Orders

With .NET & Blazor

Juan Carlos Zuluaga

2024, Semestre 1

Indice

[Links de interes 5](#_pizmy7ipqv4o)

[Matriz de funcionalidad 5](#_a8m1e8w4zrse)

[Diagrama Entidad Relación 6](#_sy0rxpvxmrea)

[Estructura básica de proyecto 6](#_mjsujqb3lu8w)

[Creando la base de datos con Entity Framework 7](#_liu6zssbsq2q)

[Creando el primer controladorCreando el primer controlador 10](#_aaeoyfm0j8d5)

[Creando nuestros primeros componentes en Blazor 11](#_8naxs9qz8kty)

[Completando las acciones de crear, editar y borrar países 19](#_7w9h64ns2dfe)

[Creando controladores genéricos y solucionando el problema de registros duplicados 26](#_dyc9u2txsmhq)

[Organizar íconos del Home 34](#_c6hb2pk5kuaa)

[CRUD de categorías 36](#_5a2ptfct5bhp)

[Creando un formulario genérico 42](#_atesh6jj5ip5)

[Configurando un repositorio para trabajo en equipo, resolver conflictos y obtener estadísticas de código 45](#_cvkzagn5hp2l)

[Adicionando un Seeder a la base de datos 46](#_h0o53ml8q5wk)

[Relación uno a muchos e índice compuesto 47](#_1uyz16hng9j)

[Creando un CRUD multinivel 58](#_dqbreos4dwee)

[Poblar los Países, Estados y Ciudades con un Backend externa 68](#_phleud5h36mh)

[Agregando paginación 73](#_iodmrt2096ac)

[Agregando filtros 94](#_kufzn83v8fia)

[Creando las tablas de usuarios 108](#_dt0z3pm7fphn)

[Creando sistema de seguridad 114](#_fyw6ksurrgt8)

[Seguridad desde el backend 117](#_bmm1y8g4pw4d)

[Habilitando tokens en swagger 122](#_pw70wbc7c0us)

[Implementando el registro de usuarios, login & logout 122](#_7ofde8qvz4kb)

[Mejorando el registro de usuarios con drop-down-lists en cascada 129](#_lg1x3k6twyj3)

[Mejorando un poco la interfaz de usuario 135](#_5dtx0bqgihy)

[Mejorando el manejo de errores en el controlador genérico 141](#_lvh1wx6gzv2l)

[Almacenando la foto del usuario 142](#_hcsd6g4i57x2)

[Editando el usuario 148](#_7t40t0rcpt7z)

[Cambiando password del usuario 155](#_mi5moqadrrxm)

[Confirmar el registro de usuarios 158](#_15sdv0fclbbu)

[Reenviar correo de confirmación 165](#_ly7zx7b5nkcq)

[Actualización de la foto del usuario luego de editar usuario 168](#_3whwml4)

[Recuperación de contraseña 169](#_g6c78gqpi6x3)

[Agregar países al SeedBd por Script 174](#_ltmh9wsbtrtw)

[Solución a la tarea de colocar un componente de filtro genérico 175](#_4ulxwbqwfc47)

[Solución a la tarea de colocar un selector con la cantidad de registros a mostrar 177](#_f8k0p2zhoqva)

[Implementación de ventanas modales 184](#_9dx8x2ms71p4)

[Creando tablas de productos y listando productos 192](#_kry7iqqtiikl)

[Creando nuevos productos 212](#_15u14z3jjbn)

[Empezar con la edición de productos y colocar las imágenes en un carrusel 219](#_51jnqrcf31y2)

[Agregando y eliminando imágenes a los productos y terminando la edición de producto 223](#_aaoukpcjbshz)

[Borrar registros relacionados de productos 228](#_g906zopqlb6b)

[Creando el “Home” de nuestra aplicación 229](#_n9uu7zu36sj5)

[Agregando productos al carro de compras 233](#_u2j0k8o5qt1d)

[Mostrando y modificando el carro de compras 244](#_hitcbqy3ltuz)

[Procesando el pedido 253](#_aysl7yxb4s9h)

[Administrar pedidos 265](#_6h11omij4gjg)

[Ver estado de mis pedidos 275](#_ac85ymb3m00s)

[Administrar usuarios y crear nuevos administradores 276](#_duyscqwxxh18)

[Corrección para que corra el App en Mac 282](#_v99e6scjo4ek)

[Fitros por categorías 283](#_ngxdah7r7zl)

[**Creando pruebas unitarias 287**](#_tyskuvfwg2ke)

[Generales 287](#_dut7bph1a7ez)

[Categorias 287](#_jl8pi46aoxna)

[Controlador 287](#_srf7qo7a2nh7)

[Unidad de Trabajo 290](#_9bpm9y9q3fcj)

[Repositorio 291](#_x5h7mwr0uccy)

[Genérico 294](#_v8zzsdlfvzqn)

[Controlador 294](#_rrgbb5k4a6xq)

[Unidad de Trabajo 296](#_lzmj3pmggjcj)

[Repositorio 298](#_rhpvkb952buj)

[Paises 305](#_divbegzi7idj)

[Controlador 305](#_4o82mkw19mvm)

[Unidad de Trabajo 309](#_rfrjzlp57qv0)

[Repositorio 311](#_o1ft4egyi9mw)

[Estados / Departamentos 314](#_lp20vw1bap0o)

[Controlador 314](#_kvvwfk4b0qbn)

[Unidad de Trabajo 318](#_28ic8arxruo5)

[Repositorio 320](#_swgagqbgcfpk)

[Ciudades 324](#_hva4mptjqjkk)

[Controlador 324](#_jq94btwqp63o)

[Unidad de Trabajo 326](#_wq8yuyfvfnjb)

[Repositorio 328](#_c4bfp9ko4e1v)

[Pedidos 330](#_vv4zitr5brys)

[Controlador 330](#_49j6bva6arua)

[Unidad de Trabajo 334](#_5zqemnhvysck)

[Repositorio 336](#_i7qevvymxcm4)

[PedidosTemporales 341](#_trm82nm7fj4k)

[Controlador 341](#_9tuj7n4mkcm4)

[Unidad de Trabajo 345](#_oxos2vcpocr)

[Repositorio 347](#_al38y63ugitq)

[Productos 352](#_wz7ugafgk672)

[Controlador 352](#_b8v4fjwl6yve)

[Unidad de Trabajo 358](#_gwbcjijvkkhu)

[Repositorio 361](#_lg1ndcjy9pok)

[Cuentas 369](#_uf5yr636get)

[Controlador 369](#_jugfc841fbkj)

[Unidad de Trabajo 386](#_7j2x5k2zcq2n)

[Repositorio 394](#_nbg0ryi6mg8)

[Helpers 403](#_4j87en7106bh)

[OrdersHelperTest 403](#_aqqqdjekgbva)

[MailHelperTest 407](#_ix1qeg7t60u6)

[**FileStorage 410**](#_aoz23g4y3u1c)

[Services 414](#_hwrmp3s35h9p)

[ApiService 414](#_yeyn9dxdsi9t)

[Otros 417](#_hd0vcc3k4cnt)

[SeedDb 417](#_rn2hc7lf50au)

[Publicación en Azure 420](#_qs4s6guwohn7)

[Fin 428](#_9xs3czapyav5)

## 

## Links de interes

* En cada capítulo ire colocando los vídeos que explican cada tema, pero en forma general en esta lista de reproducción los encontrará todos: <https://www.youtube.com/playlist?list=PLuEZQoW9bRnRBThyGs208ZMrCYBRTvIg2>
* La URL del repositorio como lo llevo en clase es: <https://github.com/Zulu55/Orders.2024.1>
* La URL del repositorio terminando lo puede encontrar en: <https://github.com/Zulu55/Orders.2024.1.Prep>

## Matriz de funcionalidad

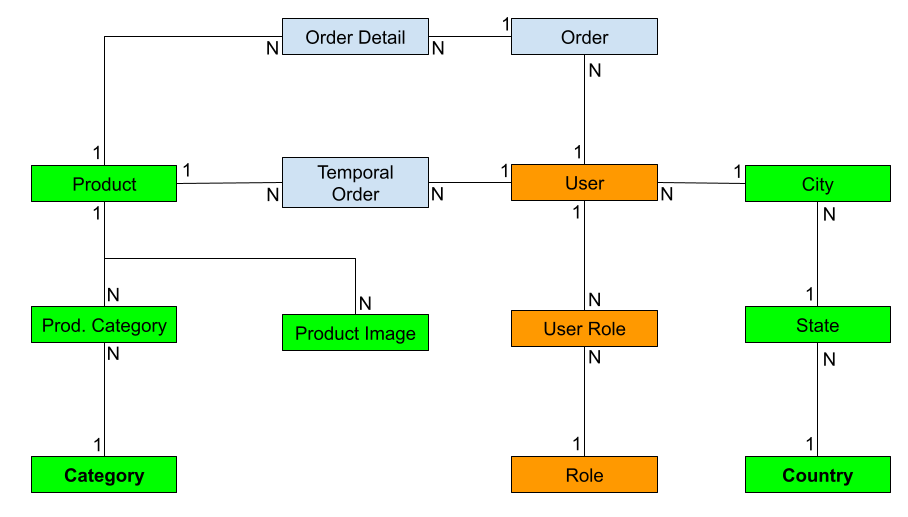
En en siguiente vídeo encontrará la explicación de esta parte, así como indicaciones de como instalar el ambiente de desarrollo: <https://www.youtube.com/watch?v=uE4VObceIeY&t=56s>

| **Funcionalidad** | **Administrador** | **Usuario** | **Anónimo** |
| --- | --- | --- | --- |
| Ingresar al sistema con email y contraseña | X | X |  |
| Editar datos de usuario (incluyendo foto de perfi) | X | X |  |
| Cambiar contraseña | X | X |  |
| Recuperar contraseña, si el usuario olvida la contraseña se le enviará un correo con un token para poder recuperar contraseña. | X | X |  |
| Administrar usuarios, el decir podrá ver todos los usuarios del sistema y crear nuevos administradores | X |  |  |
| Administras Países, Estados y Departamentos | X |  |  |
| Confirmar la cuenta con un email, cuando un usuario se de de alta, le enviaremos un correo para confirmar su cuenta. | X | X |  |
| Administrar categorías de productos, es decir, crear, modificar y borrar categorías de productos. | X |  |  |
| Administrar productos, es decir, crear, modificar y borrar productos. Donde un producto puede tener varias categorías y varias imágenes. | X |  |  |
| Ver catálogo de productos. Podrá ver todos los productos disponibles, buscarlos, hacer diferentes filtro. | X | X | X |
| Agregar productos al carro de compras, también podrá modificar e  l carro de compras. | X | X |  |
| Confirmar el pedido. | X | X |  |
| Ver el estado de mis pedidos ver como están cada uno de los pedidos echos: nuevo, en proceso, despachando, en envío, confirmado. | X | X |  |
| Administrar pedidos, el estado de cada uno de los pedidos y poder cambiar el estado de estos. | X |  |  |

## 

## Diagrama Entidad Relación

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



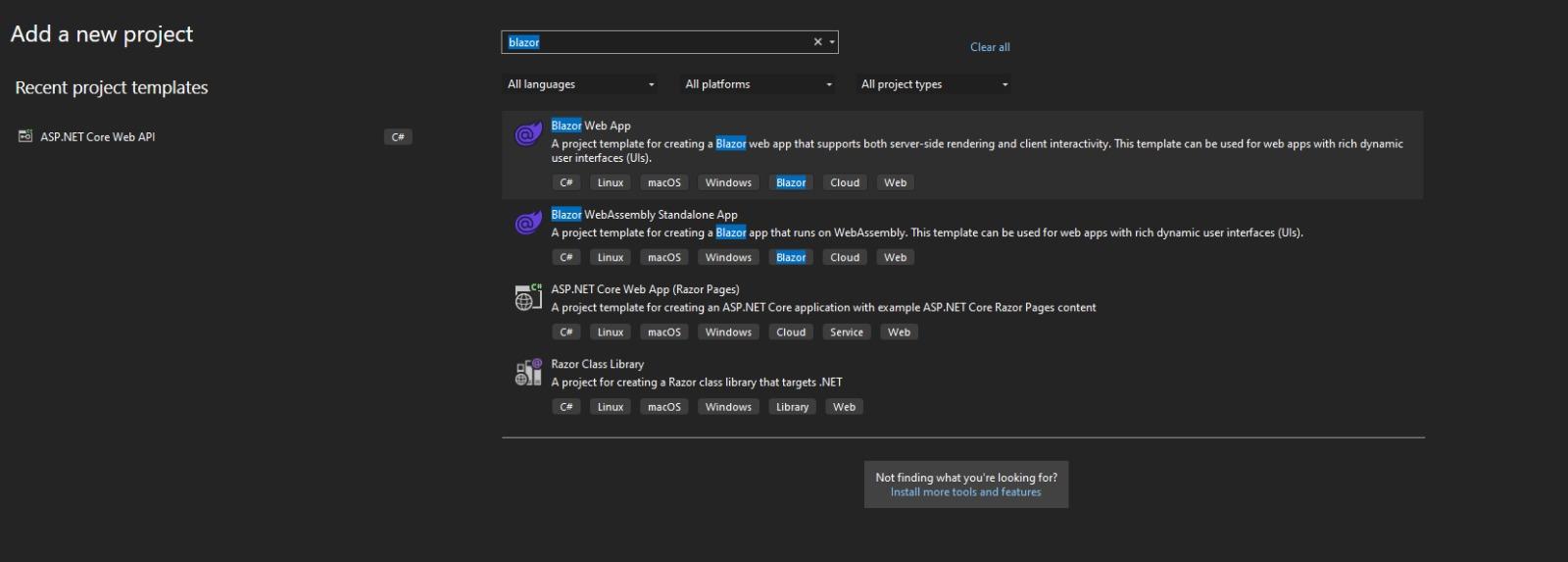
## Estructura básica de proyecto



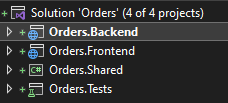
Vamos a crear esta estructura en Visual Studio (asegurese de poner todos los proyectos rn :

* Una solución en blanco llamada **Orders**.
* A la solución le agregamos un proyecto tipo: **ASP.NET Core Frontend Backend**, llamado **Orders.Backend**. (Backend)
* A la solución le agregamos un proyecto tipo: **Blazor FrontendAssembly App**, llamado **Orders. Frontend**. (Frontend)
* A la solución le agregamos un proyecto tipo: **Class Library**, llamado **Orders.Shared**.
* A la solución le agregamos un proyecto tipo: **MS Test**, llamado **Orders.Tests**.

**Nota**: en algunas instalaciones de Visual Studio no lo puedes ver como **Blazor FrontendAssembly App** sino como **Blazor WebAssembly Standalone App**, usa esta.



Debe quedar algo como esto:



Hacemos el primer commit en nuestro repositorio.

## Creando la base de datos con Entity Framework

(Explicado en el vídeo: <https://www.youtube.com/watch?v=BT7cZScDwvk>)



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

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

using System.ComponentModel.DataAnnotations;

namespace Orders.Shared.Entities

{

public class Country

{

public int Id { get; set; }

[Display(Name = "País")]

[MaxLength(100, ErrorMessage = "El campo {0} no puede tener más de {1} caracteres.")]

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

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

}

}

1. Actualizar Nuggets del proyecto **Backend**.
2. En el proyecto **Backend** creamos la carpeta **Data** y dentro de esta la clase **DataContext**:

using Microsoft.EntityFrameworkCore;

using Orders.Shared.Entities;

namespace Orders.Backend.Data

{

public class DataContext : DbContext

{

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

{

}

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

protected override void OnModelCreating(ModelBuilder modelBuilder)

{

base.OnModelCreating(modelBuilder);

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

}

}

}

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

{

"ConnectionStrings": {

"DockerConnection": "Data Source=.;Initial Catalog=Orders;User ID={Your user};Password={Your password};Connect Timeout=30;Encrypt=False;TrustServerCertificate=False;ApplicationIntent=ReadWrite;MultiSubnetFailover=False",

"LocalConnection": "Server=(localdb)\\MSSQLLocalDB;Database=Orders;Trusted\_Connection=True;MultipleActiveResultSets=true"

},

"Logging": {

"LogLevel": {

"Default": "Information",

"Microsoft.AspNetCore": "Warning"

}

},

"AllowedHosts": "\*"

}

**Nota:** dejo los 2 string de conexión para que use el que más le convenga en el vídeo de clase explico mejor cual utilizar en cada caso.

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

Microsoft.EntityFrameworkCore.SqlServer

Microsoft.EntityFrameworkCore.Tools

1. Configurar la inyección del data context en el **Program** del proyecto **Backend**:

builder.Services.AddSwaggerGen();

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

var app = builder.Build();

1. Correr los comandos:

add-migration InitialDb

update-database

1. Hacemos nuestro segundo **Commit**.

## Creando el primer controlador

(Explicado en el vídeo: <https://www.youtube.com/watch?v=1XHK0dxabco>)

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

using Microsoft.AspNetCore.Mvc;

using Microsoft.EntityFrameworkCore;

using Orders.Backend.Data;

using Orders.Shared.Entites;

namespace Orders.Backend.Controllers

{

[ApiController]

[Route("api/[controller]")]

public class CountriesController : ControllerBase

{

private readonly DataContext \_context;

public CountriesController(DataContext context)

{

\_context = context;

}

[HttpGet]

public async Task<IActionResult> GetAsync()

{

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

}

[HttpGet("{id}")]

public async Task<IActionResult> GetAsync(int id)

{

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

if (country == null)

{

return NotFound();

}

return Ok(country);

}

[HttpPost]

public async Task<IActionResult> PostAsync(Country country)

{

\_context.Add(country);

await \_context.SaveChangesAsync();

return Ok(country);

}

[HttpDelete("{id}")]

public async Task<IActionResult> DeleteAsync(int id)

{

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

if (country == null)

{

return NotFound();

}

\_context.Remove(country);

await \_context.SaveChangesAsync();

return NoContent();

}

[HttpPut]

public async Task<IActionResult> PutAsync(Country country)

{

\_context.Update(country);

await \_context.SaveChangesAsync();

return Ok(country);

}

}

}

1. Agregamos estas líneas al **Program** del proyecto **Backend** 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

(Explicado en los vídeos: <https://www.youtube.com/watch?v=4kElV4PXaZk&list=PLuEZQoW9bRnRBThyGs208ZMrCYBRTvIg2&index=5>, <https://www.youtube.com/watch?v=DMj3nvvQ2T4&list=PLuEZQoW9bRnRBThyGs208ZMrCYBRTvIg2&index=5>, <https://www.youtube.com/watch?v=S6jGtH3HoWg&list=PLuEZQoW9bRnRBThyGs208ZMrCYBRTvIg2&index=6>)

1. Le agregamos este nuget al **Fronted**: **System.Net.Http**.
2. Ahora vamos listar y crear países por la interfaz Frontend. Primero configuramos en el proyecto  **Frontend** la dirección por la cual sale nuestra **Backend**:

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

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

using System.Net;

namespace Orders.Frontend.Repositories

{

public class HttpResponseWrapper<T>

{

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

{

Response = response;

Error = error;

HttpResponseMessage = httpResponseMessage;

}

public T? Response { get; }

public bool Error { get; }

public HttpResponseMessage HttpResponseMessage { get; }

public async Task<string?> GetErrorMessageAsync()

{

if (!Error)

{

return null;

}

var statusCode = HttpResponseMessage.StatusCode;

if (statusCode == HttpStatusCode.NotFound)

{

return "Recurso no encontrado.";

}

if (statusCode == HttpStatusCode.BadRequest)

{

return await HttpResponseMessage.Content.ReadAsStringAsync();

}

if (statusCode == HttpStatusCode.Unauthorized)

{

return "Tienes que estar logueado para ejecutar esta operación.";

}

if (statusCode == HttpStatusCode.Forbidden)

{

return "No tienes permisos para hacer esta operación.";

}

return "Ha ocurrido un error inesperado.";

}

}

}

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

namespace Orders.Frontend.Repositories

{

public interface IRepository

{

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

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

Task<HttpResponseWrapper<TActionResponse>> PostAsync<T, TActionResponse>(string url, T model);

}

}

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

using System.Text;

using System.Text.Json;

namespace Orders.Frontend.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>> GetAsync<T>(string url)

{

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

if (responseHttp.IsSuccessStatusCode)

{

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

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

}

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

}

public async Task<HttpResponseWrapper<object>> PostAsync<T>(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);

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

}

public async Task<HttpResponseWrapper<TActionResponse>> PostAsync<T, TActionResponse>(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<TActionResponse>(responseHttp);

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

}

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

}

private async Task<T> UnserializeAnswer<T>(HttpActionResponseMessage responseHttp)

{

var response = await responseHttp.Content.ReadAsStringAsync();

return JsonSerializer.Deserialize<T>(response, \_jsonDefaultOptions)!;

}

}

}

8

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

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

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

await builder.Build().RunAsync();

1. En el proyecto del **Frontend** en la carpeta **Shared** creamos el componente genérico **GenericList**:

@typeparam Titem

@if(MyList is null)

{

@if(Loading is null)

{

<div class="d-flex justify-content-center 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; }

[EditorRequired]

[Parameter]

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

[EditorRequired]

[Parameter]

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

}

1. En el proyecto **Frontend** 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.GetAsync<List<Country>>("api/countries");

Countries = responseHppt.ActionResponse!;

}

}

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

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

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

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

</NavLink>

</div>

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

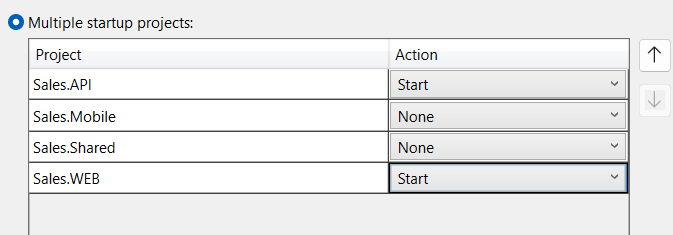
<NavLink class="nav-link" href="countries">

<span class="bi bi-list-check" aria-hidden="true"></span> Ciudades

</NavLink>

</div>

1. Configuramos nuestro proyecto para que inicie al mismo tiempo el proyecto **Backend** y el proyecto  **Frontend**:



1. Probamos.
2. Para darle un mejor manejo al código es mejor separar el código HTLM y el código C# en archivos separados. De esta manera funciona mejor el “refactor” y herramientas de autocompletación y código limpio.
3. Modificamos el **CountriesIndex.razor** para que queso así:

@page "/countries"

<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>

1. Creamos el **CountriesIndex.razor.cs**:

using Microsoft.AspNetCore.Components;

using Orders.Frontend.Repositories;

using Orders.Shared.Entities;

namespace Orders.Frontend.Pages.Countries

{

public partial class CountriesIndex

{

[Inject] private IRepository Repository { get; set; } = null!;

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

protected override async Task OnInitializedAsync()

{

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

Countries = responseHppt.ActionResponse!;

}

}

}

1. Lo mismo para el **GenericList.razor**:

@typeparam Titem

@if (MyList is null)

{

@if (Loading is null)

{

<div class="d-flex justify-content-center 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; }

[EditorRequired]

[Parameter]

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

}

1. Y creamos el **GenericList.razor.cs**:

using Microsoft.AspNetCore.Components;

namespace Orders.Frontend.Shared

{

public partial class GenericList<Titem>

{

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

}

}

1. Probamos y hacemos nuestro commit.

## Completando las acciones de crear, editar y borrar países

(Explicado en el vídeo: <https://www.youtube.com/watch?v=aSM5RjqBwnE&list=PLuEZQoW9bRnRBThyGs208ZMrCYBRTvIg2&index=7&t=12s>)

**Nota**: antes de empezar:

* Ver la nota del performance del **AsNoTracking()**.
* Colocar el componente **Loading**.

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

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

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

Task<HttpResponseWrapper<TActionResponse>> PutAsync<T, TActionResponse>(string url, T model);

1. Y los implementamos la clase **Repository** (antes renombramos el **UnserializeAnswer** a **UnserializeAnswerAsync** que nos habia quedado mal).

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

{

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

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

}

public async Task<HttpResponseWrapper<object>> PutAsync<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<TActionResponse>> PutAsync<T, TActionResponse>(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 UnserializeAnswerAsync<TActionResponse>(responseHttp);

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

}

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

}

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

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

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

</body>

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

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

builder.Services.AddSweetAlert2();

1. Creamos el componente gérico **Loading.razor**:

<div class="d-flex justify-content-center align-items-center">

<img src="https://media.tenor.com/1qrYT711uEoAAAAC/cargando.gif">

</div>

1. Modificamos el **GenericList.razor**:

@if (Loading is null)

{

<Loading/>

}

(Explicado en el vídeo:<https://www.youtube.com/watch?v=mKpkMDl85Ns&list=PLuEZQoW9bRnRBThyGs208ZMrCYBRTvIg2&index=8>)

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

using CurrieTechnologies.Razor.SweetAlert2;

using Microsoft.AspNetCore.Components.Forms;

using Microsoft.AspNetCore.Components;

using Orders.Shared.Entities;

using Microsoft.AspNetCore.Components.Routing;

namespace Orders.Frontend.Pages.Countries

{

public partial class CountryForm

{

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; }

[Inject] private SweetAlertService SweetAlertService { get; set; } = null!;

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

private async Task OnBeforeInternalNavigation(LocationChangingContext context)

{

var formWasEdited = editContext.IsModified();

if (!formWasEdited || FormPostedSuccessfully)

{

return;

}

var result = await SweetAlertService.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. Modificamos el **CountryForm.razor**:

<NavigationLock OnBeforeInternalNavigation="OnBeforeInternalNavigation"/>

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

<DataAnnotationsValidator />

<div class="mb-3">

<label>País:</label>

<div>

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

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

</div>

</div>

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

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

</EditForm>

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

using CurrieTechnologies.Razor.SweetAlert2;

using Microsoft.AspNetCore.Components;

using Orders.Frontend.Repositories;

using Orders.Shared.Entities;

namespace Orders.Frontend.Pages.Countries

{

public partial class CountryCreate

{

private CountryForm? countryForm;

[Inject] private IRepository Repository { get; set; } = null!;

[Inject] private NavigationManager NavigationManager { get; set; } = null!;

[Inject] private SweetAlertService SweetAlertService { get; set; } = null!;

private Country country = new();

private async Task CreateAsync()

{

var responseHttp = await Repository.PostAsync("/api/countries", country);

if (responseHttp.Error)

{

var message = await responseHttp.GetErrorMessageAsync();

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

return;

}

Return();

var toast = SweetAlertService.Mixin(new SweetAlertOptions

{

Toast = true,

Position = SweetAlertPosition.BottomEnd,

ShowConfirmButton = true,

Timer = 3000

});

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

}

private void Return()

{

countryForm!.FormPostedSuccessfully = true;

NavigationManager.NavigateTo("/countries");

}

}

}

1. Modificamos el **CountryCreate.razor**:

@page "/countries/create"

<h3>Crear País</h3>

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

1. Probamos la creación de países por interfaz. **Asegurate que luego de correr el proyecto, presionar Ctrl + F5, para que te tome los cambios**.

