# Índice

Contenido

[Índice 1](#_Toc135289093)

[Modelo de datos 2](#_Toc135289094)

[Arquitectura del proyecto 3](#_Toc135289095)

[Crear la BD con EF 6](#_Toc135289096)

[Creando los primeros métodos en el primer controlador 9](#_Toc135289097)

[Creando nuestros primeros componentes en Blazor 11](#_Toc135289098)

[Completando las acciones de crear, editar y borrar países 21](#_Toc135289099)

[Solucionando el problema de países con el mismo nombre y adicionando un Seeder a la base de datos 38](#_Toc135289100)

[Relación uno a muchos e índice compuesto 42](#_Toc135289101)

[Creando un CRUD multinivel 51](#_Toc135289102)

[Poblar los Países, Estados y Ciudades con un API externa 83](#_Toc135289103)

[CRUD de Categorías 94](#_Toc135289104)

[Creando tablas de productos y listando productos 95](#_Toc135289105)

[Ejemplo adicional relación de muchos a muchos para una Biblioteca 102](#_Toc135289106)

[Agregando paginación 104](#_Toc135289107)

[Agregando filtros 123](#_Toc135289108)

[Creando las tablas de usuarios 146](#_Toc135289109)

[Creando sistema de seguridad 155](#_Toc135289110)

[Seguridad desde el backend 163](#_Toc135289111)

[Implementando el registro de usuarios, login & logout 172](#_Toc135289112)

[Habilitando tokens en swagger 185](#_Toc135289113)

[Mejorando el registro de usuarios con drop-down-lists en cascada 187](#_Toc135289114)

[Mejorando un poco la interfaz de usuario 192](#_Toc135289115)

[Almacenando la foto del usuario 202](#_Toc135289116)

[Editando el usuario 213](#_Toc135289117)

[Cambiando password del usuario 225](#_Toc135289118)

[Confirmar el registro de usuarios 230](#_Toc135289119)

[Reenviar correo de confirmación 242](#_Toc135289120)

[Actualización de la foto del usuario luego de editar usuario 248](#_Toc135289121)

[Recuperación de contraseña 249](#_Toc135289122)

[Solución del problema de la paginación 258](#_Toc135289123)

# Modelo de datos

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

Diagrama

Descripción generada automáticamente

# Arquitectura del proyecto

SQL Server Azure

.NET Core 7 API

Blazor WEB Application



Crear proyectos

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/StoresG8

* Nuevo proyecto solución blank llamada **StoresG8**. Dentro de C://Projects al final la .sln queda el ícono dentro de Projects🡪(StoresG8.sln)

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

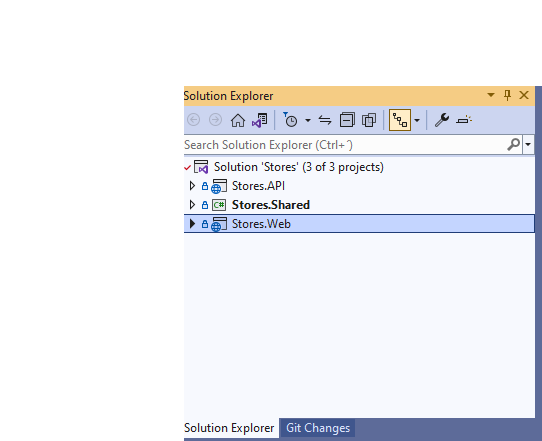
Descripción generada automáticamente

* Sobre el ícono StoresG8.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 **StoresG8.Shared**

Ubicación: C:\Projects/StoresG8

* Click derecho sobre la solución y agregamos un nuevo proyecto tipo: **ASP.NET Core Web API**, llamado **StoresG8.API**. Ubicación: C:\Projects/StoresG8
* Click derecho sobre la solución y agregamos un nuevo proyecto tipo: **Blazor WebAssembly App**, llamado **StoresG8.WEB**. Ubicación: C:\Projects/StoresG8

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

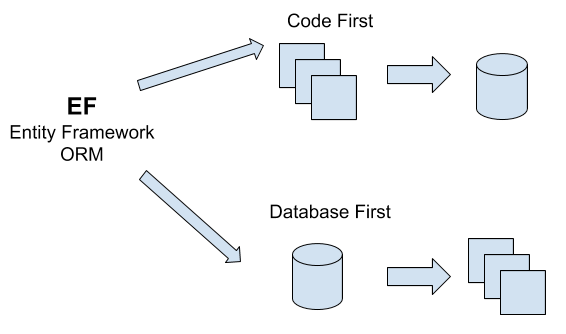


Hacemos el primer commit en nuestro repositorio. Pestaña Git Changes

\*(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 **StoresG8.Shared** la carpeta **Entities** y dentro de esta carpeta la entidad **Country**:

using System.ComponentModel.DataAnnotations;

namespace StoresG8.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 StoresG8.Shared.Entities;

namespace StoresG8.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.Development.json** y en **appsettings.json** del proyecto **API**:

{

"ConnectionStrings": {

"DefaultConnection": "Server= MyServer;Database=StoresG8;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 StoresG8.API como proyecto de inicio, abrir Package Manager Console(Tool) , e igualmente elegir StoresG8.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)

# 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 StoresG8.API.Data;

using StoresG8.Shared.Entities;

namespace StoresG8.API.Controllers

{

[ApiController]

[Route("/api/countries")]

public class CountriesController : ControllerBase

{

private readonly DataContext \_context;

public CountriesController(DataContext context)

{

\_context = context;

}

[HttpPost]

public async Task<ActionResult> Post(Country country)

{

\_context.Add(country);

await \_context.SaveChangesAsync();

return Ok(country);

}

[HttpGet]

public async Task<ActionResult> Get()

{

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

}

}

}

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 StoresG8.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 StoresG8.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 StoresG8.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://img.pikbest.com/png-images/20190918/cartoon-snail-loading-loading-gif-animation\_2734139.png!bw700 " />

</div>

}

else

{

@Loading

}

}

else if(MyList.Count == 0)

{

@if(NoRecords is null)

{

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

}

else

{

@NoRecords

}

}

else

{

@Body

}

@code {

[Parameter]

public RenderFragment? Loading { get; set; }

[Parameter]

public RenderFragment? NoRecords { get; set; }

[Parameter]

[EditorRequired]

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

[Parameter]

[EditorRequired]

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

}

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

@page "/countries"

@inject IRepository repository

<h3>Paises</h3>

<div class="mb-3">

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

</div>

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

<**Body**>

<table class="table table-striped">

<thead>

<tr>

<th>País</th>

<th></th>

</tr>

</thead>

<tbody>

@foreach (var country in Countries!)

{

<tr>

<td>

@country.Name

</td>

<td>

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

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

</td>

</tr>

}

</tbody>

</table>

</**Body**>

</**GenericList**>

@code {

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

protected async override Task OnInitializedAsync()

{

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

Countries = responseHppt.Response!;

}

}

\*Importante:

Agregamos una referencia al proyecto Web para que obtenga comunicación con el proyecto Shared (Click derecho sobre el proyecto StoresG8.WEB Add reference>

StoresG8.Shares

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

@using StoresG8.WEB.Shared

@using StoresG8.Shared.Entities

@using StoresG8.WEB.Repositories

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

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

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

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

</NavLink>

</div>

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

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

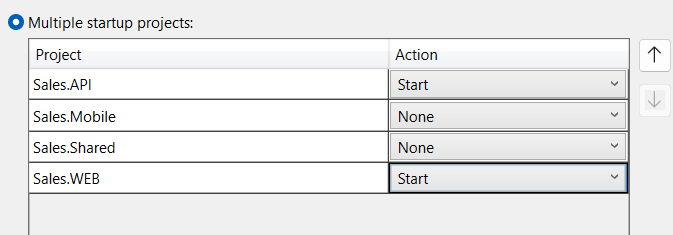
<span class="oi oi-list-rich" aria-hidden="true"></span> Países

</NavLink>

</div>

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

Vamos a las Solution StoresG8, click derecho properties:



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();

Insertar lo siguiente en el archivo imports del proyecto Web

@using Currie.Technologies.Razor.sweetAlert2

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

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

<DataAnnotationsValidator />

<div class="mb-3">

<label>País:</label>

<div>

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

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

</div>

</div>

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

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

</EditForm>

@code {

[EditorRequired]

[Parameter]

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

[EditorRequired]

[Parameter]

public EventCallback 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">Nuevo País</a>

<GenericList MyList="Countries">

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

@inject SweetAlertService swal

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

<EditForm EditContext="editContext" 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();

}

}

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

@page "/countries/create"

@inject NavigationManager navigationManager

@inject IRepository repository

@inject SweetAlertService swal

<h3>Crear País</h3>

<CountryForm @ref="countryForm" Country="country" 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**. **Asegúrate de presionar Ctrl + F5, para que te tome los cambios**.
2. Ahora creamos el componente **CountryEdit**:

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

@inject NavigationManager navigationManager

@inject IRepository repository

@inject SweetAlertService swal

<h3>Editar País</h3>

@if (country is null)

{

<p>Cargando...</p>

}

else

{

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

}

@code {

private Country? country;

private CountryForm? countryForm;

[Parameter]

public int Id { get; set; }

protected override async Task OnInitializedAsync()

{

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

if (responseHTTP.Error)

{

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

{

navigationManager.NavigateTo("countries");

}

else

{

var messageError = await responseHTTP.GetErrorMessage();

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

}

}

else

{

country = responseHTTP.Response;

}

}

private async Task Edit()

{

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

if (responseHTTP.Error)

{

var mensajeError = await responseHTTP.GetErrorMessage();

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

}

else

{

countryForm!.FormPostedSuccessfully = true;

navigationManager.NavigateTo("countries");

}

}

private void Return()

{

navigationManager.NavigateTo("countries");

}

}

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

@page "/countries"

@inject IRepository repository

@inject NavigationManager navigationManager

@inject SweetAlertService swal

<h3>Paises</h3>

<div class="mb-3">

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

</div>

<GenericList MyList="Countries">

<Body>

<table class="table table-striped">

<thead>

<tr>

<th>País</th>

<th></th>

</tr>

</thead>

<tbody>

@foreach (var country in Countries!)

{

<tr>

<td>

@country.Name

</td>

<td>

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

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

</td>

</tr>

}

</tbody>

</table>

</Body>

</GenericList>

@code {

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

protected async override Task OnInitializedAsync()

{

await Load();

}

private async Task Load()

{

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

Countries = responseHppt.Response!;

}

private async Task Delete(Country country)

{

var result = await swal.FireAsync(new SweetAlertOptions

{

Title = "Confirmación",

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

Icon = SweetAlertIcon.Question,

ShowCancelButton = true

});

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

if (confirm)

{

return;

}

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

if (responseHTTP.Error)

{

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

{

navigationManager.NavigateTo("/");

}

else

{

var mensajeError = await responseHTTP.GetErrorMessageAsync();

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

}

}

else

{

await Load();

}

}

}

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

# 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 StoresG8.Shared.Entities;

namespace StoresG8.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" });

}

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=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();

}

}

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 StoresG8.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 StoresG8.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**:

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 = "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();

}

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("{id:int}")] ///api/countries/1

public async Task<ActionResult> Get(int id)

{

var country = await \_context.Countries

.Include (x => x.States!)

.ThenInclude(x => x.Cities!)

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

if (country is null)

{

return NotFound();

}

return Ok(country);

}

[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-database**.
3. Cambiemos el **CountryIndex** 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>País</th>

<th>Departamentos/Estados</th>

<th></th>

</tr>

</thead>

<tbody>

@foreach (var country in Countries!)

{

<tr>

<

<td>

@country.Name

</td>

<td>

@country.StatesNumber

</td>

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>

</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 StoresG8.API.Data;

using StoresG8.Shared.Entities;

namespace StoresG8.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 StoresG8.API.Data;

using StoresG8.Shared.Entities;

namespace StoresG8.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}"

@using StoresG8.Shared.Entities;

@using System.Net;

@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. Modificamos CountriesController para agregar el **.Include** con States y Cities

[HttpGet("{id:int}")] ///api/countries/1

public async Task<ActionResult> Get(int id)

{

var country = await \_context.Countries

.Include (x => x.States!)

.ThenInclude(x => x.Cities!)

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

if (country is null)

{

return NotFound();

}

return Ok(country);

}

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

@using StoresG8.Shared.Entities;

<NavigationLock OnBeforeInternalNavigation="OnBeforeInternalNavigation" />

<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" type="button" @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**

@using StoresG8.Shared.Entities;

@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**

@using StoresG8.Shared.Entities;

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

@inject IRepository repository

@inject NavigationManager navigationManager

@inject SweetAlertService sweetAlertService

@inject NavigationManager navigationManager

@using System.Net;

<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}"

@using System.Net;

@using StoresG8.Shared.Entities;

@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

@using StoresG8.Shared.Entities;

<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

@using StoresG8.Shared.Entities;

<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

@using StoresG8.Shared.Entities;

@using System.Net;

<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**.

# Poblar los Países, Estados y Ciudades con un API externa

1. Para llenar la información de todos, o al menos la mayoría de países, 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 , 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** que ha recibido:

{

"ConnectionStrings": {

"DefaultConnection": "Server= OALARCON;Database=StoresG8;Encrypt=False;User Id=dba;Password=Abcd1234\*;"

},

"CountriesAPI": {

"urlBase": "https://api.countrystatecity.in",

"tokenName": "X-CSCAPI-KEY",

"tokenValue": "T29KVmZZUTIyVkRmVVF5OXV2b0tNSnBxNERvY2cyQ295YmhHT0dzQQ=="

},

"Logging": {

"LogLevel": {

"Default": "Information",

"Microsoft.AspNetCore": "Warning"

}

},

"AllowedHosts": "\*"

}

1. Dentro del proyecto **Shared** creamos la carpeta **Responses** Empecemos primero con la clase genérica para todas las respuestas , creamos dentro la clase **Response**

namespace StoresG8.Shared.Responses

{

public class Response

{

public bool IsSuccess { get; set; }

public string? Message { get; set; }

public object? Result { get; set; }

}

}

1. Luego continuamos con **CountryResponse**

**Instalamos el Nugget** Newtonsoft.Json dentro del proyecto **Shared**

using Newtonsoft.Json;

namespace StoresG8.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; }

}

}

1. Creamos la clase **StateResponse**

using Newtonsoft.Json;

namespace StoresG8.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; }

}

}

1. Y luego creamos la clase **CityResponse**

using Newtonsoft.Json;

namespace StoresG8.Shared.Responses

{

public class CityResponse

{

[JsonProperty("id")]

public long Id { get; set; }

[JsonProperty("name")]

public string? Name { get; set; }

}

}

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

using StoresG8.Shared.Responses;

namespace StoresG8.API.Services

{

public interface IApiService

{

Task<Response> GetListAsync<T>(string servicePrefix, string controller);

}

}

1. Luego en la misma carpeta creamos la implementación en el **ApiService**

using Newtonsoft.Json;

using StoresG8.Shared.Responses;

namespace StoresG8.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["CountriesAPI:urlBase"]!;

\_tokenName = \_configuration["CountriesAPI:tokenName"]!;

\_tokenValue = \_configuration["CountriesAPI: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

};

}

}

}

}

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

builder.Services.AddTransient<SeedDb>();

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

1. Luego modificamos el **SeedDb**:

using Microsoft.EntityFrameworkCore;

using StoresG8.API.Services;

using StoresG8.Shared.Entities;

using StoresG8.Shared.Responses;

namespace StoresG8.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();

}

}

}

}

}

}

}

}

1. Borramos la base de datos con **drop-database**
2. Se puede demorar varias horas para llenar la mayoría de los países con sus estados y ciudades. Digo la mayoría porque la lógica deshecha algunos países o estados que no tienen ciudades devueltas por la API.
3. Probamos y hacemos el **commit**.

## CRUD de Categorías

1. En **StoresG8G8.Shared.Entities** adicionamos la entidad **Category**:

using System.ComponentModel.DataAnnotations;

namespace StoresG8G8.Shared.Entities

{

public class Category

{

public int Id { get; set; }

[Display(Name = "Categoría")]

[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. Modificamos el **DataContext**:

public class DataContext : IdentityDbContext<User>

{

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

{

}

public DbSet<Category> Categories { get; set; }

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(x => x.Name).IsUnique();

modelBuilder.Entity<Category>().HasIndex(x => x.Name).IsUnique();

modelBuilder.Entity<State>().HasIndex("CountryId", "Name").IsUnique();

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

}

}

1. Corremos los comandos para crear la nueva migración y aplicarla:

PM> add-migration AddCategories

PM> update-database

## Creando tablas de productos y listando productos

1. Creamos la entidad **Product**:

using Microsoft.EntityFrameworkCore.Metadata.Internal;

using System.ComponentModel.DataAnnotations;

using System.ComponentModel.DataAnnotations.Schema;

namespace StoresG8G8.Shared.Entities

{

public class Product

{

public int Id { get; set; }

[Display(Name = "Nombre")]

[MaxLength(50, ErrorMessage = "El campo {0} debe tener máximo {1} caractéres.")]

