZlODlIRQ=="
  },
  "Logging": {
```

```
  "LogLevel": {
    "Default": "Information",
    "Microsoft.AspNetCore": "Warning"
  }
 },
 "AllowedHosts": "*"
}
```

3. Al proyecto **Shared** dentro de la carpeta **Responses** las clases que vamos a obtener de la API. Empecemos primero con la clase genérica para todas las respuestas **Response**:

```
namespace Sales.Shared.Responses
{
    public class Response
    {
        public bool IsSuccess { get; set; }

        public string? Message { get; set; }

        public object? Result { get; set; }
    }
}
```

4. Luego continuamos con **CountryResponse**:

```
using Newtonsoft.Json;

namespace Sales.Shared.Responses
{
    public class CountryResponse
    {
        [JsonProperty("id")]
        public long Id { get; set; }

        [JsonProperty("name")]
        public string? Name { get; set; }

        [JsonProperty("iso2")]
        public string? Iso2 { get; set; }
    }
}
```

5. Continuamos con **StateResponse**:

```
using Newtonsoft.Json;

namespace Sales.Shared.Responses
{
    public class StateResponse
    {
        [JsonProperty("id")]
        public long Id { get; set; }
```

```
        [JsonProperty("name")]
        public string? Name { get; set; }

        [JsonProperty("iso2")]
        public string? Iso2 { get; set; }
    }
}
```

6. Y luego con **CityResponse**:

```
using Newtonsoft.Json;

namespace Sales.Shared.Responses
{
    public class CityResponse
    {
        [JsonProperty("id")]
        public long Id { get; set; }

        [JsonProperty("name")]
        public string? Name { get; set; }
    }
}
```

7. En el proyecto **API** creamos la carpeta **Services** y dentro de esta, la interfaz **IApiService**:

```
using Sales.Shared.Responses;

namespace Sales.API.Services
{
    public interface IApiService
    {
        Task<Response> GetListAsync<T>(string servicePrefix, string controller);
    }
}
```

8. Luego en la misma carpeta creamos la implementación en el **ApiService**:

```
using Newtonsoft.Json;
using Sales.Shared.Responses;

namespace Sales.API.Services
{
    public class ApiService : IApiService
    {
        private readonly IConfiguration _configuration;
        private readonly string _urlBase;
        private readonly string _tokenName;
        private readonly string _tokenValue;

        public ApiService(IConfiguration configuration)
        {
            _configuration = configuration;
```

```
            _urlBase = _configuration["CoutriesAPI:urlBase"]!;
            _tokenName = _configuration["CoutriesAPI:tokenName"]!;
            _tokenValue = _configuration["CoutriesAPI:tokenValue"]!;
        }

        public async Task<Response> GetListAsync<T>(string servicePrefix, string controller)
        {
            try
            {
                HttpClient client = new()
                {
                    BaseAddress = new Uri(_urlBase),
                };

                client.DefaultRequestHeaders.Add(_tokenName, _tokenValue);
                string url = $"{servicePrefix}{controller}";
                HttpResponseMessage response = await client.GetAsync(url);
                string result = await response.Content.ReadAsStringAsync();

                if (!response.IsSuccessStatusCode)
                {
                    return new Response
                    {
                        IsSuccess = false,
                        Message = result,
                    };
                }

                List<T> list = JsonConvert.DeserializeObject<List<T>>(result)!;
                return new Response
                {
                    IsSuccess = true,
                    Result = list
                };
            }
            catch (Exception ex)
            {
                return new Response
                {
                    IsSuccess = false,
                    Message = ex.Message
                };
            }
        }
    }
}
```

9. Y la inyectamos en el **Program** del proyecto **API**:

```
builder.Services.AddTransient<SeedDb>();
builder.Services.AddScoped<IApiService, ApiService>();
```

10. Luego modificamos el **SeedDb**:

