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, debe ser privado

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

* Nuevo proyecto solución blank llamada **Stores**. Dentro de C://Projects al final la .sln queda el ícono dentro de Projects🡪(Stores.sln)

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

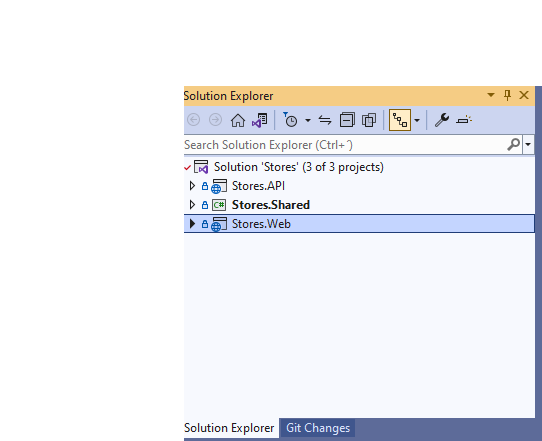
Descripción generada automáticamente

* Sobre el ícono Stores.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 **Stores.Shared**

Ubicación: C:\Projects/Stores

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

Así debe verse la estructura de los proyectos en el solution Explorer:

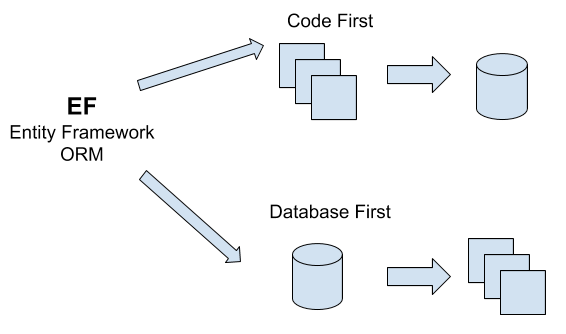


1. Hacemos el primer commit en nuestro repositorio. Pestaña Git Changes

\*(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

# Crear la BD con EF



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 **Shared** la carpeta **Entities** y dentro de estacarpeta la entidad **Country**:

using System.ComponentModel.DataAnnotations;

namespace Stores.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!;

}

}

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

using Microsoft.EntityFrameworkCore;

using Stores.Shared.Entities;

namespace Stores.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();

}

}

}

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

{

"ConnectionStrings": {

"DefaultConnection": "Server= MyServer;Database=Stores;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

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();

1. En el desplegable Startup Projects seleccionar Stores.API como proyecto de inicio, abrir Package Manager Console(Tool) , e igualmente elegir Stores.API ,como Default project
2. Correr los siguientes comandos en Package Manager Console:

add-migration InitialDb

update-database

1. Hacemos nuestro segundo **Commit**. All and Sync(Si en Git Changes no se visualiza el árbol de carpetas de los proyectos, será necesario cerrar la solución, y abrirla de nuevamente)
2. Hacemos nuestro segundo **Commit**. All and Sync(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 Stores.API.Data;

using Stores.Shared.Entities;

namespace Stores.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);

}

}

}

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:7236

builder.Services.AddScoped(sp => new HttpClient { BaseAddress = new Uri("https://localhost:7236/") });

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 Stores.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 Stores.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 Stores.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:7236/") });

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

await builder.Build().RunAsync();

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

@typeparam Titem

@if(MyList is null)

{

@if(Loading is null)

{

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

<img src="https://upload.wikimedia.org/wikipedia/commons/b/b1/Loading\_icon.gif?20151024034921" />

</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></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>

</**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 Stores.WEB Add🡪 Project reference>

Stores.Shared

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

@using Stores.WEB.Shared

@using Stores.Shared.Entities

@using Stores.WEB.Repositories

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

<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 solucion para que inicie al mismo tiempo el proyecto **API** y el proyecto **WEB**:

Vamos a las Solution StoresG1, click derecho properties:

Tabla

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**:

[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();

}

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 proyecto **WEB** configuramos la inyección del servicio de alertas:

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

builder.Services.AddSweetAlert2();

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>

</EditForm>

@code {

[EditorRequired]

[Parameter]

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

[EditorRequired]

[Parameter]

public EventCallback OnValidSubmit { 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">Adicionar 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" />

<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>

</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 **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.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**. **Asegurate 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.GetErrorMessageAsync();

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.GetErrorMessageAsync();

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">

<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"><i class="oi oi-pencil" />Editar</a>

<button class="btn btn-danger" @onclick=@(() => DeleteAsync(country))><i class="oi oi-trash" />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.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**.