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

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

[DataType(DataType.MultilineText)]

[Display(Name = "Descripción")]

[MaxLength(500, ErrorMessage = "El campo {0} debe tener máximo {1} caractéres.")]

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

[Column(TypeName = "decimal(18,2)")]

[DisplayFormat(DataFormatString = "{0:C2}")]

[Display(Name = "Precio")]

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

public decimal Price { get; set; }

[DisplayFormat(DataFormatString = "{0:N2}")]

[Display(Name = "Inventario")]

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

public float Stock { get; set; }

}

}

1. Creamos la entidad **ProductImage**:

using System.ComponentModel.DataAnnotations;

namespace StoresG8G8.Shared.Entities

{

public class ProductImage

{

public int Id { get; set; }

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

public int ProductId { get; set; }

[Display(Name = "Imagen")]

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

}

}

1. Creamos la entidad **ProductCategory**:

namespace StoresG8G8.Shared.Entities

{

public class ProductCategory

{

public int Id { get; set; }

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

public int ProductId { get; set; }

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

public int CategoryId { get; set; }

}

}

1. Modificamos la entidad **Category**:

public class Category

{

public int Id { get; set; }

[Display(Name = "Categoría")]

[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 ICollection<ProductCategory>? ProductCategories { get; set; }

[Display(Name = "Productos")]

public int ProductCategoriesNumber => ProductCategories == null ? 0 : ProductCategories.Count;

}

1. Modificamos la entidad **Product**:

public class Product

{

public int Id { get; set; }

[Display(Name = "Nombre")]

[MaxLength(50, ErrorMessage = "El campo {0} debe tener máximo {1} caractéres.")]

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

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

[DataType(DataType.MultilineText)]

[Display(Name = "Descripción")]

[MaxLength(500, ErrorMessage = "El campo {0} debe tener máximo {1} caractéres.")]

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

[Column(TypeName = "decimal(18,2)")]

[DisplayFormat(DataFormatString = "{0:C2}")]

[Display(Name = "Precio")]

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

public decimal Price { get; set; }

[DisplayFormat(DataFormatString = "{0:N2}")]

[Display(Name = "Inventario")]

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

public float Stock { get; set; }

public ICollection<ProductCategory>? ProductCategories { get; set; }

[Display(Name = "Categorías")]

public int ProductCategoriesNumber => ProductCategories == null ? 0 : ProductCategories.Count;

public ICollection<ProductImage>? ProductImages { get; set; }

[Display(Name = "Imágenes")]

public int ProductImagesNumber => ProductImages == null ? 0 : ProductImages.Count;

[Display(Name = "Imagén")]

public string MainImage => ProductImages == null ? string.Empty : ProductImages.FirstOrDefault()!.Image;

}

1. Modificamos el **DataContext**.

public class DataContext : IdentityDbContext<User>

{

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

{

}

public DbSet<Category> Categories { get; set; }

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

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

public DbSet<Product> Products { get; set; }

public DbSet<ProductCategory> ProductCategories { get; set; }

public DbSet<ProductImage> ProductImages { get; set; }

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

protected override void OnModelCreating(ModelBuilder modelBuilder)

{

base.OnModelCreating(modelBuilder);

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

modelBuilder.Entity<Category>().HasIndex(x => x.Name).IsUnique();

modelBuilder.Entity<Product>().HasIndex(x => x.Name).IsUnique();

modelBuilder.Entity<State>().HasIndex("CountryId", "Name").IsUnique();

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

}

}

1. Corremos los siguientes comandos para aplicar la migracion y correrla:

PM> add-migration AddProductsTables

PM> update-database

## Ejemplo adicional relación de muchos a muchos para una Biblioteca

using System.ComponentModel.DataAnnotations;

using System.ComponentModel.DataAnnotations.Schema;

using System.Text.Json.Serialization;

public class Libro

{

public int Id { get; set; }

[Required]

[StringLength(40)]

public string Titulo { get; set; }

[Required]

public DateTime FechaPublicacion { get; set; }

//public List<Comentario> Comentarios { get; set; }

[NotMapped]

[JsonIgnore]

public List<AutorLibro>? AutorLibros { get; set; }

}

using System.ComponentModel.DataAnnotations;

using System.ComponentModel.DataAnnotations.Schema;

using System.Text.Json.Serialization;

namespace StoresG8G8.Shared.Entities

{

public class Autor

{

public int Id { get; set; }

[Required]

[StringLength(40)]

public string Nombre { get; set; }

[NotMapped]

[JsonIgnore]

public List<AutorLibro> ?AutorLibros { get; set; }

}

}

System.ComponentModel.DataAnnotations.Schema;

using System.Text.Json.Serialization;

namespace StoresG8G8.Shared.Entities

{

public class AutorLibro

{

public int Id { get; set; }

public int Orden { get; set; }

public int ?LibroId { get; set; }

[JsonIgnore]

public Libro Libro { get; set; }

public int ?AutorId { get; set; }

[JsonIgnore]

public Autor Autor { get; set; }

}

}

## Agregando paginación

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

namespace StoresG8G8.Shared.DTOs

{

public class PaginationDTO

{

public int Id { get; set; }

public int Page { get; set; } = 1;

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

}

}

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

using StoresG8G8.Shared.DTOs;

namespace StoresG8G8.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);

}

}

}

1. Modificamos el **CountriesController** para agregar la paginación en el método **GET** y de paso agregamos el método **GetPages**:

[HttpGet]

public async Task<IActionResult> GetAsync([FromQuery] PaginationDTO pagination)

{

var queryable = \_context.Countries

.Include(x => x.States)

.AsQueryable();

return Ok(await queryable

.OrderBy(x => x.Name)

.Paginate(pagination)

.ToListAsync());

}

[HttpGet("totalPages")]

public async Task<ActionResult> GetPages([FromQuery] PaginationDTO pagination)

{

var queryable = \_context.Countries.AsQueryable();

double count = await queryable.CountAsync();

double totalPages = Math.Ceiling(count / pagination.RecordsNumber);

return Ok(totalPages);

}

1. Probamos la paginación con Swagger.
2. Creamos en el proyecto **WEB** en la carpeta **Shared** el componente **Pagination**:

<nav>

<ul class="pagination">

@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();

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

});

}

private async Task InternalSelectedPage(PageModel pageModel)

{

if (pageModel.Page == CurrentPage || pageModel.Page == 0)

{

return;

}

await SelectedPage.InvokeAsync(pageModel.Page);

}

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;

}

}

1. Modificamos nuestro componente **CountriesIndex**:

@page "/countries"

@inject IRepository repository

@inject NavigationManager navigationManager

@inject SweetAlertService sweetAlertService

<h3>Países</h3>

<Pagination CurrentPage="currentPage"

TotalPages="totalPages"

SelectedPage="SelectedPage" />

<GenericList MyList="Countries">

<Body>

<table class="table table-striped">

<thead>

<tr>

<th>País</th>

<th style="width:220px">Estados / Departamentos</th>

<th style="width:280px"></th>

</tr>

</thead>

<tbody>

@foreach (var country in Countries!)

{

<tr>

<td>

@country.Name

</td>

<td>

@country.StatesNumber

</td>

<td>

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

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

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

</td>

</tr>

}

</tbody>

</table>

</Body>

</GenericList>

@code {

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

private int currentPage = 1;

private int totalPages;

protected override async Task OnInitializedAsync()

{

await LoadAsync();

}

private async Task SelectedPage(int page)

{

currentPage = page;

await LoadAsync(page);

}

private async Task LoadAsync(int page = 1)

{

string url1 = $"api/countries?page={page}";

string url2 = $"api/countries/totalPages";

var responseHppt = await repository.Get<List<Country>>(url1);

var responseHppt2 = await repository.Get<int>(url2);

Countries = responseHppt.Response!;

totalPages = responseHppt2.Response!;

}

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/countries/{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.
2. Ahora vamos hacer lo mismo para estados. Empezamos modificando el GET del **StatesController** y de paso creamos el método para obtener el número de página:

[HttpGet]

public async Task<ActionResult> Get([FromQuery] PaginationDTO pagination)

{

var queryable = \_context.States

.Include(x => x.Cities)

.Where(x => x.Country!.Id == pagination.Id)

.AsQueryable();

return Ok(await queryable

.OrderBy(x => x.Name)

.Paginate(pagination)

.ToListAsync());

}

[HttpGet("totalPages")]

public async Task<ActionResult> GetPages([FromQuery] PaginationDTO pagination)

{

var queryable = \_context.States

.Where(x => x.Country!.Id == pagination.Id)

.AsQueryable();

double count = await queryable.CountAsync();

double totalPages = Math.Ceiling(count / pagination.RecordsNumber);

return Ok(totalPages);

}

1. Probamos en swagger:
2. Luego modificamos el **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>

<Pagination CurrentPage="currentPage"

TotalPages="totalPages"

SelectedPage="SelectedPage" />

<GenericList MyList="sates!">

<Body>

<table class="table table-striped">

<thead>

<tr>

<th>Estado / Departamento</th>

<th style="width:140px">Ciudades</th>

<th style="width:260px"></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;

private int currentPage = 1;

private int totalPages;

[Parameter]

public int Id { get; set; }

protected override async Task OnInitializedAsync()

{

await LoadAsync();

}

private async Task SelectedPage(int page)

{

currentPage = page;

await LoadAsync(page);

}

private async Task LoadAsync(int page = 1)

{

string url1 = $"api/states?id={Id}&page={page}";

string url2 = $"api/states/totalPages?id={Id}";

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

var responseHppt2 = await repository.Get<List<State>>(url1);

var responseHppt3 = await repository.Get<int>(url2);

country = responseHppt.Response;

states = responseHppt2.Response;

totalPages = responseHppt3.Response;

}

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.
2. Ahora vamos hacer lo mismo para ciudades. Empezamos modificando el GET del **CitiesController** y de paso creamos el método para obtener el número de página:

[HttpGet]

public async Task<ActionResult> Get([FromQuery] PaginationDTO pagination)

{

var queryable = \_context.Cities

.Where(x => x.State!.Id == pagination.Id)

.AsQueryable();

return Ok(await queryable

.OrderBy(x => x.Name)

.Paginate(pagination)

.ToListAsync());

}

[HttpGet("totalPages")]

public async Task<ActionResult> GetPages([FromQuery] PaginationDTO pagination)

{

var queryable = \_context.Cities

.Where(x => x.State!.Id == pagination.Id)

.AsQueryable();

double count = await queryable.CountAsync();

double totalPages = Math.Ceiling(count / pagination.RecordsNumber);

return Ok(totalPages);

}

1. Probamos en swagger:
2. Luego modificamos el **StateDetail**:

@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>

<Pagination CurrentPage="currentPage"

TotalPages="totalPages"

SelectedPage="SelectedPage" />

<GenericList MyList="cities!">

<Body>

<table class="table table-striped">

<thead>

<tr>

<th>Ciudad</th>

<th style="width:180px"></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;

private int currentPage = 1;

private int totalPages;

[Parameter]

public int StateId { get; set; }

protected override async Task OnInitializedAsync()

{

await LoadAsync();

}

private async Task SelectedPage(int page)

{

currentPage = page;

await LoadAsync(page);

}

private async Task LoadAsync(int page = 1)

{

string url1 = $"api/cities?id={StateId}&page={page}";

string url2 = $"api/cities/totalPages?id={StateId}";

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

var responseHppt2 = await repository.Get<List<City>>(url1);

var responseHppt3 = await repository.Get<int>(url2);

state = responseHppt.Response;

cities = responseHppt2.Response;

totalPages = responseHppt3.Response;

}

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();

}

private async Task CleanFilterAsync()

{

Filter = string.Empty;

await ApplyFilterAsync();

}

}

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

## Agregando filtros

1. En el projecto **Shared** modificamos la clase **PaginationDTO**:

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

public string? Filter { get; set; }

1. En el projecto **API** modificamos los métodos **Get** y **GetPages** del controlador **CountriesController**:

[HttpGet]

public async Task<IActionResult> GetAsync([FromQuery] PaginationDTO pagination)

{

var queryable = \_context.Countries

.Include(x => x.States)

.AsQueryable();

if (!string.IsNullOrWhiteSpace(pagination.Filter))

{

queryable = queryable.Where(x => x.Name.ToLower().Contains(pagination.Filter.ToLower()));

}

return Ok(await queryable

.OrderBy(x => x.Name)

.Paginate(pagination)

.ToListAsync());

}

[HttpGet("totalPages")]

public async Task<ActionResult> GetPages([FromQuery] PaginationDTO pagination)

{

var queryable = \_context.Countries.AsQueryable();

if (!string.IsNullOrWhiteSpace(pagination.Filter))

{

queryable = queryable.Where(x => x.Name.ToLower().Contains(pagination.Filter.ToLower()));

}

double count = await queryable.CountAsync();

double totalPages = Math.Ceiling(count / pagination.RecordsNumber);

return Ok(totalPages);

}

1. En el projecto **WEB** modificamos el **CountriesIndex**:

@page "/countries"

@inject IRepository repository

@inject NavigationManager navigationManager

@inject SweetAlertService sweetAlertService

<h3>Países</h3>

<div class="mb-2" style="display: flex; flex-wrap:wrap; align-items: center;">

<div>

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

</div>

<div class="mx-2">

<input style="width: 400px;" type="text" class="form-control" id="titulo" placeholder="Buscar país..." @bind-value="Filter" />

</div>

<div>

<button type="button" class="btn btn-outline-primary" @onclick="ApplyFilterAsync">Filtrar</button>

<button type="button" class="btn btn-outline-danger" @onclick="CleanFilterAsync">Limpiar</button>

</div>

</div>

<Pagination CurrentPage="currentPage"

TotalPages="totalPages"

SelectedPage="SelectedPage" />

<GenericList MyList="Countries">

<Body>

<table class="table table-striped">

<thead>

<tr>

<th>País</th>

<th style="width:220px">Estados / Departamentos</th>

<th style="width:260px"></th>

</tr>

</thead>

<tbody>

@foreach (var country in Countries!)

{

<tr>

<td>

@country.Name

</td>

<td>

@country.StatesNumber

</td>

<td>

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

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

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

</td>

</tr>

}

</tbody>

</table>

</Body>

</GenericList>

@code {

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

private int currentPage = 1;

private int totalPages;

[Parameter]

[SupplyParameterFromQuery]

public string Page { get; set; } = "";

[Parameter]

[SupplyParameterFromQuery]

public string Filter { get; set; } = "";

protected override async Task OnInitializedAsync()

{

await LoadAsync();

}

private async Task SelectedPage(int page)

{

currentPage = page;

await LoadAsync(page);

}

private async Task LoadAsync(int page = 1)

{

if (!string.IsNullOrWhiteSpace(Page))

{

page = Convert.ToInt32(Page);

}

string url1 = string.Empty;

string url2 = string.Empty;

if (string.IsNullOrEmpty(Filter))

{

url1 = $"api/countries?page={page}";

url2 = $"api/countries/totalPages";

}

else

{

url1 = $"api/countries?page={page}&filter={Filter}";

url2 = $"api/countries/totalPages?filter={Filter}";

}

var responseHppt = await repository.Get<List<Country>>(url1);

var responseHppt2 = await repository.Get<int>(url2);

Countries = responseHppt.Response!;

totalPages = responseHppt2.Response!;

}

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/countries/{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();

}

private async Task CleanFilterAsync()

{

Filter = string.Empty;

await ApplyFilterAsync();

}

private async Task ApplyFilterAsync()

{

int page = 1;

await LoadAsync(page);

await SelectedPage(page);

}

}

1. Probamos y hacemos el **commit**.
2. Replicamos para estados y ciudades, primero modificamos el **StatesController**:

[HttpGet]

public async Task<ActionResult> Get([FromQuery] PaginationDTO pagination)

{

var queryable = \_context.States

.Include(x => x.Cities)

.Where(x => x.Country!.Id == pagination.Id)

.AsQueryable();

if (!string.IsNullOrWhiteSpace(pagination.Filter))

{

queryable = queryable.Where(x => x.Name.ToLower().Contains(pagination.Filter.ToLower()));

}

return Ok(await queryable

.OrderBy(x => x.Name)

.Paginate(pagination)

.ToListAsync());

}

[HttpGet("totalPages")]

public async Task<ActionResult> GetPages([FromQuery] PaginationDTO pagination)

{

var queryable = \_context.States

.Where(x => x.Country!.Id == pagination.Id)

.AsQueryable();

if (!string.IsNullOrWhiteSpace(pagination.Filter))

{

queryable = queryable.Where(x => x.Name.ToLower().Contains(pagination.Filter.ToLower()));

}

double count = await queryable.CountAsync();

double totalPages = Math.Ceiling(count / pagination.RecordsNumber);

return Ok(totalPages);

}

1. Luego modificamos el **CitiesController**:

[HttpGet]

public async Task<ActionResult> Get([FromQuery] PaginationDTO pagination)

{

var queryable = \_context.Cities

.Where(x => x.State!.Id == pagination.Id)

.AsQueryable();

if (!string.IsNullOrWhiteSpace(pagination.Filter))

{

queryable = queryable.Where(x => x.Name.ToLower().Contains(pagination.Filter.ToLower()));

}

return Ok(await queryable

.OrderBy(x => x.Name)

.Paginate(pagination)