(Explicado en los vídeos: <https://www.youtube.com/watch?v=HEmtDVwm5pQ&list=PLuEZQoW9bRnRBThyGs208ZMrCYBRTvIg2&index=9> y <https://www.youtube.com/watch?v=NfDSDN5rRss&list=PLuEZQoW9bRnRBThyGs208ZMrCYBRTvIg2&index=10>)

1. Ahora creamos el componente **CountryEdit.razor.cs**:

using CurrieTechnologies.Razor.SweetAlert2;

using Microsoft.AspNetCore.Components;

using Orders.Frontend.Repositories;

using Orders.Shared.Entities;

namespace Orders.Frontend.Pages.Countries

{

public partial class CountryEdit

{

private Country? country;

private CountryForm? countryForm;

[Inject] private NavigationManager NavigationManager { get; set; } = null!;

[Inject] private IRepository Repository { get; set; } = null!;

[Inject] private SweetAlertService SweetAlertService { get; set; } = null!;

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

protected override async Task OnInitializedAsync()

{

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

if (responseHttp.Error)

{

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

{

NavigationManager.NavigateTo("countries");

}

else

{

var messageError = await responseHttp.GetErrorMessageAsync();

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

}

}

else

{

country = responseHttp.Response;

}

}

private async Task EditAsync()

{

var responseHttp = await Repository.PutAsync("api/countries", country);

if (responseHttp.Error)

{

var mensajeError = await responseHttp.GetErrorMessageAsync();

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

return;

}

Return();

var toast = SweetAlertService.Mixin(new SweetAlertOptions

{

Toast = true,

Position = SweetAlertPosition.BottomEnd,

ShowConfirmButton = true,

Timer = 3000

});

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

}

private void Return()

{

countryForm!.FormPostedSuccessfully = true;

NavigationManager.NavigateTo("countries");

}

}

}

1. Modificamos el **CountryEdit.razor**:

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

<h3>Editar País</h3>

@if (country is null)

{

<Loading/>

}

else

{

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

}

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

using CurrieTechnologies.Razor.SweetAlert2;

using Microsoft.AspNetCore.Components;

using Orders.Frontend.Repositories;

using Orders.Shared.Entities;

namespace Orders.Frontend.Pages.Countries

{

public partial class CountriesIndex

{

[Inject] private NavigationManager NavigationManager { get; set; } = null!;

[Inject] private SweetAlertService SweetAlertService { get; set; } = null!;

[Inject] private IRepository Repository { get; set; } = null!;

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

protected override async Task OnInitializedAsync()

{

await LoadAsync();

}

private async Task LoadAsync()

{

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

if (responseHppt.Error)

{

var message = await responseHppt.GetErrorMessageAsync();

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

return;

}

Countries = responseHppt.Response!;

}

private async Task DeleteAsync(Country country)

{

var result = await SweetAlertService.FireAsync(new SweetAlertOptions

{

Title = "Confirmación",

Text = $"¿Esta seguro que quieres borrar el país: {country.Name}?",

Icon = SweetAlertIcon.Question,

ShowCancelButton = true

});

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

if (confirm)

{

return;

}

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

if (responseHTTP.Error)

{

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

{

NavigationManager.NavigateTo("/");

}

else

{

var mensajeError = await responseHTTP.GetErrorMessageAsync();

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

}

return;

}

await LoadAsync();

var toast = SweetAlertService.Mixin(new SweetAlertOptions

{

Toast = true,

Position = SweetAlertPosition.BottomEnd,

ShowConfirmButton = true,

Timer = 3000

});

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

}

}

}

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

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

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

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

## 

## Creando controladores genéricos y solucionando el problema de registros duplicados

(Explicado en los vídeos: <https://www.youtube.com/watch?v=uo2CyjYzg5Y&list=PLuEZQoW9bRnRBThyGs208ZMrCYBRTvIg2&index=11> y <https://www.youtube.com/watch?v=Ui1pKVbQ2cs&list=PLuEZQoW9bRnRBThyGs208ZMrCYBRTvIg2&index=12>) Material complementario: <https://www.netmentor.es/entrada/repository-pattern>



1. Creamos la entidad **Category**:

using System.ComponentModel.DataAnnotations;

namespace Orders.Shared.Entities

{

public class Category

{

public int Id { get; set; }

[Display(Name = "Categoría")]

[MaxLength(100, ErrorMessage = "El campo {0} no puede tener más de {1} caracteres.")]

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

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

}

}

1. Modificamos el **DataContext**:

using Microsoft.EntityFrameworkCore;

using Orders.Shared.Entities;

namespace Orders.Backend.Data

{

public class DataContext : DbContext

{

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

{

}

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

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

protected override void OnModelCreating(ModelBuilder modelBuilder)

{

base.OnModelCreating(modelBuilder);

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

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

}

}

}

1. Agregamos la migración y actualizamos la BD.
2. En **Shared** creamos la carpeta **Responses** y dentro de esta la clase **ActionResponse**:

namespace Orders.Shared.Responses

{

public class ActionResponse<T>

{

public bool WasSuccess { get; set; }

public string? Message { get; set; }

public T? Result { get; set; }

}

}

1. En **Backend** creamos la carpeta **Repositories/Interfaces** y dentro de esta la interfaz **IGenericRepository**:

using Orders.Shared.Responses;

namespace Orders.Backend.Repositories.Interfaces

{

public interface IGenericRepository<T> where T : class

{

Task<ActionResponse<T>> GetAsync(int id);

Task<ActionResponse<IEnumerable<T>>> GetAsync();

Task<ActionResponse<T>> AddAsync(T entity);

Task<ActionResponse<T>> DeleteAsync(int id);

Task<ActionResponse<T>> UpdateAsync(T entity);

}

}

1. Creanis la carpeta **UnitsOfWork/Interfaces** y dentro de esta creamos la interfaz **IGenericUnitOfWork**:

using Orders.Shared.Responses;

namespace Orders.Backend.UnitsOfWork.Interfaces

{

public interface IGenericUnitOfWork<T> where T : class

{

Task<ActionResponse<IEnumerable<T>>> GetAsync();

Task<ActionResponse<T>> AddAsync(T model);

Task<ActionResponse<T>> UpdateAsync(T model);

Task<ActionResponse<T>> DeleteAsync(int id);

Task<ActionResponse<T>> GetAsync(int id);

}

}

1. En **Backend** creamos la carpeta **Repositories/Implementations** y dentro de esta la clase **GenericRepository**:

using Microsoft.EntityFrameworkCore;

using Orders.Backend.Data;

using Orders.Backend.Repositories.Interfaces;

using Orders.Shared.Responses;

namespace Orders.Backend.Repositories.Implementations

{

public class GenericRepository<T> : IGenericRepository<T> where T : class

{

private readonly DataContext \_context;

private readonly DbSet<T> \_entity;

public GenericRepository(DataContext context)

{

\_context = context;

\_entity = context.Set<T>();

}

public virtual async Task<ActionResponse<T>> AddAsync(T entity)

{

\_context.Add(entity);

try

{

await \_context.SaveChangesAsync();

return new ActionResponse<T>

{

WasSuccess = true,

Result = entity

};

}

catch (DbUpdateException)

{

return DbUpdateExceptionActionResponse();

}

catch (Exception exception)

{

return ExceptionActionResponse(exception);

}

}

public virtual async Task<ActionResponse<T>> DeleteAsync(int id)

{

var row = await \_entity.FindAsync(id);

if (row == null)

{

return new ActionResponse<T>

{

WasSuccess = false,

Message = "Registro no encontrado"

};

}

try

{

\_entity.Remove(row);

await \_context.SaveChangesAsync();

return new ActionResponse<T>

{

WasSuccess = true,

};

}

catch

{

return new ActionResponse<T>

{

WasSuccess = false,

Message = "No se puede borrar, porque tiene registros relacionados"

};

}

}

public virtual async Task<ActionResponse<T>> GetAsync(int id)

{

var row = await \_entity.FindAsync(id);

if (row != null)

{

return new ActionResponse<T>

{

WasSuccess = true,

Result = row

};

}

return new ActionResponse<T>

{

WasSuccess = false,

Message = "Registro no encontrado"

};

}

public virtual async Task<ActionResponse<IEnumerable<T>>> GetAsync()

{

return new ActionResponse<IEnumerable<T>>

{

WasSuccess = true,

Result = await \_entity.ToListAsync()

};

}

public virtual async Task<ActionResponse<T>> UpdateAsync(T entity)

{

try

{

\_context.Update(entity);

await \_context.SaveChangesAsync();

return new ActionResponse<T>

{

WasSuccess = true,

Result = entity

};

}

catch (DbUpdateException)

{

return DbUpdateExceptionActionResponse();

}

catch (Exception exception)

{

return ExceptionActionResponse(exception);

}

}

private ActionResponse<T> ExceptionActionResponse(Exception exception)

{

return new ActionResponse<T>

{

WasSuccess = false,

Message = exception.Message

};

}

private ActionResponse<T> DbUpdateExceptionActionResponse()

{

return new ActionResponse<T>

{

WasSuccess = false,

Message = "Ya existe el registro que estas intentando crear."

};

}

}

}

1. En **Backend** creamos la carpeta **UnitsOfWork/Implementations** y dentro de esta la clase **GenericUnitOfWork**:

using Orders.Backend.Repositories.Interfaces;

using Orders.Backend.UnitsOfWork.Interfaces;

using Orders.Shared.Responses;

namespace Orders.Backend.UnitsOfWork.Implementations

{

public class GenericUnitOfWork<T> : IGenericUnitOfWork<T> where T : class

{

private readonly IGenericRepository<T> \_repository;

public GenericUnitOfWork(IGenericRepository<T> repository)

{

\_repository = repository;

}

public virtual async Task<ActionResponse<T>> AddAsync(T model) => await \_repository.AddAsync(model);

public virtual async Task<ActionResponse<T>> DeleteAsync(int id) => await \_repository.DeleteAsync(id);

public virtual async Task<ActionResponse<IEnumerable<T>>> GetAsync() => await \_repository.GetAsync();

public virtual async Task<ActionResponse<T>> GetAsync(int id) => await \_repository.GetAsync(id);

public virtual async Task<ActionResponse<T>> UpdateAsync(T model) => await \_repository.UpdateAsync(model);

}

}

1. En **Backend** en la carpeta **Controllers** y dentro de esta la clase **GenericController**:

using Microsoft.AspNetCore.Mvc;

using Orders.Backend.UnitsOfWork.Interfaces;

namespace Orders.Backend.Controllers

{

public class GenericController<T> : Controller where T : class

{

private readonly IGenericUnitOfWork<T> \_unitOfWork;

public GenericController(IGenericUnitOfWork<T> unitOfWork)

{

\_unitOfWork = unitOfWork;

}

[HttpGet]

public virtual async Task<IActionResult> GetAsync()

{

var action = await \_unitOfWork.GetAsync();

if (action.WasSuccess)

{

return Ok(action.Result);

}

return BadRequest();

}

[HttpGet("{id}")]

public virtual async Task<IActionResult> GetAsync(int id)

{

var action = await \_unitOfWork.GetAsync(id);

if (action.WasSuccess)

{

return Ok(action.Result);

}

return NotFound();

}

[HttpPost]

public virtual async Task<IActionResult> PostAsync(T model)

{

var action = await \_unitOfWork.AddAsync(model);

if (action.WasSuccess)

{

return Ok(action.Result);

}

return BadRequest(action.Message);

}

[HttpPut]

public virtual async Task<IActionResult> PutAsync(T model)

{

var action = await \_unitOfWork.UpdateAsync(model);

if (action.WasSuccess)

{

return Ok(action.Result);

}

return BadRequest(action.Message);

}

[HttpDelete("{id}")]

public virtual async Task<IActionResult> DeleteAsync(int id)

{

var action = await \_unitOfWork.DeleteAsync(id);

if (action.WasSuccess)

{

return NoContent();

}

return BadRequest(action.Message);

}

}

}

1. Configuramos las inyecciones en el **Program** del **Backend**:

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

builder.Services.AddScoped(typeof(IGenericUnitOfWork<>), typeof(GenericUnitOfWork<>));

builder.Services.AddScoped(typeof(IGenericRepository<>), typeof(GenericRepository<>));

1. Reemplazamos el **CountriesController** por esto:

using Microsoft.AspNetCore.Mvc;

using Orders.Backend.UnitsOfWork.Interfaces;

using Orders.Shared.Entities;

namespace Orders.Backend.Controllers

{

[ApiController]

[Route("api/[controller]")]

public class CountriesController : GenericController<Country>

{

public CountriesController(IGenericUnitOfWork<Country> unit) : base(unit)

{

}

}

}

1. Creamos el **CategoriesController**:

using Microsoft.AspNetCore.Mvc;

using Orders.Backend.UnitsOfWork.Interfaces;

using Orders.Shared.Entities;

namespace Orders.Backend.Controllers

{

[ApiController]

[Route("api/[controller]")]

public class CategoriesController : GenericController<Category>

{

public CategoriesController(IGenericUnitOfWork<Category> unit) : base(unit)

{

}

}

}

1. Probamos.

## Organizar íconos del Home

(Explicado en el vídeo: <https://www.youtube.com/watch?v=h8qVvuu06_M&list=PLuEZQoW9bRnRBThyGs208ZMrCYBRTvIg2&index=14>)

1. Antes de empezar con categorías vamos a organizar los íconos. Para esto vamos a agregar el CDN de **Bootstrap Icons** a nuestro proyecto. Nos apoyamos de la página <https://www.bootstrapcdn.com/bootstrapicons/> para obtener el CDN, luego lo usamos en el **index.html**:

…

<link href="Orders.Frontend.styles.css" rel="stylesheet" />

<link href="https://cdn.jsdelivr.net/npm/bootstrap-icons@1.11.1/font/bootstrap-icons.css"

rel="stylesheet"

integrity="sha384-4LISF5TTJX/fLmGSxO53rV4miRxdg84mZsxmO8Rx5jGtp/LbrixFETvWa5a6sESd"

crossorigin="anonymous">

</head>

1. Buscamos un ícono adecuado en <https://icons.getbootstrap.com/> yo use **globe-americas**, y lo usamos en la definición del menú en **NavMenu.razor**:

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

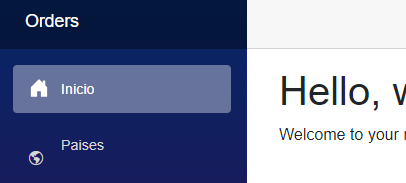
<NavLink class="nav-link" href="/countries">

<span class="bi bi-globe-americas" aria-hidden="true"></span> Paises

</NavLink>

</div>

1. Esta es la forma de usar cualquier ícono de esta libreria, pero al probar, visualmente no se ve bien, porque se ve el ícono más pequeño que el de inicio y se ve desalineado (no olvides presionar el Ctrl + F5, luego de correr para que te refresque el uso del CDN):



1. Esto es porque esta página tiene sus propios estilos **NavMenu.razor.css**, y debemo de incluir un nuevo estilo con la definición SVG del ícono. Para esto vamos tomar la definición SVG del ícono tomada de la misma página de definición del ícono <https://icons.getbootstrap.com/icons/globe-americas/> y lo agregamos a la hoja de estilos, adicionalmente aprovechemos y separemos el código C# del menú, para seguir las buenas práctias:

.bi-house-door-fill-nav-menu {

background-image: url("data:image/svg+xml,%3Csvg xmlns='http://www.w3.org/2000/svg' width='16' height='16' fill='white' class='bi bi-house-door-fill' viewBox='0 0 16 16'%3E%3Cpath d='M6.5 14.5v-3.505c0-.245.25-.495.5-.495h2c.25 0 .5.25.5.5v3.5a.5.5 0 0 0 .5.5h4a.5.5 0 0 0 .5-.5v-7a.5.5 0 0 0-.146-.354L13 5.793V2.5a.5.5 0 0 0-.5-.5h-1a.5.5 0 0 0-.5.5v1.293L8.354 1.146a.5.5 0 0 0-.708 0l-6 6A.5.5 0 0 0 1.5 7.5v7a.5.5 0 0 0 .5.5h4a.5.5 0 0 0 .5-.5Z'/%3E%3C/svg%3E");

}

.bi-globe-americas-fill-nav-menu {

background-image: url("data:image/svg+xml,%3Csvg xmlns='http://www.w3.org/2000/svg' width='16' height='16' fill='white' class='bi bi-house-door-fill' viewBox='0 0 16 16'%3E%3Cpath d='M8 0a8 8 0 1 0 0 16A8 8 0 0 0 8 0M2.04 4.326c.325 1.329 2.532 2.54 3.717 3.19.48.263.793.434.743.484q-.121.12-.242.234c-.416.396-.787.749-.758 1.266.035.634.618.824 1.214 1.017.577.188 1.168.38 1.286.983.082.417-.075.988-.22 1.52-.215.782-.406 1.48.22 1.48 1.5-.5 3.798-3.186 4-5 .138-1.243-2-2-3.5-2.5-.478-.16-.755.081-.99.284-.172.15-.322.279-.51.216-.445-.148-2.5-2-1.5-2.5.78-.39.952-.171 1.227.182.078.099.163.208.273.318.609.304.662-.132.723-.633.039-.322.081-.671.277-.867.434-.434 1.265-.791 2.028-1.12.712-.306 1.365-.587 1.579-.88A7 7 0 1 1 2.04 4.327Z'/%3E%3C/svg%3E");

}

.bi-plus-square-fill-nav-menu {

background-image: url("data:image/svg+xml,%3Csvg xmlns='http://www.w3.org/2000/svg' width='16' height='16' fill='white' class='bi bi-plus-square-fill' viewBox='0 0 16 16'%3E%3Cpath d='M2 0a2 2 0 0 0-2 2v12a2 2 0 0 0 2 2h12a2 2 0 0 0 2-2V2a2 2 0 0 0-2-2H2zm6.5 4.5v3h3a.5.5 0 0 1 0 1h-3v3a.5.5 0 0 1-1 0v-3h-3a.5.5 0 0 1 0-1h3v-3a.5.5 0 0 1 1 0z'/%3E%3C/svg%3E");

}

1. Ahora utilizamos esta nueva clase e estilo del ícono **bi-globe-americas-fill-nav-menu** en el menú:

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

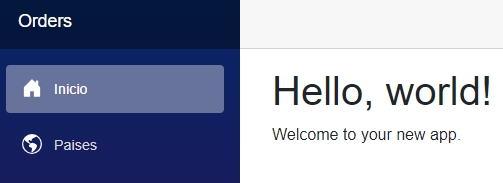
<NavLink class="nav-link" href="/countries">

<span class="bi bi-globe-americas-fill-nav-menu" aria-hidden="true"></span> Paises

</NavLink>

</div>

1. Probamos de nuevo y ahora vemos que visualmente se ve mucho mejor:



1. Esto lo debemos hacer para cualquier ícono que tengas en el menú. Ahora si vamos con las categorías.
2. Ya que estamo con los íconos busquemos un ícono para categorías yo voy a usar **bi-list-check** y como es un ícono que va en el menú obtenemos el SVG y prodecemos a colocarlo en la hora de estilos:

.bi-list-check-fill-nav-menu {

background-image: url("data:image/svg+xml,%3Csvg xmlns='http://www.w3.org/2000/svg' width='16' height='16' fill='white' class='bi bi-house-door-fill' viewBox='0 0 16 16'%3E%3Cpath d='M5 11.5a.5.5 0 0 1 .5-.5h9a.5.5 0 0 1 0 1h-9a.5.5 0 0 1-.5-.5m0-4a.5.5 0 0 1 .5-.5h9a.5.5 0 0 1 0 1h-9a.5.5 0 0 1-.5-.5m0-4a.5.5 0 0 1 .5-.5h9a.5.5 0 0 1 0 1h-9a.5.5 0 0 1-.5-.5M3.854 2.146a.5.5 0 0 1 0 .708l-1.5 1.5a.5.5 0 0 1-.708 0l-.5-.5a.5.5 0 1 1 .708-.708L2 3.293l1.146-1.147a.5.5 0 0 1 .708 0m0 4a.5.5 0 0 1 0 .708l-1.5 1.5a.5.5 0 0 1-.708 0l-.5-.5a.5.5 0 1 1 .708-.708L2 7.293l1.146-1.147a.5.5 0 0 1 .708 0m0 4a.5.5 0 0 1 0 .708l-1.5 1.5a.5.5 0 0 1-.708 0l-.5-.5a.5.5 0 0 1 .708-.708l.146.147 1.146-1.147a.5.5 0 0 1 .708 0'/%3E%3C/svg%3E");

}

1. Incluimos la nueva entrada en el menú y probamos:

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

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

<span class="bi bi-house-door-fill-nav-menu" aria-hidden="true"></span> Inicio

</NavLink>

</div>

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

<NavLink class="nav-link" href="/categories">

<span class="bi bi-list-check-fill-nav-menu" aria-hidden="true"></span> Categorías

</NavLink>

</div>

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

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

<span class="bi bi-globe-americas-fill-nav-menu" aria-hidden="true"></span> Paises

</NavLink>

</div>

## CRUD de categorías

(Explicado en el vídeo: <https://www.youtube.com/watch?v=v13wBfaqYNI&list=PLuEZQoW9bRnRBThyGs208ZMrCYBRTvIg2&index=13>)

1. En el Frontend creamos la carpeta **Categories** dentro de **Pages**, dentro de esta creamos el **CategoriesIndex.razor** y **CategoriesIndex.razor.cs**:

using CurrieTechnologies.Razor.SweetAlert2;

using Microsoft.AspNetCore.Components;

using Orders.Frontend.Repositories;

using Orders.Shared.Entities;

using System.Net;

namespace Orders.Frontend.Pages.Categories

{

public partial class CategoriesIndex

{

[Inject] private IRepository repository { get; set; } = null!;

[Inject] private SweetAlertService sweetAlertService { get; set; } = null!;

[Inject] private NavigationManager navigationManager { get; set; } = null!;

public List<Category>? Categories { get; set; }

protected override async Task OnInitializedAsync()

{

await LoadAsync();

}

private async Task LoadAsync()

{

var responseHttp = await repository.GetAsync<List<Category>>("api/categories");

if (responseHttp.Error)

{

var message = await responseHttp.GetErrorMessageAsync();

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

return;

}

Categories = responseHttp.Response;

}

private async Task DeleteAsycn(Category category)

{

var result = await sweetAlertService.FireAsync(new SweetAlertOptions

{

Title = "Confirmación",

Text = $"¿Estas seguro de querer borrar la categoría: {category.Name}?",

Icon = SweetAlertIcon.Question,

ShowCancelButton = true,

});

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

if (confirm)

{

return;

}

var responseHttp = await repository.DeleteAsync<Category>($"api/categories/{category.Id}");

if (responseHttp.Error)

{

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

{

navigationManager.NavigateTo("/categories");

}

else

{

var mensajeError = await responseHttp.GetErrorMessageAsync();

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

}

return;

}

await LoadAsync();

var toast = sweetAlertService.Mixin(new SweetAlertOptions

{

Toast = true,

Position = SweetAlertPosition.BottomEnd,

ShowConfirmButton = true,

Timer = 3000

});

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

}

}

}

1. Luego creamos el **CategoriesIndex.razor**:

@page "/categories"

<h3>Categorías</h3>

<div class="mb-3">

<a class="btn btn-primary" href="/categories/create">Nueva Categoría</a>

</div>

<GenericList MyList="Categories">

<Body>

<table class="table table-striped">

<thead>

<tr>

<th>Categoría</th>

<th></th>

</tr>

</thead>

<tbody>

@foreach (var category in Categories!)

{

<tr>

<td>@category.Name</td>

<td>

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

<button @onclick=@(() => DeleteAsycn(category)) class="btn btn-danger">Borrar</button>

</td>

</tr>

}

</tbody>

</table>

</Body>

</GenericList>

1. Probamos lo que llevamos.
2. Luego creamos el **CategoryForm.razor** y **CategoryForm.razor.cs**:

using CurrieTechnologies.Razor.SweetAlert2;

using Microsoft.AspNetCore.Components.Forms;

using Microsoft.AspNetCore.Components.Routing;

using Microsoft.AspNetCore.Components;

using Orders.Shared.Entities;

namespace Orders.Frontend.Pages.Categories

{

public partial class CategoryForm

{

private EditContext editContext = null!;

[EditorRequired, Parameter] public Category Category { get; set; } = null!;

[EditorRequired, Parameter] public EventCallback OnValidSubmit { get; set; }

[EditorRequired, Parameter] public EventCallback ReturnAction { get; set; }

[Inject] public SweetAlertService SweetAlertService { get; set; } = null!;

public bool FormPostedSuccessfully { get; set; }

protected override void OnInitialized()

{

editContext = new(Category);

}

private async Task OnBeforeInternalNavigation(LocationChangingContext context)

{

var formWasEdited = editContext.IsModified();

if (!formWasEdited || 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,

});

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

if (confirm)

{

return;

}

context.PreventNavigation();

}

}

}

1. Luego modificamos el **CategoryForm.razor**:

<NavigationLock OnBeforeInternalNavigation="OnBeforeInternalNavigation" />

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

<DataAnnotationsValidator />

<div class="mb-3">

<label>Categoría:</label>

<div>

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

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

</div>

</div>

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

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

</EditForm>

1. Luego creamos el **CategoryCreate.razor** y **CategoryCreate.razor.cs**:

using CurrieTechnologies.Razor.SweetAlert2;

using Microsoft.AspNetCore.Components;

using Orders.Frontend.Repositories;

using Orders.Shared.Entities;

namespace Orders.Frontend.Pages.Categories

{

public partial class CategoryCreate

{

private Category category = new();

private CategoryForm? categoryForm;

[Inject] private IRepository repository { get; set; } = null!;

[Inject] private SweetAlertService sweetAlertService { get; set; } = null!;

[Inject] private NavigationManager navigationManager { get; set; } = null!;

private async Task CreateAsync()

{

var responseHttp = await repository.PostAsync("/api/categories", category);

if (responseHttp.Error)

{

var message = await responseHttp.GetErrorMessageAsync();

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

return;

}

Return();

var toast = sweetAlertService.Mixin(new SweetAlertOptions

{

Toast = true,

Position = SweetAlertPosition.BottomEnd,

ShowConfirmButton = true,

Timer = 3000

});

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

}

private void Return()

{

categoryForm!.FormPostedSuccessfully = true;

navigationManager.NavigateTo("/categories");

}

}

}

1. Luego modificamos el **CategoryCreate.razor**:

@page "/categories/create"

<h3>Crear Categoría</h3>

<CategoryForm @ref="categoryForm" Category="category" OnValidSubmit="CreateAsync" ReturnAction="Return" />

1. Probamos lo que llevamos.
2. Luego creamos el **CategoryEdit.razor** y **CategoryEdit.razor.cs:**

using System.Net;

using CurrieTechnologies.Razor.SweetAlert2;

using Microsoft.AspNetCore.Components;

using Orders.Frontend.Repositories;

using Orders.Shared.Entities;

namespace Orders.Frontend.Pages.Categories

{

public partial class CategoryEdit

{

private Category? category;

private CategoryForm? categoryForm;

[Inject] private IRepository repository { get; set; } = null!;

[Inject] private SweetAlertService sweetAlertService { get; set; } = null!;

[Inject] private NavigationManager navigationManager { get; set; } = null!;

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

protected override async Task OnParametersSetAsync()

{

var responseHttp = await repository.GetAsync<Category>($"/api/categories/{Id}");

if (responseHttp.Error)

{

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

{

navigationManager.NavigateTo("/categories");

}

else

{

var messsage = await responseHttp.GetErrorMessageAsync();

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

}

}

else

{

category = responseHttp.Response;

}

}

private async Task EditAsync()

{

var responseHttp = await repository.PutAsync("/api/categories", category);

if (responseHttp.Error)

{

var message = await responseHttp.GetErrorMessageAsync();

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

return;

}

Return();

var toast = sweetAlertService.Mixin(new SweetAlertOptions

{

Toast = true,

Position = SweetAlertPosition.BottomEnd,

ShowConfirmButton = true,

Timer = 3000

});

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

}

private void Return()

{

categoryForm!.FormPostedSuccessfully = true;

navigationManager.NavigateTo("/categories");

}

}

}

1. Luego modificamos el **CategoryEdit.razor**:

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

<h3>Editar Categoría</h3>

@if(category is null)

{

<Loading/>

}

else

{

<CategoryForm @ref="categoryForm" Category="category" OnValidSubmit="EditAsync" ReturnAction="Return" />

}

1. Probamos y hacemos el commit.

## Creando un formulario genérico

(Explicado en el vídeo: <https://www.youtube.com/watch?v=u9sE2aiuHHs&list=PLuEZQoW9bRnRBThyGs208ZMrCYBRTvIg2&index=15>)

1. Las entidades **Country**, **Category**, **State** y **City** tienen en común que todas tienen un **Id** y un **Name**, para evitar duplicidad de código. Primero creamos dentro de **Shared/Interfaces** la interfaz **IEntityWithName**:

namespace Orders.Shared.Interfaces

{

public interface IEntityWithName

{

string Name { get; set; }

}

}

1. Modificamos las entidades **Country** y **Category** para que implementen esta interfaz:

public class Country : IEntityWithName

Y

public class Category : IEntityWithName

1. Vamos a crear el **FormWithName** en los componentes **Shared** para reutilizarlo en estos 4 CRUDS, procedemos a crear el **FormWithName.razor** y **FormWithName.razor.cs**:

using CurrieTechnologies.Razor.SweetAlert2;

using Microsoft.AspNetCore.Components;

using Microsoft.AspNetCore.Components.Forms;

using Microsoft.AspNetCore.Components.Routing;

using Orders.Shared.Interfaces;

namespace Orders.Frontend.Shared

{

public partial class FormWithName<TModel> where TModel : IEntityWithName

{

private EditContext editContext = null!;

[EditorRequired, Parameter] public TModel Model { get; set; } = default!;

[EditorRequired, Parameter] public string Label { get; set; } = null!;

[EditorRequired, Parameter] public EventCallback OnValidSubmit { get; set; }

[EditorRequired, Parameter] public EventCallback ReturnAction { get; set; }

[Inject] public SweetAlertService SweetAlertService { get; set; } = null!;

public bool FormPostedSuccessfully { get; set; }

protected override void OnInitialized()

{

editContext = new(Model!);

}

private async Task OnBeforeInternalNavigation(LocationChangingContext context)

{

var formWasEdited = editContext.IsModified();

if (!formWasEdited || 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,

});

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

if (confirm)

{

return;

}

context.PreventNavigation();

}

}

}

1. Modificamos el **FormWithName.razor**:

@typeparam TModel where TModel : IEntityWithName

<NavigationLock OnBeforeInternalNavigation="OnBeforeInternalNavigation" />

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

<DataAnnotationsValidator />

<div class="mb-3">

<label>@Label</label>

<div>

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

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

</div>

</div>

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

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

</EditForm>

1. Modificamos el **CountryCreate.razor**:

@page "/countries/create"

<h3>Crear País</h3>

<FormWithName @ref="countryForm" Label="País:" Model="country" OnValidSubmit="CreateAsync" ReturnAction="Return" />

1. Modificamos el **CountryCreate.razor.cs**:

private FormWithName<Country>? countryForm;

1. Modificamos el **CountryEdit.razor**:

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

<h3>Editar País</h3>

@if(country is null)

{

<Loading/>

}

else

{

<FormWithName @ref="countryForm" Label="País:" Model="country" OnValidSubmit="EditAsync" ReturnAction="Return" />

}

1. Modificamos el **CountryEdit.razor.cs**:

private FormWithName<Country>? countryForm;

1. Borramos el **CountryForm**.
2. Probamos.
3. Modificamos el **CategoryCreate.razor**:

@page "/categories/create"

<h3>Crear Categoría</h3>

<FormWithName @ref="categoryForm" Label="Categoría:" Model="category" OnValidSubmit="CreateAsync" ReturnAction="Return" />

1. Modificamos el **CategoryCreate.razor.cs**:

private FormWithName<Category>? categoryForm;

1. Modificamos el **CategoryEdit.razor**:

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

<h3>Editar Categoría</h3>

@if(category is null)

{

<Loading/>

}

else

{

<FormWithName @ref="categoryForm" Label="Categoría:" Model="category" OnValidSubmit="EditAsync" ReturnAction="Return" />

}

1. Modificamos el **CategoryEdit.razor.cs**:

private FormWithName<Category>? categoryForm;

1. Borramos el **CategoryForm**.
2. Probamos y hacemos el commit.

## Configurando un repositorio para trabajo en equipo, resolver conflictos y obtener estadísticas de código

Este tema está explicado en los vídeos:

* <https://www.youtube.com/watch?v=GtN6N11qSgA&list=PLuEZQoW9bRnRBThyGs208ZMrCYBRTvIg2&index=16>
* <https://www.youtube.com/watch?v=5ycMPV5qGMg&list=PLuEZQoW9bRnRBThyGs208ZMrCYBRTvIg2&index=17>
* <https://www.youtube.com/watch?v=-_rCQGG7lEs&list=PLuEZQoW9bRnRBThyGs208ZMrCYBRTvIg2&index=18>

## Adicionando un Seeder a la base de datos

(<https://www.youtube.com/watch?v=VD1b8yAMC7o&list=PLuEZQoW9bRnRBThyGs208ZMrCYBRTvIg2&index=20>)

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

using Orders.Shared.Entities;

namespace Orders.Backend.Data

{

public class SeedDb

{

private readonly DataContext \_context;

public SeedDb(DataContext context)

{

\_context = context;

}

public async Task SeedAsync()

{

await \_context.Database.EnsureCreatedAsync();

await CheckCountriesAsync();

await CheckCategoriesAsync();

}

private async Task CheckCountriesAsync()

{

if (!\_context.Countries.Any())

{

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

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

}

await \_context.SaveChangesAsync();

}

private async Task CheckCategoriesAsync()

{

if (!\_context.Categories.Any())

{

\_context.Categories.Add(new Category { Name = "Calzado" });

\_context.Categories.Add(new Category { Name = "Tecnología" });

}

await \_context.SaveChangesAsync();

}

}

}

1. Luego modificamos el **Program** del proyecto **Backend** 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)

{

var scopedFactory = app.Services.GetService<IServiceScopeFactory>();

using (var scope = scopedFactory!.CreateScope())

{

var 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

(<https://www.youtube.com/watch?v=1zz7kNdb-Y0&list=PLuEZQoW9bRnRBThyGs208ZMrCYBRTvIg2&index=20>)

(<https://www.youtube.com/watch?v=w_mw7qcrsqc&list=PLuEZQoW9bRnRBThyGs208ZMrCYBRTvIg2&index=21>)

(<https://www.youtube.com/watch?v=sVpsZRrp-x0&list=PLuEZQoW9bRnRBThyGs208ZMrCYBRTvIg2&index=22>)

1. Creamos la entidad **State**:

using System.ComponentModel.DataAnnotations;

namespace Orders.Shared.Entities

{

public class State : IEntityWithName

{

public int Id { get; set; }

[Display(Name = "Estado / Departamento")]

[MaxLength(100, ErrorMessage = "El campo {0} no puede tener más de {1} caracteres.")]