# 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);

}

}

1. 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 StoresG1.Shared.Entities;

namespace Stores.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 = "USA" });

\_context.Countries.Add(new Country { Name = "Canadá" });

}

await \_context.SaveChangesAsync();

}

}

}

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

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

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();

}

}

1. Borramos la base de datos con el comando **drop-database**.
2. Probamos y hacemos el **commit**.

# Relación uno a muchos e índice compuesto

1. Creamos la entidad **State**:

using System.ComponentModel.DataAnnotations;

namespace Stores.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 int CountryId { get; set; }

public Country? Country { get; set; }

}

}

1. 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;

1. Creamos la entidad **City**:

using System.ComponentModel.DataAnnotations;

namespace Stores.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 int StateId { get; set; }

public State? State { get; set; }

}

}

1. 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;

1. 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("CountryId","Name",).IsUnique();

modelBuilder.Entity<City>().HasIndex("StateId","Name").IsUnique();

}

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

Agregar este using System.Text.Json.Serialization;

builder.Services.AddControllers()

.AddJsonOptions(x => x.JsonSerializerOptions.ReferenceHandler = ReferenceHandler.IgnoreCycles);

1. 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 = "Cundinamarca",

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();

}

1. 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());

}

1. Borramos la base de datos con el comando **drop-database** para que el Seeder vuelva a ejecutarse
2. 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**.
3. Cambiemos el **Index** de países en el proyecto WEB 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>

1. Probamos y hacemos el **commit**.

# Creando un CRUD multinivel

1. 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 Stores.API.Data;

using Stores.Shared.Entities;

namespace Stores.API.Controllers

{

[ApiController]

[Route("/api/states")]

public class StatesController : ControllerBase

{

private readonly DataContext \_context;

public StatesController(DataContext context)

{

\_context = context;

}

[HttpGet]

public async Task<IActionResult> GetAsync()

{

return Ok(await \_context.States

.Include(x => x.Cities)

.ToListAsync());

}

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

public async Task<IActionResult> GetAsync(int id)

{

var state = await \_context.States

.Include(x => x.Cities)

.FirstOrDefaultAsync(x => x.Id == id);

if (state == null)

{

return NotFound();

}

return Ok(state);

}

[HttpPost]

public async Task<ActionResult> PostAsync(State state)

{

try

{

\_context.Add(state);

await \_context.SaveChangesAsync();

return Ok(state);

}

catch (DbUpdateException dbUpdateException)

{

if (dbUpdateException.InnerException!.Message.Contains("duplicate"))

{

return BadRequest("Ya existe un estado/departamento con el mismo nombre.");

}

return BadRequest(dbUpdateException.Message);

}

catch (Exception exception)

{

return BadRequest(exception.Message);

}

}

[HttpPut]

public async Task<ActionResult> PutAsync(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 estado/departamento con el mismo nombre.");

}

return BadRequest(dbUpdateException.Message);

}

catch (Exception exception)

{

return BadRequest(exception.Message);

}

}

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

public async Task<IActionResult> DeleteAsync(int id)

{

var state = await \_context.States.FirstOrDefaultAsync(x => x.Id == id);

if (state == null)

{

return NotFound();

}

\_context.Remove(state);

await \_context.SaveChangesAsync();

return NoContent();

}

}

}

1. Luego creamos el **CitiesController**:

using Microsoft.AspNetCore.Mvc;

using Microsoft.EntityFrameworkCore;

using Stores.API.Data;

using Stores.Shared.Entities;

namespace Stores.API.Controllers

{

[ApiController]