.ToListAsync());

}

[HttpGet("totalPages")]

public async Task<ActionResult> GetPages([FromQuery] PaginationDTO pagination)

{

var queryable = \_context.Cities

.Where(x => x.State!.Id == pagination.Id)

.AsQueryable();

if (!string.IsNullOrWhiteSpace(pagination.Filter))

{

queryable = queryable.Where(x => x.Name.ToLower().Contains(pagination.Filter.ToLower()));

}

double count = await queryable.CountAsync();

double totalPages = Math.Ceiling(count / pagination.RecordsNumber);

return Ok(totalPages);

}

1. Modificamos el **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" style="display: flex; flex-wrap:wrap; align-items: center;">

<div>

<a class="btn btn-primary" href="/states/create/@country.Id">Nuevo Estado/Departamento</a>

<a class="btn btn-success" href="/countries">Regresar</a>

</div>

<div class="mx-2">

<input style="width: 400px;" type="text" class="form-control" id="titulo" placeholder="Buscar estado/departamento..." @bind-value="Filter" />

</div>

<div>

<button type="button" class="btn btn-outline-primary" @onclick="ApplyFilterAsync">Filtrar</button>

<button type="button" class="btn btn-outline-danger" @onclick="CleanFilterAsync">Limpiar</button>

</div>

</div>

<Pagination CurrentPage="currentPage"

TotalPages="totalPages"

SelectedPage="SelectedPage" />

<GenericList MyList="states!">

<Body>

<table class="table table-striped">

<thead>

<tr>

<th>Estado / Departamento</th>

<th style="width:140px">Ciudades</th>

<th style="width:260px"></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;

private int currentPage = 1;

private int totalPages;

[Parameter]

public int Id { get; set; }

[Parameter]

[SupplyParameterFromQuery]

public string Page { get; set; } = "";

[Parameter]

[SupplyParameterFromQuery]

public string Filter { get; set; } = "";

protected override async Task OnInitializedAsync()

{

await LoadAsync();

}

private async Task SelectedPage(int page)

{

currentPage = page;

await LoadAsync(page);

}

private async Task LoadAsync(int page = 1)

{

if (!string.IsNullOrWhiteSpace(Page))

{

page = Convert.ToInt32(Page);

}

string url1 = string.Empty;

string url2 = string.Empty;

if (string.IsNullOrEmpty(Filter))

{

url1 = $"api/states?id={Id}&page={page}";

url2 = $"api/states/totalPages?id={Id}";

}

else

{

url1 = $"api/states?id={Id}&page={page}&filter={Filter}";

url2 = $"api/states/totalPages?id={Id}&filter={Filter}";

}

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

var responseHppt2 = await repository.Get<List<State>>(url1);

var responseHppt3 = await repository.Get<int>(url2);

country = responseHppt.Response;

states = responseHppt2.Response;

totalPages = responseHppt3.Response;

}

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();

}

private async Task CleanFilterAsync()

{

Filter = string.Empty;

await ApplyFilterAsync();

}

private async Task ApplyFilterAsync()

{

int page = 1;

await LoadAsync(page);

await SelectedPage(page);

}

}

1. Modificamos el **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" style="display: flex; flex-wrap:wrap; align-items: center;">

<div>

<a class="btn btn-primary" href="/cities/create/@StateId">Nueva Ciudad</a>

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

</div>

<div class="mx-2">

<input style="width: 400px;" type="text" class="form-control" id="titulo" placeholder="Buscar ciudad..." @bind-value="Filter" />

</div>

<div>

<button type="button" class="btn btn-outline-primary" @onclick="ApplyFilterAsync">Filtrar</button>

<button type="button" class="btn btn-outline-danger" @onclick="CleanFilterAsync">Limpiar</button>

</div>

</div>

<Pagination CurrentPage="currentPage"

TotalPages="totalPages"

SelectedPage="SelectedPage" />

<GenericList MyList="cities!">

<Body>

<table class="table table-striped">

<thead>

<tr>

<th>Ciudad</th>

<th style="width:180px"></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;

private int currentPage = 1;

private int totalPages;

[Parameter]

public int StateId { get; set; }

[Parameter]

[SupplyParameterFromQuery]

public string Page { get; set; } = "";

[Parameter]

[SupplyParameterFromQuery]

public string Filter { get; set; } = "";

protected override async Task OnInitializedAsync()

{

await LoadAsync();

}

private async Task SelectedPage(int page)

{

currentPage = page;

await LoadAsync(page);

}

private async Task LoadAsync(int page = 1)

{

if (!string.IsNullOrWhiteSpace(Page))

{

page = Convert.ToInt32(Page);

}

string url1 = string.Empty;

string url2 = string.Empty;

if (string.IsNullOrEmpty(Filter))

{

url1 = $"api/cities?id={StateId}&page={page}";

url2 = $"api/cities/totalPages?id={StateId}";

}

else

{

url1 = $"api/cities?id={StateId}&page={page}&filter={Filter}";

url2 = $"api/cities/totalPages?id={StateId}&filter={Filter}";

}

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

var responseHppt2 = await repository.Get<List<City>>(url1);

var responseHppt3 = await repository.Get<int>(url2);

state = responseHppt.Response;

cities = responseHppt2.Response;

totalPages = responseHppt3.Response;

}

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();

}

private async Task CleanFilterAsync()

{

Filter = string.Empty;

await ApplyFilterAsync();

}

private async Task ApplyFilterAsync()

{

int page = 1;

await LoadAsync(page);

await SelectedPage(page);

}

}

## Creando las tablas de usuarios

1. Como vamos a tener dos tipos de usuarios; administradores y usuarios. Vamos a crear una enumeración para diferenciarlos. Creamos la carpeta **Enums** en el proyecto **Shared** y dentro de esta carpeta la enumeración **UserType**:

namespace StoresG8.Shared.Enums

{

public enum UserType

{

Admin,

User

}

}

1. En el proyecto **Shared** instalar el nuget **Microsoft.AspNetCore.Identity.EntityFrameworkCore** última versión (hoy es 7.0.5)
2. En el proyecto **Shared** en la carpeta **Entities**, crear la entidad **User**:

using Microsoft.AspNetCore.Identity;

using StoresG8.Shared.Enums;

using System.ComponentModel.DataAnnotations;

namespace StoresG8.Shared.Entities

{

public class User : IdentityUser

{

[Display(Name = "Documento")]

[MaxLength(20, ErrorMessage = "El campo {0} debe tener máximo {1} caractéres.")]

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

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

[Display(Name = "Nombres")]

[MaxLength(50, ErrorMessage = "El campo {0} debe tener máximo {1} caractéres.")]

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

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

[Display(Name = "Apellidos")]

[MaxLength(50, ErrorMessage = "El campo {0} debe tener máximo {1} caractéres.")]

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

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

[Display(Name = "Dirección")]

[MaxLength(200, ErrorMessage = "El campo {0} debe tener máximo {1} caractéres.")]

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

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

[Display(Name = "Foto")]

public string? Photo { get; set; }

[Display(Name = "Tipo de usuario")]

public UserType UserType { get; set; }

public City? City { get; set; }

[Display(Name = "Ciudad")]

[Range(1, int.MaxValue, ErrorMessage = "Debes seleccionar una {0}.")]

public int CityId { get; set; }

[Display(Name = "Usuario")]

public string FullName => $"{FirstName} {LastName}";

}

}

1. Modificamos la entidad **City** para definir la relación a ambos lados de esta:

public State? State { get; set; }

public ICollection<User>? Users { get; set; }

1. En el proyecto **API** instalar el nugget **Microsoft.AspNetCore.Identity.EntityFrameworkCore** la última versión(hoy es la 7.0.5)
2. Modificar el **DataContext**:

public class DataContext : IdentityDbContext<User>

1. Crear la interfaz **IUserHelper** en **API.Helpers**:

using Microsoft.AspNetCore.Identity;

using StoresG8.Shared.Entities;

namespace StoresG8.API.Helpers

{

public interface IUserHelper

{

Task<User> GetUserAsync(string email);

Task<IdentityResult> AddUserAsync(User user, string password);

Task CheckRoleAsync(string roleName);

Task AddUserToRoleAsync(User user, string roleName);

Task<bool> IsUserInRoleAsync(User user, string roleName);

}

}

1. Luego hacemos la implementación de dicha interfaz:

using Microsoft.AspNetCore.Identity;

using Microsoft.EntityFrameworkCore;

using StoresG8.API.Data;

using StoresG8.Shared.Entities;

namespace StoresG8.API.Helpers

{

public class UserHelper : IUserHelper

{

private readonly DataContext \_context;

private readonly UserManager<User> \_userManager;

private readonly RoleManager<IdentityRole> \_roleManager;

public UserHelper(DataContext context, UserManager<User> userManager, RoleManager<IdentityRole> roleManager)

{

\_context = context;

\_userManager = userManager;

\_roleManager = roleManager;

}

public async Task<IdentityResult> AddUserAsync(User user, string password)

{

return await \_userManager.CreateAsync(user, password);

}

public async Task AddUserToRoleAsync(User user, string roleName)

{

await \_userManager.AddToRoleAsync(user, roleName);

}

public async Task CheckRoleAsync(string roleName)

{

bool roleExists = await \_roleManager.RoleExistsAsync(roleName);

if (!roleExists)

{

await \_roleManager.CreateAsync(new IdentityRole

{

Name = roleName

});

}

}

public async Task<User> GetUserAsync(string email)

{

return await \_context.Users

.Include(u => u.City)

.ThenInclude(c => c.State)

.ThenInclude(s => s.Country)

.FirstOrDefaultAsync(x => x.Email == email);

}

public async Task<bool> IsUserInRoleAsync(User user, string roleName)

{

return await \_userManager.IsInRoleAsync(user, roleName);

}

}

}

1. Modificamos el **Program** del proyecto **API**:

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

builder.Services.AddIdentity<User, IdentityRole>(x =>

{

x.User.RequireUniqueEmail = true;

x.Password.RequireDigit = false;

x.Password.RequiredUniqueChars = 0;

x.Password.RequireLowercase = false;

x.Password.RequireNonAlphanumeric = false;

x.Password.RequireUppercase = false;

})

.AddEntityFrameworkStores<DataContext>()

.AddDefaultTokenProviders();

builder.Services.AddScoped<IUserHelper, UserHelper>();

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();

}

}

if (app.Environment.IsDevelopment())

{

app.UseSwagger();

app.UseSwaggerUI();

}

app.UseHttpsRedirection();

app.UseAuthentication();

app.UseAuthorization();

1. Modificamos el **SeedDb**:

public class SeedDb

{

private readonly DataContext \_context;

private readonly IApiService \_apiService;

private readonly IUserHelper \_userHelper;

public SeedDb(DataContext context, IApiService apiService, IUserHelper userHelper)

{

\_context = context;

\_apiService = apiService;

\_userHelper = userHelper;

}

public async Task SeedAsync()

{

await \_context.Database.EnsureCreatedAsync();

await CheckCountriesAsync();

await CheckRolesAsync();

await CheckUserAsync("1", "OAP", "OAP", "oap@yopmail.com", "300445555", "CR 78 9687", UserType.Admin);

}

private async Task<User> CheckUserAsync(string document, string firstName, string lastName, string email, string phone, string address, UserType userType)

{

var user = await \_userHelper.GetUserAsync(email);

if (user == null)

{

user = new User

{

FirstName = firstName,

LastName = lastName,

Email = email,

UserName = email,

PhoneNumber = phone,

Address = address,

Document = document,

City = \_context.Cities.FirstOrDefault(),

UserType = userType,

};

await \_userHelper.AddUserAsync(user, "123456");

await \_userHelper.AddUserToRoleAsync(user, userType.ToString());

}

return user;

}

private async Task CheckRolesAsync()

{

await \_userHelper.CheckRoleAsync(UserType.Admin.ToString());

await \_userHelper.CheckRoleAsync(UserType.User.ToString());

}

1. Corremos los siguientes comandos:

PM> drop-database

PM> add-migration Users

PM> update-database

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

## Creando sistema de seguridad

1. Al proyecto **WEB** agregamos el paquete: **Microsoft.AspNetCore.Components.WebAssembly.Authentication** ver **7.0.5**.
2. Agregamos este using en el **\_Imports**:

@using Microsoft.AspNetCore.Components.Authorization

1. En el proyecto **WEB** creamos la carpeta **Auth** y dentro de esta la clase **AuthenticationProviderTest**:

using Microsoft.AspNetCore.Components.Authorization;

using System.Security.Claims;

namespace StoresG8WEB.Auth

{

public class AuthenticationProviderTest : AuthenticationStateProvider

{

public override async Task<AuthenticationState> GetAuthenticationStateAsync()

{

var anonimous = new ClaimsIdentity();

return await Task.FromResult(new AuthenticationState(new ClaimsPrincipal(anonimous)));

}

}

}

1. Modificamos el **Program** del proyecto **WEB**:

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

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

builder.Services.AddSweetAlert2();

builder.Services.AddAuthorizationCore();

builder.Services.AddScoped<AuthenticationStateProvider, AuthenticationProviderTest>();

1. Modificamos el **App.razor**:

@using Microsoft.AspNetCore.Components.Authorization

<Router AppAssembly="@typeof(App).Assembly">

<Found Context="routeData">

<AuthorizeRouteView RouteData="@routeData" DefaultLayout="@typeof(MainLayout)" />

<FocusOnNavigate RouteData="@routeData" Selector="h1" />

</Found>

<NotFound>

<CascadingAuthenticationState>

<PageTitle>No encontrado</PageTitle>

<LayoutView Layout="@typeof(MainLayout)">

<p role="alert">Lo sentimos no hay nada en esta ruta.</p>

</LayoutView>

</CascadingAuthenticationState>

</NotFound>

</Router>

1. Probamos y vemos que aparentemente no pasa nada, ahora a nuestro **AuthenticationProviderTest** le vamos a colocar un tiempo de espera:

public override async Task<AuthenticationState> GetAuthenticationStateAsync()

{

await Task.Delay(3000);

var anonimous = new ClaimsIdentity();

return await Task.FromResult(new AuthenticationState(new ClaimsPrincipal(anonimous)));

}

1. Probamos de nuevo y vemos que tarda los 3 segundos haciendo la autorización.
2. Si queremos cambiar el mensaje, modificamos el **App.razor**:

<AuthorizeRouteView RouteData="@routeData" DefaultLayout="@typeof(MainLayout)">

<Authorizing>

<p>Autorizando...</p>

</Authorizing>

</AuthorizeRouteView>

1. Probamos de nuevo.
2. Modificamos el **Index.razor**.

@page "/"

@using Microsoft.AspNetCore.Components.Authorization

<PageTitle>Index</PageTitle>

<AuthorizeView>

<p>Estas autenticado</p>

</AuthorizeView>

<h1>Hello, world!</h1>

Welcome to your new app.

<SurveyPrompt Title="How is Blazor working for you?" />

1. Modificamos el **AuthenticationProviderTest**:

public override async Task<AuthenticationState> GetAuthenticationStateAsync()

{

var anonimous = new ClaimsIdentity();

var oapUser = new ClaimsIdentity(authenticationType: "test");

return await Task.FromResult(new AuthenticationState(new ClaimsPrincipal(oapUser)));

}

1. Cambiamos el **Index.razor**.

<AuthorizeView>

<Authorized>

<p>Estas autenticado</p>

</Authorized>

<NotAuthorized>

<p>No estas autorizado</p>

</NotAuthorized>

</AuthorizeView>

1. Y jugamos con el **AuthenticationProviderTest** para ver que pasa con el usuario **anonimous** y con el usuario **oapUser**.
2. Modificamos nuestro **AuthenticationProviderTest**, para agregar algunos **Claims**:

public override async Task<AuthenticationState> GetAuthenticationStateAsync()

{

var anonimous = new ClaimsIdentity();

var oapUser = new ClaimsIdentity(new List<Claim>

{

new Claim("FirstName", "Luis"),

new Claim("LastName", "O"),

new Claim(ClaimTypes.Name, "oap@yopmail.com")

},

authenticationType: "test");

return await Task.FromResult(new AuthenticationState(new ClaimsPrincipal(oapUser)));

}

1. Modificamos el **Index.razor** y probamos:

<AuthorizeView>

<Authorized>

<p>Estas autenticado, @context.User.Identity?.Name</p>

</Authorized>

<NotAuthorized>

<p>No estas autorizado</p>

</NotAuthorized>

</AuthorizeView>

1. Modificamos de nuevo el **Index.razor** para crear un **Role** y probamos:

<AuthorizeView Roles="Admin">

<Authorized>

<p>Estas autenticado y autorizado, @context.User.Identity?.Name</p>

</Authorized>

<NotAuthorized>

<p>No estas autorizado</p>

</NotAuthorized>

</AuthorizeView>

1. Modificamos nuestro **AuthenticationProviderTest**, para agregar el **Claim** de **Role** y probamos:

var oapUser = new ClaimsIdentity(new List<Claim>

{

new Claim("FirstName", "Juan"),

new Claim("LastName", "Oap"),

new Claim(ClaimTypes.Name, "oap@yopmail.com"),

new Claim(ClaimTypes.Role, "Admin")

},

authenticationType: "test");

1. Ahora cambiamos nuestro **NavMenu** para mostrar la opción de países solo a los administradores, y jugamos con nuestro **AuthenticationProviderTest** para cambiarle el rol al usuario:

@using Microsoft.AspNetCore.Components.Authorization

<div class="@NavMenuCssClass nav-scrollable" @onclick="ToggleNavMenu">

<nav class="flex-column">

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

<NavLink class="nav-link" href="" Match="NavLinkMatch.All">

<span class="oi oi-home" aria-hidden="true"></span> Home

</NavLink>

</div>

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

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

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

</NavLink>

</div>

<AuthorizeView Roles="Admin">

<Authorized>

<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>

</Authorized>

</AuthorizeView>

</nav>

</div>

1. Pero nótese que solo estamos ocultando la opción, si el usuario por la URL introduce la dirección de países, pues podrá acceder a nuestras páginas, lo cual es algo que no queremos.
2. Para evitar esto le colocamos este atributo a todos los componentes a los que navegamos y queremos proteger Countries:

@using Microsoft.AspNetCore.Authorization;

@attribute [Authorize(Roles = "Admin")]

1. Ahora si queremos personalizar el mensaje de autorización, podemos modificar todo nuestro **App.razor**:

@using Microsoft.AspNetCore.Components.Authorization

<Router AppAssembly="@typeof(App).Assembly">

<Found Context="routeData">

<AuthorizeRouteView RouteData="@routeData" DefaultLayout="@typeof(MainLayout)">

<Authorizing>

<p>Autorizando...</p>

</Authorizing>

<NotAuthorized>

<p>No estas autorizado para ver este contenido...</p>

</NotAuthorized>

</AuthorizeRouteView>

<FocusOnNavigate RouteData="@routeData" Selector="h1" />

</Found>

<NotFound>

<CascadingAuthenticationState>

<PageTitle>No encontrado</PageTitle>

<LayoutView Layout="@typeof(MainLayout)">

<p role="alert">Lo sentimos no hay nada en esta ruta.</p>

</LayoutView>

</CascadingAuthenticationState>

</NotFound>

</Router>

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

## Seguridad desde el backend

1. Antes de empezar corrijamos el Warnig del **GetUserAsync** en el **UserHelper**,

public async Task<User> GetUserAsync(string email)

{

var user = await \_context.Users

.Include(u => u.City!)