[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 || States.Count == 0 ? 0 : States.Count;

1. Creamos la entidad **City**:

using System.ComponentModel.DataAnnotations;

namespace Orders.Shared.Entities

{

public class City : IEntityWithName

{

public int Id { get; set; }

[Display(Name = "Ciudad")]

[MaxLength(100, ErrorMessage = "El campo {0} no puede tener más de {1} caracteres.")]

[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; }

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

[Display(Name = "Ciudades")]

public int CitiesNumber => Cities == null || Cities.Count == 0 ? 0 : Cities.Count;

1. Modificamos el **DataContext**:

using Microsoft.EntityFrameworkCore;

using Orders.Shared.Entities;

namespace Orders.Backend.Data

{

public class DataContext : DbContext

{

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<Category>().HasIndex(c => c.Name).IsUnique();

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

modelBuilder.Entity<State>().HasIndex(s => new { s.CountryId, s.Name }).IsUnique();

modelBuilder.Entity<City>().HasIndex(c => new { c.StateId, c.Name }).IsUnique();

DisableCascadingDelete(modelBuilder);

}

private void DisableCascadingDelete(ModelBuilder modelBuilder)

{

var relationships = modelBuilder.Model.GetEntityTypes().SelectMany(e => e.GetForeignKeys());

foreach (var relationship in relationships)

{

relationship.DeleteBehavior = DeleteBehavior.Restrict;

}

}

}

}

1. Luego de esto agregamos una nueva migración y la aplicamos..
2. Para evitar la redundancia ciclica en la respuesta de los JSON vamos a agregar la siguiente configuración, modificamos el **Program** del **Backend**:

builder.Services.AddControllers()

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

1. Colocamos el modificador **virtual** a todos los métodos público del **GenericRepository** para poderlos sobre escribir (se nos habia olvidado en este, fijese que el **GenericUnicOfWork** y **GenericController** si los tiene modificadores **virtual**).
2. Creamos el **ICountriesRepository**:

using Orders.Shared.Entities;

using Orders.Shared.Responses;

namespace Orders.Backend.Repositories.Interfaces

{

public interface ICountriesRepository

{

Task<ActionResponse<Country>> GetAsync(int id);

Task<ActionResponse<IEnumerable<Country>>> GetAsync();

}

}

1. Creamos el **CountriesRepository**:

using Microsoft.EntityFrameworkCore;

using Orders.Backend.Data;

using Orders.Backend.Repositories.Interfaces;

using Orders.Shared.Entities;

using Orders.Shared.Responses;

namespace Orders.Backend.Repositories.Implementations

{

public class CountriesRepository : GenericRepository<Country>, ICountriesRepository

{

private readonly DataContext \_context;

public CountriesRepository(DataContext context) : base(context)

{

\_context = context;

}

public override async Task<ActionResponse<IEnumerable<Country>>> GetAsync()

{

var countries = await \_context.Countries

.Include(c => c.States)

.ToListAsync();

return new ActionResponse<IEnumerable<Country>>

{

WasSuccess = true,

Result = countries

};

}

public override async Task<ActionResponse<Country>> GetAsync(int id)

{

var country = await \_context.Countries

.Include(c => c.States!)

.ThenInclude(s => s.Cities)

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

if (country == null)

{

return new ActionResponse<Country>

{

WasSuccess = false,

Message = "País no existe"

};

}

return new ActionResponse<Country>

{

WasSuccess = true,

Result = country

};

}

}

}

1. Creamos el **ICountriesUnitOfWork**:

using Orders.Shared.Entities;

using Orders.Shared.Responses;

namespace Orders.Backend.UnitsOfWork.Interfaces

{

public interface ICountriesUnitOfWork

{

Task<ActionResponse<Country>> GetAsync(int id);

Task<ActionResponse<IEnumerable<Country>>> GetAsync();

}

}

1. Creamos el **CountriesUnitOfWork**:

using Orders.Backend.Repositories.Interfaces;

using Orders.Backend.UnitsOfWork.Interfaces;

using Orders.Shared.Entities;

using Orders.Shared.Responses;

namespace Orders.Backend.UnitsOfWork.Implementations

{

public class CountriesUnitOfWork : GenericUnitOfWork<Country>, ICountriesUnitOfWork

{

private readonly ICountriesRepository \_countriesRepository;

public CountriesUnitOfWork(IGenericRepository<Country> repository, ICountriesRepository countriesRepository) : base(repository)

{

\_countriesRepository = countriesRepository;

}

public override async Task<ActionResponse<IEnumerable<Country>>> GetAsync() => await \_countriesRepository.GetAsync();

public override async Task<ActionResponse<Country>> GetAsync(int id) => await \_countriesRepository.GetAsync(id);

}

}

1. Agregamos las nuevas inyecciones en el **Program**:

builder.Services.AddScoped(typeof(IGenericUnitOfWork<>), typeof(GenericUnitOfWork<>));

builder.Services.AddScoped(typeof(IGenericRepository<>), typeof(GenericRepository<>));

builder.Services.AddScoped<ICountriesRepository, CountriesRepository>();

builder.Services.AddScoped<ICountriesUnitOfWork, CountriesUnitOfWork>();

builder.Services.AddTransient<SeedDb>();

1. Modificamos el **CountriesController**:

using Microsoft.AspNetCore.Mvc;

using Orders.Backend.UnitsOfWork;

using Orders.Shared.Entites;

namespace Orders.Backend.Controllers

{

[ApiController]

[Route("api/[controller]")]

public class CountriesController : GenericController<Country>

{

private readonly ICountriesUnitOfWork \_countriesUnitOfWork;

public CountriesController(IGenericUnitOfWork<Country> unit, ICountriesUnitOfWork countriesUnitOfWork) : base(unit)

{

\_countriesUnitOfWork = countriesUnitOfWork;

}

[HttpGet]

public override async Task<IActionResult> GetAsync()

{

var response = await \_countriesUnitOfWork.GetAsync();

if (response.WasSuccess)

{

return Ok(response.Result);

}

return BadRequest();

}

[HttpGet("{id}")]

public override async Task<IActionResult> GetAsync(int id)

{

var response = await \_countriesUnitOfWork.GetAsync(id);

if (response.WasSuccess)

{

return Ok(response.Result);

}

return NotFound(response.Message);

}

}

}

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. Probamos los cambios por el swagger.
2. Creamos el **IStatesRepository**:

using Orders.Shared.Entities;

using Orders.Shared.Responses;

namespace Orders.Backend.Repositories.Interfaces

{

public interface IStatesRepository

{

Task<ActionResponse<State>> GetAsync(int id);

Task<ActionResponse<IEnumerable<State>>> GetAsync();

}

}

1. Creamos el **StatesRepository**:

using Microsoft.EntityFrameworkCore;

using Orders.Backend.Data;

using Orders.Backend.Repositories.Interfaces;

using Orders.Shared.Entities;

using Orders.Shared.Responses;

namespace Orders.Backend.Repositories.Implementations

{

public class StatesRepository : GenericRepository<State>, IStatesRepository

{

private readonly DataContext \_context;

public StatesRepository(DataContext context) : base(context)

{

\_context = context;

}

public override async Task<ActionResponse<IEnumerable<State>>> GetAsync()

{

var states = await \_context.States

.Include(s => s.Cities)

.ToListAsync();

return new ActionResponse<IEnumerable<State>>

{

WasSuccess = true,

Result = states

};

}

public override async Task<ActionResponse<State>> GetAsync(int id)

{

var state = await \_context.States

.Include(s => s.Cities)

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

if (state == null)

{

return new ActionResponse<State>

{

WasSuccess = false,

Message = "Estado no existe"

};

}

return new ActionResponse<State>

{

WasSuccess = true,

Result = state

};

}

}

}

1. Creamos el **IStatesUnitOfWork**:

using Orders.Shared.Entities;

using Orders.Shared.Responses;

namespace Orders.Backend.UnitsOfWork.Interfaces

{

public interface IStatesUnitOfWork

{

Task<ActionResponse<State>> GetAsync(int id);

Task<ActionResponse<IEnumerable<State>>> GetAsync();

}

}

1. Creamos el **StatesUnitOfWork**:

using Orders.Backend.Repositories.Interfaces;

using Orders.Backend.UnitsOfWork.Interfaces;

using Orders.Shared.Entities;

using Orders.Shared.Responses;

namespace Orders.Backend.UnitsOfWork.Implementations

{

public class StatesUnitOfWork : GenericUnitOfWork<State>, IStatesUnitOfWork

{

private readonly IStatesRepository \_statesRepository;

public StatesUnitOfWork(IGenericRepository<State> repository, IStatesRepository statesRepository) : base(repository)

{

\_statesRepository = statesRepository;

}

public override async Task<ActionResponse<IEnumerable<State>>> GetAsync() => await \_statesRepository.GetAsync();

public override async Task<ActionResponse<State>> GetAsync(int id) => await \_statesRepository.GetAsync(id);

}

}

1. Agregamos las nuevas inyecciones en el **Program**:

builder.Services.AddScoped<ICountriesRepository, CountriesRepository>();

builder.Services.AddScoped<IStatesRepository, StatesRepository>();

builder.Services.AddScoped<ICountriesUnitOfWork, CountriesUnitOfWork>();

builder.Services.AddScoped<IStatesUnitOfWork, StatesUnitOfWork>();

1. Creamos el **StatesController**:

using Microsoft.AspNetCore.Mvc;

using Orders.Backend.UnitsOfWork.Interfaces;

using Orders.Shared.Entities;

namespace Orders.Backend.Controllers

{

[ApiController]

[Route("api/[controller]")]

public class StatesController : GenericController<State>

{

private readonly IStatesUnitOfWork \_statesUnitOfWork;

public StatesController(IGenericUnitOfWork<State> unitOfWork, IStatesUnitOfWork statesUnitOfWork) : base(unitOfWork)

{

\_statesUnitOfWork = statesUnitOfWork;

}

[HttpGet]

public override async Task<IActionResult> GetAsync()

{

var response = await \_statesUnitOfWork.GetAsync();

if (response.WasSuccess)

{

return Ok(response.Result);

}

return BadRequest();

}

[HttpGet("{id}")]

public override async Task<IActionResult> GetAsync(int id)

{

var response = await \_statesUnitOfWork.GetAsync(id);

if (response.WasSuccess)

{

return Ok(response.Result);

}

return NotFound(response.Message);

}

}

}

1. Creamos el **CitiesController**:

using Microsoft.AspNetCore.Mvc;

using Orders.Backend.UnitsOfWork.Interfaces;

using Orders.Shared.Entities;

namespace Orders.Backend.Controllers

{

[ApiController]

[Route("api/[controller]")]

public class CitiesController : GenericController<City>

{

public CitiesController(IGenericUnitOfWork<City> unitOfWork) : base(unitOfWork)

{

}

}

}

1. Probamos en Swagger lo que llevamos.
2. Cambiemos el **CountriesIndex.razor** para ver el número de departamentos/estados de cada país y adicionar el botón de detalles (también le ponemos el **btn-sm** a los botónes de **CategoriesIndex.razor**):

<GenericList MyList="countries">

<NoRecords>

<p>Aun no hay paises registrados.</p>

</NoRecords>

<Body>

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

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

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

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

</td>

</tr>

}

</tbody>

</table>

</Body>

</GenericList>

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

## Creando un CRUD multinivel

(<https://www.youtube.com/watch?v=sVpsZRrp-x0&list=PLuEZQoW9bRnRBThyGs208ZMrCYBRTvIg2&index=22>)

(<https://www.youtube.com/watch?v=O7wocU3wnvU&list=PLuEZQoW9bRnRBThyGs208ZMrCYBRTvIg2&index=24>)

(<https://www.youtube.com/watch?v=90wSCbMO5NI&list=PLuEZQoW9bRnRBThyGs208ZMrCYBRTvIg2&index=25>)

(<https://www.youtube.com/watch?v=a2OT5yXnjQ8&list=PLuEZQoW9bRnRBThyGs208ZMrCYBRTvIg2&index=26>)

(<https://www.youtube.com/watch?v=dCy0-3C-9Dk&list=PLuEZQoW9bRnRBThyGs208ZMrCYBRTvIg2&index=27>)

1. En el proyecto **Frontend** en la carpeta **Pages/Countries** vamos a crear la página **CountryDetails.razor** y **CountryDetails.razor.cs**:

using System.Net;

using CurrieTechnologies.Razor.SweetAlert2;

using Microsoft.AspNetCore.Components;

using Orders.Frontend.Repositories;

using Orders.Shared.Entities;

namespace Orders.Frontend.Pages.Countries

{

public partial class CountryDetails

{

private Country? country;

[Inject] private NavigationManager NavigationManager { get; set; } = null!;

[Inject] private SweetAlertService SweetAlertService { get; set; } = null!;

[Inject] private IRepository Repository { get; set; } = null!;

[Parameter]

public int CountryId { get; set; }

protected override async Task OnInitializedAsync()

{

await LoadAsync();

}

private async Task LoadAsync()

{

var responseHttp = await Repository.GetAsync<Country>($"/api/countries/{CountryId}");

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;

}

private async Task DeleteAsync(State state)

{

var result = await SweetAlertService.FireAsync(new SweetAlertOptions

{

Title = "Confirmación",

Text = $"¿Realmente deseas eliminar el departamento/estado? {state.Name}",

Icon = SweetAlertIcon.Question,

ShowCancelButton = true,

CancelButtonText = "No",

ConfirmButtonText = "Si"

});

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

if (confirm)

{

return;

}

var responseHttp = await Repository.DeleteAsync<State>($"/api/states/{state.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();

var toast = SweetAlertService.Mixin(new SweetAlertOptions

{

Toast = true,

Position = SweetAlertPosition.BottomEnd,

ShowConfirmButton = true,

Timer = 3000

});

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

}

}

}

1. Modificamos página **CountryDetails.razor**:

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

@if (country is null)

{

<Loading />

}

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>

<h4>Estados/Departamentos</h4>

<GenericList MyList="country.States!.ToList()">

<Body>

<table class="table table-striped">

<thead>

<tr>

<th>Estado / Departamento</th>

<th>Ciudades</th>

<th></th>

</tr>

</thead>

<tbody>

@foreach (var state in country.States!)

{

<tr>

<td>

@state.Name

</td>

<td>

@state.CitiesNumber

</td>

<td>

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

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

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

</td>

</tr>

}

</tbody>

</table>

</Body>

</GenericList>

}

1. Modificamos el **CountriesIndex.razor**:

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

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

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

1. Probamos lo que llevamos hasta el momento.
2. En el proyecto **Frontend** en la carpeta **Pages**/**States** y dentro de esta creamos el componente **StateCreate.razor** y **StateCreate.razor.cs**:

using CurrieTechnologies.Razor.SweetAlert2;

using Microsoft.AspNetCore.Components;

using Orders.Frontend.Repositories;

using Orders.Shared.Entities;

namespace Orders.Frontend.Pages.States

{

public partial class StateCreate

{

private State state = new();

private FormWithName<State>? stateForm;

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

[Inject] private IRepository Repository { get; set; } = null!;

[Inject] private NavigationManager NavigationManager { get; set; } = null!;

[Inject] private SweetAlertService SweetAlertService { get; set; } = null!;

private async Task CreateAsync()

{

state.CountryId = CountryId;

var responseHttp = await Repository.PostAsync("/api/states", state);

if (responseHttp.Error)

{

var message = await responseHttp.GetErrorMessageAsync();

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

return;

}

Return();

var toast = SweetAlertService.Mixin(new SweetAlertOptions

{

Toast = true,

Position = SweetAlertPosition.BottomEnd,

ShowConfirmButton = true,

Timer = 3000

});

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

}

private void Return()

{

stateForm!.FormPostedSuccessfully = true;

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

}

}

}

1. Luego modificamos el **StateCreate.razor**:

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

<h3>Nuevo Estado/Departamento</h3>

<FormWithName @ref="stateForm" Label="Estado/Departamento:" Model="state" OnValidSubmit="CreateAsync" ReturnAction="Return" />

1. Probamos lo que llevamos hasta el momento.
2. En el proyecto **Frontend** en la carpeta **Pages**/**States** y dentro de esta creamos el componente **StateEdit.razor** y **StateEdit.razor.cs**:

using System.Net;

using CurrieTechnologies.Razor.SweetAlert2;

using Microsoft.AspNetCore.Components;

using Orders.Frontend.Repositories;

using Orders.Shared.Entities;

namespace Orders.Frontend.Pages.States

{

public partial class StateEdit

{

private State? state;

private FormWithName<State>? stateForm;

[Inject] private IRepository Repository { get; set; } = null!;

[Inject] private NavigationManager NavigationManager { get; set; } = null!;

[Inject] private SweetAlertService SweetAlertService { get; set; } = null!;

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

protected override async Task OnParametersSetAsync()

{

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

if (responseHttp.Error)

{

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

{

Return();

}

var message = await responseHttp.GetErrorMessageAsync();

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

return;

}

state = responseHttp.Response;

}

private async Task SaveAsync()

{

var responseHttp = await Repository.PutAsync($"/api/states", state);

if (responseHttp.Error)

{

var message = await responseHttp.GetErrorMessageAsync();

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

return;

}

Return();

var toast = SweetAlertService.Mixin(new SweetAlertOptions

{

Toast = true,

Position = SweetAlertPosition.BottomEnd,

ShowConfirmButton = true,

Timer = 3000

});

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

}

private void Return()

{

stateForm!.FormPostedSuccessfully = true;

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

}

}

}

1. Luego modificamos el **StateEdit.razor**:

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

<h3>Editar Estado/Departamento</h3>

@if (state is null)

{

<Loading />

}

else

{

<FormWithName @ref="stateForm" Label="Estado/Departamento:" Model="state" OnValidSubmit="SaveAsync" ReturnAction="Return" />

}

1. Probamos lo que llevamos.
2. En el proyecto **Frontend** en la carpeta **Pages**/**States** y dentro de esta creamos el componente **StateDetails.razor** y **StateDetails.razor.cs**:

using CurrieTechnologies.Razor.SweetAlert2;

using Microsoft.AspNetCore.Components;

using Orders.Frontend.Repositories;

using Orders.Shared.Entities;

using System.Net;

namespace Orders.Frontend.Pages.States

{

public partial class StateDetails

{

private State? state;

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

[Inject] private NavigationManager NavigationManager { get; set; } = null!;

[Inject] private SweetAlertService SweetAlertService { get; set; } = null!;

[Inject] private IRepository Repository { get; set; } = null!;

protected override async Task OnInitializedAsync()

{

await LoadAsync();

}

private async Task LoadAsync()

{

var responseHttp = await Repository.GetAsync<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 DeleteAsync(City city)

{

var result = await SweetAlertService.FireAsync(new SweetAlertOptions

{

Title = "Confirmación",

Text = $"¿Realmente deseas eliminar la ciudad? {city.Name}",

Icon = SweetAlertIcon.Question,

ShowCancelButton = true,

CancelButtonText = "No",

ConfirmButtonText = "Si"

});

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

if (confirm)

{

return;

}

var responseHttp = await Repository.DeleteAsync<City>($"/api/cities/{city.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();

var toast = SweetAlertService.Mixin(new SweetAlertOptions

{

Toast = true,

Position = SweetAlertPosition.BottomEnd,

ShowConfirmButton = true,

Timer = 3000

});

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

}

}

}

1. Luego moidificamos el **StateDetails.razor**:

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

@if (state is null)

{

<Loading />

}

else

{

<h3>@state.Name</h3>

<div class="mb-2">

<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>

<h4>Ciudades</h4>

<GenericList MyList="state.Cities!.ToList()">

<Body>

<table class="table table-striped">

<thead>

<tr>

<th>Ciudad</th>

<th></th>

</tr>

</thead>

<tbody>

@foreach (var city in state.Cities!)

{

<tr>

<td>

@city.Name

</td>

<td>

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

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

</td>

</tr>

}

</tbody>

</table>

</Body>

</GenericList>

}

1. Probamos.
2. En el proyecto **Frontend** en la carpeta **Pages**/**Cities** y dentro de esta creamos el componente **CityCreate.razor** y **CityCreate.razor.cs**:

using CurrieTechnologies.Razor.SweetAlert2;

using Microsoft.AspNetCore.Components;

using Orders.Frontend.Repositories;

using Orders.Shared.Entities;

namespace Orders.Frontend.Pages.Cities

{

public partial class CityCreate

{

private City city = new();

private FormWithName<City>? cityForm;

[Inject] private IRepository Repository { get; set; } = null!;

[Inject] private NavigationManager NavigationManager { get; set; } = null!;

[Inject] private SweetAlertService SweetAlertService { get; set; } = null!;

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

private async Task CreateAsync()

{

city.StateId = StateId;

var responseHttp = await Repository.PostAsync("/api/cities", city);

if (responseHttp.Error)

{

var message = await responseHttp.GetErrorMessageAsync();

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

return;

}

Return();

var toast = SweetAlertService.Mixin(new SweetAlertOptions

{

Toast = true,

Position = SweetAlertPosition.BottomEnd,

ShowConfirmButton = true,

Timer = 3000

});

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

}

private void Return()

{

cityForm!.FormPostedSuccessfully = true;

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

}

}

}

1. Luego modificamos el **CityCreate.razor**:

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

<h3>Nueva Ciudad</h3>

<FormWithName @ref="cityForm" Label="Ciudad:" Model="city" OnValidSubmit="CreateAsync" ReturnAction="Return" />

1. Probamos.
2. En el proyecto **Frontend** en la carpeta **Pages**/**Cities** y dentro de esta creamos el componente **CityEdit.razor** y **CityEdit.razor.cs**:

using CurrieTechnologies.Razor.SweetAlert2;

using Microsoft.AspNetCore.Components;

using Orders.Frontend.Repositories;

using Orders.Shared.Entities;

using System.Net;

namespace Orders.Frontend.Pages.Cities

{

public partial class CityEdit

{

private City? city;

private FormWithName<City>? cityForm;

[Inject] private NavigationManager NavigationManager { get; set; } = null!;

[Inject] private IRepository Repository { get; set; } = null!;

[Inject] private SweetAlertService SweetAlertService { get; set; } = null!;

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

protected override async Task OnParametersSetAsync()

{

var responseHttp = await Repository.GetAsync<City>($"/api/cities/{CityId}");

if (responseHttp.Error)

{

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

{

Return();

}

var message = await responseHttp.GetErrorMessageAsync();

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

return;

}

city = responseHttp.Response;

}

private async Task SaveAsync()

{

var response = await Repository.PutAsync($"/api/cities", city);

if (response.Error)

{

var message = await response.GetErrorMessageAsync();

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

return;

}

Return();

var toast = SweetAlertService.Mixin(new SweetAlertOptions

{

Toast = true,

Position = SweetAlertPosition.BottomEnd,

ShowConfirmButton = true,

Timer = 3000

});

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

}

private void Return()

{

cityForm!.FormPostedSuccessfully = true;

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

}

}

}

1. Luego modificamos el **CityEdit.razor**:

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

<h3>Editar Ciudad</h3>

@if (city is null)

{

<Loading />

}

else

{

<FormWithName @ref="cityForm" Label="Ciudad:" Model="city" OnValidSubmit="SaveAsync" ReturnAction="Return" />

}

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

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

1. Para llenar la información de todos, o al menos la mayorìa de paìses, estados y ciudades del mundo. Vamos a utilizar esta Backend: <https://countrystatecity.in/docs/api/all-countries/> Para poderla utilizar vas a necesitar un token, puedes solicitar tu propio token en: <https://docs.google.com/forms/d/e/1FAIpQLSciOf_227-3pKGKJok6TM0QF2PZhSgfQwy-F-bQaBj0OUgMmA/viewform> llena el formulario y en pocas horas te lo enviarán (la menos eso paso conmigo), luego de tener tu token has los siguientes cambios al proyecto:
2. Al proyecto **Backend** agrega al **appstettings.json** los siguientes parámetros. No olvides cambiar el valor del **TokenValue** por la obtenida por usted en el paso anterior:

{

"ConnectionStrings": {

"DockerConnection": "Data Source=.;Initial Catalog=OrdersPrep;User ID={Your user};Password={Your password};Connect Timeout=30;Encrypt=False;TrustServerCertificate=False;ApplicationIntent=ReadWrite;MultiSubnetFailover=False"

},

"CoutriesBackend": {

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

"tokenName": "X-CSCBackend-KEY",

"tokenValue": "{Your token value}"

},

"Logging": {

"LogLevel": {

"Default": "Information",

"Microsoft.AspNetCore": "Warning"

}

},

"AllowedHosts": "\*"

}

1. Al proyecto **Shared** dentro de la carpeta **Responses** las clases que vamos a obtener de la Backend. Empecemos con **CountryResponse**:

namespace Orders.Shared.Responses

{

public class CountryResponse

{

public long Id { get; set; }

public string? Name { get; set; }

public string? Iso2 { get; set; }

}

}

1. Continuamos con **StateResponse**:

namespace Orders.Shared.Responses

{

public class StateResponse

{

public long Id { get; set; }

public string? Name { get; set; }

public string? Iso2 { get; set; }

}

}

1. Y luego con **CityResponse**:

namespace Orders.Shared.Responses

{

public class CityResponse

{

public long Id { get; set; }

public string? Name { get; set; }

}

}

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

using Orders.Shared.Responses;

namespace Orders.Backend.Services

{

public interface IApiService

{

Task<ActionResponse<T>> GetAsync<T>(string servicePrefix, string controller);

}

}

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

using System.Text.Json;

using Orders.Shared.Responses;

namespace Orders.Backend.Services

{

public class ApiService : IApiService

{

private readonly string \_urlBase;

private readonly string \_tokenName;

private readonly string \_tokenValue;

public ApiService(IConfiguration configuration)

{

\_urlBase = configuration["CoutriesBackend:urlBase"]!;

\_tokenName = configuration["CoutriesBackend:tokenName"]!;

\_tokenValue = configuration["CoutriesBackend:tokenValue"]!;

}

private JsonSerializerOptions \_jsonDefaultOptions => new JsonSerializerOptions

{

PropertyNameCaseInsensitive = true,

};

public async Task<ActionResponse<T>> GetAsync<T>(string servicePrefix, string controller)

{

try

{

var client = new HttpClient()

{

BaseAddress = new Uri(\_urlBase),

};

client.DefaultRequestHeaders.Add(\_tokenName, \_tokenValue);

var url = $"{servicePrefix}{controller}";

var responseHttp = await client.GetAsync(url);

var response = await responseHttp.Content.ReadAsStringAsync();

if (!responseHttp.IsSuccessStatusCode)

{

return new ActionResponse<T>

{

WasSuccess = false,

Message = response

};

}

return new ActionResponse<T>

{

WasSuccess = true,

Result = JsonSerializer.Deserialize<T>(response, \_jsonDefaultOptions)!