[Route("/api/cities")]

public class CitiesController : ControllerBase

{

private readonly DataContext \_context;

public CitiesController(DataContext context)

{

\_context = context;

}

[HttpGet]

public async Task<IActionResult> GetAsync()

{

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

}

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

public async Task<IActionResult> GetAsync(int id)

{

var city = await \_context.Cities.FirstOrDefaultAsync(x => x.Id == id);

if (city == null)

{

return NotFound();

}

return Ok(city);

}

[HttpPost]

public async Task<ActionResult> PostAsync(City city)

{

try

{

\_context.Add(city);

await \_context.SaveChangesAsync();

return Ok(city);

}

catch (DbUpdateException dbUpdateException)

{

if (dbUpdateException.InnerException!.Message.Contains("duplicate"))

{

return BadRequest("Ya existe una ciudad con el mismo nombre.");

}

return BadRequest(dbUpdateException.Message);

}

catch (Exception exception)

{

return BadRequest(exception.Message);

}

}

[HttpPut]

public async Task<ActionResult> PutAsync(City city)

{

try

{

\_context.Update(city);

await \_context.SaveChangesAsync();

return Ok(city);

}

catch (DbUpdateException dbUpdateException)

{

if (dbUpdateException.InnerException!.Message.Contains("duplicate"))

{

return BadRequest("Ya existe una ciudad con el mismo nombre.");

}

return BadRequest(dbUpdateException.Message);

}

catch (Exception exception)

{

return BadRequest(exception.Message);

}

}

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

public async Task<IActionResult> DeleteAsync(int id)

{

var city = await \_context.Cities.FirstOrDefaultAsync(x => x.Id == id);

if (city == null)

{

return NotFound();

}

\_context.Remove(city);

await \_context.SaveChangesAsync();

return NoContent();

}

}

}

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

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

@inject IRepository repository

@inject NavigationManager navigationManager

@inject SweetAlertService sweetAlertService

@if(country is null)

{

<p>Cargando...</p>

} else

{

<h3>@country.Name</h3>

<div class="mb-2">

<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">

<Body>

<table class="table table-striped">

<thead>

<tr>

<th>Estado / Departamento</th>

<th>Ciudades</th>

<th></th>

</tr>

</thead>

<tbody>

@foreach (var state in states!)

{

<tr>

<td>

@state.Name

</td>

<td>

@state.CitiesNumber

</td>

<td>

<a class="btn btn-info" href="/states/details/@state.Id">Detalles</a>

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

<button class="btn btn-danger" @onclick=@(() => DeleteAsync(state.Id))>Borrar</button>

</td>

</tr>

}

</tbody>

</table>

</Body>

</GenericList>

}

@code {

private Country? country;

private List<State>? states;

[Parameter]

public int Id { get; set; }

protected override async Task OnInitializedAsync()

{

await LoadAsync();

}

private async Task LoadAsync()

{

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

if (responseHttp.Error)

{

if (responseHttp.HttpResponseMessage.StatusCode == HttpStatusCode.NotFound)

{

navigationManager.NavigateTo("/countries");

return;

}

var message = await responseHttp.GetErrorMessageAsync();

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

return;

}

country = responseHttp.Response;

states = country!.States!.ToList();

}

private async Task DeleteAsync(int id)

{

var result = await sweetAlertService.FireAsync(new SweetAlertOptions

{

Title = "Confirmación",

Text = "¿Realmente deseas eliminar el registro?",

Icon = SweetAlertIcon.Question,

ShowCancelButton = true,

CancelButtonText = "No",

ConfirmButtonText = "Si"

});

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

if (confirm)

{

return;

}

var responseHttp = await repository.Delete($"/api/states/{id}");

if (responseHttp.Error)

{

if (responseHttp.HttpResponseMessage.StatusCode != HttpStatusCode.NotFound)

{

var message = await responseHttp.GetErrorMessageAsync();

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

return;

}

}

await LoadAsync();