.ThenInclude(c => c.State!)

.ThenInclude(s => s.Country!)

.FirstOrDefaultAsync(u => u.Email! == email);

return user!;

}

1. Agregamos el paquete **Microsoft.AspNetCore.Authentication.JwtBearer** al proyecto **API**, versión 7.0.5.
2. Creamos el parámetro **jwtKey** en el appsettings del proyecto **API** (cualquier cosa, entre más larga mejor):

"AllowedHosts": "\*",

"jwtKey": "sagdsadgfeSDF674545REFG$%FEfgdslkjfglkjhfgdkljhdR5454545\_4TGRGtyo!!kjytkljty"

}

1. Modificamos el **Program** del proyecto **API**:

builder.Services.AddScoped<IUserHelper, UserHelper>();

builder.Services.AddAuthentication(JwtBearerDefaults.AuthenticationScheme)

.AddJwtBearer(x => x.TokenValidationParameters = new TokenValidationParameters

{

ValidateIssuer = false,

ValidateAudience = false,

ValidateLifetime = true,

ValidateIssuerSigningKey = true,

IssuerSigningKey = new SymmetricSecurityKey(Encoding.UTF8.GetBytes(builder.Configuration["jwtKey"]!)),

ClockSkew = TimeSpan.Zero

});

var app = builder.Build();

1. En el proyecto **Shared** en la carpeta **DTOs** creamos el **UserDTO**:

using StoresG8.Shared.Entities;

using System.ComponentModel.DataAnnotations;

using System.Xml.Linq;

namespace StoresG8.Shared.DTOs

{

public class UserDTO : User

{

[DataType(DataType.Password)]

[Display(Name = "Contraseña")]

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

[StringLength(20, MinimumLength = 6, ErrorMessage = "El campo {0} debe tener entre {2} y {1} carácteres.")]

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

[Compare("Password", ErrorMessage = "La contraseña y la confirmación no son iguales.")]

[Display(Name = "Confirmación de contraseña")]

[DataType(DataType.Password)]

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

[StringLength(20, MinimumLength = 6, ErrorMessage = "El campo {0} debe tener entre {2} y {1} carácteres.")]

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

}

}

1. En el proyecto **Shared** en la carpeta **DTOs** creamos el **TokenDTO**:

using StoresG8.Shared.Entities;

namespace StoresG8.Shared.DTOs

{

public class TokenDTO

{

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

public DateTime Expiration { get; set; }

}

}

1. En el proyecto **Shared** en la carpeta **DTOs** creamos el **LoginDTO**:

using System.ComponentModel.DataAnnotations;

namespace StoresG8.Shared.DTOs

{

public class LoginDTO

{

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

[EmailAddress(ErrorMessage = "Debes ingresar un correo válido.")]

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

[Display(Name = "Contraseña")]

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

[MinLength(6, ErrorMessage = "El campo {0} debe tener al menos {1} carácteres.")]

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

}

}

1. Agregamos estos métodos al **IUserHelper**:

Task<SignInResult> LoginAsync(LoginDTO model);

Task LogoutAsync();

1. Los implementamos en el **UserHelper**:

private readonly DataContext \_context;

private readonly UserManager<User> \_userManager;

private readonly RoleManager<IdentityRole> \_roleManager;

private readonly SignInManager<User> \_signInManager;

public UserHelper(DataContext context, UserManager<User> userManager, RoleManager<IdentityRole> roleManager, SignInManager<User> signInManager)

{

\_context = context;

\_userManager = userManager;

\_roleManager = roleManager;

\_signInManager = signInManager;

}

…

public async Task<SignInResult> LoginAsync(LoginDTO model)

{

return await \_signInManager.PasswordSignInAsync(model.Email, model.Password, false, false);

}

public async Task LogoutAsync()

{

await \_signInManager.SignOutAsync();

}

1. Creamos el **AccountsController**:

using Microsoft.AspNetCore.Mvc;

using Microsoft.IdentityModel.Tokens;

using StoresG8.API.Helpers;

using StoresG8.Shared.DTOs;

using StoresG8.Shared.Entities;

using System.IdentityModel.Tokens.Jwt;

using System.Security.Claims;

using System.Text;

namespace StoresG8.API.Controllers

{

[ApiController]

[Route("/api/accounts")]

public class AccountsController : ControllerBase

{

private readonly IUserHelper \_userHelper;

private readonly IConfiguration \_configuration;

public AccountsController(IUserHelper userHelper, IConfiguration configuration)

{

\_userHelper = userHelper;

\_configuration = configuration;

}

[HttpPost("CreateUser")]

public async Task<ActionResult> CreateUser([FromBody] UserDTO model)

{

User user = model;

var result = await \_userHelper.AddUserAsync(user, model.Password);

if (result.Succeeded)

{

await \_userHelper.AddUserToRoleAsync(user, user.UserType.ToString());

return Ok(BuildToken(user));

}

return BadRequest(result.Errors.FirstOrDefault());

}

[HttpPost("Login")]

public async Task<ActionResult> Login([FromBody] LoginDTO model)

{

var result = await \_userHelper.LoginAsync(model);

if (result.Succeeded)

{

var user = await \_userHelper.GetUserAsync(model.Email);

return Ok(BuildToken(user));

}

return BadRequest("Email o contraseña incorrectos.");

}

private TokenDTO BuildToken(User user)

{

var claims = new List<Claim>

{

new Claim(ClaimTypes.Name, user.Email!),

new Claim(ClaimTypes.Role, user.UserType.ToString()),

new Claim("Document", user.Document),

new Claim("FirstName", user.FirstName),

new Claim("LastName", user.LastName),

new Claim("Address", user.Address),

new Claim("Photo", user.Photo ?? string.Empty),

new Claim("CityId", user.CityId.ToString())

};

var key = new SymmetricSecurityKey(Encoding.UTF8.GetBytes(\_configuration["jwtKey"]!));

var credentials = new SigningCredentials(key, SecurityAlgorithms.HmacSha256);

var expiration = DateTime.UtcNow.AddDays(30);

var token = new JwtSecurityToken(

issuer: null,

audience: null,

claims: claims,

expires: expiration,

signingCredentials: credentials);

return new TokenDTO

{

Token = new JwtSecurityTokenHandler().WriteToken(token),

Expiration = expiration

};

}

}

}

1. Luego le colocamos autorización a los 3 controladores **CountriesController**, **StatesController** y **CitiesController**:

[Authorize(AuthenticationSchemes = JwtBearerDefaults.AuthenticationScheme)]

1. Modificamos el **CountriesIndex**:

try

{

var responseHppt = await repository.Get<List<Country>>(url1);

var responseHppt2 = await repository.Get<int>(url2);

Countries = responseHppt.Response!;

totalPages = responseHppt2.Response!;

}

catch (Exception ex)

{

await sweetAlertService.FireAsync("Error", ex.Message, SweetAlertIcon.Error);

}

1. Podemos probar por **POSTMAN** como está funcionando nuestro token, y con <https://jwt.io/> probamos como está quedando nuestro token.
2. Probamos en la interfaz web, y nos debe salir un error porque aun no le mandamos ningún token a nuestra API. Hacemos el **commit**.

## Implementando el registro de usuarios, login & logout

1. En el proyecto **WEB** Instalamos el paquete: **System.IdentityModel.Tokens.Jwt**.
2. En el proyecto **WEB** en la carpeta **Helpers** creamos el **IJSRuntimeExtensionMethods**:

using Microsoft.JSInterop;

namespace StoresG8.WEB.Helpers

{

public static class IJSRuntimeExtensionMethods

{

public static ValueTask<object> SetLocalStorage(this IJSRuntime js, string key, string content)

{

return js.InvokeAsync<object>("localStorage.setItem", key, content);

}

public static ValueTask<object> GetLocalStorage(this IJSRuntime js, string key)

{

return js.InvokeAsync<object>("localStorage.getItem", key);

}

public static ValueTask<object> RemoveLocalStorage(this IJSRuntime js, string key)

{

return js.InvokeAsync<object>("localStorage.removeItem", key);

}

}

}

1. En el proyecto **WEB** en la carpeta **Auth** creamos la interface **ILoginService**:

namespace StoresG8.WEB.Auth

{

public interface ILoginService

{

Task LoginAsync(string token);

Task LogoutAsync();

}

}

1. En el proyecto **WEB** en la carpeta **Auth** creamos el **AuthenticationProviderJWT**:

using Microsoft.AspNetCore.Components.Authorization;

using Microsoft.JSInterop;

using StoresG8.WEB.Helpers;

using System.IdentityModel.Tokens.Jwt;

using System.Net.Http.Headers;

using System.Security.Claims;

namespace StoresG8.WEB.Auth

{

public class AuthenticationProviderJWT : AuthenticationStateProvider, ILoginService

{

private readonly IJSRuntime \_jSRuntime;

private readonly HttpClient \_httpClient;

private readonly String \_tokenKey;

private readonly AuthenticationState \_anonimous;

public AuthenticationProviderJWT(IJSRuntime jSRuntime, HttpClient httpClient)

{

\_jSRuntime = jSRuntime;

\_httpClient = httpClient;

\_tokenKey = "TOKEN\_KEY";

\_anonimous = new AuthenticationState(new ClaimsPrincipal(new ClaimsIdentity()));

}

public async override Task<AuthenticationState> GetAuthenticationStateAsync()

{

var token = await \_jSRuntime.GetLocalStorage(\_tokenKey);

if (token is null)

{

return \_anonimous;

}

return BuildAuthenticationState(token.ToString()!);

}

private AuthenticationState BuildAuthenticationState(string token)

{

\_httpClient.DefaultRequestHeaders.Authorization = new AuthenticationHeaderValue("bearer", token);

var claims = ParseClaimsFromJWT(token);

return new AuthenticationState(new ClaimsPrincipal(new ClaimsIdentity(claims, "jwt")));

}

private IEnumerable<Claim> ParseClaimsFromJWT(string token)

{

var jwtSecurityTokenHandler = new JwtSecurityTokenHandler();

var unserializedToken = jwtSecurityTokenHandler.ReadJwtToken(token);

return unserializedToken.Claims;

}

public async Task LoginAsync(string token)

{

await \_jSRuntime.SetLocalStorage(\_tokenKey, token);

var authState = BuildAuthenticationState(token);

NotifyAuthenticationStateChanged(Task.FromResult(authState));

}

public async Task LogoutAsync()

{

await \_jSRuntime.RemoveLocalStorage(\_tokenKey);

\_httpClient.DefaultRequestHeaders.Authorization = null;

NotifyAuthenticationStateChanged(Task.FromResult(\_anonimous));

}

}

}

1. Modificamos el **Program** del **WEB** para usar nuestro nuevo proveedor de autenticación:

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

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

builder.Services.AddSweetAlert2();

builder.Services.AddAuthorizationCore();

builder.Services.AddScoped<AuthenticationProviderJWT>();

builder.Services.AddScoped<AuthenticationStateProvider, AuthenticationProviderJWT>(x => x.GetRequiredService<AuthenticationProviderJWT>());

builder.Services.AddScoped<ILoginService, AuthenticationProviderJWT>(x => x.GetRequiredService<AuthenticationProviderJWT>());

1. Creamos un componente razor en la carpeta Shared del proyecto WEB, llamado **AuthLinks**:

<AuthorizeView>

<Authorized>

<span>Hola, @context.User.Identity!.Name</span>

<a href="Logout" class="nav-link btn btn-link">Cerrar Sesión</a>

</Authorized>

<NotAuthorized>

<a href="Register" class="nav-link btn btn-link">Registro</a>

<a href="Login" class="nav-link btn btn-link">Iniciar Sesión</a>

</NotAuthorized>

</AuthorizeView>

1. Llamamos el nuevo componente desde el **MainLayout**:.

@inherits LayoutComponentBase

<div class="page">

<div class="sidebar">

<NavMenu />

</div>

<main>

<div class="top-row px-4">

<AuthLinks/>

<a href="https://docs.microsoft.com/aspnet/" target="\_blank">Acerca de</a>

</div>

<article class="content px-4">

@Body

</article>

</main>

</div>

1. Probamos lo que llevamos.
2. Dentro de **Pages** creamos la carpeta **Auth** y dentro de esta el componente **Register**:

@page "/Register"

@using StoresG8.Shared.DTOs;

@using StoresG8.Shared.Enums;

@using StoresG8.WEB.Auth;

@inject IRepository repository

@inject SweetAlertService sweetAlertService

@inject NavigationManager navigationManager

@inject ILoginService loginService

<h3>Registrar Nuevo Usuario</h3>

<EditForm Model="userDTO" OnValidSubmit="CreteUserAsync">

<DataAnnotationsValidator/>

<div class="row">

<div class="col-6">

<div class="mb-3">

<label>Nombres:</label>

<div>

<InputText class="form-control" @bind-Value="@userDTO.FirstName" />

<ValidationMessage For="@(() => userDTO.FirstName)" />

</div>

</div>

<div class="mb-3">

<label>Apellidos:</label>

<div>

<InputText class="form-control" @bind-Value="@userDTO.LastName" />

<ValidationMessage For="@(() => userDTO.LastName)" />

</div>

</div>

<div class="mb-3">

<label>Documento:</label>

<div>

<InputText class="form-control" @bind-Value="@userDTO.Document" />

<ValidationMessage For="@(() => userDTO.Document)" />

</div>

</div>

<div class="mb-3">

<label>Teléfono:</label>

<div>

<InputText class="form-control" @bind-Value="@userDTO.PhoneNumber" />

<ValidationMessage For="@(() => userDTO.PhoneNumber)" />

</div>

</div>

<div class="mb-3">

<label>Dirección:</label>

<div>

<InputText class="form-control" @bind-Value="@userDTO.Address" />

<ValidationMessage For="@(() => userDTO.Address)" />

</div>

</div>

<div class="mb-3">

<label>Email:</label>

<div>

<InputText class="form-control" @bind-Value="@userDTO.Email" />

<ValidationMessage For="@(() => userDTO.Email)" />

</div>

</div>

</div>

<div class="col-6">

<div class="mb-3">

<label>Ciudad:</label>

<div>

<InputNumber class="form-control" @bind-Value="@userDTO.CityId" />

<ValidationMessage For="@(() => userDTO.CityId)" />

</div>

</div>

<div class="mb-3">

<label>Foto:</label>

<div>

<InputText class="form-control" @bind-Value="@userDTO.Photo" />

<ValidationMessage For="@(() => userDTO.Photo)" />

</div>

</div>

<div class="mb-3">

<label>Contraseña:</label>

<div>

<InputText type="password" class="form-control" @bind-Value="@userDTO.Password" />

<ValidationMessage For="@(() => userDTO.Password)" />

</div>

</div>

<div class="mb-3">

<label>Confirmación de contraseña:</label>

<div>

<InputText type="password" class="form-control" @bind-Value="@userDTO.PasswordConfirm" />