};

}

catch (Exception ex)

{

return new ActionResponse<T>

{

WasSuccess = false,

Message = ex.Message

};

}

}

}

}

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

builder.Services.AddTransient<SeedDb>();

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

1. Luego modificamos el **SeedDb**:

using Microsoft.EntityFrameworkCore;

using Orders.Backend.Services;

using Orders.Shared.Entities;

using Orders.Shared.Responses;

namespace Orders.Backend.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())

{

var responseCountries = await \_apiService.GetAsync<List<CountryResponse>>("/v1", "/countries");

if (responseCountries.WasSuccess)

{

var countries = responseCountries.Result!;

foreach (var CountryResponse in countries)

{

var country = await \_context.Countries.FirstOrDefaultAsync(c => c.Name == CountryResponse.Name!)!;

if (country == null)

{

country = new() { Name = CountryResponse.Name!, States = new List<State>() };

var responseStates = await \_apiService.GetAsync<List<StateResponse>>("/v1", $"/countries/{CountryResponse.Iso2}/states");

if (responseStates.WasSuccess)

{

var states = responseStates.Result!;

foreach (var StateResponse in states!)

{

var state = country.States!.FirstOrDefault(s => s.Name == StateResponse.Name!)!;

if (state == null)

{

state = new() { Name = StateResponse.Name!, Cities = new List<City>() };

var responseCities = await \_apiService.GetAsync<List<CityResponse>>("/v1", $"/countries/{CountryResponse.Iso2}/states/{StateResponse.Iso2}/cities");

if (responseCities.WasSuccess)

{

var cities = responseCities.Result!;

foreach (var CityResponse in cities)

{

if (CityResponse.Name == "Mosfellsbær" || CityResponse.Name == "Șăulița")

{

continue;

}

var 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 los paises que tengamos en la BD.
2. Se puede demorar varios minutos para llenar la mayoría de países con sus estados y ciudades. Digo la mayorìa porque la lógica deshecha algunos paises o estados que no tienen ciudades devueltas por la API.
3. Probamos y hacemos el **commit**.

## Agregando paginación

(<https://www.youtube.com/watch?v=dX9R2VhtHAY&list=PLuEZQoW9bRnRBThyGs208ZMrCYBRTvIg2&index=28>)

(<https://www.youtube.com/watch?v=b-7LZudXVGg&list=PLuEZQoW9bRnRBThyGs208ZMrCYBRTvIg2&index=29>)

(<https://www.youtube.com/watch?v=PW0avOOKUQs&list=PLuEZQoW9bRnRBThyGs208ZMrCYBRTvIg2&index=30>)

(<https://www.youtube.com/watch?v=BNlv62A4m8o&list=PLuEZQoW9bRnRBThyGs208ZMrCYBRTvIg2&index=31>)

(<https://www.youtube.com/watch?v=MLsfLpRsCJc&list=PLuEZQoW9bRnRBThyGs208ZMrCYBRTvIg2&index=32>)

(<https://www.youtube.com/watch?v=xv-9M0P8Bo4&list=PLuEZQoW9bRnRBThyGs208ZMrCYBRTvIg2&index=33>)

(<https://www.youtube.com/watch?v=xv-9M0P8Bo4&list=PLuEZQoW9bRnRBThyGs208ZMrCYBRTvIg2&index=34>)

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

namespace Orders.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 **Backend** creamos el folder **Helpers** y dentro de este la clase **QueryableExtensions**:

using Orders.Shared.DTOs;

namespace Orders.Backend.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 **IGenericRepository**, agregandole otra sobre carga el GET.

Task<ActionResponse<IEnumerable<T>>> GetAsync();

Task<ActionResponse<IEnumerable<T>>> GetAsync(PaginationDTO pagination);

Task<ActionResponse<int>> GetTotalPagesAsync(PaginationDTO pagination);

1. Modificamos el **GenericRepository**:

public virtual async Task<ActionResponse<IEnumerable<T>>> GetAsync(PaginationDTO pagination)

{

var queryable = \_entity.AsQueryable();

return new ActionResponse<IEnumerable<T>>

{

WasSuccess = true,

Result = await queryable

.Paginate(pagination)

.ToListAsync()

};

}

public virtual async Task<ActionResponse<int>> GetTotalPagesAsync(PaginationDTO pagination)

{

var queryable = \_entity.AsQueryable();

double count = await queryable.CountAsync();

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

return new ActionResponse<int>

{

WasSuccess = true,

Result = totalPages

};

}

1. Modificamos el **IGenericUnitOfWork**:

Task<ActionResponse<IEnumerable<T>>> GetAsync();

Task<ActionResponse<IEnumerable<T>>> GetAsync(PaginationDTO pagination);

Task<ActionResponse<int>> GetTotalPagesAsync(PaginationDTO pagination);

1. Modificamos el **IGenericUnitOfWork**:

public virtual async Task<ActionResponse<IEnumerable<T>>> GetAsync(PaginationDTO pagination) => await \_repository.GetAsync(pagination);

public virtual async Task<ActionResponse<int>> GetTotalPagesAsync(PaginationDTO pagination) => await \_repository.GetTotalPagesAsync(pagination);

1. Modificamos el **GenericController**:

[HttpGet("full")]

public virtual async Task<IActionResult> GetAsync()

{

var action = await \_unitOfWork.GetAsync();

if (action.WasSuccess)

{

return Ok(action.Result);

}

return BadRequest();

}

[HttpGet]

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

{

var action = await \_unitOfWork.GetAsync(pagination);

if (action.WasSuccess)

{

return Ok(action.Result);

}

return BadRequest();

}

[HttpGet("totalPages")]

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

{

var action = await \_unitOfWork.GetTotalPagesAsync(pagination);

if (action.WasSuccess)

{

return Ok(action.Result);

}

return BadRequest();

}

1. Modificamos el **ICountriesRepository**:

Task<ActionResponse<IEnumerable<Country>>> GetAsync();

Task<ActionResponse<IEnumerable<Country>>> GetAsync(PaginationDTO pagination);

1. Modificamos el **CountriesRepository**:

public override async Task<ActionResponse<IEnumerable<Country>>> GetAsync()

{

var countries = await \_context.Countries

.OrderBy(x => x.Name)

.ToListAsync();

return new ActionResponse<IEnumerable<Country>>

{

WasSuccess = true,

Result = countries

};

}

public override async Task<ActionResponse<IEnumerable<Country>>> GetAsync(PaginationDTO pagination)

{

var queryable = \_context.Countries

.Include(c => c.States)

.AsQueryable();

return new ActionResponse<IEnumerable<Country>>

{

WasSuccess = true,

Result = await queryable

.OrderBy(x => x.Name)

.Paginate(pagination)

.ToListAsync()

};

}

1. Modificamos el **ICountriesUnitOfWork**:

Task<ActionResponse<IEnumerable<Country>>> GetAsync();

Task<ActionResponse<IEnumerable<Country>>> GetAsync(PaginationDTO pagination);

1. Modificamos el **CountriesUnitOfWork**:

public override async Task<ActionResponse<IEnumerable<Country>>> GetAsync() => await \_countriesRepository.GetAsync();

public override async Task<ActionResponse<IEnumerable<Country>>> GetAsync(PaginationDTO pagination) => await \_countriesRepository.GetAsync(pagination);

1. Modificamos el **CountriesController**:

[HttpGet("full")]

public override async Task<IActionResult> GetAsync()

{

var response = await \_countriesUnitOfWork.GetAsync();

if (response.WasSuccess)

{

return Ok(response.Result);

}

return BadRequest();

}

[HttpGet]

public override async Task<IActionResult> GetAsync(PaginationDTO pagination)

{

var response = await \_countriesUnitOfWork.GetAsync(pagination);

if (response.WasSuccess)

{

return Ok(response.Result);

}

return BadRequest();

}

1. Modificamos el **IStatesRepository**:

Task<ActionResponse<IEnumerable<State>>> GetAsync();

Task<ActionResponse<IEnumerable<State>>> GetAsync(PaginationDTO pagination);

Task<ActionResponse<int>> GetTotalPagesAsync(PaginationDTO pagination);

1. Modificamos el **StatesRepository**:

public override async Task<ActionResponse<IEnumerable<State>>> GetAsync()

{

var states = await \_context.States

.OrderBy(x => x.Name)

.ToListAsync();

return new ActionResponse<IEnumerable<State>>

{

WasSuccess = true,

Result = states

};

}

public override async Task<ActionResponse<IEnumerable<State>>> GetAsync(PaginationDTO pagination)

{

var queryable = \_context.States

.Include(x => x.Cities)

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

.AsQueryable();

return new ActionResponse<IEnumerable<State>>

{

WasSuccess = true,

Result = await queryable

.OrderBy(x => x.Name)

.Paginate(pagination)

.ToListAsync()

};

}

public async override Task<ActionResponse<int>> GetTotalPagesAsync(PaginationDTO pagination)

{

var queryable = \_context.States

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

.AsQueryable();

double count = await queryable.CountAsync();

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

return new ActionResponse<int>

{

WasSuccess = true,

Result = totalPages

};

}

1. Modificamos el **IStatesUnitOfWork**:

Task<ActionResponse<IEnumerable<State>>> GetAsync();

Task<ActionResponse<IEnumerable<State>>> GetAsync(PaginationDTO pagination);

Task<ActionResponse<int>> GetTotalPagesAsync(PaginationDTO pagination);

1. Modificamos el **StatesUnitOfWork**:

public override async Task<ActionResponse<IEnumerable<State>>> GetAsync() => await \_statesRepository.GetAsync();

public override async Task<ActionResponse<IEnumerable<State>>> GetAsync(PaginationDTO pagination) => await \_statesRepository.GetAsync(pagination);

public override async Task<ActionResponse<int>> GetTotalPagesAsync(PaginationDTO pagination) => await \_statesRepository.GetTotalPagesAsync(pagination);

1. Modificamos el **StatesController**:

[HttpGet("full")]

public override async Task<IActionResult> GetAsync()

{

var response = await \_statesUnitOfWork.GetAsync();

if (response.WasSuccess)

{

return Ok(response.Result);

}

return BadRequest();

}

[HttpGet]

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

{

var response = await \_statesUnitOfWork.GetAsync(pagination);

if (response.WasSuccess)

{

return Ok(response.Result);

}

return BadRequest();

}

[HttpGet("totalPages")]

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

{

var action = await \_statesUnitOfWork.GetTotalPagesAsync(pagination);

if (action.WasSuccess)

{

return Ok(action.Result);

}

return BadRequest();

}

1. Creamos el **ICitiesRepository**:

using Orders.Shared.DTOs;

using Orders.Shared.Entites;

using Orders.Shared.Responses;

namespace Orders.Backend.Repositories

{

public interface ICitiesRepository

{

Task<ActionResponse<IEnumerable<City>>> GetAsync(PaginationDTO pagination);

Task<ActionResponse<int>> GetTotalPagesAsync(PaginationDTO pagination);

}

}

1. Creamos el **CitiesRepository**:

using Microsoft.EntityFrameworkCore;

using Orders.Backend.Data;

using Orders.Backend.Helpers;

using Orders.Shared.DTOs;

using Orders.Shared.Entites;

using Orders.Shared.Responses;

namespace Orders.Backend.Repositories

{

public class CitiesRepository : GenericRepository<City>, ICitiesRepository

{

private readonly DataContext \_context;

public CitiesRepository(DataContext context) : base(context)

{

\_context = context;

}

public override async Task<ActionResponse<IEnumerable<City>>> GetAsync(PaginationDTO pagination)

{

var queryable = \_context.Cities

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

.AsQueryable();

return new ActionResponse<IEnumerable<City>>

{

WasSuccess = true,

Result = await queryable

.OrderBy(x => x.Name)

.Paginate(pagination)

.ToListAsync()

};

}

public override async Task<ActionResponse<int>> GetTotalPagesAsync(PaginationDTO pagination)

{

var queryable = \_context.Cities

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

.AsQueryable();

double count = await queryable.CountAsync();

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

return new ActionResponse<int>

{

WasSuccess = true,

Result = totalPages

};

}

}

}

1. Creamos el **ICitiesUnitOfWork**:

using Orders.Shared.DTOs;

using Orders.Shared.Entites;

using Orders.Shared.Responses;

namespace Orders.Backend.UnitsOfWork

{

public interface ICitiesUnitOfWork

{

Task<ActionResponse<IEnumerable<City>>> GetAsync(PaginationDTO pagination);

Task<ActionResponse<int>> GetTotalPagesAsync(PaginationDTO pagination);

}

}

1. Creamos el **CitiesUnitOfWork**:

using Orders.Backend.Repositories;

using Orders.Shared.DTOs;

using Orders.Shared.Entites;

using Orders.Shared.Responses;

namespace Orders.Backend.UnitsOfWork

{

public class CitiesUnitOfWork : GenericUnitOfWork<City>, ICitiesUnitOfWork

{

private readonly ICitiesRepository \_citiesRepository;

public CitiesUnitOfWork(IGenericRepository<City> repository, ICitiesRepository citiesRepository) : base(repository)

{

\_citiesRepository = citiesRepository;

}

public override async Task<ActionResponse<IEnumerable<City>>> GetAsync(PaginationDTO pagination) => await \_citiesRepository.GetAsync(pagination);

public override async Task<ActionResponse<int>> GetTotalPagesAsync(PaginationDTO pagination) => await \_citiesRepository.GetTotalPagesAsync(pagination);

}

}

1. Agregamos las nuevaa inyecciones en el **Program**:

builder.Services.AddScoped<ICitiesRepository, CitiesRepository>();

builder.Services.AddScoped<ICountriesRepository, CountriesRepository>();

builder.Services.AddScoped<IStatesRepository, StatesRepository>();

builder.Services.AddScoped<ICitiesUnitOfWork, CitiesUnitOfWork>();

builder.Services.AddScoped<ICountriesUnitOfWork, CountriesUnitOfWork>();

builder.Services.AddScoped<IStatesUnitOfWork, StatesUnitOfWork>();

1. Modificamos el **CitiesController**:

using Microsoft.AspNetCore.Mvc;

using Orders.Backend.UnitsOfWork;

using Orders.Shared.DTOs;

using Orders.Shared.Entites;

namespace Orders.Backend.Controllers

{

[ApiController]

[Route("api/[controller]")]

public class CitiesController : GenericController<City>

{

private readonly ICitiesUnitOfWork \_citiesUnitOfWork;

public CitiesController(IGenericUnitOfWork<City> unitOfWork, ICitiesUnitOfWork citiesUnitOfWork) : base(unitOfWork)

{

\_citiesUnitOfWork = citiesUnitOfWork;

}

[HttpGet]

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

{

var response = await \_citiesUnitOfWork.GetAsync(pagination);

if (response.WasSuccess)

{

return Ok(response.Result);

}

return BadRequest();

}

[HttpGet("totalPages")]

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

{

var action = await \_citiesUnitOfWork.GetTotalPagesAsync(pagination);

if (action.WasSuccess)

{

return Ok(action.Result);

}

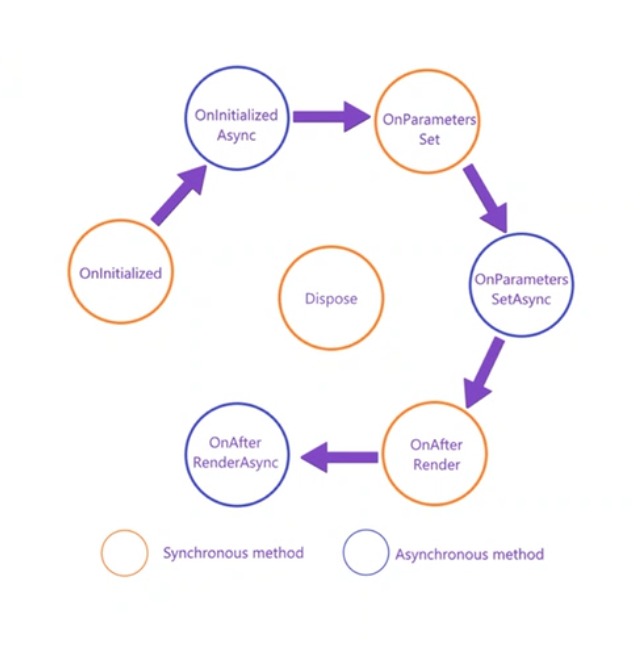
return BadRequest();

}

}

}

1. Probamos la paginación por el Swagger.
2. Este es el ciclo de vida en Blazor:



1. Creamos en el proyecto **Frontend** en la carpeta **Shared** el componente **Pagination.razor** y **Pagination.razor.cs**:

using Microsoft.AspNetCore.Components;

namespace Orders.Frontend.Shared

{

public partial class Pagination

{

private List<PageModel> links = new();

[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; }

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

});

}

private 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 el **Pagination.razor**:

<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>

1. Modificamos la clase **CountriesIndex.razor.cs**:

…

private int currentPage = 1;

private int totalPages;

[Inject] private NavigationManager NavigationManager { get; set; } = null!;

[Inject] private SweetAlertService SweetAlertService { get; set; } = null!;

[Inject] private IRepository Repository { get; set; } = null!;

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

protected override async Task OnInitializedAsync()

{

await LoadAsync();

}

private async Task SelectedPageAsync(int page)

{

currentPage = page;

await LoadAsync(page);

}

private async Task LoadAsync(int page = 1)

{

var ok = await LoadListAsync(page);

if (ok)

{

await LoadPagesAsync();

}

}

private async Task<bool> LoadListAsync(int page)

{

var responseHttp = await Repository.GetAsync<List<Country>>($"api/countries?page={page}");

if (responseHttp.Error)

{

var message = await responseHttp.GetErrorMessageAsync();

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

return false;

}

Countries = responseHttp.Response;

return true;

}

private async Task LoadPagesAsync()

{

var responseHttp = await Repository.GetAsync<int>("api/countries/totalPages");

if (responseHttp.Error)

{

var message = await responseHttp.GetErrorMessageAsync();

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

return;

}

totalPages = responseHttp.Response;

}

…

1. Modificamos nuestro componente **CountriesIndex.razor**:

@page "/countries"

<h3>Paises</h3>

<div class="mb-3">

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

</div>

<GenericList MyList="Countries">

<Body>

<Pagination CurrentPage="currentPage"

TotalPages="totalPages"

SelectedPage="SelectedPageAsync" />

<table class="table table-striped">

<thead>

<tr>

<th>País</th>

<th style="width:210px">Estados / Departamentos</th>

<th style="width:210px"></th>

</tr>

</thead>

<tbody>

@foreach (var country in Countries!)

{

<tr>

<td>@country.Name</td>

<td>@country.StatesNumber</td>

<td>

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

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

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

</td>

</tr>

}

</tbody>

</table>

</Body>

</GenericList>

1. Probamos lo que llevamos hasta el momento.
2. Luego modificamos el **CountryDetails.razor.cs**:

…

private Country? country;

private List<State>? states;

private int currentPage = 1;

private int totalPages;

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

[Inject] private NavigationManager NavigationManager { get; set; } = null!;

[Inject] private SweetAlertService SweetAlertService { get; set; } = null!;

[Inject] private IRepository Repository { get; set; } = null!;

protected override async Task OnInitializedAsync()

{

await LoadAsync();

}

private async Task SelectedPageAsync(int page)

{

currentPage = page;

await LoadAsync(page);

}

private async Task LoadAsync(int page = 1)

{

var ok = await LoadCountryAsync();

if (ok)

{

ok = await LoadStatesAsync(page);

if (ok)

{

await LoadPagesAsync();

}

}

}

private async Task LoadPagesAsync()

{

var responseHttp = await Repository.GetAsync<int>($"api/states/totalPages?id={CountryId}");

if (responseHttp.Error)

{

var message = await responseHttp.GetErrorMessageAsync();

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

return;

}

totalPages = responseHttp.Response;

}

private async Task<bool> LoadStatesAsync(int page)

{

var responseHttp = await Repository.GetAsync<List<State>>($"api/states?id={CountryId}&page={page}");

if (responseHttp.Error)

{

var message = await responseHttp.GetErrorMessageAsync();

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

return false;

}

states = responseHttp.Response;

return true;

}

private async Task<bool> LoadCountryAsync()

{

var responseHttp = await Repository.GetAsync<Country>($"/api/countries/{CountryId}");

if (responseHttp.Error)

{

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

{

NavigationManager.NavigateTo("/countries");

return false;

}

var message = await responseHttp.GetErrorMessageAsync();

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

return false;

}

country = responseHttp.Response;

return true;

}

…

1. Luego modificamos el **CountryDetails.razor**:

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

@if (country is null)

{

<Loading />

}

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>

<h4>Estados/Departamentos</h4>

<GenericList MyList="states!">

<Body>

<Pagination CurrentPage="currentPage"

TotalPages="totalPages"

SelectedPage="SelectedPageAsync" />

<table class="table table-striped">

<thead>

<tr>

<th>Estado / Departamento</th>

<th style="width:90px">Ciudades</th>

<th style="width:210px"></th>

</tr>

</thead>

<tbody>

@foreach (var state in states!)

{

<tr>

<td>

@state.Name

</td>

<td>

@state.CitiesNumber

</td>

<td>

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

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

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

</td>

</tr>

}

</tbody>

</table>

</Body>

</GenericList>

}

1. Probamos.
2. Luego modificamos el **StateDetail.razor.cs**:

…

private State? state;

private List<City>? cities;

private int currentPage = 1;

private int totalPages;

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

[Inject] private NavigationManager NavigationManager { get; set; } = null!;

[Inject] private SweetAlertService SweetAlertService { get; set; } = null!;

[Inject] private IRepository Repository { get; set; } = null!;

protected override async Task OnInitializedAsync()

{

await LoadAsync();

}

private async Task SelectedPageAsync(int page)

{

currentPage = page;

await LoadAsync(page);

}

private async Task LoadAsync(int page = 1)

{

var ok = await LoadStateAsync();

if (ok)

{

ok = await LoadCitiesAsync(page);

if (ok)

{

await LoadPagesAsync();

}

}

}

private async Task LoadPagesAsync()

{

var responseHttp = await Repository.GetAsync<int>($"api/cities/totalPages?id={StateId}");

if (responseHttp.Error)

{

var message = await responseHttp.GetErrorMessageAsync();

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

return;

}

totalPages = responseHttp.Response;

}

private async Task<bool> LoadCitiesAsync(int page)

{

var responseHttp = await Repository.GetAsync<List<City>>($"api/cities?id={StateId}&page={page}");

if (responseHttp.Error)

{

var message = await responseHttp.GetErrorMessageAsync();

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

return false;

}

cities = responseHttp.Response;

return true;

}

private async Task<bool> LoadStateAsync()

{

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

if (responseHttp.Error)

{

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

{

NavigationManager.NavigateTo("/countries");

return false;

}

var message = await responseHttp.GetErrorMessageAsync();

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

return false;

}

state = responseHttp.Response;

return true;

}

…

1. Luego modificamos el **StateDetail.razor**:

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

@if (state is null)

{

<Loading />

}

else

{

<h3>@state.Name</h3>

<div class="mb-2">

<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>

<h4>Ciudades</h4>

<GenericList MyList="cities!">

<Body>

<Pagination CurrentPage="currentPage"

TotalPages="totalPages"

SelectedPage="SelectedPageAsync" />

<table class="table table-striped">

<thead>

<tr>

<th>Ciudad</th>

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

</tr>

</thead>

<tbody>

@foreach (var city in cities!)

{

<tr>

<td>

@city.Name

</td>

<td>

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

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

</td>

</tr>

}

</tbody>

</table>

</Body>

</GenericList>

}

1. Probamos.
2. Creamos más registros en el **SeedBd** para que las categorías paginen:

private async Task CheckCategoriesAsync()

{

if (!\_context.Categories.Any())

{

\_context.Categories.Add(new Category { Name = "Apple" });

\_context.Categories.Add(new Category { Name = "Autos" });

\_context.Categories.Add(new Category { Name = "Belleza" });

\_context.Categories.Add(new Category { Name = "Calzado" });

\_context.Categories.Add(new Category { Name = "Comida" });

\_context.Categories.Add(new Category { Name = "Cosmeticos" });

\_context.Categories.Add(new Category { Name = "Deportes" });

\_context.Categories.Add(new Category { Name = "Erótica" });

\_context.Categories.Add(new Category { Name = "Ferreteria" });

\_context.Categories.Add(new Category { Name = "Gamer" });

\_context.Categories.Add(new Category { Name = "Hogar" });

\_context.Categories.Add(new Category { Name = "Jardín" });

\_context.Categories.Add(new Category { Name = "Jugetes" });

\_context.Categories.Add(new Category { Name = "Lenceria" });

\_context.Categories.Add(new Category { Name = "Mascotas" });

\_context.Categories.Add(new Category { Name = "Nutrición" });

\_context.Categories.Add(new Category { Name = "Ropa" });

\_context.Categories.Add(new Category { Name = "Tecnología" });

await \_context.SaveChangesAsync();

}

}

1. Luego modificamos el **CategoriesIndex.razor.cs**:

…

private int currentPage = 1;

private int totalPages;

[Inject] private NavigationManager NavigationManager { get; set; } = null!;

[Inject] private SweetAlertService SweetAlertService { get; set; } = null!;

[Inject] private IRepository Repository { get; set; } = null!;

public List<Category>? Categories { get; set; }

protected override async Task OnInitializedAsync()

{

await LoadAsync();

}

private async Task SelectedPageAsync(int page)

{

currentPage = page;

await LoadAsync(page);

}

private async Task LoadAsync(int page = 1)

{

var ok = await LoadListAsync(page);

if (ok)

{

await LoadPagesAsync();

}

}

private async Task<bool> LoadListAsync(int page)

{

var responseHttp = await Repository.GetAsync<List<Category>>($"api/categories?page={page}");

if (responseHttp.Error)

{

var message = await responseHttp.GetErrorMessageAsync();

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

return false;

}

Categories = responseHttp.Response;

return true;

}

private async Task LoadPagesAsync()

{

var responseHttp = await Repository.GetAsync<int>("api/categories/totalPages");

if (responseHttp.Error)

{

var message = await responseHttp.GetErrorMessageAsync();

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

return;

}

totalPages = responseHttp.Response;

}

…

1. Luego modificamos el **CategoriesIndex.razor**:

@page "/categories"

<h3>Categorías</h3>

<div class="mb-3">

<a class="btn btn-primary" href="/categories/create">Nueva Categoría</a>

</div>

<GenericList MyList="Categories">

<Body>

<Pagination CurrentPage="currentPage"

TotalPages="totalPages"

SelectedPage="SelectedPageAsync" />

<table class="table table-striped">

<thead>

<tr>

<th>Categoría</th>

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

</tr>

</thead>

<tbody>

@foreach (var category in Categories!)