}

}

1. Probamos lo que llevamos hasta el momento.
2. 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 sweetAlertService

<NavigationLock OnBeforeInternalNavigation="OnBeforeInternalNavigation" />

<EditForm EditContext="editContext" OnValidSubmit="OnValidSubmit">

<DataAnnotationsValidator/>

<div class="mb-3">

<label>Estado/Departamenteo:</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!;

[Parameter]

[EditorRequired]

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

[Parameter]

[EditorRequired]

public EventCallback OnValidSubmit { get; set; }

[Parameter]

[EditorRequired]

public EventCallback ReturnAction { get; set; }

public bool FormPostedSuccessfully { get; set; }

protected override void OnInitialized()

{

editContext = new(State);

}

private async Task OnBeforeInternalNavigation(LocationChangingContext context)

{

var formWasMofied = editContext.IsModified();

if (!formWasMofied || FormPostedSuccessfully)

{

return;

}

var result = await sweetAlertService.FireAsync(new SweetAlertOptions

{

Title = "Confirmación",

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

Icon = SweetAlertIcon.Question,

ShowCancelButton = true,

CancelButtonText = "No",

ConfirmButtonText = "Si"

});

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

if (confirm)

{

return;

}

context.PreventNavigation();

}

}

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

@page "/states/create/{CountryId:int}"

@inject IRepository repository

@inject NavigationManager navigationManager

@inject SweetAlertService sweetAlertService

<h3>Crear Estado/Departamento</h3>

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

@code {

private State state = new();

private StateForm? stateForm;

[Parameter]

public int CountryId { get; set; }

private async Task CreateAsync()

{

state.CountryId = CountryId;

var httpResponse = await repository.Post("/api/states", state);

if (httpResponse.Error)

{

var message = await httpResponse.GetErrorMessageAsync();

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

return;

}

Return();

}

private void Return()

{

stateForm!.FormPostedSuccessfully = true;

navigationManager.NavigateTo($"/countries/details/{CountryId}");

}

}

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

@page "/states/edit/{StateId:int}"

@inject IRepository repository

@inject NavigationManager navigationManager

@inject SweetAlertService sweetAlertService

@inject NavigationManager navigationManager

<h3>Editar Estado/Departamento</h3>

@if (state is null)

{

<p>Cargando...</p>

}

else

{

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

}

@code {

private State? state;

private StateForm? stateForm;

[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 == HttpStatusCode.NotFound)

{

navigationManager.NavigateTo("/countries");

return;

}

var message = await responseHttp.GetErrorMessageAsync();

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

return;

}

state = responseHttp.Response;

}

private async Task EditAsync()

{

var responseHttp = await repository.Put("/api/states", state);

if (responseHttp.Error)

{

var message = await responseHttp.GetErrorMessageAsync();

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

return;

}

Return();

}

private void Return()

{

stateForm!.FormPostedSuccessfully = true;

navigationManager.NavigateTo($"/countries/details/{state!.CountryId}");

}

}

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

@page "/states/details/{StateId:int}"

@inject IRepository repository

@inject NavigationManager navigationManager

@inject SweetAlertService sweetAlertService

@if (state is null)

{

<p>Cargando...</p>

}

else

{

<h3>@state.Name</h3>

<div class="mb-2">

<a class="btn btn-primary" href="/cities/create/@state.Id">Nueva Ciuadad</a>

<a class="btn btn-success" href="/countries/details/@state.CountryId">Regresar</a>

</div>

<GenericList MyList="cities">

<Body>

<table class="table table-striped">

<thead>

<tr>

<th>Ciudad</th>

<th></th>

</tr>

</thead>

<tbody>

@foreach (var city in cities!)