<ValidationMessage For="@(() => userDTO.PasswordConfirm)" />

</div>

</div>

</div>

</div>

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

</EditForm>

@code {

private UserDTO userDTO = new();

private async Task CreteUserAsync()

{

userDTO.UserName = userDTO.Email;

userDTO.UserType = UserType.User;

var responseHttp = await repository.Post<UserDTO, TokenDTO>("/api/accounts/CreateUser", userDTO);

if (responseHttp.Error)

{

var message = await responseHttp.GetErrorMessageAsync();

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

return;

}

await loginService.LoginAsync(responseHttp.Response!.Token);

navigationManager.NavigateTo("/");

}

}

1. Dentro de **Pages** en la carpeta **Auth** creamos el componente **Login**:

@page "/Login"

@using StoresG8.Shared.DTOs;

@using StoresG8.WEB.Auth;

@inject IRepository repository

@inject SweetAlertService sweetAlertService

@inject NavigationManager navigationManager

@inject ILoginService loginService

<h3>Iniciar Sesión</h3>

<EditForm Model="loginDTO" OnValidSubmit="LoginAsync">

<DataAnnotationsValidator />

<div class="row">

<div class="col-4">

<div class="mb-3">

<label>Email:</label>

<div>

<InputText class="form-control" @bind-Value="@loginDTO.Email" />

<ValidationMessage For="@(() => loginDTO.Email)" />

</div>

</div>

<div class="mb-3">

<label>Contraseña:</label>

<div>

<InputText type="password" class="form-control" @bind-Value="@loginDTO.Password" />

<ValidationMessage For="@(() => loginDTO.Password)" />

</div>

</div>

<button class="btn btn-primary" type="submit">Iniciar Sesión</button>

</div>

</div>

</EditForm>

@code {

private LoginDTO loginDTO = new();

private async Task LoginAsync()

{

var responseHttp = await repository.Post<LoginDTO, TokenDTO>("/api/accounts/Login", loginDTO);

if (responseHttp.Error)

{

var message = await responseHttp.GetErrorMessageAsync();

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

return;

}

await loginService.LoginAsync(responseHttp.Response!.Token);

navigationManager.NavigateTo("/");

}

}

1. Probemos lo que llevamos.
2. Dentro de **Pages** en la carpeta **Auth** creamos el componente **Logout**:

@page "/logout"

@using StoresG8.WEB.Auth;

@inject ILoginService loginService

@inject NavigationManager navigationManager

<p>Cerrando sesión...</p>

@code {

protected override async Task OnInitializedAsync()

{

await loginService.LogoutAsync();

navigationManager.NavigateTo("/");

}

}

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

## Habilitando tokens en swagger

1. Modificamos el **Program** del **API**:

builder.Services.AddSwaggerGen();

builder.Services.AddSwaggerGen(c =>

{

c.SwaggerDoc("v1", new OpenApiInfo { Title = "StoresG8 API", Version = "v1" });

c.AddSecurityDefinition("Bearer", new OpenApiSecurityScheme

{

Description = @"JWT Authorization header using the Bearer scheme. <br /> <br />

Enter 'Bearer' [space] and then your token in the text input below.<br /> <br />

Example: 'Bearer 12345abcdef'<br /> <br />",

Name = "Authorization",

In = ParameterLocation.Header,

Type = SecuritySchemeType.ApiKey,

Scheme = "Bearer"

});

c.AddSecurityRequirement(new OpenApiSecurityRequirement()

{

{

new OpenApiSecurityScheme

{

Reference = new OpenApiReference

{

Type = ReferenceType.SecurityScheme,

Id = "Bearer"

},

Scheme = "oauth2",

Name = "Bearer",

In = ParameterLocation.Header,

},

new List<string>()

}

});

});

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

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

## Mejorando el registro de usuarios con drop-down-lists en cascada

1. Creamos el método **GetCombo** en el **CountriesController**:

[AllowAnonymous]

[HttpGet("combo")]

public async Task<ActionResult> GetCombo()

{

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

}

1. Creamos el método **GetCombo** en el **StatesController**:

[AllowAnonymous]

[HttpGet("combo/{countryId:int}")]

public async Task<ActionResult> GetCombo(int countryId)

{

return Ok(await \_context.States

.Where(x => x.CountryId == countryId)

.ToListAsync());

}

1. Creamos el método **GetCombo** en el **CitiesController**:

[AllowAnonymous]

[HttpGet("combo/{stateId:int}")]

public async Task<ActionResult> GetCombo(int stateId)

{

return Ok(await \_context.Cities

.Where(x => x.StateId == stateId)

.ToListAsync());

}

1. Modificamos el **Register.razor**:

…

<div class="col-6">

Eliminamos todo el <Div> de Ciudad, y copiamos el siguiente código:

<div class="mb-3">

<label>País:</label>

<div>

<select class="form-select" @onchange="CountryChangedAsync">

<option value="0">-- Seleccione un país --</option>

@if (countries is not null)

{

@foreach (var country in countries)

{

<option value="@country.Id">@country.Name</option>

}

}

</select>

</div>

</div>

<div class="mb-3">

<label>Estado/Departamento:</label>

<div>

<select class="form-select" @onchange="StateChangedAsync">

<option value="0">-- Seleccione un estado/departamento --</option>

@if (states is not null)

{

@foreach (var state in states)

{

<option value="@state.Id">@state.Name</option>

}

}

</select>

</div>

</div>

<div class="mb-3">

<label>Ciudad:</label>

<div>

<select class="form-select" @bind="userDTO.CityId">

<option value="0">-- Seleccione una ciudad --</option>

@if (cities is not null)

{

@foreach (var city in cities)

{

<option value="@city.Id">@city.Name</option>

}

}

</select>

<ValidationMessage For="@(() => userDTO.CityId)" />

</div>

</div>

<div class="mb-3">

<label>Foto:</label>

…

@code {

private UserDTO userDTO = new();

private List<Country>? countries;

private List<State>? states;

private List<City>? cities;

protected override async Task OnInitializedAsync()

{

await LoadCountriesAsync();

}

private async Task CountryChangedAsync(ChangeEventArgs e)

{

var selectedCountry = Convert.ToInt32(e.Value!);

await LoadStatesAsyn(selectedCountry);

}

private async Task StateChangedAsync(ChangeEventArgs e)

{

var selectedState = Convert.ToInt32(e.Value!);

await LoadCitiesAsyn(selectedState);

}

private async Task LoadCountriesAsync()

{

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

if (responseHttp.Error)

{

var message = await responseHttp.GetErrorMessageAsync();

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

return;

}

countries = responseHttp.Response;

}

private async Task LoadStatesAsyn(int countryId)

{

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

if (responseHttp.Error)

{

var message = await responseHttp.GetErrorMessageAsync();

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

return;

}

states = responseHttp.Response;

}

private async Task LoadCitiesAsyn(int stateId)

{

var responseHttp = await repository.Get<List<City>>($"/api/cities/combo/{stateId}");

if (responseHttp.Error)

{

var message = await responseHttp.GetErrorMessageAsync();

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

return;

}

cities = responseHttp.Response;

}

private async Task CreteUserAsync()

…

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

## Mejorando un poco la interfaz de usuario

1. Primero vamos a agregar estas líneas al final de nuestro **app.css**:

.spinner {

border: 16px solid silver;

border-top: 16px solid #337AB7;

border-radius: 50%;

width: 80px;

height: 80px;

animation: spin 700ms linear infinite;

top: 40%;

left: 55%;

position: absolute;

}

@keyframes spin {

0% {

transform: rotate(0deg)

}

100% {

transform: rotate(360deg)

}

}

1. Luego modificamos nuestro **CountriesIndex**:

…

@if (Countries is null)

{

<div class="spinner"/>

}

else

{

<GenericList MyList="Countries">

<div class="card">

<div class="card-header">

<span>

<i class="oi oi-globe"></i> Países

<a class="btn btn-sm btn-primary float-end" href="/countries/create"><i class="oi oi-plus"></i> Adicionar País</a>

</span>

</div>

<div class="card-body">

<div class="mb-2" style="display: flex; flex-wrap:wrap; align-items: center;">

<div>

<input style="width: 400px;" type="text" class="form-control" id="titulo" placeholder="Buscar país..." @bind-value="Filter" />

</div>

<div class="mx-1">

<button type="button" class="btn btn-outline-primary" @onclick="ApplyFilterAsync"><i class="oi oi-layers" /> Filtrar</button>

<button type="button" class="btn btn-outline-danger" @onclick="CleanFilterAsync"><i class="oi oi-ban" /> Limpiar</button>

</div>

</div>

<Pagination CurrentPage="currentPage"

TotalPages="totalPages"

SelectedPage="SelectedPage" />

<table class="table table-striped">

<thead>

<tr>

<th>País</th>

<th style="width:220px">Departamentos / Estados</th>

<th style="width:310px"></th>

</tr>

</thead>

<tbody>

@foreach (var country in Countries!)

{

<tr>

<td>

@country.Name

</td>

<td>

@country.StatesNumber

</td>

<td>

<a href="/countries/details/@country.Id" class="btn btn-info btn-sm"><i class="oi oi-list" /> Detalles</a>

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

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

</td>

</tr>

}

</tbody>

</table>

</div>

</div>

</GenericList>

}

…

1. Replica el cambio para el resto de la solución. Si quieres una lista de íconos que puedes usar te dejo este link: <https://kordamp.org/ikonli/cheat-sheet-openiconic.html>
2. Este es un ejemplo de como puede quedar la página de **Register**:

<EditForm Model="userDTO" OnValidSubmit="CreteUserAsync">

<DataAnnotationsValidator />

<div class="card">

<div class="card-header">

<span>

<i class="oi oi-person" /> Registrar Nuevo Usuario

<button class="btn btn-sm btn-primary float-end" type="submit"><i class="oi oi-check" /> Registrar</button>

</span>

</div>

<div class="card-body">

<div class="row">

<div class="col-6">

<div class="mb-3">

<label>Nombres:</label>

<div>

<InputText class="form-control" @bind-Value="@userDTO.FirstName" />

<ValidationMessage For="@(() => userDTO.FirstName)" />

</div>

</div>

<div class="mb-3">

<label>Apellidos:</label>

<div>

<InputText class="form-control" @bind-Value="@userDTO.LastName" />

<ValidationMessage For="@(() => userDTO.LastName)" />

</div>

</div>

<div class="mb-3">

<label>Documento:</label>

<div>

<InputText class="form-control" @bind-Value="@userDTO.Document" />

<ValidationMessage For="@(() => userDTO.Document)" />

</div>

</div>

<div class="mb-3">

<label>Teléfono:</label>

<div>

<InputText class="form-control" @bind-Value="@userDTO.PhoneNumber" />

<ValidationMessage For="@(() => userDTO.PhoneNumber)" />

</div>

</div>

<div class="mb-3">

<label>Dirección:</label>

<div>

<InputText class="form-control" @bind-Value="@userDTO.Address" />

<ValidationMessage For="@(() => userDTO.Address)" />

</div>

</div>

<div class="mb-3">

<label>Email:</label>

<div>

<InputText class="form-control" @bind-Value="@userDTO.Email" />

<ValidationMessage For="@(() => userDTO.Email)" />

</div>

</div>

</div>

<div class="col-6">

<div class="mb-3">

<label>País:</label>

<div>

<select class="form-select" @onchange="CountryChangedAsync">

<option value="0">-- Seleccione un país --</option>

@if (countries is not null)

{

@foreach (var country in countries)

{

<option value="@country.Id">@country.Name</option>

}

}

</select>

</div>

</div>

<div class="mb-3">

<label>Estado/Departamento:</label>

<div>

<select class="form-select" @onchange="StateChangedAsync">

<option value="0">-- Seleccione un estado/departamento --</option>

@if (states is not null)

{

@foreach (var state in states)

{

<option value="@state.Id">@state.Name</option>

}

}

</select>

</div>

</div>

<div class="mb-3">

<label>Ciudad:</label>

<div>

<select class="form-select" @bind="userDTO.CityId">

<option value="0">-- Seleccione una ciudad --</option>

@if (cities is not null)

{

@foreach (var city in cities)

{

<option value="@city.Id">@city.Name</option>

}

}

</select>

<ValidationMessage For="@(() => userDTO.CityId)" />

</div>

</div>

<div class="mb-3">

<label>Foto:</label>

<div>

<InputText class="form-control" @bind-Value="@userDTO.Photo" />

<ValidationMessage For="@(() => userDTO.Photo)" />

</div>

</div>

<div class="mb-3">

<label>Contraseña:</label>

<div>

<InputText type="password" class="form-control" @bind-Value="@userDTO.Password" />

<ValidationMessage For="@(() => userDTO.Password)" />

</div>

</div>

<div class="mb-3">

<label>Confirmación de contraseña:</label>

<div>

<InputText type="password" class="form-control" @bind-Value="@userDTO.PasswordConfirm" />

<ValidationMessage For="@(() => userDTO.PasswordConfirm)" />

</div>

</div>

</div>

</div>

</div>

</div>

</EditForm>

1. Y este es un ejemplo de como puede quedar la página de **Login**:

@page "/Login"

@inject IRepository repository

@inject SweetAlertService sweetAlertService

@inject NavigationManager navigationManager

@inject ILoginService loginService

<div class="row">

<div class="col-md-4 offset-md-4">

<EditForm Model="loginDTO" OnValidSubmit="LoginAsync">

<DataAnnotationsValidator />

<div class="card bg-light">

<div class="card-header justify-content-center">

<span>

<i class="oi oi-account-login" /> Iniciar Sesión

<button class="btn btn-sm btn-primary float-end" type="submit"><i class="oi oi-check" /> Iniciar Sesión</button>

</span>

</div>

<div class="card-body">

<div class="mb-3">

<label>Email:</label>

<div>

<InputText class="form-control" @bind-Value="@loginDTO.Email" />

<ValidationMessage For="@(() => loginDTO.Email)" />

</div>

</div>

<div class="card-body">

<label>Contraseña:</label>

<div>

<InputText type="password" class="form-control" @bind-Value="@loginDTO.Password" />

<ValidationMessage For="@(() => loginDTO.Password)" />

</div>

</div>

</div>

</div>

</EditForm>

</div>

</div>

1. También cambiemos todos los **<p>Cargando…</p>** por **<div class="spinner" />**
2. Hacemos el **commit**.

## ´Almacenando la foto del usuario

1. Creamos el componente genérico **InputImg.razor**:

<div>

<label>@Label</label>

<div>

<InputFile OnChange="OnChange" accept=".jpg,.jpeg,.png" />

</div>

</div>

<div>

@if (imageBase64 is not null)

{

<div>

<div style="margin: 10px">

<img src="data:image/jpeg;base64, @imageBase64" style="width:400px" />

</div>

</div>

}

@if (ImageURL is not null)

{

<div>

<div style="margin: 10px">

<img src="@ImageURL" style="width:400px" />

</div>

</div>

}

</div>

@code {

[Parameter] public string Label { get; set; } = "Imagen";

[Parameter] public string? ImageURL { get; set; }

[Parameter] public EventCallback<string> ImageSelected { get; set; }

private string? imageBase64;

async Task OnChange(InputFileChangeEventArgs e)

{

var imagenes = e.GetMultipleFiles();

foreach (var imagen in imagenes)

{

var arrBytes = new byte[imagen.Size];

await imagen.OpenReadStream().ReadAsync(arrBytes);

imageBase64 = Convert.ToBase64String(arrBytes);

ImageURL = null;

await ImageSelected.InvokeAsync(imageBase64);

StateHasChanged();

}

}

}

1. Modificamos nuestra página de **Register.razor**:

@using StoresG8.WEB.Shared;

…

<div class="mb-3">

<label>Confirmación de contraseña:</label>

<div>

<InputText type="password" class="form-control" @bind-Value="@userDTO.PasswordConfirm" />

<ValidationMessage For="@(() => userDTO.PasswordConfirm)" />

</div>

</div>

<div class="mb-3">

<InputImg Label="Foto" ImageSelected="ImageSelected" ImageURL="@imageUrl" />

</div>

</div>

</div>

</div>

</div>

</EditForm>

@code {

private UserDTO userDTO = new();

private List<Country>? countries;

private List<State>? states;

private List<City>? cities;

private bool loading;

private string? imageUrl;

protected override async Task OnInitializedAsync()

{

await LoadCountriesAsync();

if (!string.IsNullOrEmpty(userDTO.Photo))

{

imageUrl = userDTO.Photo;

userDTO.Photo = null;

}

}

private void ImageSelected(string imagenBase64)