```
using Microsoft.EntityFrameworkCore;
using Sales.API.Services;
using Sales.Shared.Entities;
using Sales.Shared.Responses;

namespace Sales.API.Data
{
    public class SeedDb
    {
        private readonly DataContext _context;
        private readonly IApiService _apiService;

        public SeedDb(DataContext context, IApiService apiService)
        {
            _context = context;
            _apiService = apiService;
        }

        public async Task SeedAsync()
        {
            await _context.Database.EnsureCreatedAsync();
            await CheckCountriesAsync();
        }

        private async Task CheckCountriesAsync()
        {
            if (!_context.Countries.Any())
            {
                Response responseCountries = await _apiService.GetListAsync<CountryResponse>("/v1", "/countries");
                if (responseCountries.IsSuccess)
                {
                    List<CountryResponse> countries = (List<CountryResponse>)responseCountries.Result!;
                    foreach (CountryResponse countryResponse in countries)
                    {
                        Country country = await _context.Countries!.FirstOrDefaultAsync(c => c.Name ==
countryResponse.Name!)!;
                        if (country == null)
                        {
                            country = new() { Name = countryResponse.Name!, States = new List<State>() };
                            Response responseStates = await _apiService.GetListAsync<StateResponse>("/v1",
$"/countries/{countryResponse.Iso2}/states");
                            if (responseStates.IsSuccess)
                            {
                                List<StateResponse> states = (List<StateResponse>)responseStates.Result!;
                                foreach (StateResponse stateResponse in states!)
                                {
                                    State state = country.States!.FirstOrDefault(s => s.Name == stateResponse.Name!)!;
                                    if (state == null)
                                    {
                                        state = new() { Name = stateResponse.Name!, Cities = new List<City>() };
                                        Response responseCities = await _apiService.GetListAsync<CityResponse>("/v1",
$"/countries/{countryResponse.Iso2}/states/{stateResponse.Iso2}/cities");
```

```
                    if (responseCities.IsSuccess)
                    {
                        List<CityResponse> cities = (List<CityResponse>)responseCities.Result!;
                        foreach (CityResponse cityResponse in cities)
                        {
                            if (cityResponse.Name == "Mosfellsbær" || cityResponse.Name == "Șăulița")
                            {
                                continue;
                            }
                            City city = state.Cities!.FirstOrDefault(c => c.Name == cityResponse.Name!)!;
                            if (city == null)
                            {
                                state.Cities.Add(new City() { Name = cityResponse.Name! });
                            }
                        }
                    }
                    if (state.CitiesNumber > 0)
                    {
                        country.States.Add(state);
                    }
                }
            }
            if (country.StatesNumber > 0)
            {
                _context.Countries.Add(country);
                await _context.SaveChangesAsync();
            }
        }
    }
}
```

11. Borramos la base de datos con **drop-database**.

12. Se puede demorar varias horas para llenar la mayoría de países con sus estados y ciudades. Digo la mayorìa porque la lógica deshecha algunos paises o estados que no tienen ciudades devueltas por la API.

13. Probamos y hacemos el **commit**.

Agregando paginación

1. En el projecto **Shared** creamos la carpeta **DTOs** y dentro de esta creamos la clase **PaginationDTO**:

```
namespace Sales.Shared.DTOs
{
    public class PaginationDTO
    {
        public int Page { get; set; } = 1;
```

```
        public int RecordsNumber { get; set; } = 10;
    }
}
```

2. En el proyecto **API** creamos el folder **Helpers** y dentro de este la clase **HttpContextExtensions**:

```
using Microsoft.EntityFrameworkCore;

namespace Sales.API.Helpers
{
    public static class HttpContextExtensions
    {
        public async static Task InsertPaginationParametersInResponse<T>(this HttpContext context, IQueryable<T>
queryable, int numberRecordsToShow)
        {
            if (context is null)
            {
                throw new ArgumentNullException(nameof(context));
            }

            double count = await queryable.CountAsync();
            double pageTotal = Math.Ceiling(count / numberRecordsToShow);
            context.Response.Headers.Add("count", count.ToString());
            context.Response.Headers.Add("pageTotal", pageTotal.ToString());
        }
    }
}
```

3. En la misma carpeta creamos la clase **QueryableExtensions**:

```
using Sales.Shared.DTOs;

namespace Sales.API.Helpers
{
    public static class QueryableExtensions
    {
        public static IQueryable<T> Paginate<T>(this IQueryable<T> queryable, PaginationDTO pagination)
        {
            return queryable
                .Skip((pagination.Page - 1) * pagination.RecordsNumber)
                .Take(pagination.RecordsNumber);
        }
    }
}
```

4. Modificamos el **CountriesController** para agregar la paginación en los métodos **GET**:

```
[HttpGet]
public async Task<ActionResult> Get([FromQuery] PaginationDTO pagination)
{
    var queryable = _context.Countries
        .Include(x => x.States)
```

```
        .AsQueryable();
    await HttpContext
        .InsertPaginationParametersInResponse(queryable, pagination.RecordsNumber);
    return Ok(await queryable
        .OrderBy(x => x.Name)
        .Paginar(pagination)
        .ToListAsync());
}

[HttpGet("full")]
public async Task<ActionResult> GetFull([FromQuery] PaginationDTO pagination)
{
    var queryable = _context.Countries
        .Include(x => x.States!)
        .ThenInclude(x => x.Cities)
        .AsQueryable();
    await HttpContext
        .InsertPaginationParametersInResponse(queryable, pagination.RecordsNumber);
    return Ok(await queryable
        .OrderBy(x => x.Name)
        .Paginar(pagination)
        .ToListAsync());
}
```

5. Probamos la paginación por el Swagger.

6. Creamos en el proyecto **WEB** en la carpeta **Shared** el componente **Pagination**:

```
<nav>
    <ul class="pagination justify-content-center">

        @foreach (var link in Links)
        {
            <li @onclick=@(() => InternalSelectedPage(link)) style="cursor: pointer" class="page-item @(link.Enable ? null :
"disabled") @(link.Enable ? "active" : null)">
                <a class="page-link">@link.Text</a>
            </li>
        }
    </ul>
</nav>

@code {
    [Parameter] public int CurrentPage { get; set; } = 1;
    [Parameter] public int TotalPages { get; set; }
    [Parameter] public int Radio { get; set; } = 5;
    [Parameter] public EventCallback<int> SelectedPage { get; set; }
    List<PageModel> Links = new();

    private async Task InternalSelectedPage(PageModel pageModel)
    {
        if (pageModel.Page == CurrentPage)
        {
            return;
```

```
        }

        if (!pageModel.Enable)
        {
            return;
        }

        await SelectedPage.InvokeAsync(pageModel.Page);
    }

    protected override void OnParametersSet()
    {
        Links = new List<PageModel>();

        var previousLinkEnable = CurrentPage != 1;
        var previousLinkPage = CurrentPage - 1;
        Links.Add(new PageModel
            {
                Text = "Anterior",
                Page = previousLinkPage,
                Enable = previousLinkEnable
            });

        for (int i = 1; i <= TotalPages; i++)
        {
            if (i >= CurrentPage - Radio && i <= CurrentPage + Radio)
            {
                Links.Add(new PageModel
                    {
                        Page = i,
                        Enable = CurrentPage == i,
                        Text = $"{i}"
                    });
            }
        }

        var linkNextEnable = CurrentPage != TotalPages;
        var linkNextPage = CurrentPage + 1;
        Links.Add(new PageModel
            {
                Text = "Siguiente",
                Page = linkNextPage,
                Enable = linkNextEnable
            });

    }

    class PageModel
    {
        public string Text { get; set; } = null!;
        public int Page { get; set; }
        public bool Enable { get; set; } = true;
        public bool Active { get; set; } = false;
```

```
    }
}
```

7. Modificamos nuestro componente **CountriesIndex**:

8. Probamos y hacemos el **commit**.

Acá vamos