{

<tr>

<td>@category.Name</td>

<td>

<a href="/categories/edit/@category.Id" class="btn btn-sm btn-warning">Editar</a>

<button @onclick=@(() => DeleteAsycn(category)) class="btn btn-sm btn-danger">Borrar</button>

</td>

</tr>

}

</tbody>

</table>

</Body>

</GenericList>

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

## Agregando filtros

(<https://www.youtube.com/watch?v=DO5DrGUEEJw&list=PLuEZQoW9bRnRBThyGs208ZMrCYBRTvIg2&index=35>)

(<https://www.youtube.com/watch?v=NDd_HUAvzPU&list=PLuEZQoW9bRnRBThyGs208ZMrCYBRTvIg2&index=36>)

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

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

public string? Filter { get; set; }

1. Modificamos el **ICountriesRepository**:

Task<ActionResponse<int>> GetTotalPagesAsync(PaginationDTO pagination);

1. Modificamos el **CountriesRepository**:

public override async Task<ActionResponse<IEnumerable<Country>>> GetAsync(PaginationDTO pagination)

{

var queryable = \_context.Countries

.Include(c => c.States)

.AsQueryable();

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

{

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

}

return new ActionResponse<IEnumerable<Country>>

{

WasSuccess = true,

Result = await queryable

.OrderBy(c => c.Name)

.Paginate(pagination)

.ToListAsync()

};

}

public override async Task<ActionResponse<int>> GetTotalPagesAsync(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();

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

return new ActionResponse<int>

{

WasSuccess = true,

Result = totalPages

};

}

1. Modificamos el **StatesRepository**:

public override async Task<ActionResponse<IEnumerable<State>>> GetAsync(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 new ActionResponse<IEnumerable<State>>

{

WasSuccess = true,

Result = await queryable

.OrderBy(x => x.Name)

.Paginate(pagination)

.ToListAsync()

};

}

public override async Task<ActionResponse<int>> GetTotalPagesAsync(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();

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

return new ActionResponse<int>

{

WasSuccess = true,

Result = totalPages

};

}

1. Modificamos el **CitiesRepository**:

public override async Task<ActionResponse<IEnumerable<City>>> GetAsync(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 new ActionResponse<IEnumerable<City>>

{

WasSuccess = true,

Result = await queryable

.OrderBy(x => x.Name)

.Paginate(pagination)

.ToListAsync()

};

}

public override async Task<ActionResponse<int>> GetTotalPagesAsync(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();

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

return new ActionResponse<int>

{

WasSuccess = true,

Result = totalPages

};

}

1. Agregamos el **ICategoriesRepository**:

using Orders.Shared.DTOs;

using Orders.Shared.Entites;

using Orders.Shared.Responses;

namespace Orders.Backend.Repositories

{

public interface ICategoriesRepository

{

Task<ActionResponse<IEnumerable<Category>>> GetAsync(PaginationDTO pagination);

Task<ActionResponse<int>> GetTotalPagesAsync(PaginationDTO pagination);

}

}

1. Creamos el **CategoriesRepository**:

using Microsoft.EntityFrameworkCore;

using Orders.Backend.Data;

using Orders.Backend.Helpers;

using Orders.Shared.DTOs;

using Orders.Shared.Entites;

using Orders.Shared.Responses;

namespace Orders.Backend.Repositories

{

public class CategoriesRepository : GenericRepository<Category>, ICategoriesRepository

{

private readonly DataContext \_context;

public CategoriesRepository(DataContext context) : base(context)

{

\_context = context;

}

public override async Task<ActionResponse<IEnumerable<Category>>> GetAsync(PaginationDTO pagination)

{

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

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

{

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

}

return new ActionResponse<IEnumerable<Category>>

{

WasSuccess = true,

Result = await queryable

.OrderBy(x => x.Name)

.Paginate(pagination)

.ToListAsync()

};

}

public override async Task<ActionResponse<int>> GetTotalPagesAsync(PaginationDTO pagination)

{

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

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

{

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

}

double count = await queryable.CountAsync();

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

return new ActionResponse<int>

{

WasSuccess = true,

Result = totalPages

};

}

}

}

1. Modificamos el **ICountriesUnitOfWork**:

Task<ActionResponse<int>> GetTotalPagesAsync(PaginationDTO pagination);

1. Modificamos el **CountriesUnitOfWork**:

public override async Task<ActionResponse<int>> GetTotalPagesAsync(PaginationDTO pagination) => await \_countriesRepository.GetTotalPagesAsync(pagination);

1. Agregamos el **ICategoriesUnitOfWork**:

using Orders.Shared.DTOs;

using Orders.Shared.Entites;

using Orders.Shared.Responses;

namespace Orders.Backend.UnitsOfWork

{

public interface ICategoriesUnitOfWork

{

Task<ActionResponse<IEnumerable<Category>>> GetAsync(PaginationDTO pagination);

Task<ActionResponse<int>> GetTotalPagesAsync(PaginationDTO pagination);

}

}

1. Agregamos el **CategoriesUnitOfWork**:

using Orders.Backend.Repositories;

using Orders.Shared.DTOs;

using Orders.Shared.Entites;

using Orders.Shared.Responses;

namespace Orders.Backend.UnitsOfWork

{

public class CategoriesUnitOfWork : GenericUnitOfWork<Category>, ICategoriesUnitOfWork

{

private readonly ICategoriesRepository \_categoriesRepository;

public CategoriesUnitOfWork(IGenericRepository<Category> repository, ICategoriesRepository categoriesRepository) : base(repository)

{

\_categoriesRepository = categoriesRepository;

}

public override async Task<ActionResponse<IEnumerable<Category>>> GetAsync(PaginationDTO pagination) => await \_categoriesRepository.GetAsync(pagination);

public override async Task<ActionResponse<int>> GetTotalPagesAsync(PaginationDTO pagination) => await \_categoriesRepository.GetTotalPagesAsync(pagination);

}

}

1. Agregamos las nuevas inyecciones al **Program**:

builder.Services.AddScoped<ICategoriesRepository, CategoriesRepository>();

builder.Services.AddScoped<ICitiesRepository, CitiesRepository>();

builder.Services.AddScoped<ICountriesRepository, CountriesRepository>();

builder.Services.AddScoped<IStatesRepository, StatesRepository>();

builder.Services.AddScoped<ICategoriesUnitOfWork, CategoriesUnitOfWork>();

builder.Services.AddScoped<ICitiesUnitOfWork, CitiesUnitOfWork>();

builder.Services.AddScoped<ICountriesUnitOfWork, CountriesUnitOfWork>();

builder.Services.AddScoped<IStatesUnitOfWork, StatesUnitOfWork>();

1. Modificamos el controlador **CountriesController**:

[HttpGet("totalPages")]

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

{

var action = await \_countriesUnitOfWork.GetTotalPagesAsync(pagination);

if (action.WasSuccess)

{

return Ok(action.Result);

}

return BadRequest();

}

1. Modificamos el controlador **CategoriesController**:

using Microsoft.AspNetCore.Mvc;

using Orders.Backend.UnitsOfWork;

using Orders.Shared.DTOs;

using Orders.Shared.Entites;

namespace Orders.Backend.Controllers

{

[ApiController]

[Route("api/[controller]")]

public class CategoriesController : GenericController<Category>

{

private readonly ICategoriesUnitOfWork \_categoriesUnitOfWork;

public CategoriesController(IGenericUnitOfWork<Category> unit, ICategoriesUnitOfWork categoriesUnitOfWork) : base(unit)

{

\_categoriesUnitOfWork = categoriesUnitOfWork;

}

[HttpGet]

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

{

var response = await \_categoriesUnitOfWork.GetAsync(pagination);

if (response.WasSuccess)

{

return Ok(response.Result);

}

return BadRequest();

}

[HttpGet("totalPages")]

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

{

var action = await \_categoriesUnitOfWork.GetTotalPagesAsync(pagination);

if (action.WasSuccess)

{

return Ok(action.Result);

}

return BadRequest();

}

}

}

1. Probamos los filtros por Swagger.
2. En el projecto **Frontend** modificamos el **CountriesIndex.razor.cs**:

…

private int currentPage = 1;

private int totalPages;

[Inject] private NavigationManager NavigationManager { get; set; } = null!;

[Inject] private SweetAlertService SweetAlertService { get; set; } = null!;

[Inject] private IRepository Repository { get; set; } = null!;

[Parameter, SupplyParameterFromQuery] public string Page { get; set; } = string.Empty;

[Parameter, SupplyParameterFromQuery] public string Filter { get; set; } = string.Empty;

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

protected override async Task OnInitializedAsync()

{

await LoadAsync();

}

private async Task SelectedPageAsync(int page)

{

currentPage = page;

await LoadAsync(page);

}

private async Task LoadAsync(int page = 1)

{

if (!string.IsNullOrWhiteSpace(Page))

{

page = Convert.ToInt32(Page);

}

var ok = await LoadListAsync(page);

if (ok)

{

await LoadPagesAsync();

}

}

private async Task<bool> LoadListAsync(int page)

{

var url = $"api/countries?page={page}";

if (!string.IsNullOrEmpty(Filter))

{

url += $"&filter={Filter}";

}

var responseHttp = await Repository.GetAsync<List<Country>>(url);

if (responseHttp.Error)

{

var message = await responseHttp.GetErrorMessageAsync();

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

return false;

}

Countries = responseHttp.Response;

return true;

}

private async Task LoadPagesAsync()

{

var url = "api/countries/totalPages";

if (!string.IsNullOrEmpty(Filter))

{

url += $"?filter={Filter}";

}

var responseHttp = await Repository.GetAsync<int>(url);

if (responseHttp.Error)

{

var message = await responseHttp.GetErrorMessageAsync();

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

return;

}

totalPages = responseHttp.Response;

}

private async Task CleanFilterAsync()

{

Filter = string.Empty;

await ApplyFilterAsync();

}

private async Task ApplyFilterAsync()

{

int page = 1;

await LoadAsync(page);

await SelectedPageAsync(page);

}

…

1. En el projecto **Frontend** modificamos el **CountriesIndex.razor**:

…

<Pagination CurrentPage="currentPage"

TotalPages="totalPages"

SelectedPage="SelectedPageAsync" />

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

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

<button type="button" class="btn btn-outline-primary mx-1" @onclick="ApplyFilterAsync">Filtrar</button>

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

</div>

<table class="table table-striped">

…

1. Probamos lo que llevamos.
2. En el projecto **Frontend** modificamos el **CountryDetails.razor.cs**:

…

private Country? country;

private List<State>? states;

private int currentPage = 1;

private int totalPages;

[Inject] private NavigationManager NavigationManager { get; set; } = null!;

[Inject] private SweetAlertService SweetAlertService { get; set; } = null!;

[Inject] private IRepository Repository { get; set; } = null!;

[Parameter, SupplyParameterFromQuery] public string Page { get; set; } = string.Empty;

[Parameter, SupplyParameterFromQuery] public string Filter { get; set; } = string.Empty;

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

protected override async Task OnInitializedAsync()

{

await LoadAsync();

}

private async Task SelectedPageAsync(int page)

{

currentPage = page;

await LoadAsync(page);

}

private async Task LoadAsync(int page = 1)

{

if (!string.IsNullOrWhiteSpace(Page))

{

page = Convert.ToInt32(Page);

}

var ok = await LoadCountryAsync();

if (ok)

{

ok = await LoadStatesAsync(page);

if (ok)

{

await LoadPagesAsync();

}

}

}

private async Task LoadPagesAsync()

{

var url = $"api/states/totalPages?id={CountryId}";

if (!string.IsNullOrEmpty(Filter))

{

url += $"&filter={Filter}";

}

var responseHttp = await Repository.GetAsync<int>(url);

if (responseHttp.Error)

{

var message = await responseHttp.GetErrorMessageAsync();

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

return;

}

totalPages = responseHttp.Response;

}

private async Task<bool> LoadStatesAsync(int page)

{

var url = $"api/states?id={CountryId}&page={page}";

if (!string.IsNullOrEmpty(Filter))

{

url += $"&filter={Filter}";

}

var responseHttp = await Repository.GetAsync<List<State>>(url);

if (responseHttp.Error)

{

var message = await responseHttp.GetErrorMessageAsync();

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

return false;

}

states = responseHttp.Response;

return true;

}

private async Task CleanFilterAsync()

{

Filter = string.Empty;

await ApplyFilterAsync();

}

private async Task ApplyFilterAsync()

{

int page = 1;

await LoadAsync(page);

await SelectedPageAsync(page);

}

…

1. En el projecto **Frontend** modificamos el **StateDetails.razor.cs**:

private State? state;

private List<City>? cities;

private int currentPage = 1;

private int totalPages;

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

[Inject] private NavigationManager NavigationManager { get; set; } = null!;

[Inject] private SweetAlertService SweetAlertService { get; set; } = null!;

[Inject] private IRepository Repository { get; set; } = null!;

[Parameter, SupplyParameterFromQuery] public string Page { get; set; } = string.Empty;

[Parameter, SupplyParameterFromQuery] public string Filter { get; set; } = string.Empty;

protected override async Task OnInitializedAsync()

{

await LoadAsync();

}

private async Task SelectedPageAsync(int page)

{

currentPage = page;

await LoadAsync(page);

}

private async Task LoadAsync(int page = 1)

{

if (!string.IsNullOrWhiteSpace(Page))

{

page = Convert.ToInt32(Page);

}

var ok = await LoadStateAsync();

if (ok)

{

ok = await LoadCitiesAsync(page);

if (ok)

{

await LoadPagesAsync();

}

}

}

private async Task LoadPagesAsync()

{

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

if (!string.IsNullOrEmpty(Filter))

{

url += $"&filter={Filter}";

}

var responseHttp = await Repository.GetAsync<int>(url);

if (responseHttp.Error)

{

var message = await responseHttp.GetErrorMessageAsync();

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

return;

}

totalPages = responseHttp.Response;

}

private async Task<bool> LoadCitiesAsync(int page)

{

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

if (!string.IsNullOrEmpty(Filter))

{

url += $"&filter={Filter}";

}

var responseHttp = await Repository.GetAsync<List<City>>(url);

if (responseHttp.Error)

{

var message = await responseHttp.GetErrorMessageAsync();

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

return false;

}

cities = responseHttp.Response;

return true;

}

private async Task CleanFilterAsync()

{

Filter = string.Empty;

await ApplyFilterAsync();

}

private async Task ApplyFilterAsync()

{

int page = 1;

await LoadAsync(page);

await SelectedPageAsync(page);

}

…

1. En el projecto **Frontend** modificamos el **StateDetails.razor**:

…

<Pagination CurrentPage="currentPage"

TotalPages="totalPages"

SelectedPage="SelectedPageAsync" />

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

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

<button type="button" class="btn btn-outline-primary mx-1" @onclick="ApplyFilterAsync">Filtrar</button>

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

</div>

<table class="table table-striped">

…

1. Probamos.
2. En el projecto **Frontend** modificamos el **CategoriesIndex.razor.cs**:

…

private int currentPage = 1;

private int totalPages;

[Inject] private NavigationManager NavigationManager { get; set; } = null!;

[Inject] private SweetAlertService SweetAlertService { get; set; } = null!;

[Inject] private IRepository Repository { get; set; } = null!;

[Parameter, SupplyParameterFromQuery] public string Page { get; set; } = string.Empty;

[Parameter, SupplyParameterFromQuery] public string Filter { get; set; } = string.Empty;

public List<Category>? Categories { get; set; }

protected override async Task OnInitializedAsync()

{

await LoadAsync();

}

private async Task SelectedPageAsync(int page)

{

currentPage = page;

await LoadAsync(page);

}

private async Task LoadAsync(int page = 1)

{

if (!string.IsNullOrWhiteSpace(Page))

{

page = Convert.ToInt32(Page);

}

var ok = await LoadListAsync(page);

if (ok)

{

await LoadPagesAsync();

}

}

private async Task<bool> LoadListAsync(int page)

{

var url = $"api/categories/?page={page}";

if (!string.IsNullOrEmpty(Filter))

{

url += $"&filter={Filter}";

}

var responseHttp = await Repository.GetAsync<List<Category>>(url);

if (responseHttp.Error)

{

var message = await responseHttp.GetErrorMessageAsync();

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

return false;

}

Categories = responseHttp.Response;

return true;

}

private async Task LoadPagesAsync()

{

var url = $"api/categories/totalPages";

if (!string.IsNullOrEmpty(Filter))

{

url += $"?filter={Filter}";

}

var responseHttp = await Repository.GetAsync<int>(url);

if (responseHttp.Error)

{

var message = await responseHttp.GetErrorMessageAsync();

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

return;

}

totalPages = responseHttp.Response;

}

private async Task CleanFilterAsync()

{

Filter = string.Empty;

await ApplyFilterAsync();

}

private async Task ApplyFilterAsync()

{

int page = 1;

await LoadAsync(page);

await SelectedPageAsync(page);

}

…

1. En el projecto **Frontend** modificamos el **CategoriesIndex.razor**:

…

<Pagination CurrentPage="currentPage"

TotalPages="totalPages"

SelectedPage="SelectedPageAsync" />

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

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

<button type="button" class="btn btn-outline-primary mx-1" @onclick="ApplyFilterAsync">Filtrar</button>

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

</div>

…

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

## Creando las tablas de usuarios

(<https://www.youtube.com/watch?v=hQA64AXO_gQ&list=PLuEZQoW9bRnRBThyGs208ZMrCYBRTvIg2&index=37>)

(<https://www.youtube.com/watch?v=SFBhK20hpp8&list=PLuEZQoW9bRnRBThyGs208ZMrCYBRTvIg2&index=39>)

(<https://www.youtube.com/watch?v=HfcpwpvUFOg&list=PLuEZQoW9bRnRBThyGs208ZMrCYBRTvIg2&index=40>)

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**:

using System.ComponentModel;

namespace Orders.Shared.Enums

{

public enum UserType

{

[Description("Administrador")]

Admin,

[Description("Usuario")]

User

}

}

1. En el proyecto **Shared** el nuget **Microsoft.AspNetCore.Identity.EntityFrameworkCore**.
2. En el proyecto **Shared** en la carpeta **Entities**, crear la entidad **User**:

using Microsoft.AspNetCore.Identity;

using Orders.Shared.Enums;

using System.ComponentModel.DataAnnotations;

namespace Orders.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 **Backend** instalar el nugget **Microsoft.AspNetCore.Identity.EntityFrameworkCore**.
2. Modificar el **DataContext**:

public class DataContext : IdentityDbContext<User>

1. Creamos el **IUsersRepository**:

using Microsoft.AspNetCore.Identity;

using Orders.Shared.Entities;

namespace Orders.Backend.Repositories.Interfaces

{

public interface IUsersRepository

{

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. Creamos el **UsersRepository**:

using Microsoft.AspNetCore.Identity;

using Microsoft.EntityFrameworkCore;

using Orders.Backend.Data;

using Orders.Backend.Repositories.Interfaces;

using Orders.Shared.Entities;

namespace Orders.Backend.Repositories.Implementations

{

public class UsersRepository : IUsersRepository

{

private readonly DataContext \_context;

private readonly UserManager<User> \_userManager;

private readonly RoleManager<IdentityRole> \_roleManager;

public UsersRepository(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)

{

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

if (!roleExists)

{

await \_roleManager.CreateAsync(new IdentityRole

{

Name = roleName

});

}

}

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<bool> IsUserInRoleAsync(User user, string roleName)

{

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

}

}

}

1. Creamos el **IUsersUnitOfWork**:

using Microsoft.AspNetCore.Identity;

using Orders.Shared.Entities;

namespace Orders.Backend.UnitsOfWork.Interfaces

{

public interface IUsersUnitOfWork

{

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. Creamos el **UsersUnitOfWork**:

using Microsoft.AspNetCore.Identity;

using Orders.Backend.Repositories.Interfaces;

using Orders.Backend.UnitsOfWork.Interfaces;

using Orders.Shared.Entities;

namespace Orders.Backend.UnitsOfWork.Implementations

{

public class UsersUnitOfWork : IUsersUnitOfWork

{

private readonly IUsersRepository \_usersRepository;

public UsersUnitOfWork(IUsersRepository usersRepository)

{

\_usersRepository = usersRepository;

}

public async Task<IdentityResult> AddUserAsync(User user, string password) => await \_usersRepository.AddUserAsync(user, password);

public async Task AddUserToRoleAsync(User user, string roleName) => await \_usersRepository.AddUserToRoleAsync(user, roleName);

public async Task CheckRoleAsync(string roleName) => await \_usersRepository.CheckRoleAsync(roleName);

public async Task<User> GetUserAsync(string email) => await \_usersRepository.GetUserAsync(email);

public async Task<bool> IsUserInRoleAsync(User user, string roleName) => await \_usersRepository.IsUserInRoleAsync(user, roleName);

}

}

1. Matriculamos la nueva inyección en el **Program** del proyecto **Backend**, y otras modificaciones para configurar el manejo de usuarios:

builder.Services.AddScoped<ICategoriesRepository, CategoriesRepository>();

builder.Services.AddScoped<ICitiesRepository, CitiesRepository>();

builder.Services.AddScoped<ICountriesRepository, CountriesRepository>();

builder.Services.AddScoped<IStatesRepository, StatesRepository>();

builder.Services.AddScoped<IUsersRepository, UsersRepository>();

builder.Services.AddScoped<ICategoriesUnitOfWork, CategoriesUnitOfWork>();

builder.Services.AddScoped<ICitiesUnitOfWork, CitiesUnitOfWork>();

builder.Services.AddScoped<ICountriesUnitOfWork, CountriesUnitOfWork>();

builder.Services.AddScoped<IStatesUnitOfWork, StatesUnitOfWork>();

builder.Services.AddScoped<IUsersUnitOfWork, UsersUnitOfWork>();

builder.Services.AddTransient<SeedDb>();

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();

var app = builder.Build();

1. Modificamos el **SeedDb**:

…

private readonly DataContext \_context;

private readonly IApiService \_apiService;

private readonly IUsersUnitOfWork \_usersUnitOfWork;

public SeedDb(DataContext context, IApiService apiService, IUsersUnitOfWork usersUnitOfWork)

{

\_context = context;

\_apiService = apiService;

\_usersUnitOfWork = usersUnitOfWork;

}

public async Task SeedAsync()

{

await \_context.Database.EnsureCreatedAsync();

//await CheckCountriesAsync();

await CheckCountriesFullAsync();

await CheckCategoriesAsync();

await CheckRolesAsync();

await CheckUserAsync("1010", "Juan", "Zuluaga", "zulu@yopmail.com", "322 311 4620", "Calle Luna Calle Sol", UserType.Admin);

}

private async Task CheckRolesAsync()

{

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

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

}

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

{

var user = await \_usersUnitOfWork.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 \_usersUnitOfWork.AddUserAsync(user, "123456");

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

}

return user;

}

…

1. Corremos los siguientes comandos:

PM> drop-database

PM> add-migration AddUsersEntities

PM> update-database

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

## Creando sistema de seguridad

(<https://www.youtube.com/watch?v=HfcpwpvUFOg&list=PLuEZQoW9bRnRBThyGs208ZMrCYBRTvIg2&index=40>)

1. Al proyecto **Frontend** agregamos el paquete:

**Microsoft.AspNetCore.Components.WebAssembly.Authentication**.

1. Agregamos este using en el **\_Imports**:

@using Microsoft.AspNetCore.Components.Authorization

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

using System.Security.Claims;

using Microsoft.AspNetCore.Components.Authorization;

namespace Orders.Frontend.AuthenticationProviders

{

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 **Frontend**:

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**:

<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. Modificacmos el **Home.razor**.

@page "/"

<AuthorizeView>

<p>Estas autenticado</p>

</AuthorizeView>

1. Modificamos el **AuthenticationProviderTest**:

public override async Task<AuthenticationState> GetAuthenticationStateAsync()

{

var anonimous = new ClaimsIdentity();

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

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

}

1. Cambiamos el **Home.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 **user**.
2. Modificamos nuestro **AuthenticationProviderTest**, para agregar algunos **Claims**:

public override async Task<AuthenticationState> GetAuthenticationStateAsync()

{

var anonimous = new ClaimsIdentity();

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

var admin = new ClaimsIdentity(new List<Claim>

{

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

new Claim("LastName", "Zulu"),

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

},

authenticationType: "test");

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

}

1. Modificamos el **Home.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 admin = new ClaimsIdentity(new List<Claim>

{

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

new Claim("LastName", "Zulu"),

new Claim(ClaimTypes.Name, "zulu@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:

<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="bi bi-house-door-fill-nav-menu" aria-hidden="true"></span> Inicio

</NavLink>

</div>

<AuthorizeView Roles="Admin">

<Authorized>

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

<NavLink class="nav-link" href="/categories">

<span class="bi bi-list-check-fill-nav-menu" aria-hidden="true"></span> Categorías

</NavLink>

</div>

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

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

<span class="bi bi-globe-americas-fill-nav-menu" aria-hidden="true"></span> Paises

</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:

[Authorize(Roles = "Admin")]

1. Ahora si queremos personalizar el mensaje podemos modificar nuestro **App.razor**:

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

<Authorizing>

<p>Autorizando...</p>

</Authorizing>

<NotAuthorized>

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

</NotAuthorized>

</AuthorizeRouteView>

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

## Seguridad desde el backend

(<https://www.youtube.com/watch?v=ou_I4fY8ing&list=PLuEZQoW9bRnRBThyGs208ZMrCYBRTvIg2&index=41>)

(<https://www.youtube.com/watch?v=jrDSAQgPumU&list=PLuEZQoW9bRnRBThyGs208ZMrCYBRTvIg2&index=42>)

(<https://www.youtube.com/watch?v=jrDSAQgPumU&list=PLuEZQoW9bRnRBThyGs208ZMrCYBRTvIg2&index=43>)

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

"jwtKey": "[Put your own long key]",

"Logging": {

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

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 Orders.Shared.Entities;

using System.ComponentModel.DataAnnotations;

using System.Xml.Linq;

namespace Orders.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 Orders.Shared.Entities;

namespace Orders.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 Orders.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 **IUsersRepository**:

Task<SignInResult> LoginAsync(LoginDTO model);

Task LogoutAsync();

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

…

private readonly DataContext \_context;

private readonly UserManager<User> \_userManager;

private readonly RoleManager<IdentityRole> \_roleManager;

private readonly SignInManager<User> \_signInManager;

public UsersRepository(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. Agregamos estos métodos al **IUsersUnitOfWork**:

Task<SignInResult> LoginAsync(LoginDTO model);

Task LogoutAsync();

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

public async Task<SignInResult> LoginAsync(LoginDTO model) => await \_usersRepository.LoginAsync(model);

public async Task LogoutAsync() => await \_usersRepository.LogoutAsync();

1. Creamos el **AccountsController**:

using Microsoft.AspNetCore.Mvc;

using Microsoft.IdentityModel.Tokens;

using Orders.Backend.UnitsOfWork.Interfaces;

using Orders.Shared.DTOs;

using Orders.Shared.Entities;

using System.IdentityModel.Tokens.Jwt;

using System.Security.Claims;

using System.Text;

namespace Orders.Backend.Controllers

{

[ApiController]

[Route("/api/accounts")]

public class AccountsController : ControllerBase

{

private readonly IUsersUnitOfWork \_usersUnitOfWork;

private readonly IConfiguration \_configuration;

public AccountsController(IUsersUnitOfWork usersUnitOfWork, IConfiguration configuration)

{

\_usersUnitOfWork = usersUnitOfWork;

\_configuration = configuration;

}

[HttpPost("CreateUser")]

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

{

User user = model;

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

if (result.Succeeded)

{

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

return Ok(BuildToken(user));

}

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

}

[HttpPost("Login")]

public async Task<IActionResult> LoginAsync([FromBody] LoginDTO model)

{

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

if (result.Succeeded)

{

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

return Ok(BuildToken(user));

}

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

}

private TokenDTO BuildToken(User user)

{

var claims = new List<Claim>

{

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

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

new("Document", user.Document),

new("FirstName", user.FirstName),

new("LastName", user.LastName),

new("Address", user.Address),

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

new("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 4 controladores **CountriesController**, **StatesController,** **CitiesController** y **CategoriesController**:

[Authorize(AuthenticationSchemes = JwtBearerDefaults.AuthenticationScheme)]

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 Frontend, y nos debe salir un error porque aun no le mandamos ningún token a nuestra Backend. Hacemos el **commit**.

## Habilitando tokens en swagger

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

builder.Services.AddSwaggerGen(c =>

{

c.SwaggerDoc("v1", new OpenApiInfo { Title = "Orders Backend", 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**.