{

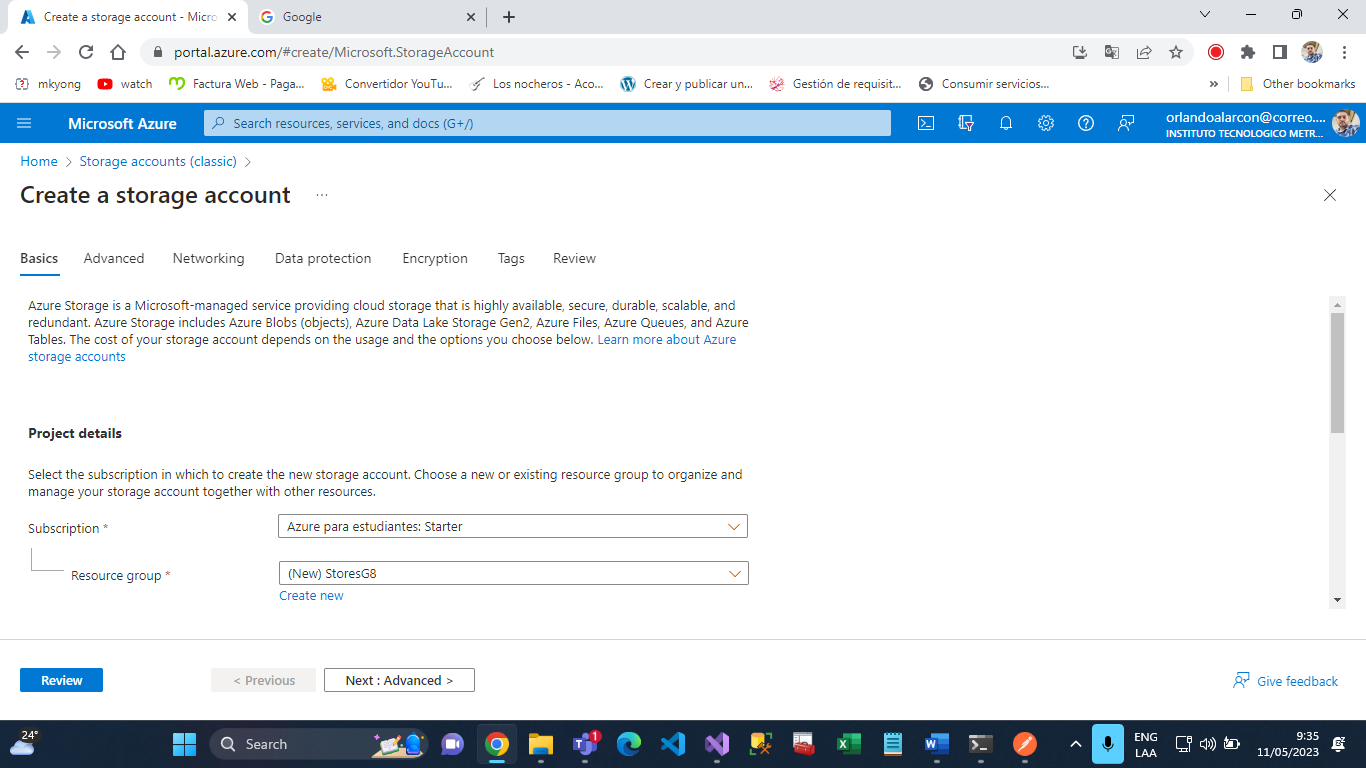
userDTO.Photo = imagenBase64;

imageUrl = null;

}

…

1. Probamos lo que llevamos hasta el momento.
2. Ahora vamos a crear el **blob** en **Azure**:

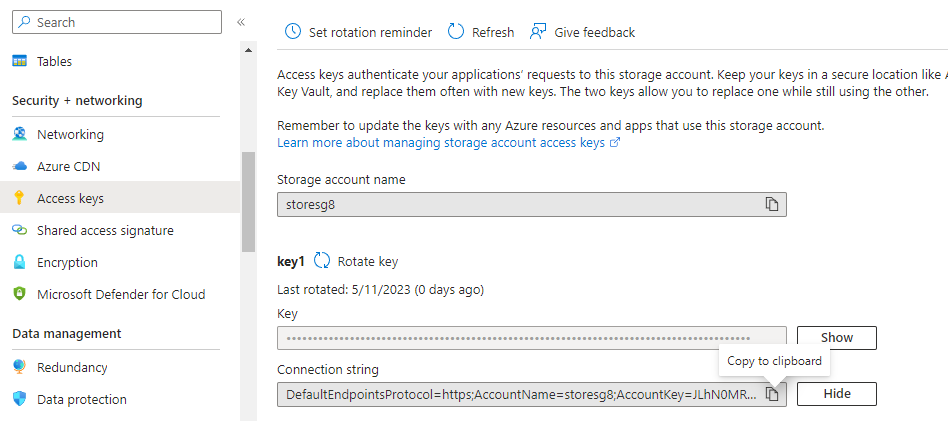


1. Y luego creamos los contenedores para **users** y **products**:

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

1. Luego que termine oprimimos el botón **GO TO RESOURCE**, luego buscamos en el menú lateral izquierdo Access Key y copiamos el Connection String que necesitamos para acceder a nuestro blob storage, en mi caso es:



DefaultEndpointsProtocol=https;AccountName=storesG8;AccountKey=JLhN0MRAddCMbjk0a+uQDg0E4pz7mRuva/9hsVGfVurU4DdUV1ILMyRJEeh5Npfl8kyZjpko65WQ+AStqqBcFw==;EndpointSuffix=core.windows.net

1. Agregamos ese connection string en el **appsettings** de nuestro proyecto **API**:

"ConnectionStrings": {

"DefaultConnection": "Server= OALARCON;Database=StoresG8;Encrypt=False;User Id=dba;Password=Abcd1234\*;"

"AzureStorage": "DefaultEndpointsProtocol=https;AccountName=storesG8;AccountKey=JLhN0MRAddCMbjk0a+uQDg0E4pz7mRuva/9hsVGfVurU4DdUV1ILMyRJEeh5Npfl8kyZjpko65WQ+AStqqBcFw==;EndpointSuffix=core.windows.net"

},

1. En el proyecto **API** en la carpeta **Helpers** creamos la interfaz **IFileStorage**:

namespace StoresG8.API.Helpers

{

public interface IFileStorage

{

Task<string> SaveFileAsync(byte[] content, string extention, string containerName);

Task RemoveFileAsync(string path, string nombreContenedor);

async Task<string> EditFileAsync(byte[] content, string extention, string containerName, string path)

{

if (path is not null)

{

await RemoveFileAsync(path, containerName);

}

return await SaveFileAsync(content, extention, containerName);

}

}

}

1. En la misma carpeta creamos la implementation **FileStorage**:

Instalamos el nugget llamado Azure.Storage,Blobs en el proyecto API

using Azure.Storage.Blobs;

using Azure.Storage.Blobs.Models;

namespace StoresG8.API.Helpers

{

public class FileStorage : IFileStorage

{

private readonly string connectionString;

public FileStorage(IConfiguration configuration)

{

connectionString = configuration.GetConnectionString("AzureStorage")!;

}

public async Task RemoveFileAsync(string path, string containerName)

{

var client = new BlobContainerClient(connectionString, containerName);

await client.CreateIfNotExistsAsync();

var fileName = Path.GetFileName(path);

var blob = client.GetBlobClient(fileName);

await blob.DeleteIfExistsAsync();

}

public async Task<string> SaveFileAsync(byte[] content, string extention, string containerName)

{

var client = new BlobContainerClient(connectionString, containerName);

await client.CreateIfNotExistsAsync();

client.SetAccessPolicy(PublicAccessType.Blob);

var fileName = $"{Guid.NewGuid()}{extention}";

var blob = client.GetBlobClient(fileName);

using (var ms = new MemoryStream(content))

{

await blob.UploadAsync(ms);

}

return blob.Uri.ToString();

}

}

}

1. Configuramos la nueva inyección en el **Program** del **API**:

builder.Services.AddScoped<IUserHelper, UserHelper>();

builder.Services.AddScoped<IFileStorage, FileStorage>();

1. Modificamos el **AccountsController**:

[ApiController]

[Route("/api/accounts")]

public class AccountsController : ControllerBase

{

private readonly IUserHelper \_userHelper;

private readonly IConfiguration \_configuration;

private readonly IFileStorage \_fileStorage;

private readonly string \_container;

public AccountsController(IUserHelper userHelper, IConfiguration configuration, IFileStorage fileStorage)

{

\_userHelper = userHelper;

\_configuration = configuration;

\_fileStorage = fileStorage;

\_container = "users";

}

[HttpPost("CreateUser")]

public async Task<ActionResult> CreateUser([FromBody] UserDTO model)

{

User user = model;

if(!string.IsNullOrEmpty(model.Photo))

{

var photoUser = Convert.FromBase64String(model.Photo);

model.Photo = await \_fileStorage.SaveFileAsync(photoUser, ".jpg", \_container);

}

var result = await \_userHelper.AddUserAsync(user, model.Password);

if (result.Succeeded)

{

await \_userHelper.AddUserToRoleAsync(user, user.UserType.ToString());

return Ok(BuildToken(user));

}

return BadRequest(result.Errors.FirstOrDefault());

}

1. Modificamos el **AuthLinks.razor**:

<AuthorizeView>

<Authorized>

<span>Hola, @context.User.Identity!.Name</span>

@if (!string.IsNullOrEmpty(photoUser))

{

<div class="mx-2">

<img src="@photoUser" width="50" height="50" style="border-radius:50%" />

</div>

}

<a href="Logout" class="nav-link btn btn-link">Cerrar Sesión</a>

</Authorized>

<NotAuthorized>

<a href="Register" class="nav-link btn btn-link">Registro</a>

<a href="Login" class="nav-link btn btn-link">Iniciar Sesión</a>

</NotAuthorized>

</AuthorizeView>

@code {

private string? photoUser;

[CascadingParameter]

private Task<AuthenticationState> authenticationStateTask { get; set; } = null!;

protected async override Task OnParametersSetAsync()

{

var authenticationState = await authenticationStateTask;

var claims = authenticationState.User.Claims.ToList();

var photoClaim = claims.FirstOrDefault(x => x.Type == "Photo");

if (photoClaim is not null)

{

photoUser = photoClaim.Value;

}

}

}

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

## Editando el usuario

1. A la interfaz **IUserHelper** le adicionamos los siguientes métodos:

Task<IdentityResult> ChangePasswordAsync(User user, string currentPassword, string newPassword);

Task<IdentityResult> UpdateUserAsync(User user);

Task<User> GetUserAsync(Guid userId);

1. Implementamos los nuevos métodos en el **UserHelper**:

public async Task<User> GetUserAsync(string email)

{

var user = await \_context.Users

.Include(u => u.City!)

.ThenInclude(c => c.State!)

.ThenInclude(s => s.Country!)

.FirstOrDefaultAsync(x => x.Email == email);

return user!;

}

public async Task<User> GetUserAsync(Guid userId)

{

var user = await \_context.Users

.Include(u => u.City!)

.ThenInclude(c => c.State!)

.ThenInclude(s => s.Country!)

.FirstOrDefaultAsync(x => x.Id == userId.ToString());

return user!;

}

public async Task<IdentityResult> ChangePasswordAsync(User user, string currentPassword, string newPassword)

{

return await \_userManager.ChangePasswordAsync(user, currentPassword, newPassword);

}

public async Task<IdentityResult> UpdateUserAsync(User user)

{

return await \_userManager.UpdateAsync(user);

}

1. Creamos estos métodos en el **AccountsController**:

[HttpPut]

[Authorize(AuthenticationSchemes = JwtBearerDefaults.AuthenticationScheme)]

public async Task<ActionResult> Put(User user)

{

try

{

if (!string.IsNullOrEmpty(user.Photo))

{

var photoUser = Convert.FromBase64String(user.Photo);

user.Photo = await \_fileStorage.SaveFileAsync(photoUser, ".jpg", \_container);

}

var currentUser = await \_userHelper.GetUserAsync(user.Email!);

if (currentUser == null)

{

return NotFound();

}

currentUser.Document = user.Document;

currentUser.FirstName = user.FirstName;

currentUser.LastName = user.LastName;

currentUser.Address = user.Address;

currentUser.PhoneNumber = user.PhoneNumber;

currentUser.Photo = !string.IsNullOrEmpty(user.Photo) && user.Photo != currentUser.Photo ? user.Photo : currentUser.Photo;

currentUser.CityId = user.CityId;

var result = await \_userHelper.UpdateUserAsync(currentUser);

if (result.Succeeded)

{

return NoContent();

}

return BadRequest(result.Errors.FirstOrDefault());

}

catch (Exception ex)

{

return BadRequest(ex.Message);

}

}

[HttpGet]

[Authorize(AuthenticationSchemes = JwtBearerDefaults.AuthenticationScheme)]

public async Task<ActionResult> Get()

{

return Ok(await \_userHelper.GetUserAsync(User.Identity!.Name!));

}

1. Modificamos el **AuthLinks**:

<Authorized>

Hola, <a href="EditUser" class="nav-link btn btn-link">@context.User.Identity!.Name</a>

@if (!string.IsNullOrEmpty(photoUser))

{

<div class="mx-2">

<img src="@photoUser" width="50" height="50" style="border-radius:50%" />

</div>

}

<a href="Logout" class="nav-link btn btn-link">Cerrar Sesión</a>

</Authorized>

1. Creamos el **EditUser.razor**:

@page "/EditUser"

@using CurrieTechnologies.Razor.SweetAlert2;

@using StoresG8.Shared.Entities;

@using StoresG8.WEB.Auth;

@using StoresG8.WEB.Repositories;

@inject IRepository repository

@inject SweetAlertService sweetAlertService

@inject NavigationManager navigationManager

@inject ILoginService loginService

@if (user is null)

{

<div class="spinner" />

}

else

{

<EditForm Model="user" OnValidSubmit="SaveUserAsync">

<DataAnnotationsValidator />

<div class="card">

<div class="card-header">

<span>

<i class="oi oi-person" /> Editar Usuario

<a class="btn btn-sm btn-secondary float-end" href="/changePassword"><i class="oi oi-key" /> Cambiar Contraseña</a>

<button class="btn btn-sm btn-primary float-end mx-2" type="submit"><i class="oi oi-check" /> Guardar Cambios</button>

</span>

</div>

<div class="card-body">

<div class="row">

<div class="col-6">

<div class="mb-3">

<label>Nombres:</label>

<div>

<InputText class="form-control" @bind-Value="@user.FirstName" />

<ValidationMessage For="@(() => user.FirstName)" />

</div>

</div>

<div class="mb-3">

<label>Apellidos:</label>

<div>

<InputText class="form-control" @bind-Value="@user.LastName" />

<ValidationMessage For="@(() => user.LastName)" />

</div>

</div>

<div class="mb-3">

<label>Documento:</label>

<div>

<InputText class="form-control" @bind-Value="@user.Document" />

<ValidationMessage For="@(() => user.Document)" />

</div>

</div>

<div class="mb-3">

<label>Teléfono:</label>

<div>

<InputText class="form-control" @bind-Value="@user.PhoneNumber" />

<ValidationMessage For="@(() => user.PhoneNumber)" />

</div>

</div>

<div class="mb-3">

<label>Dirección:</label>

<div>

<InputText class="form-control" @bind-Value="@user.Address" />

<ValidationMessage For="@(() => user.Address)" />

</div>

</div>

</div>

<div class="col-6">

<div class="mb-3">

<label>País:</label>

<div>

<select class="form-select" @onchange="CountryChangedAsync">

<option value="0">-- Seleccione un país --</option>

@if (countries is not null)

{

@foreach (var country in countries)

{

<option value="@country.Id" selected="@(country.Id == user.City!.State!.Country!.Id)">@country.Name</option>

}

}

</select>

</div>

</div>

<div class="mb-3">

<label>Estado/Departamento:</label>

<div>

<select class="form-select" @onchange="StateChangedAsync">

<option value="0">-- Seleccione un estado/departamento --</option>

@if (states is not null)

{

@foreach (var state in states)

{

<option value="@state.Id" selected="@(state.Id == user.City!.State!.Id)">@state.Name</option>

}

}

</select>

</div>

</div>

<div class="mb-3">

<label>Ciudad:</label>

<div>

<select class="form-select" @bind="user.CityId">

<option value="0">-- Seleccione una ciudad --</option>

@if (cities is not null)

{

@foreach (var city in cities)

{

<option value="@city.Id" selected="@(city.Id == user.City!.Id)">@city.Name</option>

}

}

</select>

<ValidationMessage For="@(() => user.CityId)" />

</div>

</div>

<div class="mb-3">

<InputImg Label="Foto" ImageSelected="ImageSelected" ImageURL="@imageUrl" />

</div>

</div>

</div>

</div>

</div>

</EditForm>

}

@code {

private User? user;

private List<Country>? countries;

private List<State>? states;

private List<City>? cities;

private string? imageUrl;

protected override async Task OnInitializedAsync()

{

await LoadUserAsyc();

await LoadCountriesAsync();

await LoadStatesAsyn(user!.City!.State!.Country!.Id);

await LoadCitiesAsyn(user!.City!.State!.Id);

if (!string.IsNullOrEmpty(user!.Photo))

{

imageUrl = user.Photo;

user.Photo = null;

}

}

private async Task LoadUserAsyc()

{

var responseHTTP = await repository.Get<User>($"/api/accounts");

if (responseHTTP.Error)

{

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

{

navigationManager.NavigateTo("/");

return;

}

var messageError = await responseHTTP.GetErrorMessageAsync();

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

return;

}

user = responseHTTP.Response;

}

private void ImageSelected(string imagenBase64)

{

user!.Photo = imagenBase64;

imageUrl = null;

}

private async Task CountryChangedAsync(ChangeEventArgs e)

{

var selectedCountry = Convert.ToInt32(e.Value!);

await LoadStatesAsyn(selectedCountry);

}

private async Task StateChangedAsync(ChangeEventArgs e)

{

var selectedState = Convert.ToInt32(e.Value!);

await LoadCitiesAsyn(selectedState);

}

private async Task LoadCountriesAsync()

{

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

if (responseHttp.Error)

{

var message = await responseHttp.GetErrorMessageAsync();

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

return;

}

countries = responseHttp.Response;

}

private async Task LoadStatesAsyn(int countryId)

{

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

if (responseHttp.Error)

{

var message = await responseHttp.GetErrorMessageAsync();

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

return;

}

states = responseHttp.Response;

}

private async Task LoadCitiesAsyn(int stateId)

{

var responseHttp = await repository.Get<List<City>>($"/api/cities/combo/{stateId}");

if (responseHttp.Error)

{

var message = await responseHttp.GetErrorMessageAsync();

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

return;

}

cities = responseHttp.Response;

}

private async Task SaveUserAsync()

{

var responseHttp = await repository.Put<User>("/api/accounts", user!);

if (responseHttp.Error)

{

var message = await responseHttp.GetErrorMessageAsync();

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

return;

}

navigationManager.NavigateTo("/");

}

}

1. Probamos.

## Cambiando password del usuario

1. Dentro de **StoresG8.Shared.DTOs** creamos el **ChangePasswordDTO**:

using System.ComponentModel.DataAnnotations;

namespace StoresG8.Shared.DTOs

{

public class ChangePasswordDTO

{

[DataType(DataType.Password)]

[Display(Name = "Contraseña actual")]

[StringLength(20, MinimumLength = 6, ErrorMessage = "El campo {0} debe tener entre {2} y {1} carácteres.")]