{

<tr>

<td>

@city.Name

</td>

<td>

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

<button class="btn btn-danger" @onclick=@(() => DeleteAsync(city.Id))>Borrar</button>

</td>

</tr>

}

</tbody>

</table>

</Body>

</GenericList>

}

@code {

private State? state;

private List<City>? cities;

[Parameter]

public int StateId { get; set; }

protected override async Task OnInitializedAsync()

{

await LoadAsync();

}

private async Task LoadAsync()

{

var responseHttp = await repository.Get<State>($"/api/states/{StateId}");

if (responseHttp.Error)

{

if (responseHttp.HttpResponseMessage.StatusCode == HttpStatusCode.NotFound)

{

navigationManager.NavigateTo("/countries");

return;

}

var message = await responseHttp.GetErrorMessageAsync();

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

return;

}

state = responseHttp.Response;

cities = state!.Cities!.ToList();

}

private async Task DeleteAsync(int cityId)

{

var result = await sweetAlertService.FireAsync(new SweetAlertOptions

{

Title = "Confirmación",

Text = "¿Realmente deseas eliminar el registro?",

Icon = SweetAlertIcon.Question,

ShowCancelButton = true,

CancelButtonText = "No",

ConfirmButtonText = "Si"

});

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

if (confirm)

{

return;

}

var responseHttp = await repository.Delete($"/api/cities/{cityId}");

if (responseHttp.Error)

{

if (responseHttp.HttpResponseMessage.StatusCode != HttpStatusCode.NotFound)

{

var message = await responseHttp.GetErrorMessageAsync();

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

return;

}

}

await LoadAsync();

}

}

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

@inject SweetAlertService sweetAlertService

<NavigationLock OnBeforeInternalNavigation="OnBeforeInternalNavigation" />

<EditForm EditContext="editContext" OnValidSubmit="OnValidSubmit">

<DataAnnotationsValidator />

<div class="mb-3">

<label>Cuidad:</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!;

[Parameter]

[EditorRequired]

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

[Parameter]

[EditorRequired]

public EventCallback OnValidSubmit { get; set; }

[Parameter]

[EditorRequired]

public EventCallback ReturnAction { get; set; }

public bool FormPostedSuccessfully { get; set; }

protected override void OnInitialized()

{

editContext = new(City);

}

private async Task OnBeforeInternalNavigation(LocationChangingContext context)

{

var formWasMofied = editContext.IsModified();

if (!formWasMofied || FormPostedSuccessfully)

{

return;

}

var result = await sweetAlertService.FireAsync(new SweetAlertOptions

{

Title = "Confirmación",

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

Icon = SweetAlertIcon.Question,

ShowCancelButton = true,

CancelButtonText = "No",

ConfirmButtonText = "Si"

});

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

if (confirm)

{

return;

}

context.PreventNavigation();

}

}

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

@page "/cities/create/{StateId:int}"

@inject IRepository repository

@inject NavigationManager navigationManager

@inject SweetAlertService sweetAlertService

<h3>Crear Ciudad</h3>

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

@code {

private City city = new();

private CityForm? cityForm;

[Parameter]

public int StateId { get; set; }

private async Task CreateAsync()

{

city.StateId = StateId;

var httpResponse = await repository.Post("/api/cities", city);

if (httpResponse.Error)

{

var message = await httpResponse.GetErrorMessageAsync();

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

return;

}

Return();

}

private void Return()

{

cityForm!.FormPostedSuccessfully = true;

navigationManager.NavigateTo($"/states/details/{StateId}");

}

}

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

@page "/cities/edit/{CityId:int}"

@inject IRepository repository

@inject NavigationManager navigationManager

@inject SweetAlertService sweetAlertService

@inject NavigationManager navigationManager

<h3>Editar Ciudad</h3>

@if (city is null)

{

<p>Cargando...</p>

}

else

{

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

}

@code {

private City? city;

private CityForm? cityForm;

[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 == HttpStatusCode.NotFound)

{

navigationManager.NavigateTo("/countries");

return;

}

var message = await responseHttp.GetErrorMessageAsync();

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

return;

}

city = responseHttp.Response;

}

private async Task EditAsync()

{

var responseHttp = await repository.Put("/api/cities", city);

if (responseHttp.Error)

{

var message = await responseHttp.GetErrorMessageAsync();

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

return;

}

Return();

}

private void Return()

{

cityForm!.FormPostedSuccessfully = true;

navigationManager.NavigateTo($"/states/details/{city!.StateId}");

}

}

1. Probamos y hacemos el **commit**.