## Implementando el registro de usuarios, login & logout

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

using Microsoft.JSInterop;

namespace Orders. Frontend.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  **Frontend** en la carpeta **Services** creamos el **ILoginService**:

namespace Orders. Frontend.Auth

{

public interface ILoginService

{

Task LoginAsync(string token);

Task LogoutAsync();

}

}

1. En el proyecto  **Frontend** en la carpeta **AuthenticationProviders** creamos el **AuthenticationProviderJWT**:

using System.IdentityModel.Tokens.Jwt;

using System.Net.Http.Headers;

using System.Security.Claims;

using Microsoft.AspNetCore.Components.Authorization;

using Microsoft.JSInterop;

using Orders.Frontend.Helpers;

using Orders.Frontend.Services;

namespace Orders.Frontend.AuthenticationProviders

{

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 override async 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  **Frontend** 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 el componente compartido **AuthLinks.razor**:

<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.razor** y **Register.razor.cs**:

using CurrieTechnologies.Razor.SweetAlert2;

using Microsoft.AspNetCore.Components;

using Orders.Frontend.Repositories;

using Orders.Frontend.Services;

using Orders.Shared.DTOs;

using Orders.Shared.Enums;

namespace Orders.Frontend.Pages.Auth

{

public partial class Register

{

private UserDTO userDTO = new();

[Inject] private NavigationManager NavigationManager { get; set; } = null!;

[Inject] private SweetAlertService SweetAlertService { get; set; } = null!;

[Inject] private IRepository Repository { get; set; } = null!;

[Inject] private ILoginService LoginService { get; set; } = null!;

private async Task CreteUserAsync()

{

userDTO.UserName = userDTO.Email;

userDTO.UserType = UserType.User;

var responseHttp = await Repository.PostAsync<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. Luego modificamos el **Register.razor**:

@page "/Register"

<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>

1. Probamos.
2. Dentro de **Pages** en la carpeta **Auth** creamos el componente **Login.razor** y **Login.razor.cs**:

using CurrieTechnologies.Razor.SweetAlert2;

using Microsoft.AspNetCore.Components;

using Orders.Frontend.Repositories;

using Orders.Frontend.Services;

using Orders.Shared.DTOs;

namespace Orders.Frontend.Pages.Auth

{

public partial class Login

{

private LoginDTO loginDTO = new();

[Inject] private NavigationManager NavigationManager { get; set; } = null!;

[Inject] private SweetAlertService SweetAlertService { get; set; } = null!;

[Inject] private IRepository Repository { get; set; } = null!;

[Inject] private ILoginService LoginService { get; set; } = null!;

private async Task LoginAsync()

{

var responseHttp = await Repository.PostAsync<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. Luego modificamos el **Login.razor**:

@page "/Login"

<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>

1. Dentro de **Pages** en la carpeta **Auth** creamos el componente **Logout.razor** y **Logout.razor.cs**:

using Microsoft.AspNetCore.Components;

using Orders.Frontend.Services;

namespace Orders.Frontend.Pages.Auth

{

public partial class Logout

{

[Inject] private NavigationManager NavigationManager { get; set; } = null!;

[Inject] private ILoginService LoginService { get; set; } = null!;

protected override async Task OnInitializedAsync()

{

await LoginService.LogoutAsync();

NavigationManager.NavigateTo("/");

}

}

}

1. Modificamos el **Logout.razor**:

@page "/logout"

<p>Cerrando sesión...</p>

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

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

1. Modificamos el **ICountriesRepository**:

Task<IEnumerable<Country>> GetComboAsync();

1. Modificamos el **CountriesRepository**:

public async Task<IEnumerable<Country>> GetComboAsync()

{

return await \_context.Countries

.OrderBy(c => c.Name)

.ToListAsync();

}

1. Modificamos el **ICountriesUnitOfWork**:

Task<IEnumerable<Country>> GetComboAsync();

1. Modificamos el **CountriesUnitOfWork**:

public async Task<IEnumerable<Country>> GetComboAsync() => await \_countriesRepository.GetComboAsync();

1. Modificamos el **IStatesRepository**:

Task<IEnumerable<State>> GetComboAsync(int countryId);

1. Modificamos el **StatesRepository**:

public async Task<IEnumerable<State>> GetComboAsync(int countryId)

{

return await \_context.States

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

.OrderBy(s => s.Name)

.ToListAsync();

}

1. Modificamos el **IStatesUnitOfWork**:

Task<IEnumerable<State>> GetComboAsync(int countryId);

1. Modificamos el **StatesUnitOfWork**:

public async Task<IEnumerable<State>> GetComboAsync(int countryId) => await \_statesRepository.GetComboAsync(countryId);

1. Modificamos el **ICitiesRepository**:

Task<IEnumerable<City>> GetComboAsync(int stateId);

1. Modificamos el **CitiesRepository**:

public async Task<IEnumerable<City>> GetComboAsync(int stateId)

{

return await \_context.Cities

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

.OrderBy(c => c.Name)

.ToListAsync();

}

1. Modificamos el **ICitiesUnitOfWork**:

Task<IEnumerable<City>> GetComboAsync(int stateId);

1. Modificamos el **CitiesUnitOfWork**:

public async Task<IEnumerable<City>> GetComboAsync(int stateId) => await \_citiesRepository.GetComboAsync(stateId);

1. Modificamos el **ICategoriesRepository**:

Task<IEnumerable<Category>> GetComboAsync();

1. Modificamos el **CategoriesRepository**:

public async Task<IEnumerable<Category>> GetComboAsync()

{

return await \_context.Categories

.OrderBy(c => c.Name)

.ToListAsync();

}

1. Modificamos el **ICategoriesUnitOfWork**:

Task<IEnumerable<Category>> GetComboAsync();

1. Modificamos el **CategoriesUnitOfWork**:

public async Task<IEnumerable<Category>> GetComboAsync() => await \_categoriesRepository.GetComboAsync();

1. Modificamos el **CountriesController**:

[AllowAnonymous]

[HttpGet("combo")]

public async Task<IActionResult> GetComboAsync()

{

return Ok(await \_countriesUnitOfWork.GetComboAsync());

}

1. Modificamos el **StatesController**:

[AllowAnonymous]

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

public async Task<IActionResult> GetComboAsync(int countryId)

{

return Ok(await \_statesUnitOfWork.GetComboAsync(countryId));

}

1. Modificamos el **CitiesController**:

[AllowAnonymous]

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

public async Task<IActionResult> GetComboAsync(int stateId)

{

return Ok(await \_citiesUnitOfWork.GetComboAsync(stateId));

}

1. Modificamos el **CategoriesController**:

[AllowAnonymous]

[HttpGet("combo")]

public async Task<IActionResult> GetComboAsync()

{

return Ok(await \_categoriesUnitOfWork.GetComboAsync());

}

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

…

private UserDTO userDTO = new();

private List<Country>? countries;

private List<State>? states;

private List<City>? cities;

private bool loading;

[Inject] private NavigationManager NavigationManager { get; set; } = null!;

[Inject] private SweetAlertService SweetAlertService { get; set; } = null!;

[Inject] private IRepository Repository { get; set; } = null!;

[Inject] private ILoginService LoginService { get; set; } = null!;

protected override async Task OnInitializedAsync()

{

await LoadCountriesAsync();

}

private async Task CountryChangedAsync(ChangeEventArgs e)

{

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

states = null;

cities = null;

userDTO.CityId = 0;

await LoadStatesAsyn(selectedCountry);

}

private async Task StateChangedAsync(ChangeEventArgs e)

{

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

cities = null;

userDTO.CityId = 0;

await LoadCitiesAsyn(selectedState);

}

private async Task LoadCountriesAsync()

{

var responseHttp = await Repository.GetAsync<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.GetAsync<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.GetAsync<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()

{

userDTO.UserName = userDTO.Email;

userDTO.UserType = UserType.User;

loading = true;

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

loading = false;

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. Modificamos el **Register.razor**:

…

<h3>Registrar Nuevo Usuario</h3>

@if (loading)

{

<Loading />

}

else

{

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

…

<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>

…

</EditForm>

}

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

## Mejorando un poco la interfaz de usuario

1. Luego modificamos nuestro **CountriesIndex.razor**:

@page "/countries"

<div class="card">

<div class="card-header">

<span>

<i class="bi bi-globe-americas"/> Países

<a class="btn btn-primary btn-sm float-end" href="/countries/create"><i class="bi bi-plus-square" /> Nuevo País</a>

</span>

</div>

<div class="card-body">

<GenericList MyList="Countries">

<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="bi bi-funnel" /> Filtrar</button>

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

</div>

</div>

<Pagination CurrentPage="currentPage"

TotalPages="totalPages"

SelectedPage="SelectedPageAsync" />

<table class="table table-striped">

<thead>

<tr>

<th>País</th>

<th style="width:210px">Estados / Departamentos</th>

<th style="width:168px"></th>

</tr>

</thead>

<tbody>

@foreach (var country in Countries!)

{

<tr>

<td><a href="/countries/details/@country.Id"> @country.Name</a></td>

<td>@country.StatesNumber</td>

<td>

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

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

</td>

</tr>

}

</tbody>

</table>

</Body>

</GenericList>

</div>

</div>

1. Luego modificamos nuestro **CountryDetails**:

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

@if (country is null)

{

<Loading />

}

else

{

<div class="card">

<div class="card-header">

<span>

<i class="bi bi-globe-americas" /> @country.Name

<a class="btn btn-sm btn-primary float-end mx-1" href="/states/create/@country.Id"><i class="bi bi-plus-square" /> Adicionar Estado/Departamento</a>

<a class="btn btn-sm btn-success float-end" href="/countries"><i class="bi bi-arrow-left" /> Regresar</a>

</span>

</div>

<div class="card-body">

<GenericList MyList="states!">

<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 estado/departamento..." @bind-value="Filter" />

</div>

<div class="mx-1">

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

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

</div>

</div>

<Pagination CurrentPage="currentPage"

TotalPages="totalPages"

SelectedPage="SelectedPageAsync" />

<table class="table table-striped">

<thead>

<tr>

<th>Estado / Departamento</th>

<th style="width:90px">Ciudades</th>

<th style="width:168px"></th>

</tr>

</thead>

<tbody>

@foreach (var state in states!)

{

<tr>

<td><a href="/states/details/@state.Id">@state.Name</a></td>

<td>@state.CitiesNumber</td>

<td>

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

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

</td>

</tr>

}

</tbody>

</table>

</Body>

</GenericList>

</div>

</div>

}

1. Luego modificamos nuestro **StateDetails**:

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

@if (state is null)

{

<Loading />

}

else

{

<div class="card">

<div class="card-header">

<span>

<i class="bi bi-globe-americas" /> @state.Name

<a class="btn btn-sm btn-primary float-end mx-1" href="/cities/create/@StateId"><i class="bi bi-plus-square"></i> Adicionar Ciudad</a>

<a class="btn btn-sm btn-success float-end" href="/countries/details/@state.CountryId"><i class="bi bi-arrow-left" /> Regresar</a>

</span>

</div>

<div class="card-body">

<GenericList MyList="cities!">

<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 ciudad..." @bind-value="Filter" />

</div>

<div class="mx-1">

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

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

</div>

</div>

<Pagination CurrentPage="currentPage"

TotalPages="totalPages"

SelectedPage="SelectedPageAsync" />

<table class="table table-striped">

<thead>

<tr>

<th>Ciudad</th>

<th style="width:168px"></th>

</tr>

</thead>

<tbody>

@foreach (var city in cities!)

{

<tr>

<td>

@city.Name

</td>

<td>

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

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

</td>

</tr>

}

</tbody>

</table>

</Body>

</GenericList>

</div>

</div>

}

1. Luego modificamos nuestro **CategoriesIndex**:

@page "/categories"

<div class="card">

<div class="card-header">

<span>

<i class="bi bi-list-check"></i> Categorias

<a class="btn btn-sm btn-primary float-end" href="/categories/create"><i class="bi bi-plus-square"></i> Adicionar Categoría</a>

</span>

</div>

<div class="card-body">

<GenericList MyList="Categories">

<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 categoría..." @bind-value="Filter" />

</div>

<div class="mx-1">

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

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

</div>

</div>

<Pagination CurrentPage="currentPage"

TotalPages="totalPages"

SelectedPage="SelectedPageAsync" />

<table class="table table-striped">

<thead>

<tr>

<th>Categoría</th>

<th style="width:168px"></th>

</tr>

</thead>

<tbody>

@foreach (var category in Categories!)

{

<tr>

<td>@category.Name</td>

<td>

<a href="/categories/edit/@category.Id" class="btn btn-sm btn-warning"><i class="bi bi-pencil" /> Editar</a>

<button @onclick=@(() => DeleteAsycn(category)) class="btn btn-sm btn-danger"><i class="bi bi-trash" /> Borrar</button>

</td>

</tr>

}

</tbody>

</table>

</Body>

</GenericList>

</div>

</div>

1. Este es un ejemplo de como puede quedar la página de **Register**:

@page "/Register"

@if (loading)

{

<Loading />

}

else

{

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

<DataAnnotationsValidator />

<div class="card">

<div class="card-header">

<span>

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

<button class="btn btn-sm btn-primary float-end" type="submit"><i class="bi bi-person-add" /> Registrar</button>

</span>

</div>

<div class="card-body">

<div class="row">

<div class="col-6">

…

</div>

</div>

</div>

</div>

</EditForm>

}

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

@page "/Login"

<div class="row mt-5">

<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="bi bi-box-arrow-in-left" /> Iniciar Sesión

<button class="btn btn-sm btn-primary float-end" type="submit"><i class="bi bi-box-arrow-in-right" /> 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>

</EditForm>

</div>

</div>

1. Modificamos el **FormWithName.razor**:

@typeparam TModel where TModel : IEntityWithName

<NavigationLock OnBeforeInternalNavigation="OnBeforeInternalNavigation" />

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

<DataAnnotationsValidator />

<div class="mb-3">

<label>@Label</label>

<div>

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

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

</div>

</div>

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

<button class="btn btn-success" @onclick="ReturnAction"><i class="bi bi-arrow-left" /> Regresar</button>

</EditForm>

1. Hacemos el **commit**.

## Mejorando el manejo de errores en el controlador genérico

1. Modificamos el GenericRepository cambiando estas líneas en el catch del **AddAsync** y **UpdateAsync**:

catch (DbUpdateException ex)

{

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

{

return DbUpdateExceptionActionResponse();

}

return new ActionResponse<T>

{

WasSuccess = false,

Message = ex.Message

};

}

1. Probamos.

## Almacenando la foto del usuario

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

using Microsoft.AspNetCore.Components;

using Microsoft.AspNetCore.Components.Forms;

namespace Orders.Frontend.Shared

{

public partial class InputImg

{

private string? imageBase64;

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

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

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

private 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 el **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>

1. Modificamos la clase de **Register.razor.cs**:

…

private UserDTO userDTO = new();

private List<Country>? countries;

private List<State>? states;

private List<City>? cities;

private bool loading;

private string? imageUrl;

[Inject] private NavigationManager NavigationManager { get; set; } = null!;

[Inject] private SweetAlertService SweetAlertService { get; set; } = null!;

[Inject] private IRepository Repository { get; set; } = null!;

[Inject] private ILoginService LoginService { get; set; } = null!;

protected override async Task OnInitializedAsync()

{

await LoadCountriesAsync();

}

private void ImageSelected(string imagenBase64)

{

userDTO.Photo = imagenBase64;

imageUrl = null;

}

…

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

…

<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>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>

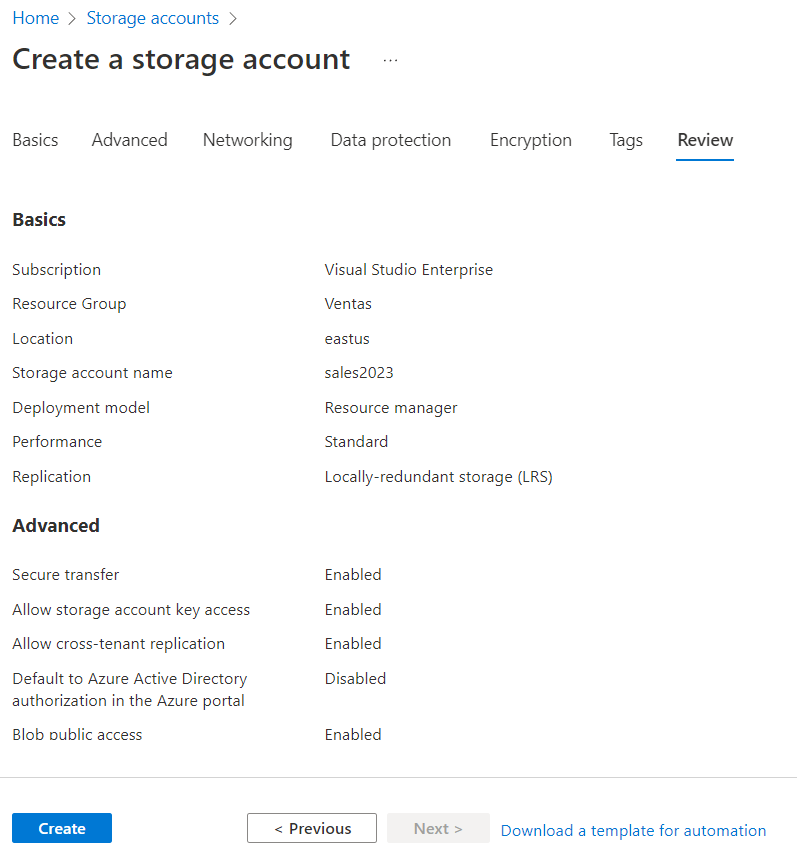
<div class="mb-3">

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

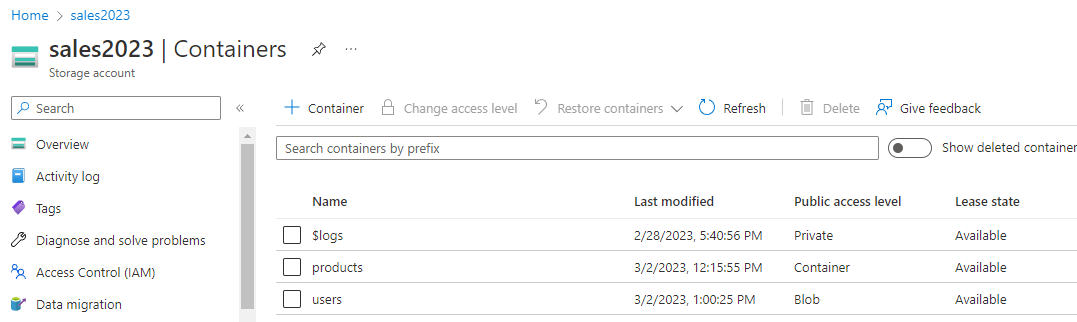
</div>

…

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**:



1. Luego que termine copiamos el connection string que necesitamos para acceder a nuestro blob storage:
2. Agregamos ese connection string en el **appsettings** de nuestro proyecto **Backend**:

"ConnectionStrings": {

"DockerConnection": "Data Source=.;Initial Catalog=Orders;User ID={Your user};Password={Your password};Connect Timeout=30;Encrypt=False;TrustServerCertificate=False;ApplicationIntent=ReadWrite;MultiSubnetFailover=False",

"LocalConnection": "Server=(localdb)\\MSSQLLocalDB;Database=Orders2023;Trusted\_Connection=True;MultipleActiveResultSets=true",

"AzureStorage": "{Your azure connection string}"

},

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

namespace Orders.Backend.Helpers

{

public interface IFileStorage

{

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

Task RemoveFileAsync(string path, string containerName);

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**:

using Azure.Storage.Blobs;

using Azure.Storage.Blobs.Models;

namespace Orders.Backend.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 **Backend**:

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<IActionResult> 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 \_usersUnitOfWork.AddUserAsync(user, model.Password);

if (result.Succeeded)

{

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

return Ok(BuildToken(user));

}

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

}

1. Adicionamos el **AuthLinks.razor.cs**:

using Microsoft.AspNetCore.Components;

using Microsoft.AspNetCore.Components.Authorization;

namespace Orders.Frontend.Shared

{

public partial class AuthLinks

{

private string? photoUser;

[CascadingParameter]

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

protected override async 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. 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>

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

## Editando el usuario

1. Modificamos el **IUsersRepository**:

Task<User> GetUserAsync(Guid userId);

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

Task<IdentityResult> UpdateUserAsync(User user);

1. Modificamos el **UsersRepository**:

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. Modificamos el **IUsersUnitOfWork**:

Task<User> GetUserAsync(Guid userId);

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

Task<IdentityResult> UpdateUserAsync(User user);

1. Modificamos el **UsersUnitOfWork**:

public async Task<User> GetUserAsync(Guid userId) => await \_usersRepository.GetUserAsync(userId);

public async Task<IdentityResult> ChangePasswordAsync(User user, string currentPassword, string newPassword) => await \_usersRepository.ChangePasswordAsync(user, currentPassword, newPassword);

public async Task<IdentityResult> UpdateUserAsync(User user) => await \_usersRepository.UpdateUserAsync(user);

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

[HttpPut]

[Authorize(AuthenticationSchemes = JwtBearerDefaults.AuthenticationScheme)]

public async Task<IActionResult> PutAsync(User user)

{

try

{

var currentUser = await \_usersUnitOfWork.GetUserAsync(User.Identity!.Name!);

if (currentUser == null)

{

return NotFound();

}

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

{

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

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

}

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 \_usersUnitOfWork.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<IActionResult> GetAsync()

{

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

}

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

<Authorized>

<span class="d-flex align-items-center">Hola, <a href="EditUser" class="nav-link btn btn-link">@context.User.Identity!.Name</a></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>

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

using System.Net;

using CurrieTechnologies.Razor.SweetAlert2;

using Microsoft.AspNetCore.Components;

using Orders.Frontend.Repositories;

using Orders.Frontend.Services;

using Orders.Shared.Entities;

namespace Orders.Frontend.Pages.Auth

{

public partial class EditUser

{

private User? user;

private List<Country>? countries;

private List<State>? states;

private List<City>? cities;

private string? imageUrl;

[Inject] private NavigationManager NavigationManager { get; set; } = null!;

[Inject] private SweetAlertService SweetAlertService { get; set; } = null!;

[Inject] private IRepository Repository { get; set; } = null!;

[Inject] private ILoginService LoginService { get; set; } = null!;

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.GetAsync<User>($"/api/accounts");

if (responseHttp.Error)

{

if (responseHttp.HttpResponseMessage.StatusCode == 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!);

states = null;

cities = null;

user!.CityId = 0;

await LoadStatesAsyn(selectedCountry);

}

private async Task StateChangedAsync(ChangeEventArgs e)

{

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

cities = null;

user!.CityId = 0;

await LoadCitiesAsyn(selectedState);

}

private async Task LoadCountriesAsync()

{

var responseHttp = await Repository.GetAsync<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.GetAsync<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.GetAsync<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.PutAsync<User>("/api/accounts", user!);

if (responseHttp.Error)

{

var message = await responseHttp.GetErrorMessageAsync();

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

return;

}

NavigationManager.NavigateTo("/");

}

}

}

1. Modificamos **EditUser.razor**:

@page "/EditUser"

@if (user is null)

{

<Loading />

}

else

{

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

<DataAnnotationsValidator />

<div class="card">

<div class="card-header">

<span>

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

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

<button class="btn btn-sm btn-primary float-end mx-2" type="submit"><i class="bi bi-floppy" /> 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>

}

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

## Cambiando password del usuario

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

using System.ComponentModel.DataAnnotations;

namespace Orders.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 **Orders.Backend.Controllers** en el controlador **AccountsController** adicionamos este método:

[HttpPost("changePassword")]

[Authorize(AuthenticationSchemes = JwtBearerDefaults.AuthenticationScheme)]

public async Task<IActionResult> ChangePasswordAsync(ChangePasswordDTO model)

{

if (!ModelState.IsValid)

{

return BadRequest(ModelState);

}

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

if (user == null)

{

return NotFound();

}

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

if (!result.Succeeded)

{

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

}

return NoContent();

}

1. Dentro de **Orders. Frontend.Pages** creamos el **ChangePassword.razor** y **ChangePassword.razor.cs**:

using CurrieTechnologies.Razor.SweetAlert2;

using Microsoft.AspNetCore.Components;

using Orders.Frontend.Repositories;

using Orders.Shared.DTOs;

namespace Orders.Frontend.Pages.Auth

{

public partial class ChangePassword

{

private ChangePasswordDTO changePasswordDTO = new();

private bool loading;

[Inject] private NavigationManager NavigationManager { get; set; } = null!;

[Inject] private SweetAlertService SweetAlertService { get; set; } = null!;

[Inject] private IRepository Repository { get; set; } = null!;

private async Task ChangePasswordAsync()

{

loading = true;

var responseHttp = await Repository.PostAsync("/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.BottomEnd,

ShowConfirmButton = true,

Timer = 3000

});

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

}

}

}

1. Luego modificamos **ChangePassword.razor**:

@page "/changePassword"

@if (loading)

{

<Loading />

}

<div class="row">

<div class="col-6">

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

<DataAnnotationsValidator />

<div class="card">

<div class="card-header">

<span>

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

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

<button class="btn btn-sm btn-primary float-end mx-2" type="submit"><i class="bi bi-floppy" /> 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>

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

## Confirmar el registro de usuarios

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

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;

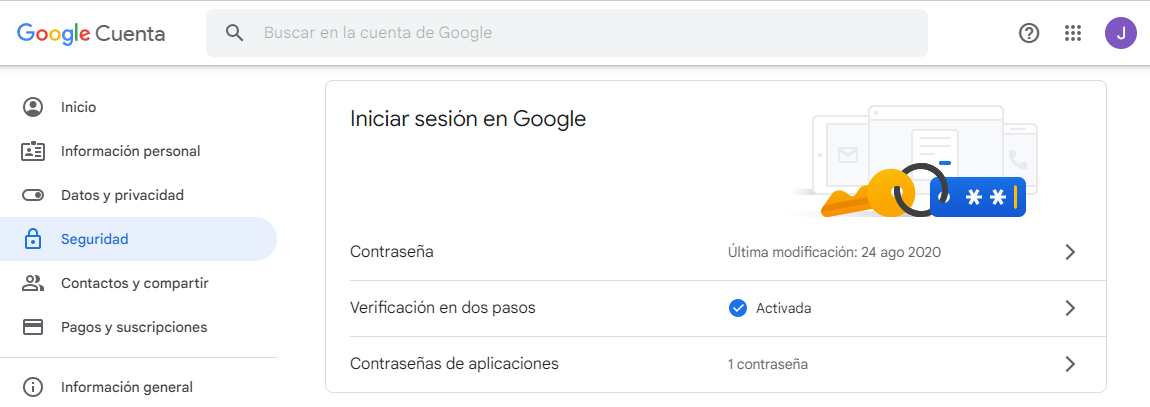
x.Lockout.AllowedForNewUsers = true;

})

.AddEntityFrameworkStores<DataContext>()

.AddDefaultTokenProviders();

1. Verificamos que la cuenta de Gmail con la que vamos a mandar los correos tenga lo siguiente:



1. Adicionamos estos parámetros a la configuración del **Backend**:

"Mail": {

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

"Name": "Soporte Orders",

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

"Port": 587,

"Password": "{Your password}"

},

"Url Frontend": "localhost:7175"

**Nota**: reemplazar el 7175 por el puerto donde sale tu App Frontend, y reemplazar el password por el generado de tu cuenta.