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

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

[DataType(DataType.Password)]

[Display(Name = "Nueva contraseña")]

[StringLength(20, MinimumLength = 6, ErrorMessage = "El campo {0} debe tener entre {2} y {1} carácteres.")]

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

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

[Compare("NewPassword", ErrorMessage = "La nueva contraseña y la confirmación no son iguales.")]

[DataType(DataType.Password)]

[Display(Name = "Confirmación nueva contraseña")]

[StringLength(20, MinimumLength = 6, ErrorMessage = "El campo {0} debe tener entre {2} y {1} carácteres.")]

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

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

}

}

1. En **StoresG8.API.Controllers** en el controlador **AccountsController** adicionamos este método:

[HttpPost("changePassword")]

[Authorize(AuthenticationSchemes = JwtBearerDefaults.AuthenticationScheme)]

public async Task<ActionResult> ChangePasswordAsync(ChangePasswordDTO model)

{

if (!ModelState.IsValid)

{

return BadRequest(ModelState);

}

var user = await \_userHelper.GetUserAsync(User.Identity!.Name!);

if (user == null)

{

return NotFound();

}

var result = await \_userHelper.ChangePasswordAsync(user, model.CurrentPassword, model.NewPassword);

if (!result.Succeeded)

{

return BadRequest(result.Errors.FirstOrDefault().Description);

}

return NoContent();

}

1. Dentro de **StoresG8.WEB.Pages**.**Auth** creamos el **ChangePassword.razor**:

@page "/changePassword"

@using StoresG8.Shared.DTOs;

@inject IRepository repository

@inject SweetAlertService sweetAlertService

@inject NavigationManager navigationManager

@if (loading)

{

<div class="spinner" />

}

<div class="row">

<div class="col-6">

<EditForm Model="changePasswordDTO" OnValidSubmit="ChangePasswordAsync">

<DataAnnotationsValidator />

<div class="card">

<div class="card-header">

<span>

<i class="oi oi-key" /> Cambiar Contraseña

<a class="btn btn-sm btn-success float-end" href="/editUser"><i class="oi oi-arrow-thick-left" /> Regresar</a>

<button class="btn btn-sm btn-primary float-end mx-2" type="submit"><i class="oi oi-check" /> Guardar Cambios</button>

</span>

</div>

<div class="card-body">

<div class="mb-3">

<label>Contraseña actual:</label>

<div>

<InputText type="password" class="form-control" @bind-Value="@changePasswordDTO.CurrentPassword" />

<ValidationMessage For="@(() => changePasswordDTO.CurrentPassword)" />

</div>

</div>

<div class="mb-3">

<label>Nueva contraseña:</label>

<div>

<InputText type="password" class="form-control" @bind-Value="@changePasswordDTO.NewPassword" />

<ValidationMessage For="@(() => changePasswordDTO.CurrentPassword)" />

</div>

</div>

<div class="mb-3">

<label>Confirmación de nueva contraseña:</label>

<div>

<InputText type="password" class="form-control" @bind-Value="@changePasswordDTO.Confirm" />

<ValidationMessage For="@(() => changePasswordDTO.Confirm)" />

</div>

</div>

</div>

</div>

</EditForm>

</div>

</div>

@code {

private ChangePasswordDTO changePasswordDTO = new();

private bool loading;

private async Task ChangePasswordAsync()

{

loading = true;

var responseHttp = await repository.Post("/api/accounts/changePassword", changePasswordDTO);

if (responseHttp.Error)

{

var message = await responseHttp.GetErrorMessageAsync();

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

loading = false;

return;

}

loading = false;

navigationManager.NavigateTo("/editUser");

var toast = sweetAlertService.Mixin(new SweetAlertOptions

{

Toast = true,

Position = SweetAlertPosition.TopEnd,

ShowConfirmButton = true,

Timer = 5000

});

await toast.FireAsync(icon: SweetAlertIcon.Success, message: "Contraseña cambiada con éxito.");

}

}

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

## Confirmar el registro de usuarios

1. Cambiamos la configuración de usuarios en el **Program** del **API**:

builder.Services.AddIdentity<User, IdentityRole>(x =>

{

x.Tokens.AuthenticatorTokenProvider = TokenOptions.DefaultAuthenticatorProvider;

x.SignIn.RequireConfirmedEmail = true;

x.User.RequireUniqueEmail = true;

x.Password.RequireDigit = false;

x.Password.RequiredUniqueChars = 0;

x.Password.RequireLowercase = false;

x.Password.RequireNonAlphanumeric = false;

x.Password.RequireUppercase = false;

x.Lockout.DefaultLockoutTimeSpan = TimeSpan.FromMinutes(5);

x.Lockout.MaxFailedAccessAttempts = 3;

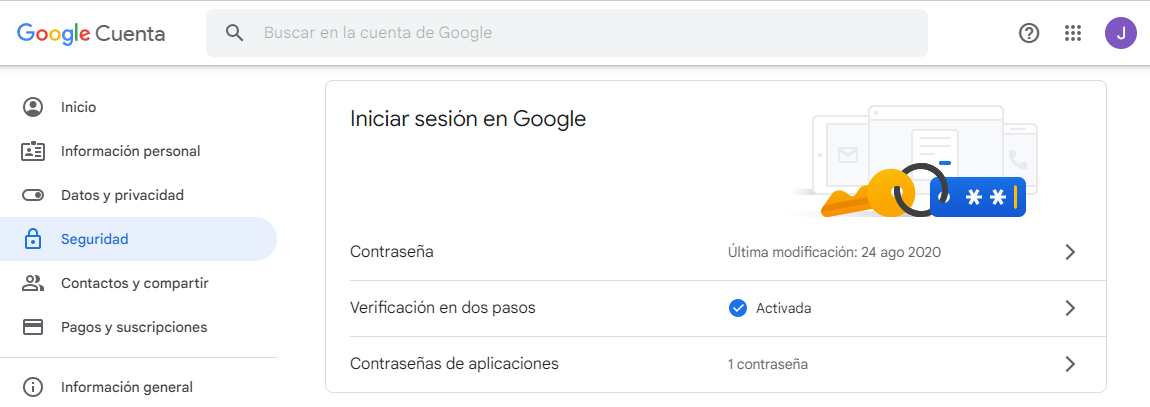
x.Lockout.AllowedForNewUsers = true;

})

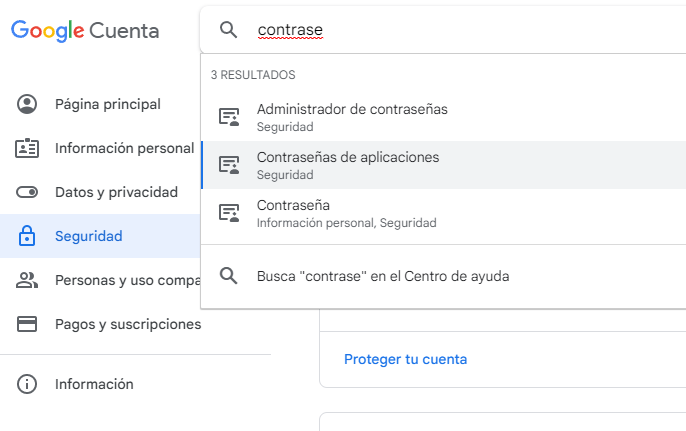
.AddEntityFrameworkStores<DataContext>()

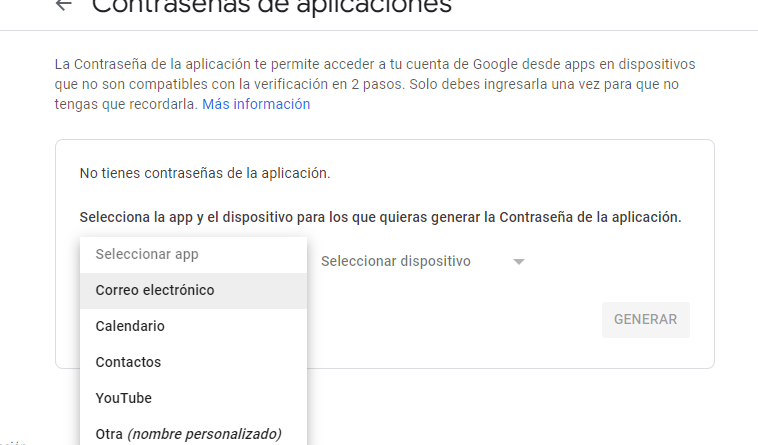
.AddDefaultTokenProviders();

1. Verificamos que la cuenta de Google tenga activa la verificación en dos pasos.



Debemos digitar en la lupa de la cuenta de Google lo siguiente: Contraseñas de aplicaciones, y generamos una nueva contraseña de aplicación, y le colocamos como nombre personalizado: Send Mail, copiamos la contraseña generada y la copiamos en el siguiente paso.



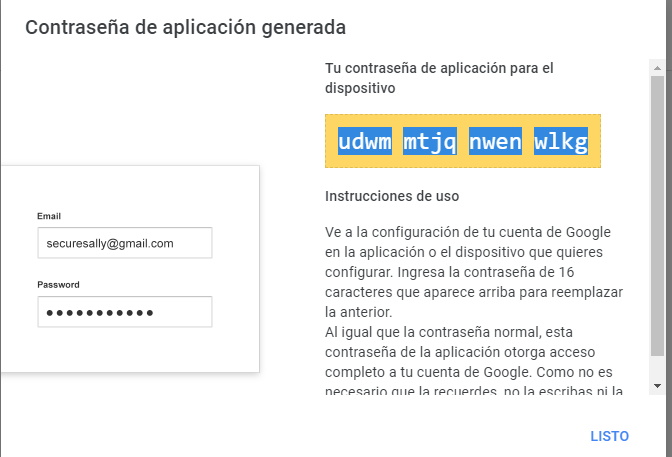


Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente



1. Adicionamos estos parámetros al Appsettings del **API**:

"Mail": {

"From": "Stores@gmail.com",

"Name": "Soporte StoresG8",

"Smtp": "smtp.gmail.com",

"Port": 587,

"Password": "nniufszzppfuzhxe"

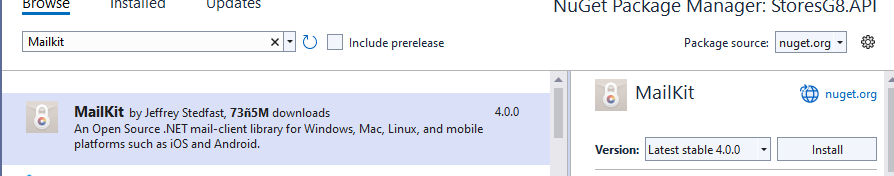
},

"UrlWEB": "localhost:7057"

}

**Nota**: reemplazar el 7057 por el puerto donde sale tu App WEB, y reemplazar el password por el generado de tu cuenta.

1. Adicionamos el nuget “**Mailkit**” al proyecto **API**:



1. En los **Helpers** del **API** adicionamos la interzar **IMailHelper**:

using StoresG8.Shared.Responses;

public interface IMailHelper

{

Response SendMail(string toName, string toEmail, string subject, string body);

}

1. Luego agregamos la implementation **MailHelper**:

using MailKit.Net.Smtp;

using MimeKit;

using StoresG8.Shared.Responses;

namespace StoresG8.API.Helpers

{

public class MailHelper : IMailHelper

{

private readonly IConfiguration \_configuration;

public MailHelper(IConfiguration configuration)

{

\_configuration = configuration;

}

public Response SendMail(string toName, string toEmail, string subject, string body)

{

try

{

var from = \_configuration["Mail:From"];

var name = \_configuration["Mail:Name"];

var smtp = \_configuration["Mail:Smtp"];

var port = \_configuration["Mail:Port"];

var password = \_configuration["Mail:Password"];

var message = new MimeMessage();

message.From.Add(new MailboxAddress(name, from));

message.To.Add(new MailboxAddress(toName, toEmail));

message.Subject = subject;

BodyBuilder bodyBuilder = new BodyBuilder

{

HtmlBody = body

};

message.Body = bodyBuilder.ToMessageBody();

using (var client = new SmtpClient())

{

client.Connect(smtp, int.Parse(port!), false);

client.Authenticate(from, password);

client.Send(message);

client.Disconnect(true);

}

return new Response { IsSuccess = true };

}

catch (Exception ex)

{

return new Response

{

IsSuccess = false,

Message = ex.Message,

Result = ex

};

}

}

}

}

1. Configuramos la inyección del servicio:

builder.Services.AddScoped<IMailHelper, MailHelper>();

1. Add those methods to **IUserHelper**:

Task<string> GenerateEmailConfirmationTokenAsync(User user);

Task<IdentityResult> ConfirmEmailAsync(User user, string token);

Y la implementación en la clase UserHelper:

public async Task<string> GenerateEmailConfirmationTokenAsync(User user)

{

return await \_userManager.GenerateEmailConfirmationTokenAsync(user);

}

public async Task<IdentityResult> ConfirmEmailAsync(User user, string token)

{

return await \_userManager.ConfirmEmailAsync(user, token);

}

1. Modificamos el método **CreateUser** del controlador **AccountsController**

(primero inyectamos el **IMailHelper**):

private readonly IMailHelper \_mailHelper;

public AccountsController(IUserHelper userHelper, IConfiguration configuration, IFileStorage fileStorage,IMailHelper mailHelper)

{

\_userHelper = userHelper;

\_configuration = configuration;

\_fileStorage = fileStorage;

\_container = "users";

this.\_mailHelper = mailHelper;

}

[HttpPost("CreateUser")]

public async Task<ActionResult> CreateUser([FromBody] UserDTO model)

{

User user = model;

if (!string.IsNullOrEmpty(model.Photo))

{

var photoUser = Convert.FromBase64String(model.Photo);

model.Photo = await \_fileStorage.SaveFileAsync(photoUser, ".jpg", \_container);

}

var result = await \_userHelper.AddUserAsync(user, model.Password);

if (result.Succeeded)

{

await \_userHelper.AddUserToRoleAsync(user, user.UserType.ToString());

eliminamos las siguientes líneas:

return Ok(BuildToken(user));

}

return BadRequest(result.Errors.FirstOrDefault());

}

y las reemplazamos por el siguiente código

var myToken = await \_userHelper.GenerateEmailConfirmationTokenAsync(user);

var tokenLink = Url.Action("ConfirmEmail", "accounts", new

{

userid = user.Id,

token = myToken

}, HttpContext.Request.Scheme, \_configuration["UrlWEB"]);

var response = \_mailHelper.SendMail(user.FullName, user.Email!,

$"StoresG8s- Confirmación de cuenta",

$"<h1>StoresG8 - Confirmación de cuenta</h1>" +

$"<p>Para habilitar el usuario, por favor hacer clic 'Confirmar Email':</p>" +

$"<b><a href ={tokenLink}>Confirmar Email</a></b>");

if (response.IsSuccess)

{

return NoContent();

}

return BadRequest(response.Message);

}

return BadRequest(result.Errors.FirstOrDefault());

}

1. Crear el método para confirmar el email en el **AccountsController**:

[HttpGet("ConfirmEmail")]

public async Task<ActionResult> ConfirmEmailAsync(string userId, string token)

{

token = token.Replace(" ", "+");

var user = await \_userHelper.GetUserAsync(new Guid(userId));

if (user == null)

{

return NotFound();

}

var result = await \_userHelper.ConfirmEmailAsync(user, token);

if (!result.Succeeded)

{

return BadRequest(result.Errors.FirstOrDefault());

}

return NoContent();

}

1. Modificamos el método **Login** en el **AccountsController**:

[HttpPost("Login")]

public async Task<ActionResult> Login([FromBody] LoginDTO model)

{

var result = await \_userHelper.LoginAsync(model);

if (result.Succeeded)

{

var user = await \_userHelper.GetUserAsync(model.Email);

return Ok(BuildToken(user));

}

if (result.IsLockedOut)

{

return BadRequest("Ha superado el máximo número de intentos, su cuenta está bloqueada, intente de nuevo en 5 minutos.");

}

if (result.IsNotAllowed)

{

return BadRequest("El usuario no ha sido habilitado, debes de seguir las instrucciones del correo enviado para poder habilitar el usuario.");

}

return BadRequest("Email o contraseña incorrectos.");

}

1. Agregamos este método al **IRepository del proyecto Web**:

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

1. Lo implementamos en el **Repository**:

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

{

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

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

}

1. Dentro de **Pages/Auth** creamos la página **ConfirmEmail.razor**:

@page "/api/accounts/ConfirmEmail"

@inject IRepository repository

@inject SweetAlertService sweetAlertService

@inject NavigationManager navigationManager

<h3>Confirmación de email</h3>

<p>Presione el botón para confirmar su cuenta</p>

<button class="btn btn-primary" @onclick="ConfirmAccountAsync">Confirmar Cuenta</button>

@code {

private string? message;

[Parameter]

[SupplyParameterFromQuery]

public string UserId { get; set; } = "";

[Parameter]

[SupplyParameterFromQuery]

public string Token { get; set; } = "";

protected async Task ConfirmAccountAsync()

{

var responseHttp = await repository.Get($"/api/accounts/ConfirmEmail/?userId={UserId}&token={Token}");

if (responseHttp.Error)

{

message = await responseHttp.GetErrorMessageAsync();

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

navigationManager.NavigateTo("/");

}

else

{

await sweetAlertService.FireAsync("Confirmación", "Gracias por confirmar su email, ahora puedes ingresar al sistema.", SweetAlertIcon.Info);

navigationManager.NavigateTo("/Login");

}

}

}

1. Borramos los usuarios de la base de datos.
2. Modificamos el alimentador de la base de datos:

private async Task<User> CheckUserAsync(string document, string firstName, string lastName, string email, string phone, string address, UserType userType)

{

var user = await \_userHelper.GetUserAsync(email);

if (user == null)

{

var city = await \_context.Cities.FirstOrDefaultAsync(x => x.Name == "Medellín");

if (city == null)

{

city = await \_context.Cities.FirstOrDefaultAsync();

}

user = new User

{

FirstName = firstName,

LastName = lastName,

Email = email,

UserName = email,

PhoneNumber = phone,

Address = address,

Document = document,

City = city,

UserType = userType,

};

await \_userHelper.AddUserAsync(user, "123456");

await \_userHelper.AddUserToRoleAsync(user, userType.ToString());

var token = await \_userHelper.GenerateEmailConfirmationTokenAsync(user);

await \_userHelper.ConfirmEmailAsync(user, token);

}

return user;

}

1. Modificamos el **Register.razor**:

private async Task CreteUserAsync()

{

loading = true;

userDTO.UserName = userDTO.Email;

userDTO.UserType = UserType.User;

var responseHttp = await repository.Post<UserDTO>("/api/accounts/CreateUser", userDTO);

if (responseHttp.Error)

{

var message = await responseHttp.GetErrorMessageAsync();

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

loading = false;

return;

}

loading = false;

await sweetAlertService.FireAsync("Confirmación", "Su cuenta ha sido creada con éxito. Se te ha enviado un correo electrónico con las instrucciones para activar tu usuario.", SweetAlertIcon.Info);

navigationManager.NavigateTo("/");

}

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

## Reenviar correo de confirmación

1. En **StoresG8.Shared.DTOs** creamos la clase **EmailDTO**:

using System.ComponentModel.DataAnnotations;

namespace StoresG8.Shared.DTOs

{

public class EmailDTO

{

[Display(Name = "Email")]

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

[EmailAddress(ErrorMessage = "Debes ingresar un correo válido.")]

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

}

}

1. En el **API** creamos este método en el **AccountsController**:

[HttpPost("ResedToken")]

public async Task<ActionResult> ResedToken([FromBody] EmailDTO model)

{

User user = await \_userHelper.GetUserAsync(model.Email);

if (user == null)

{

return NotFound();

}

var myToken = await \_userHelper.GenerateEmailConfirmationTokenAsync(user);

var tokenLink = Url.Action("ConfirmEmail", "accounts", new

{

userid = user.Id,

token = myToken

}, HttpContext.Request.Scheme, \_configuration["UrlWEB"]);

var response = \_mailHelper.SendMail(user.FullName, user.Email!,

$"StoresG8s- Confirmación de cuenta",

$"<h1>StoresG8 - Confirmación de cuenta</h1>" +

$"<p>Para habilitar el usuario, por favor hacer clic 'Confirmar Email':</p>" +

$"<b><a href ={tokenLink}>Confirmar Email</a></b>");

if (response.IsSuccess)

{

return NoContent();

}

return BadRequest(response.Message);

}

1. Modificamos nuestro **Login.razor**:

<div class="row">

<div class="col-md-4 offset-md-4">

<EditForm Model="loginDTO" OnValidSubmit="LoginAsync">

<DataAnnotationsValidator />

<div class="card bg-light">

<div class="card-header justify-content-center">

<span>

<i class="oi oi-account-login" /> Iniciar Sesión

<button class="btn btn-sm btn-primary float-end" type="submit"><i class="oi oi-check" /> Iniciar Sesión</button>

</span>

</div>

<div class="card-body">

<div class="mb-3">

<label>Email:</label>

<div>

<InputText class="form-control" @bind-Value="@loginDTO.Email" />

<ValidationMessage For="@(() => loginDTO.Email)" />

</div>

</div>

<div class="mb-3">

<label>Contraseña:</label>

<div>

<InputText type="password" class="form-control" @bind-Value="@loginDTO.Password" />

<ValidationMessage For="@(() => loginDTO.Password)" />

</div>

</div>

</div>

<div class="card-footer">

<a class="bbtn btn-link" href="/ResendToken">Reenviar correro de activación de cuenta</a>

</div>

</div>

</EditForm>

</div>

</div>

1. Dentro de **Pages/Auth** creamos el componente **ResendConfirmationEmailToken.razor**:

@page "/ResendToken"

@using StoresG8.Shared.DTOs;

@inject IRepository repository

@inject SweetAlertService sweetAlertService

@inject NavigationManager navigationManager

@if (loading)

{

<div class="spinner" />

}

<div class="row">

<div class="col-6">

<EditForm Model="emailDTO" OnValidSubmit="ResendConfirmationEmailTokenAsync">

<DataAnnotationsValidator />

<div class="card">

<div class="card-header">

<span>

<i class="oi oi-key" /> Reenviar correo de confirmación de contraseña

<button class="btn btn-sm btn-primary float-end mx-2" type="submit"><i class="oi oi-loop-square" /> Reenviar</button>

</span>

</div>

<div class="card-body">

<div class="mb-3">

<label>Email:</label>

<div>

<InputText class="form-control" @bind-Value="@emailDTO.Email" />

<ValidationMessage For="@(() => emailDTO.Email)" />

</div>

</div>

</div>

</div>

</EditForm>

</div>

</div>

@code {

private EmailDTO emailDTO = new();

private bool loading;

private async Task ResendConfirmationEmailTokenAsync()

{

loading = true;

var responseHttp = await repository.Post("/api/accounts/ResedToken", emailDTO);

if (responseHttp.Error)

{

var message = await responseHttp.GetErrorMessageAsync();

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

loading = false;

return;

}

loading = false;

await sweetAlertService.FireAsync("Confirmación", "Se te ha enviado un correo electrónico con las instrucciones para activar tu usuario.", SweetAlertIcon.Info);

navigationManager.NavigateTo("/");

}

}

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

## Actualización de la foto del usuario luego de editar usuario

1. Modificamos el método **PUT** del **AccountsController**:

var result = await \_userHelper.UpdateUserAsync(currentUser);

if (result.Succeeded)

{

return Ok(BuildToken(currentUser));

}

1. Agregamos este método al **IRepository**, si ya está creado, no lo tocamos:

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

1. Y su implementación en el **Repository,** si ya está creado, no lo tocamos::

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. Modificamos el **EditUser**:

@using StoresG8.Shared.DTOs;

private async Task SaveUserAsync()

{

var responseHttp = await repository.Put<User, TokenDTO>("/api/accounts", user!);

if (responseHttp.Error)

{

var message = await responseHttp.GetErrorMessageAsync();

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

return;

}

await loginService.LoginAsync(responseHttp.Response!.Token);

navigationManager.NavigateTo("/");

}

1. Probamos y hacemos el **Commit**.

## Recuperación de contraseña

1. Modificamos el **Login.razor**:

<div class="card-footer">

<p><a class="bbtn btn-link" href="/ResendToken">Reenviar correro de activación de cuenta</a></p>

<p><a class="bbtn btn-link" href="/RecoverPassword">¿Has olvidado tu contraseña?</a></p>

</div>

1. Adicionamos en **StoresG8.Shared.DTOs** la clase **ResetPasswordDTO**:

using System.ComponentModel.DataAnnotations;

namespace StoresG8.Shared.DTOs

{

public class ResetPasswordDTO

{

[Display(Name = "Email")]

[EmailAddress(ErrorMessage = "Debes ingresar un correo válido.")]

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

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

[DataType(DataType.Password)]

[Display(Name = "Contraseña")]

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

[StringLength(20, MinimumLength = 6, ErrorMessage = "El campo {0} debe tener entre {2} y {1} carácteres.")]

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

[Compare("Password", ErrorMessage = "La nueva contraseña y la confirmación no son iguales.")]

[DataType(DataType.Password)]

[Display(Name = "Confirmación de contraseña")]

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

[StringLength(20, MinimumLength = 6, ErrorMessage = "El campo {0} debe tener entre {2} y {1} carácteres.")]

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

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

}

}

1. Adicionamos estos métodos al **IUserHelper**:

Task<string> GeneratePasswordResetTokenAsync(User user);

Task<IdentityResult> ResetPasswordAsync(User user, string token, string password);

Y la implementación:

public async Task<string> GeneratePasswordResetTokenAsync(User user)

{

return await \_userManager.GeneratePasswordResetTokenAsync(user);

}

public async Task<IdentityResult> ResetPasswordAsync(User user, string token, string password)

{

return await \_userManager.ResetPasswordAsync(user, token, password);

}

1. Adicionamos estos métodos al **AccountController**:

[HttpPost("RecoverPassword")]

public async Task<ActionResult> RecoverPassword([FromBody] EmailDTO model)

{

User user = await \_userHelper.GetUserAsync(model.Email);

if (user == null)

{

return NotFound();

}

var myToken = await \_userHelper.GeneratePasswordResetTokenAsync(user);

var tokenLink = Url.Action("ResetPassword", "accounts", new

{

userid = user.Id,

token = myToken

}, HttpContext.Request.Scheme, \_configuration["UrlWEB"]);

var response = \_mailHelper.SendMail(user.FullName, user.Email!,

$"StoresG8 - Recuperación de contraseña",

$"<h1>StoresG8 - Recuperación de contraseña</h1>" +

$"<p>Para recuperar su contraseña, por favor hacer clic 'Recuperar Contraseña':</p>" +

$"<b><a href ={tokenLink}>Recuperar Contraseña</a></b>");

if (response.IsSuccess)

{

return NoContent();

}

return BadRequest(response.Message);

}

[HttpPost("ResetPassword")]

public async Task<ActionResult> ResetPassword([FromBody] ResetPasswordDTO model)

{

User user = await \_userHelper.GetUserAsync(model.Email);

if (user == null)

{

return NotFound();

}

var result = await \_userHelper.ResetPasswordAsync(user, model.Token, model.Password);

if (result.Succeeded)

{

return NoContent();

}

return BadRequest(result.Errors.FirstOrDefault()!.Description);

}

1. Dentro de **Pages/Auth** creamos el **RecoverPassword.razor**:

@using StoresG8.Shared.DTOs;

@page "/RecoverPassword"

@inject IRepository repository

@inject SweetAlertService sweetAlertService

@inject NavigationManager navigationManager

@if (loading)

{

<div class="spinner" />

}

<div class="row">

<div class="col-6">

<EditForm Model="emailDTO" OnValidSubmit="SendRecoverPasswordEmailTokenAsync">

<DataAnnotationsValidator />

<div class="card">

<div class="card-header">

<span>

<i class="oi oi-key" /> Enviar email para recuperación de contraseña

<button class="btn btn-sm btn-primary float-end mx-2" type="submit"><i class="oi oi-loop-square" /> Enviar</button>

</span>

</div>

<div class="card-body">

<div class="mb-3">

<label>Email:</label>

<div>

<InputText class="form-control" @bind-Value="@emailDTO.Email" />

<ValidationMessage For="@(() => emailDTO.Email)" />

</div>

</div>

</div>

</div>

</EditForm>

</div>

</div>

@code {

private EmailDTO emailDTO = new();

private bool loading;

private async Task SendRecoverPasswordEmailTokenAsync()

{

loading = true;

var responseHttp = await repository.Post("/api/accounts/RecoverPassword", emailDTO);

if (responseHttp.Error)

{

var message = await responseHttp.GetErrorMessageAsync();

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

loading = false;

return;

}

loading = false;

await sweetAlertService.FireAsync("Confirmación", "Se te ha enviado un correo electrónico con las instrucciones para recuperar su contraseña.", SweetAlertIcon.Info);

navigationManager.NavigateTo("/");

}

}

1. Dentro de **Pages/Auth** creamos el **ResetPassword.razor**:

@using StoresG8.Shared.DTOs;

@page "/api/accounts/ResetPassword"

@inject IRepository repository

@inject SweetAlertService sweetAlertService

@inject NavigationManager navigationManager

@if (loading)

{

<div class="spinner" />

}

<div class="row">

<div class="col-6">

<EditForm Model="resetPasswordDTO" OnValidSubmit="ChangePasswordAsync">

<DataAnnotationsValidator />

<div class="card">

<div class="card-header">

<span>

<i class="oi oi-key" /> Cambiar Contraseña

<button class="btn btn-sm btn-primary float-end mx-2" type="submit"><i class="oi oi-check" /> Cambiar Contrasña</button>

</span>

</div>

<div class="card-body">

<div class="mb-3">

<label>Email:</label>

<div>

<InputText class="form-control" @bind-Value="@resetPasswordDTO.Email" />

<ValidationMessage For="@(() => resetPasswordDTO.Email)" />

</div>

</div>

<div class="mb-3">

<label>Nueva contraseña:</label>

<div>

<InputText type="password" class="form-control" @bind-Value="@resetPasswordDTO.Password" />

<ValidationMessage For="@(() => resetPasswordDTO.Password)" />

</div>

</div>

<div class="mb-3">

<label>Confirmar contraseña:</label>

<div>

<InputText type="password" class="form-control" @bind-Value="@resetPasswordDTO.ConfirmPassword" />

<ValidationMessage For="@(() => resetPasswordDTO.ConfirmPassword)" />

</div>

</div>

</div>

</div>

</EditForm>

</div>

</div>

@code {

private ResetPasswordDTO resetPasswordDTO = new();

private bool loading;

[Parameter]

[SupplyParameterFromQuery]

public string Token { get; set; } = "";

private async Task ChangePasswordAsync()

{

loading = true;

resetPasswordDTO.Token = Token;

var responseHttp = await repository.Post("/api/accounts/ResetPassword", resetPasswordDTO);

if (responseHttp.Error)

{

var message = await responseHttp.GetErrorMessageAsync();

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

loading = false;

return;

}

loading = false;

await sweetAlertService.FireAsync("Confirmación", "Contraseña cambiada con éxito, ahora puede ingresar con su nueva contraseña.", SweetAlertIcon.Info);

navigationManager.NavigateTo("/Login");

}

}

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

## Solución del problema de la paginación

1. Modificamos el componente de **Pagination**:

<nav>

<ul class="pagination">

@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; } = 10;

[Parameter] public EventCallback<int> SelectedPage { get; set; }

List<PageModel> Links = new();

private async Task InternalSelectedPage(PageModel pageModel)

{

if (pageModel.Page == CurrentPage || pageModel.Page == 0)

{

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 (TotalPages <= Radio)

{

Links.Add(new PageModel

{

Page = i,

Enable = CurrentPage == i,

Text = $"{i}"

});

}

if (TotalPages > Radio && i <= Radio && CurrentPage <= Radio)

{

Links.Add(new PageModel

{

Page = i,

Enable = CurrentPage == i,

Text = $"{i}"

});

}

if (CurrentPage > Radio && i > CurrentPage - Radio && i <= CurrentPage)

{

Links.Add(new PageModel

{

Page = i,

Enable = CurrentPage == i,

Text = $"{i}"

});

}

}

var linkNextEnable = CurrentPage != TotalPages;

var linkNextPage = CurrentPage != TotalPages ? CurrentPage + 1 : CurrentPage;

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;

}

}