1. Adicionamos el nuget “**Mailkit**” al proyecto **Backend**:
2. En los **Helpers** del **Backend** adicionamos la interzar **IMailHelper**:

using Orders.Shared.Responses;

namespace Orders.Backend.Helpers

{

public interface IMailHelper

{

ActionResponse<string> SendMail(string toName, string toEmail, string subject, string body);

}

}

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

using MailKit.Net.Smtp;

using MimeKit;

using Orders.Shared.Responses;

namespace Orders.Backend.Helpers

{

public class MailHelper : IMailHelper

{

private readonly IConfiguration \_configuration;

public MailHelper(IConfiguration configuration)

{

\_configuration = configuration;

}

public ActionResponse<string> 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 ActionResponse<string> { WasSuccess = true };

}

catch (Exception ex)

{

return new ActionResponse<string>

{

WasSuccess = false,

Message = ex.Message,

};

}

}

}

}

1. Configuramos la inyección del servicio:

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

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

Task<string> GenerateEmailConfirmationTokenAsync(User user);

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

Y la implementación:

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. Add those methods to **IUsersUnitOfWork**:

Task<string> GenerateEmailConfirmationTokenAsync(User user);

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

Y la implementación:

public async Task<string> GenerateEmailConfirmationTokenAsync(User user) => await \_usersRepository.GenerateEmailConfirmationTokenAsync(user);

public async Task<IdentityResult> ConfirmEmailAsync(User user, string token) => await \_usersRepository.ConfirmEmailAsync(user, token);

1. Modificamos el método **CreateUser** del controlador **AccountsController** (primero inyectamos el **IMailHelper**):

[HttpPost("CreateUser")]

public async Task<IActionResult> 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 \_usersUnitOfWork.AddUserAsync(user, model.Password);

if (result.Succeeded)

{

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

var response = await SendConfirmationEmailAsync(user);

if (response.WasSuccess)

{

return NoContent();

}

return BadRequest(response.Message);

}

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

}

…

private async Task<ActionResponse<string>> SendConfirmationEmailAsync(User user)

{

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

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

{

userid = user.Id,

token = myToken

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

return \_mailHelper.SendMail(user.FullName, user.Email!,

$"Orders - Confirmación de cuenta",

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

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

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

}

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

[HttpGet("ConfirmEmail")]

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

{

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

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

if (user == null)

{

return NotFound();

}

var result = await \_usersUnitOfWork.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<IActionResult> Login([FromBody] LoginDTO model)

{

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

if (result.Succeeded)

{

var user = await \_usersUnitOfWork.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. Modificar el UserRepository para que el usuario se bloque por número de intentos fallidos:

public async Task<SignInResult> LoginAsync(LoginDTO model)

{

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

}

1. Agregamos este método al **IRepository** en el **Frontend**:

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

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

public async Task<HttpResponseWrapper<object>> GetAsync(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** y **ConfirmEmail.razor.cs**:

using CurrieTechnologies.Razor.SweetAlert2;

using Microsoft.AspNetCore.Components;

using Orders.Frontend.Repositories;

namespace Orders.Frontend.Pages.Auth

{

public partial class ConfirmEmail

{

private string? message;

[Inject] private NavigationManager NavigationManager { get; set; } = null!;

[Inject] private SweetAlertService SweetAlertService { get; set; } = null!;

[Inject] private IRepository Repository { get; set; } = null!;

[Parameter, SupplyParameterFromQuery] public string UserId { get; set; } = string.Empty;

[Parameter, SupplyParameterFromQuery] public string Token { get; set; } = string.Empty;

protected async Task ConfirmAccountAsync()

{

var responseHttp = await Repository.GetAsync($"/api/accounts/ConfirmEmail/?userId={UserId}&token={Token}");

if (responseHttp.Error)

{

message = await responseHttp.GetErrorMessageAsync();

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

NavigationManager.NavigateTo("/");

return;

}

await SweetAlertService.FireAsync("Confirmación", "Gracias por confirmar su email, ahora puedes ingresar al sistema.", SweetAlertIcon.Info);

NavigationManager.NavigateTo("/Login");

}

}

}

1. Luego modicamos **ConfirmEmail.razor**:

@page "/api/accounts/ConfirmEmail"

<h3>Confirmación de email</h3>

<p>Presione el botón para confirmar su cuenta</p>

<button class="btn btn-primary" @onclick="ConfirmAccountAsync">Confirmar Cuenta</button>

1. Borramos los usuarios de la base de datos. Pueden usar estas instrucciones:

DELETE FROM AspNetUserRoles

DELETE FROM AspNetUsers

1. 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 \_usersUnitOfWork.GetUserAsync(email);

if (user == null)

{

var city = await \_context.Cities.FirstOrDefaultAsync(x => x.Name == "Medellín");

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 \_usersUnitOfWork.AddUserAsync(user, "123456");

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

var token = await \_usersUnitOfWork.GenerateEmailConfirmationTokenAsync(user);

await \_usersUnitOfWork.ConfirmEmailAsync(user, token);

}

return user;

}

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

private async Task CreteUserAsync()

{

loading = true;

userDTO.UserName = userDTO.Email;

userDTO.UserType = UserType.User;

var responseHttp = await Repository.PostAsync<UserDTO>("/api/accounts/CreateUser", userDTO);

loading = false;

if (responseHttp.Error)

{

var message = await responseHttp.GetErrorMessageAsync();

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

return;

}

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 **Orders.Shared.DTOs** creamos la clase **EmailDTO**:

using System.ComponentModel.DataAnnotations;

namespace Orders.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 **Backend** creamos este método en el **AccountsController**:

[HttpPost("ResedToken")]

public async Task<IActionResult> ResedTokenAsync([FromBody] EmailDTO model)

{

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

if (user == null)

{

return NotFound();

}

var response = await SendConfirmationEmailAsync(user);

if (response.WasSuccess)

{

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="bi bi-box-arrow-in-left" /> Iniciar Sesión

<button class="btn btn-sm btn-primary float-end" type="submit"><i class="bi bi-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="btn btn-link" href="/ResendToken">Reenviar correro de activación de cuenta</a>

</div>

</div>

</EditForm>

</div>

</div>

1. Dentro de **Pages/Auth** creamos el **ResendConfirmationEmailToken.razor** y **ResendConfirmationEmailToken.cs**:

using CurrieTechnologies.Razor.SweetAlert2;

using Microsoft.AspNetCore.Components;

using Orders.Frontend.Repositories;

using Orders.Shared.DTOs;

namespace Orders.Frontend.Pages.Auth

{

public partial class ResendConfirmationEmailToken

{

private EmailDTO emailDTO = new();

private bool loading;

[Inject] private NavigationManager NavigationManager { get; set; } = null!;

[Inject] private SweetAlertService SweetAlertService { get; set; } = null!;

[Inject] private IRepository Repository { get; set; } = null!;

private async Task ResendConfirmationEmailTokenAsync()

{

loading = true;

var responseHttp = await Repository.PostAsync("/api/accounts/ResedToken", emailDTO);

loading = false;

if (responseHttp.Error)

{

var message = await responseHttp.GetErrorMessageAsync();

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

loading = false;

return;

}

await SweetAlertService.FireAsync("Confirmación", "Se te ha enviado un correo electrónico con las instrucciones para activar tu usuario.", SweetAlertIcon.Info);

NavigationManager.NavigateTo("/");

}

}

}

1. Modificamos el **ResendConfirmationEmailToken.razor**:

@page "/ResendToken"

@if (loading)

{

<Loading />

}

<div class="row">

<div class="col-6">

<EditForm Model="emailDTO" OnValidSubmit="ResendConfirmationEmailTokenAsync">

<DataAnnotationsValidator />

<div class="card">

<div class="card-header">

<span>

<i class="bi bi-key" /> Reenviar correo de confirmación de contraseña

<button class="btn btn-sm btn-primary float-end mx-2" type="submit"><i class="bi bi-send" /> 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>

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

## Actualización de la foto del usuario luego de editar usuario

1. Modificamos el **PUT** del **AccountsController**:

…

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

if (result.Succeeded)

{

return Ok(BuildToken(currentUser));

}

…

1. Modificamos el **EditUser**:

private async Task SaveUserAsync()

{

var responseHttp = await Repository.PutAsync<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="btn btn-link" href="/ResendToken">Reenviar correro de activación de cuenta</a></p>

<p><a class="btn btn-link" href="/RecoverPassword">¿Has olvidado tu contraseña?</a></p>

</div>

1. Adicionamos en **Orders.Shared.DTOs** la clase **ResetPasswordDTO**:

using System.ComponentModel.DataAnnotations;

namespace Orders.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 **IUsersRepository**:

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 **IUsersUnitOfWork**:

Task<string> GeneratePasswordResetTokenAsync(User user);

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

Y la implementación:

public async Task<string> GeneratePasswordResetTokenAsync(User user) => await \_usersRepository.GeneratePasswordResetTokenAsync(user);

public async Task<IdentityResult> ResetPasswordAsync(User user, string token, string password) => await \_usersRepository.ResetPasswordAsync(user, token, password);

1. Adicionamos estos métodos al **AccountController**:

[HttpPost("RecoverPassword")]

public async Task<IActionResult> RecoverPasswordAsync([FromBody] EmailDTO model)

{

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

if (user == null)

{

return NotFound();

}

var myToken = await \_usersUnitOfWork.GeneratePasswordResetTokenAsync(user);

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

{

userid = user.Id,

token = myToken

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

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

$"Orders - Recuperación de contraseña",

$"<h1>Orders - 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.WasSuccess)

{

return NoContent();

}

return BadRequest(response.Message);

}

[HttpPost("ResetPassword")]

public async Task<IActionResult> ResetPasswordAsync([FromBody] ResetPasswordDTO model)

{

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

if (user == null)

{

return NotFound();

}

var result = await \_usersUnitOfWork.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** y **RecoverPassword.razor.cs**:

using CurrieTechnologies.Razor.SweetAlert2;

using Microsoft.AspNetCore.Components;

using Orders.Frontend.Repositories;

using Orders.Shared.DTOs;

namespace Orders.Frontend.Pages.Auth

{

public partial class RecoverPassword

{

private EmailDTO emailDTO = new();

private bool loading;

[Inject] private NavigationManager NavigationManager { get; set; } = null!;

[Inject] private SweetAlertService SweetAlertService { get; set; } = null!;

[Inject] private IRepository Repository { get; set; } = null!;

private async Task SendRecoverPasswordEmailTokenAsync()

{

loading = true;

var responseHttp = await Repository.PostAsync("/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. Creamos el **RecoverPassword.razor.cs**:

@page "/RecoverPassword"

@if (loading)

{

<Loading />

}

<div class="row">

<div class="col-6">

<EditForm Model="emailDTO" OnValidSubmit="SendRecoverPasswordEmailTokenAsync">

<DataAnnotationsValidator />

<div class="card">

<div class="card-header">

<span>

<i class="bi bi-key" /> Enviar email para recuperación de contraseña

<button class="btn btn-sm btn-primary float-end mx-2" type="submit"><i class="bi bi-send" /> 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>

1. Dentro de **Pages/Auth** creamos el **ResetPassword.razor** y **ResetPassword.razor.cs**:

using CurrieTechnologies.Razor.SweetAlert2;

using Microsoft.AspNetCore.Components;

using Orders.Frontend.Repositories;

using Orders.Shared.DTOs;

namespace Orders.Frontend.Pages.Auth

{

public partial class ResetPassword

{

private ResetPasswordDTO resetPasswordDTO = new();

private bool loading;

[Inject] private NavigationManager NavigationManager { get; set; } = null!;

[Inject] private SweetAlertService SweetAlertService { get; set; } = null!;

[Inject] private IRepository Repository { get; set; } = null!;

[Parameter, SupplyParameterFromQuery] public string Token { get; set; } = string.Empty;

private async Task ChangePasswordAsync()

{

resetPasswordDTO.Token = Token;

loading = true;

var responseHttp = await Repository.PostAsync("/api/accounts/ResetPassword", resetPasswordDTO);

loading = false;

if (responseHttp.Error)

{

var message = await responseHttp.GetErrorMessageAsync();

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

loading = false;

return;

}

await SweetAlertService.FireAsync("Confirmación", "Contraseña cambiada con éxito, ahora puede ingresar con su nueva contraseña.", SweetAlertIcon.Info);

NavigationManager.NavigateTo("/Login");

}

}

}

1. Creamos el **ResetPassword.razor.cd**:

@page "/api/accounts/ResetPassword"

@if (loading)

{

<Loading />

}

<div class="row">

<div class="col-6">

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

<DataAnnotationsValidator />

<div class="card">

<div class="card-header">

<span>

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

<button class="btn btn-sm btn-primary float-end mx-2" type="submit"><i class="bi bi-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>

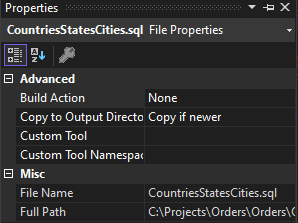
</div>

</div>

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

## Agregar países al SeedBd por Script

1. Agregamos al **Backend/Data** el archivo **CountriesStatesCities.sql** (tome el archivo del repositorio).
2. Colocar a este archivo la propiedad **Copy if newer**:



1. Modificar el **SeedDb**:

public async Task SeedAsync()

{

await \_context.Database.EnsureCreatedAsync();

await CheckCountriesFullAsync();

await CheckCountriesAsync();

await CheckCatregoriesAsync();

await CheckRolesAsync();

await CheckUserAsync("1010", "Juan", "Zuluaga", "zulu@yopmail.com", "322 311 4620", "Calle Luna Calle Sol", UserType.Admin);

}

private async Task CheckCountriesFullAsync()

{

if (!\_context.Countries.Any())

{

var countriesStatesCitiesSQLScript = File.ReadAllText("Data\\CountriesStatesCities.sql");

await \_context.Database.ExecuteSqlRawAsync(countriesStatesCitiesSQLScript);

}

}

1. Borramos la base de datos para poder probar, probamos y hacemos el **commit**.

## Solución a la tarea de colocar un componente de filtro genérico

1. Creamos en el **Frontend/Shared** el componente **Filter.razor** y el **Filter.razor.cs**:

using Microsoft.AspNetCore.Components;

namespace Orders.Frontend.Shared

{

public partial class Filter

{

[Parameter, SupplyParameterFromQuery] public string TextToFilter { get; set; } = string.Empty;

[Parameter] public string PlaceHolder { get; set; } = string.Empty;

[Parameter] public Func<string, Task> Callback { get; set; } = async (text) => await Task.CompletedTask;

private async Task CleanFilterAsync()

{

await Callback(string.Empty);

}

private async Task ApplyFilterAsync()

{

await Callback(TextToFilter);

}

}

}

1. Modificamos el **Filter.razor**:

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

<input style="width:400px;" type="text" class="form-control" placeholder=@PlaceHolder @bind-value="TextToFilter">

<button type="button" class="btn btn-outline-primary mx-1" @onclick="ApplyFilterAsync"><i class="bi bi-funnel" /> Filtrar</button>

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

</div>

1. Modificamos el **CategoriesIndex.razor.cs**:

private async Task FilterCallBack(string filter)

{

Filter = filter;

await ApplyFilterAsync();

StateHasChanged();

}

1. Modificamos el **CategoriesIndex.razor**:

…

<div class="card">

<div class="card-header">

<span>

<i class="bi bi-list-check" /></i> Categorias

<a class="btn btn-sm btn-primary float-end" @onclick=@(() => ShowModalAsync())><i class="bi bi-plus-circle" /> Adicionar Categoría</a>

</span>

</div>

<div class="card-body">

<Filter PlaceHolder="Buscar categoría..." Callback=@FilterCallBack />

<GenericList MyList="Categories">

<Body>

<Pagination CurrentPage="currentPage"

TotalPages="totalPages"

SelectedPage="SelectedPageAsync" />

…

1. Probamos lo que llevamos hasta el momento.
2. Modificamos el **CountriesIndex.razor.cs**:

private async Task FilterCallBack(string filter)

{

Filter = filter;

await ApplyFilterAsync();

StateHasChanged();

}

1. Modificamos el **CountriesIndex.razor**:

…

<div class="card">

<div class="card-header">

<span>

<i class="bi bi-globe-americas" /> Países

<a class="btn btn-sm btn-primary float-end" @onclick=@(() => ShowModalAsync())><i class="bi bi-plus-circle" /> Adicionar País</a>

</span>

</div>

<div class="card-body">

<Filter PlaceHolder="Buscar país..." Callback=@FilterCallBack />

<GenericList MyList="Countries">

<Body>

<Pagination CurrentPage="currentPage"

TotalPages="totalPages"

SelectedPage="SelectedPageAsync" />

…

1. Probamos lo que llevamos hasta el momento.
2. Modificamos el **CountryDetails.razor.cs**:

private async Task FilterCallBack(string filter)

{

Filter = filter;

await ApplyFilterAsync();

StateHasChanged();

}

1. Modificamos el **CountryDetails.razor**:

…

<div class="card-header">

<span>

<i class="bi bi-globe-americas" /> @country.Name

<a class="btn btn-sm btn-primary float-end mx-1" @onclick=@(() => ShowModalAsync())><i class="bi bi-plus-circle" /> Adicionar Estado/Departamento</a>

<a class="btn btn-sm btn-success float-end" href="/countries"><i class="bi bi-arrow-left" /> Regresar</a>

</span>

</div>

<div class="card-body">

<Filter PlaceHolder="Buscar estado/departamento..." Callback=@FilterCallBack />

<GenericList MyList="states!">

<Body>

<Pagination CurrentPage="currentPage"

TotalPages="totalPages"

SelectedPage="SelectedPageAsync" />

…

1. Probamos lo que llevamos hasta el momento.
2. Modificamos el **StateDetails.razor.cs**:

private async Task FilterCallBack(string filter)

{

Filter = filter;

await ApplyFilterAsync();

StateHasChanged();

}

1. Modificamos el **StateDetails.razor**:

…

<div class="card-body">

<Filter PlaceHolder="Buscar ciudad..." Callback=@FilterCallBack />

<GenericList MyList="cities!">

<Body>

<Pagination CurrentPage="currentPage"

TotalPages="totalPages"

SelectedPage="SelectedPageAsync" />

…

1. Probamos y hacemos el commit.

## Solución a la tarea de colocar un selector con la cantidad de registros a mostrar

1. Modificamos el **Pagination.razor.cs**:

using Microsoft.AspNetCore.Components;

namespace Orders.Frontend.Shared

{

public partial class Pagination

{

private List<PageModel> links = [];

private List<OptionModel> options = [];

private int selectedOptionValue = 10;

[Parameter] public int CurrentPage { get; set; } = 1;

[Parameter] public int Radio { get; set; } = 10;

[Parameter] public EventCallback<int> RecordsNumber { get; set; }

[Parameter] public EventCallback<int> SelectedPage { get; set; }

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

protected override void OnParametersSet()

{

BuildPages();

BuildOptions();

}

private void BuildPages()

{

links = [];

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

});

}

private void BuildOptions()

{

options =

[

new OptionModel { Value = 10, Name = "10" },

new OptionModel { Value = 25, Name = "25" },

new OptionModel { Value = 50, Name = "50" },

new OptionModel { Value = int.MaxValue, Name = "Todos" },

];

}

private async Task InternalRecordsNumberSelected(ChangeEventArgs e)

{

if (e.Value != null)

{

selectedOptionValue = Convert.ToInt32(e.Value.ToString());

}

await RecordsNumber.InvokeAsync(selectedOptionValue);

}

private async Task InternalSelectedPage(PageModel pageModel)

{

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

{

return;

}

await SelectedPage.InvokeAsync(pageModel.Page);

}

private class OptionModel

{

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

public int Value { get; set; }

}

private class PageModel

{

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

public bool Enable { get; set; } = true;

public int Page { get; set; }

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

}

}

}

1. Modificamos el **Pagination.razor**:

<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>

}

<li class="mx-2">

<select class="form-select custom-select" @onchange="InternalRecordsNumberSelected">

@foreach (var option in options)

{

<option value="@option.Value">@option.Name</option>

}

</select>

</li>

</ul>

</nav>

1. Modificamos el **CountriesIndex.razor.cs**:

…

[Parameter, SupplyParameterFromQuery] public int RecordsNumber { get; set; } = 10;

…

private async Task SelectedRecordsNumberAsync(int recordsnumber)

{

RecordsNumber = recordsnumber;

int page = 1;

await LoadAsync(page);

await SelectedPageAsync(page);

}

…

private async Task<bool> LoadListAsync(int page)

{

ValidateRecordsNumber(RecordsNumber);

var url = $"api/countries?page={page}&recordsnumber={RecordsNumber}";

if (!string.IsNullOrEmpty(Filter))

{

url += $"&filter={Filter}";

}

…

private void ValidateRecordsNumber(int recordsnumber)

{

if (recordsnumber == 0)

{

RecordsNumber = 10;

}

}

…

private async Task LoadPagesAsync()

{

ValidateRecordsNumber(RecordsNumber);

var url = $"api/countries/totalPages?recordsnumber={RecordsNumber}";

if (!string.IsNullOrEmpty(Filter))

{

url += $"&filter={Filter}";

}

…

1. Modificamos el **CountriesIndex.razor**:

…

<Pagination CurrentPage="currentPage"

TotalPages="totalPages"

SelectedPage="SelectedPageAsync"

RecordsNumber="SelectedRecordsNumberAsync" />

…

1. Modificamos el **CountryDetails.razor.cs**:

…

[Parameter, SupplyParameterFromQuery] public int RecordsNumber { get; set; } = 10;

…

private async Task SelectedRecordsNumberAsync(int recordsnumber)

{

RecordsNumber = recordsnumber;

int page = 1;

await LoadAsync(page);

await SelectedPageAsync(page);

}

…

private async Task<bool> LoadStatesAsync(int page)

{

ValidateRecordsNumber(RecordsNumber);

var url = $"api/states?id={CountryId}&page={page}&recordsnumber={RecordsNumber}";

if (!string.IsNullOrEmpty(Filter))

{

url += $"&filter={Filter}";

}

…

private void ValidateRecordsNumber(int recordsnumber)

{

if (recordsnumber == 0)

{

RecordsNumber = 10;

}

}

…

private async Task LoadPagesAsync()

{

ValidateRecordsNumber(RecordsNumber);

var url = $"api/states/totalPages?id={CountryId}&recordsnumber={RecordsNumber}";

if (!string.IsNullOrEmpty(Filter))

{

url += $"&filter={Filter}";

}

…

1. Modificamos el **CountryDetails.razor**:

…

<Pagination CurrentPage="currentPage"

TotalPages="totalPages"

SelectedPage="SelectedPageAsync"

RecordsNumber="SelectedRecordsNumberAsync" />

…

1. Modificamos el **StateDetails.razor.cs**:

…

[Parameter, SupplyParameterFromQuery] public int RecordsNumber { get; set; } = 10;

…

private async Task SelectedRecordsNumberAsync(int recordsnumber)

{

RecordsNumber = recordsnumber;

int page = 1;

await LoadAsync(page);

await SelectedPageAsync(page);

}

…

private async Task<bool> LoadCitiesAsync(int page)

{

ValidateRecordsNumber(RecordsNumber);

var url = $"api/cities?id={StateId}&page={page}&recordsnumber={RecordsNumber}";

if (!string.IsNullOrEmpty(Filter))

{

url += $"&filter={Filter}";

}

…

private void ValidateRecordsNumber(int recordsnumber)

{

if (recordsnumber == 0)

{

RecordsNumber = 10;

}

}

…

private async Task LoadPagesAsync()

{

ValidateRecordsNumber(RecordsNumber);

var url = $"api/cities/totalPages?id={StateId}&recordsnumber={RecordsNumber}";

if (!string.IsNullOrEmpty(Filter))

{

url += $"&filter={Filter}";

}

…

1. Modificamos el **StateDetails.razor**:

…

<Pagination CurrentPage="currentPage"

TotalPages="totalPages"

SelectedPage="SelectedPageAsync"

RecordsNumber="SelectedRecordsNumberAsync" />

…

1. Modificamos el **CategoriesIndex.razor.cs**:

…

[Parameter, SupplyParameterFromQuery] public int RecordsNumber { get; set; } = 10;

…

private async Task SelectedRecordsNumberAsync(int recordsnumber)

{

RecordsNumber = recordsnumber;

int page = 1;

await LoadAsync(page);

await SelectedPageAsync(page);

}

…

private async Task<bool> LoadListAsync(int page)

{

ValidateRecordsNumber(RecordsNumber);

var url = $"api/categories?page={page}&recordsnumber={RecordsNumber}";

if (!string.IsNullOrEmpty(Filter))

{

url += $"&filter={Filter}";

}

…

private void ValidateRecordsNumber(int recordsnumber)

{

if (recordsnumber == 0)

{

RecordsNumber = 10;

}

}

…

private async Task LoadPagesAsync()

{

ValidateRecordsNumber(RecordsNumber);

var url = $"api/categories/totalPages?recordsnumber={RecordsNumber}";

if (!string.IsNullOrEmpty(Filter))

{

url += $"&filter={Filter}";

}

…

1. Modificamos el **CategoriesIndex.razor**:

…

<Pagination CurrentPage="currentPage"

TotalPages="totalPages"

SelectedPage="SelectedPageAsync"

RecordsNumber="SelectedRecordsNumberAsync" />

…

1. Probamos y hacemos el commit.

## Implementación de ventanas modales

Documentación oficial en:<https://blazored.github.io/Modal/>

1. Instalar el paquete **Blazored.Modal** en el **Frontend**:
2. Modificamos el **Program** del proyecto  **Frontend**:

builder.Services.AddBlazoredModal();

1. Modificamos el **\_Imports.razor**:

@using Blazored.Modal

@using Blazored.Modal.Services

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

<CascadingBlazoredModal Position="ModalPosition.Middle" Size="ModalSize.Large" HideHeader="true" DisableBackgroundCancel="true" AnimationType="ModalAnimationType.FadeInOut">

<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>

</CascadingBlazoredModal>

1. Modificamos el **CategoriesIndex.razor.cs**:

…

[CascadingParameter] IModalService Modal { get; set; } = default!;

…

private async Task ShowModalAsync(int id = 0, bool isEdit = false)

{

IModalReference modalReference;

if (isEdit)

{

modalReference = Modal.Show<CategoryEdit>(string.Empty, new ModalParameters().Add("Id", id));

}

else

{

modalReference = Modal.Show<CategoryCreate>();

}

var result = await modalReference.Result;

if (result.Confirmed)

{

await LoadAsync();

}

}

…

1. Modificamos el **CategoriesIndex.razor**:

…

<a class="btn btn-sm btn-primary float-end" @onclick=@(() => ShowModalAsync())><i class="bi bi-plus-circle" /> Adicionar Categoría</a>

…

<a @onclick=@(() => ShowModalAsync(category.Id, true)) class="btn btn-warning"><i class="bi bi-pencil" /> Editar</a>

…

1. Modificamos el **CategoriesEdit.razor.cs**:

…

[CascadingParameter] BlazoredModalInstance BlazoredModal { get; set; } = default!;

…

private async Task EditAsync()

{

var responseHTTP = await Repository.PutAsync("api/categories", category);

if (responseHTTP.Error)

{

var mensajeError = await responseHTTP.GetErrorMessageAsync();

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

return;

}

await BlazoredModal.CloseAsync(ModalResult.Ok());

Return();

var toast = SweetAlertService.Mixin(new SweetAlertOptions

{

Toast = true,

Position = SweetAlertPosition.BottomEnd,

ShowConfirmButton = true,

Timer = 3000

});

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

}

…

1. Modificamos el **CategoriesCreate.razor.cs**:

…

[CascadingParameter] BlazoredModalInstance BlazoredModal { get; set; } = default!;

…

private async Task CreateAsync()

{

var responseHttp = await Repository.PostAsync("/api/categories", category);

if (responseHttp.Error)

{

var message = await responseHttp.GetErrorMessageAsync();

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

return;

}

await BlazoredModal.CloseAsync(ModalResult.Ok());

Return();

var toast = SweetAlertService.Mixin(new SweetAlertOptions

{

Toast = true,

Position = SweetAlertPosition.BottomEnd,

ShowConfirmButton = true,

Timer = 3000

});

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

}

…

1. Probamos (Corremos la App con Ctrl + F5).
2. Modificamos el **CountriesIndex.razor.cs**:

…

[CascadingParameter] IModalService Modal { get; set; } = default!;

…

…

1. Modificamos el **CountriesIndex.razor**:

…

<a class="btn btn-sm btn-primary float-end" @onclick=@(() => ShowModalAsync())><i class="bi bi-plus-circle" /> Adicionar País</a>

…

<a class="btn btn-warning btn-sm" @onclick=@(() => ShowModalAsync(country.Id, true))><i class="bi bi-pencil" /> Editar</a>

…

1. Modificamos el **CountryCreate.razor.cs**:

…

[CascadingParameter] BlazoredModalInstance BlazoredModal { get; set; } = default!;

…

await BlazoredModal.CloseAsync(ModalResult.Ok());

Return();

…

1. Modificamos el **CountryEdit.razor.cs**:

…

[CascadingParameter] BlazoredModalInstance BlazoredModal { get; set; } = default!;

…

await BlazoredModal.CloseAsync(ModalResult.Ok());

Return();

…

1. Probamos (Corremos la App con Ctrl + F5).
2. Modificamos el **CountryDetails.razor.cs**:

…

[CascadingParameter] IModalService Modal { get; set; } = default!;

…

private async Task ShowModalAsync(int id = 0, bool isEdit = false)

{

IModalReference modalReference;

if (isEdit)

{

modalReference = Modal.Show<StateEdit>(string.Empty, new ModalParameters().Add("StateId", id));

}

else

{

modalReference = Modal.Show<StateCreate>(string.Empty, new ModalParameters().Add("CountryId", CountryId));

}

var result = await modalReference.Result;

if (result.Confirmed)

{

await LoadAsync();

}

}

…

1. Modificamos el **CountryDetails.razor**:

…

<a class="btn btn-sm btn-primary float-end mx-1" @onclick=@(() => ShowModalAsync())><i class="bi bi-plus-square" /> Adicionar Estado/Departamento</a>

…

<a class="btn btn-warning btn-sm" @onclick=@(() => ShowModalAsync(state.Id, true))><i class="bi bi-pencil" /> Editar</a>

…

1. Modificamos el **StateCreate.razor.cs**:

…

[CascadingParameter] BlazoredModalInstance BlazoredModal { get; set; } = default!;

…

await BlazoredModal.CloseAsync(ModalResult.Ok());

Return();

…

1. Modificamos el **StateEdit.razor**:

…

[CascadingParameter] BlazoredModalInstance BlazoredModal { get; set; } = default!;

…

await BlazoredModal.CloseAsync(ModalResult.Ok());

Return();

…

1. Probamos (Corremos la App con Ctrl + F5).
2. Modificamos el **StateDetails.razor.cs**:

…

[CascadingParameter] IModalService Modal { get; set; } = default!;

…

private async Task ShowModalAsync(int id = 0, bool isEdit = false)

{

IModalReference modalReference;

if (isEdit)

{

modalReference = Modal.Show<CityEdit>(string.Empty, new ModalParameters().Add("CityId", id));

}

else

{

modalReference = Modal.Show<CityCreate>(string.Empty, new ModalParameters().Add("StateId", StateId));

}

var result = await modalReference.Result;

if (result.Confirmed)

{

await LoadAsync();

}

}

…

1. Modificamos el **StateDetails.razor**:

…

<a class="btn btn-sm btn-primary float-end mx-1" @onclick=@(() => ShowModalAsync())><i class="bi bi-plus-square"></i> Adicionar Ciudad</a>

…

<a class="btn btn-warning btn-sm" @onclick=@(() => ShowModalAsync(city.Id, true))><i class="bi bi-pencil" /> Editar</a>

…

1. Modificamos el **CityCreate.razor.cs**:

…

[CascadingParameter] BlazoredModalInstance BlazoredModal { get; set; } = default!;

…

await BlazoredModal.CloseAsync(ModalResult.Ok());

Return();

…

1. Modificamos el **CityEdit.razor.cs**:

…

[CascadingParameter] BlazoredModalInstance BlazoredModal { get; set; } = default!;

…

await BlazoredModal.CloseAsync(ModalResult.Ok());

Return();

…

1. Probamos (Corremos la App con Ctrl + F5).
2. Modificamos el **ConfirmEmail.razor.cs**:

…

[CascadingParameter] IModalService Modal { get; set; } = default!;

…

await sweetAlertService.FireAsync("Confirmación", "Gracias por confirmar su email, ahora puedes ingresar al sistema.", SweetAlertIcon.Info);

Modal.Show<Login>();

…

1. Modificamos el **EditUser.razor.cs**:

…

[CascadingParameter] IModalService Modal { get; set; } = default!;

…

private void ShowModal()

{

Modal.Show<ChangePassword>();

}

…

1. Modificamos el **EditUser.razor**:

…

<a class="btn btn-sm btn-secondary float-end" @onclick=@(() => ShowModal())><i class="bi bi-key" /> Cambiar Contraseña</a>

…

1. Modificamos el **ResetPassword.razor.cs**:

…

[CascadingParameter] IModalService Modal { get; set; } = default!;

…

await sweetAlertService.FireAsync("Confirmación", "Contraseña cambiada con éxito, ahora puede ingresar con su nueva contraseña.", SweetAlertIcon.Info);

Modal.Show<Login>();

…

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

…

[CascadingParameter] IModalService Modal { get; set; } = default!;

…

private void ShowModal()

{

Modal.Show<Login>();

}

…

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

…

<a @onclick=@(() => ShowModal()) class="nav-link btn btn-link">Iniciar Sesión</a>

…

1. Modificamos el **ChangePassword.razor.cs**:

…

[CascadingParameter] BlazoredModalInstance BlazoredModal { get; set; } = default!;

…

loading = false;

await BlazoredModal.CloseAsync(ModalResult.Ok());

…

1. Modificamos el **ChangePassword.razor**:

…

<div class="row">

<div class="col-12">

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

…

1. Modificamos el **Login.razor.cs**:

…

private LoginDTO loginDTO = new();

private bool wasClose;

[CascadingParameter] BlazoredModalInstance BlazoredModal { get; set; } = default!;

…

private async Task CloseModalAsync()

{

wasClose = true;

await BlazoredModal.CloseAsync(ModalResult.Ok());

}

private async Task LoginAsync()

{

if (wasClose)

{

NavigationManager.NavigateTo("/");

return;

}

var responseHttp = await repository.PostAsync<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. Modificamos el **Login.razor**:

…

<div class="row">

<div class="col-12">

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

…

<button class="btn btn-sm btn-primary float-end" @onclick=@(() => LoginAsync())><i class="bi bi-box-arrow-in-right" /> Iniciar Sesión</button>

<button class="btn btn-sm mx-1 btn-danger float-end" @onclick=@(() => CloseModalAsync())><i class="bi bi-x-circle-fill" /> Cancelar</button>

…

<div class="card-footer">

<p><a class="btn btn-link" href="/Register">¿No eres usuario aún? Resgitrate aquí</a></p>

<p><a class="btn btn-link" href="/ResendToken">Reenviar correro de activación de cuenta</a></p>

<p><a class="btn btn-link" href="/RecoverPassword">¿Has olvidado tu contraseña?</a></p>

</div>

…

1. Probamos (Corremos la App con Ctrl + F5) y hacemos el commit.

## 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 Orders.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 Orders.Shared.Entities

{

public class ProductImage

{

public int Id { get; set; }

public Product? Product { get; set; };

public int ProductId { get; set; }

[Display(Name = "Imagen")]

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

}

}

1. Creamos la entidad **ProductCategory**:

namespace Orders.Shared.Entities

{

public class ProductCategory

{

public int Id { get; set; }

public Product? Product { get; set; }

public int ProductId { get; set; }

public Category? Category { get; set; }

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 || ProductCategories.Count == 0 ? 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 || ProductCategories.Count == 0 ? 0 : ProductCategories.Count;

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

[Display(Name = "Imágenes")]

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

[Display(Name = "Imagén")]

public string MainImage => ProductImages == null || ProductImages.Count == 0 ? 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();

DisableCascadingDelete(modelBuilder);

}

private void DisableCascadingDelete(ModelBuilder modelBuilder)

{

var relationships = modelBuilder.Model.GetEntityTypes().SelectMany(e => e.GetForeignKeys());

foreach (var relationship in relationships)

{

relationship.DeleteBehavior = DeleteBehavior.Restrict;

}

}

}

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

PM> add-migration AddProductsTables

PM> update-database

1. Dentro del proyecto **Backend** copiamos el folder **Images** el cual puedes obtener de mi repositorio.
2. Borramos de la base de datos las **categorías** y **usuarios** que tengamos.
3. Modificamos el **SeedDb** para agregar registros a las nuevas tablas y de paso aprovechamos y creamos los usuarios y productos con fotos:

…

public class SeedDb

{

private readonly DataContext \_context;

private readonly IApiService \_apiService;

private readonly IUsersUnitOfWork \_usersUnitOfWork;

private readonly IFileStorage \_fileStorage;

public SeedDb(DataContext context, IApiService apiService, IUsersUnitOfWork usersUnitOfWork, IFileStorage fileStorage)

{

\_context = context;

\_apiService = apiService;

\_usersUnitOfWork = usersUnitOfWork;

\_fileStorage = fileStorage;

}

public async Task SeedAsync()

{

await \_context.Database.EnsureCreatedAsync();

//await CheckCountriesAsync();

await CheckCountriesFullAsync();

await CheckCategoriesAsync();

await CheckRolesAsync();

await CheckProductsAsync();

await CheckUserAsync("0001", "Juan", "Zuluaga", "zulu@yopmail.com", "322 311 4620", "Calle Luna Calle Sol", "JuanZuluaga.jpg", UserType.Admin);

await CheckUserAsync("0002", "Ledys", "Bedoya", "ledys@yopmail.com", "322 311 4620", "Calle Luna Calle Sol", "LedysBedoya.jpg", UserType.User);

await CheckUserAsync("0003", "Brad", "Pitt", "brad@yopmail.com", "322 311 4620", "Calle Luna Calle Sol", "Brad.jpg", UserType.User);

await CheckUserAsync("0004", "Angelina", "Jolie", "angelina@yopmail.com", "322 311 4620", "Calle Luna Calle Sol", "Angelina.jpg", UserType.User);

await CheckUserAsync("0005", "Bob", "Marley", "bob@yopmail.com", "322 311 4620", "Calle Luna Calle Sol", "bob.jpg", UserType.User);

await CheckUserAsync("0006", "Celia", "Cruz", "celia@yopmail.com", "322 311 4620", "Calle Luna Calle Sol", "celia.jpg", UserType.Admin);

await CheckUserAsync("0007", "Fredy", "Mercury", "fredy@yopmail.com", "322 311 4620", "Calle Luna Calle Sol", "fredy.jpg", UserType.User);

await CheckUserAsync("0008", "Hector", "Lavoe", "hector@yopmail.com", "322 311 4620", "Calle Luna Calle Sol", "hector.jpg", UserType.User);

await CheckUserAsync("0009", "Liv", "Taylor", "liv@yopmail.com", "322 311 4620", "Calle Luna Calle Sol", "liv.jpg", UserType.User);

await CheckUserAsync("0010", "Otep", "Shamaya", "otep@yopmail.com", "322 311 4620", "Calle Luna Calle Sol", "otep.jpg", UserType.User);

await CheckUserAsync("0011", "Ozzy", "Osbourne", "ozzy@yopmail.com", "322 311 4620", "Calle Luna Calle Sol", "ozzy.jpg", UserType.User);

await CheckUserAsync("0012", "Selena", "Quintanilla", "selenba@yopmail.com", "322 311 4620", "Calle Luna Calle Sol", "selena.jpg", UserType.User);

}

…

private async Task CheckCategoriesAsync()

{

if (!\_context.Categories.Any())

{

\_context.Categories.Add(new Category { Name = "Apple" });

\_context.Categories.Add(new Category { Name = "Autos" });

\_context.Categories.Add(new Category { Name = "Belleza" });

\_context.Categories.Add(new Category { Name = "Calzado" });

\_context.Categories.Add(new Category { Name = "Comida" });

\_context.Categories.Add(new Category { Name = "Cosmeticos" });

\_context.Categories.Add(new Category { Name = "Deportes" });

\_context.Categories.Add(new Category { Name = "Gamer" });

\_context.Categories.Add(new Category { Name = "Jugetes" });

\_context.Categories.Add(new Category { Name = "Mascotas" });

\_context.Categories.Add(new Category { Name = "Nutrición" });

\_context.Categories.Add(new Category { Name = "Ropa" });

\_context.Categories.Add(new Category { Name = "Tecnología" });

await \_context.SaveChangesAsync();

}

}

private async Task CheckProductsAsync()

{

if (!\_context.Products.Any())

{

await AddProductAsync("Adidas Barracuda", 270000M, 12F, new List<string>() { "Calzado", "Deportes" }, new List<string>() { "adidas\_barracuda.png" });

await AddProductAsync("Adidas Superstar", 250000M, 12F, new List<string>() { "Calzado", "Deportes" }, new List<string>() { "Adidas\_superstar.png" });

await AddProductAsync("Aguacate", 5000M, 500F, new List<string>() { "Comida" }, new List<string>() { "Aguacate1.png", "Aguacate2.png", "Aguacate3.png" });

await AddProductAsync("AirPods", 1300000M, 12F, new List<string>() { "Tecnología", "Apple" }, new List<string>() { "airpos.png", "airpos2.png" });

await AddProductAsync("Akai APC40 MKII", 2650000M, 12F, new List<string>() { "Tecnología" }, new List<string>() { "Akai1.png", "Akai2.png", "Akai3.png" });

await AddProductAsync("Apple Watch Ultra", 4500000M, 24F, new List<string>() { "Apple", "Tecnología" }, new List<string>() { "AppleWatchUltra1.png", "AppleWatchUltra2.png" });

await AddProductAsync("Audifonos Bose", 870000M, 12F, new List<string>() { "Tecnología" }, new List<string>() { "audifonos\_bose.png" });

await AddProductAsync("Bicicleta Ribble", 12000000M, 6F, new List<string>() { "Deportes" }, new List<string>() { "bicicleta\_ribble.png" });

await AddProductAsync("Camisa Cuadros", 56000M, 24F, new List<string>() { "Ropa" }, new List<string>() { "camisa\_cuadros.png" });

await AddProductAsync("Casco Bicicleta", 820000M, 12F, new List<string>() { "Deportes" }, new List<string>() { "casco\_bicicleta.png", "casco.png" });

await AddProductAsync("Gafas deportivas", 160000M, 24F, new List<string>() { "Deportes" }, new List<string>() { "Gafas1.png", "Gafas2.png", "Gafas3.png" });

await AddProductAsync("Hamburguesa triple carne", 25500M, 240F, new List<string>() { "Comida" }, new List<string>() { "Hamburguesa1.png", "Hamburguesa2.png", "Hamburguesa3.png" });

await AddProductAsync("iPad", 2300000M, 6F, new List<string>() { "Tecnología", "Apple" }, new List<string>() { "ipad.png" });

await AddProductAsync("iPhone 13", 5200000M, 6F, new List<string>() { "Tecnología", "Apple" }, new List<string>() { "iphone13.png", "iphone13b.png", "iphone13c.png", "iphone13d.png" });

await AddProductAsync("Johnnie Walker Blue Label 750ml", 1266700M, 18F, new List<string>() { "Licores" }, new List<string>() { "JohnnieWalker3.png", "JohnnieWalker2.png", "JohnnieWalker1.png" });

await AddProductAsync("KOOY Disfraz inflable de gallo para montar", 150000M, 28F, new List<string>() { "Juguetes" }, new List<string>() { "KOOY1.png", "KOOY2.png", "KOOY3.png" });

await AddProductAsync("Mac Book Pro", 12100000M, 6F, new List<string>() { "Tecnología", "Apple" }, new List<string>() { "mac\_book\_pro.png" });

await AddProductAsync("Mancuernas", 370000M, 12F, new List<string>() { "Deportes" }, new List<string>() { "mancuernas.png" });

await AddProductAsync("Mascarilla Cara", 26000M, 100F, new List<string>() { "Belleza" }, new List<string>() { "mascarilla\_cara.png" });

await AddProductAsync("New Balance 530", 180000M, 12F, new List<string>() { "Calzado", "Deportes" }, new List<string>() { "newbalance530.png" });

await AddProductAsync("New Balance 565", 179000M, 12F, new List<string>() { "Calzado", "Deportes" }, new List<string>() { "newbalance565.png" });

await AddProductAsync("Nike Air", 233000M, 12F, new List<string>() { "Calzado", "Deportes" }, new List<string>() { "nike\_air.png" });

await AddProductAsync("Nike Zoom", 249900M, 12F, new List<string>() { "Calzado", "Deportes" }, new List<string>() { "nike\_zoom.png" });

await AddProductAsync("Buso Adidas Mujer", 134000M, 12F, new List<string>() { "Ropa", "Deportes" }, new List<string>() { "buso\_adidas.png" });

await AddProductAsync("Suplemento Boots Original", 15600M, 12F, new List<string>() { "Nutrición" }, new List<string>() { "Boost\_Original.png" });

await AddProductAsync("Whey Protein", 252000M, 12F, new List<string>() { "Nutrición" }, new List<string>() { "whey\_protein.png" });

await AddProductAsync("Arnes Mascota", 25000M, 12F, new List<string>() { "Mascotas" }, new List<string>() { "arnes\_mascota.png" });

await AddProductAsync("Cama Mascota", 99000M, 12F, new List<string>() { "Mascotas" }, new List<string>() { "cama\_mascota.png" });

await AddProductAsync("Teclado Gamer", 67000M, 12F, new List<string>() { "Gamer", "Tecnología" }, new List<string>() { "teclado\_gamer.png" });

await AddProductAsync("Ring de Lujo 17", 1600000M, 33F, new List<string>() { "Autos" }, new List<string>() { "Ring1.png", "Ring2.png" });

await AddProductAsync("Silla Gamer", 980000M, 12F, new List<string>() { "Gamer", "Tecnología" }, new List<string>() { "silla\_gamer.png" });

await AddProductAsync("Mouse Gamer", 132000M, 12F, new List<string>() { "Gamer", "Tecnología" }, new List<string>() { "mouse\_gamer.png" });

await \_context.SaveChangesAsync();

}

}

private async Task AddProductAsync(string name, decimal price, float stock, List<string> categories, List<string> images)

{

Product prodcut = new()

{

Description = name,

Name = name,

Price = price,

Stock = stock,

ProductCategories = new List<ProductCategory>(),

ProductImages = new List<ProductImage>()

};

foreach (var categoryName in categories)

{

var category = await \_context.Categories.FirstOrDefaultAsync(c => c.Name == categoryName);

if (category != null)

{

prodcut.ProductCategories.Add(new ProductCategory { Category = category });

}

}

foreach (string? image in images)

{

var filePath = $"{Environment.CurrentDirectory}\\Images\\products\\{image}";

var fileBytes = File.ReadAllBytes(filePath);

var imagePath = await \_fileStorage.SaveFileAsync(fileBytes, "jpg", "products");

prodcut.ProductImages.Add(new ProductImage { Image = imagePath });

}

\_context.Products.Add(prodcut);

}

…

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

{

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

if (user == null)

{

var city = await \_context.Cities.FirstOrDefaultAsync(x => x.Name == "Medellín");

if (city == null)

{

city = await \_context.Cities.FirstOrDefaultAsync();

}

var filePath = $"{Environment.CurrentDirectory}\\Images\\users\\{image}";

var fileBytes = File.ReadAllBytes(filePath);

var imagePath = await \_fileStorage.SaveFileAsync(fileBytes, "jpg", "users");

user = new User

{

FirstName = firstName,

LastName = lastName,

Email = email,

UserName = email,

PhoneNumber = phone,

Address = address,

Document = document,

City = city,

UserType = userType,

Photo= imagePath,

};

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

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

var token = await \_usersUnitOfWork.GenerateEmailConfirmationTokenAsync(user);

await \_usersUnitOfWork.ConfirmEmailAsync(user, token);

}

return user;

}

…

1. Probamos lo que llevamos.
2. Creamos el **ProductDTO**:

using Microsoft.EntityFrameworkCore.Metadata.Internal;

using System.ComponentModel.DataAnnotations;

using System.ComponentModel.DataAnnotations.Schema;

namespace Orders.Shared.DTOs

{

public class ProductDTO

{

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 List<int>? ProductCategoryIds { get; set; }

public List<string>? ProductImages { get; set; }

}

}

1. Creamos el **IProductsRepository**:

using Orders.Shared.DTOs;

using Orders.Shared.Entities;

using Orders.Shared.Responses;

namespace Orders.Backend.Repositories.Interfaces

{

public interface IProductsRepository

{

Task<ActionResponse<Product>> GetAsync(int id);

Task<ActionResponse<IEnumerable<Product>>> GetAsync(PaginationDTO pagination);

Task<ActionResponse<int>> GetTotalPagesAsync(PaginationDTO pagination);

Task<ActionResponse<Product>> AddFullAsync(ProductDTO productDTO);

Task<ActionResponse<Product>> UpdateFullAsync(ProductDTO productDTO);

}

}

1. Creamos el **ProductsRepository**:

using Microsoft.EntityFrameworkCore;

using Orders.Backend.Data;

using Orders.Backend.Helpers;

using Orders.Backend.Repositories.Interfaces;

using Orders.Shared.DTOs;

using Orders.Shared.Entities;

using Orders.Shared.Responses;

namespace Orders.Backend.Repositories.Implementations

{

public class ProductsRepository : GenericRepository<Product>, IProductsRepository

{

private readonly DataContext \_context;

private readonly IFileStorage \_fileStorage;

public ProductsRepository(DataContext context, IFileStorage fileStorage) : base(context)

{

\_context = context;

\_fileStorage = fileStorage;

}

public override async Task<ActionResponse<IEnumerable<Product>>> GetAsync(PaginationDTO pagination)

{

var queryable = \_context.Products

.Include(x => x.ProductImages)

.Include(x => x.ProductCategories)

.AsQueryable();

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

{

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

}

return new ActionResponse<IEnumerable<Product>>

{

WasSuccess = true,

Result = await queryable

.OrderBy(x => x.Name)

.Paginate(pagination)

.ToListAsync()

};

}

public override async Task<ActionResponse<int>> GetTotalPagesAsync(PaginationDTO pagination)

{

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

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

{

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

}

double count = await queryable.CountAsync();

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

return new ActionResponse<int>

{

WasSuccess = true,

Result = totalPages

};

}

public override async Task<ActionResponse<Product>> GetAsync(int id)

{

var product = await \_context.Products

.Include(x => x.ProductImages)

.Include(x => x.ProductCategories!)

.ThenInclude(x => x.Category)

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

if (product == null)

{

return new ActionResponse<Product>

{

WasSuccess = false,

Message = "Producto no existe"

};

}

return new ActionResponse<Product>

{

WasSuccess = true,

Result = product

};

}

public async Task<ActionResponse<Product>> AddFullAsync(ProductDTO productDTO)

{

try

{

var newProduct = new Product

{

Name = productDTO.Name,

Description = productDTO.Description,

Price = productDTO.Price,

Stock = productDTO.Stock,

ProductCategories = new List<ProductCategory>(),

ProductImages = new List<ProductImage>()

};

foreach (var productImage in productDTO.ProductImages!)

{

var photoProduct = Convert.FromBase64String(productImage);

newProduct.ProductImages.Add(new ProductImage { Image = await \_fileStorage.SaveFileAsync(photoProduct, ".jpg", "products") });

}

foreach (var productCategoryId in productDTO.ProductCategoryIds!)

{

var category = await \_context.Categories.FirstOrDefaultAsync(x => x.Id == productCategoryId);

if (category != null)

{

newProduct.ProductCategories.Add(new ProductCategory { Category = category });

}

}

\_context.Add(newProduct);

await \_context.SaveChangesAsync();

return new ActionResponse<Product>

{

WasSuccess = true,

Result = newProduct

};

}

catch (DbUpdateException)

{

return new ActionResponse<Product>

{

WasSuccess = false,

Message = "Ya existe un producto con el mismo nombre."

};

}

catch (Exception exception)

{

return new ActionResponse<Product>

{

WasSuccess = false,

Message = exception.Message

};

}

}

public async Task<ActionResponse<Product>> UpdateFullAsync(ProductDTO productDTO)

{

try

{

var product = await \_context.Products

.Include(x => x.ProductCategories!)

.ThenInclude(x => x.Category)

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

if (product == null)

{

return new ActionResponse<Product>

{

WasSuccess = false,

Message = "Producto no existe"

};

}

product.Name = productDTO.Name;

product.Description = productDTO.Description;

product.Price = productDTO.Price;

product.Stock = productDTO.Stock;

\_context.ProductCategories.RemoveRange(product.ProductCategories!);

product.ProductCategories = new List<ProductCategory>();

foreach (var productCategoryId in productDTO.ProductCategoryIds!)

{

var category = await \_context.Categories.FindAsync(productCategoryId);

if (category != null)

{

\_context.ProductCategories.Add(new ProductCategory { CategoryId = category.Id, ProductId = product.Id });

}

}

\_context.Update(product);

await \_context.SaveChangesAsync();

return new ActionResponse<Product>

{

WasSuccess = true,

Result = product

};

}

catch (DbUpdateException)

{

return new ActionResponse<Product>

{

WasSuccess = false,

Message = "Ya existe un producto con el mismo nombre."

};

}

catch (Exception exception)

{

return new ActionResponse<Product>

{

WasSuccess = false,

Message = exception.Message

};

}

}

}

}

1. Creamos el **IProductsUnitOfWork**:

using Orders.Shared.DTOs;

using Orders.Shared.Entities;

using Orders.Shared.Responses;

namespace Orders.Backend.UnitsOfWork.Interfaces

{

public interface IProductsUnitOfWork

{

Task<ActionResponse<Product>> GetAsync(int id);

Task<ActionResponse<IEnumerable<Product>>> GetAsync(PaginationDTO pagination);

Task<ActionResponse<int>> GetTotalPagesAsync(PaginationDTO pagination);

Task<ActionResponse<Product>> AddFullAsync(ProductDTO productDTO);

Task<ActionResponse<Product>> UpdateFullAsync(ProductDTO productDTO);

}

}

1. Creamos el **ProductsUnitOfWork**:

using Orders.Backend.Repositories.Interfaces;

using Orders.Backend.UnitsOfWork.Interfaces;

using Orders.Shared.DTOs;

using Orders.Shared.Entities;

using Orders.Shared.Responses;

namespace Orders.Backend.UnitsOfWork.Implementations

{

public class ProductsUnitOfWork : GenericUnitOfWork<Product>, IProductsUnitOfWork

{

private readonly IProductsRepository \_productsRepository;

public ProductsUnitOfWork(IGenericRepository<Product> repository, IProductsRepository productsRepository) : base(repository)

{

\_productsRepository = productsRepository;

}

public override async Task<ActionResponse<IEnumerable<Product>>> GetAsync(PaginationDTO pagination) => await \_productsRepository.GetAsync(pagination);

public override async Task<ActionResponse<int>> GetTotalPagesAsync(PaginationDTO pagination) => await \_productsRepository.GetTotalPagesAsync(pagination);

public override async Task<ActionResponse<Product>> GetAsync(int id) => await \_productsRepository.GetAsync(id);

public async Task<ActionResponse<Product>> AddFullAsync(ProductDTO productDTO) => await \_productsRepository.AddFullAsync(productDTO);

public async Task<ActionResponse<Product>> UpdateFullAsync(ProductDTO productDTO) => await \_productsRepository.UpdateFullAsync(productDTO);

}

}

1. Adicinamos las nuevas inyecciones en el **Program** del **Backend**:

builder.Services.AddScoped<ICategoriesRepository, CategoriesRepository>();

builder.Services.AddScoped<ICitiesRepository, CitiesRepository>();

builder.Services.AddScoped<ICountriesRepository, CountriesRepository>();

builder.Services.AddScoped<IProductsRepository, ProductsRepository>();

builder.Services.AddScoped<IStatesRepository, StatesRepository>();

builder.Services.AddScoped<IUsersRepository, UsersRepository>();

builder.Services.AddScoped<ICategoriesUnitOfWork, CategoriesUnitOfWork>();

builder.Services.AddScoped<ICitiesUnitOfWork, CitiesUnitOfWork>();

builder.Services.AddScoped<ICountriesUnitOfWork, CountriesUnitOfWork>();

builder.Services.AddScoped<IProductsUnitOfWork, ProductsUnitOfWork>();

builder.Services.AddScoped<IStatesUnitOfWork, StatesUnitOfWork>();

builder.Services.AddScoped<IUsersUnitOfWork, UsersUnitOfWork>();

1. Creamos el **ProductsController**:

using Microsoft.AspNetCore.Authentication.JwtBearer;

using Microsoft.AspNetCore.Authorization;

using Microsoft.AspNetCore.Mvc;

using Orders.Backend.UnitsOfWork.Interfaces;

using Orders.Shared.DTOs;

using Orders.Shared.Entities;

namespace Orders.Backend.Controllers

{

[ApiController]

[Authorize(AuthenticationSchemes = JwtBearerDefaults.AuthenticationScheme)]

[Route("api/[controller]")]

public class ProductsController : GenericController<Product>

{

private readonly IProductsUnitOfWork \_productsUnitOfWork;

public ProductsController(IGenericUnitOfWork<Product> unitOfWork, IProductsUnitOfWork productsUnitOfWork) : base(unitOfWork)

{

\_productsUnitOfWork = productsUnitOfWork;

}

[HttpGet]

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

{

var response = await \_productsUnitOfWork.GetAsync(pagination);

if (response.WasSuccess)

{

return Ok(response.Result);

}

return BadRequest();

}

[HttpGet("totalPages")]

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

{

var action = await \_productsUnitOfWork.GetTotalPagesAsync(pagination);

if (action.WasSuccess)

{

return Ok(action.Result);

}

return BadRequest();

}

[HttpGet("{id}")]

public override async Task<IActionResult> GetAsync(int id)

{

var action = await \_productsUnitOfWork.GetAsync(id);

if (action.WasSuccess)

{

return Ok(action.Result);

}

return NotFound(action.Message);

}

[HttpPost("full")]

public async Task<IActionResult> PostFullAsync(ProductDTO productDTO)

{

var action = await \_productsUnitOfWork.AddFullAsync(productDTO);

if (action.WasSuccess)

{

return Ok(action.Result);

}

return NotFound(action.Message);

}

[HttpPut("full")]

public async Task<IActionResult> PutFullAsync(ProductDTO productDTO)

{

var action = await \_productsUnitOfWork.UpdateFullAsync(productDTO);

if (action.WasSuccess)

{

return Ok(action.Result);

}

return NotFound(action.Message);

}

}

}

1. Dentro de **Pages** creamos la carpeta **Products** y dentro de esta creamos el **ProductsIndex.razor** y **ProductsIndex.razor.cs**:

using CurrieTechnologies.Razor.SweetAlert2;

using Microsoft.AspNetCore.Authorization;

using Microsoft.AspNetCore.Components;

using Orders.Frontend.Repositories;

using Orders.Shared.Entities;

namespace Orders.Frontend.Pages.Products

{

[Authorize(Roles = "Admin")]

public partial class ProductsIndex

{

private int currentPage = 1;

private int totalPages;

[Inject] private NavigationManager NavigationManager { get; set; } = null!;

[Inject] private SweetAlertService SweetAlertService { get; set; } = null!;

[Inject] private IRepository Repository { get; set; } = null!;

public List<Product>? Products { get; set; }

[Parameter, SupplyParameterFromQuery] public string Page { get; set; } = string.Empty;

[Parameter, SupplyParameterFromQuery] public string Filter { get; set; } = string.Empty;

[Parameter, SupplyParameterFromQuery] public int RecordsNumber { get; set; } = 10;

protected override async Task OnInitializedAsync()

{

await LoadAsync();

}

private async Task SelectedRecordsNumberAsync(int recordsnumber)

{

RecordsNumber = recordsnumber;

int page = 1;

await LoadAsync(page);

await SelectedPageAsync(page);

}

private async Task FilterCallBack(string filter)

{

Filter = filter;

await ApplyFilterAsync();

StateHasChanged();

}

private async Task SelectedPageAsync(int page)

{

currentPage = page;

await LoadAsync(page);

}

private async Task LoadAsync(int page = 1)

{

if (!string.IsNullOrWhiteSpace(Page))

{

page = Convert.ToInt32(Page);

}

var ok = await LoadListAsync(page);

if (ok)

{

await LoadPagesAsync();

}

}

private void ValidateRecordsNumber(int recordsnumber)

{

if (recordsnumber == 0)

{

RecordsNumber = 10;

}

}

private async Task<bool> LoadListAsync(int page)

{

ValidateRecordsNumber(RecordsNumber);

var url = $"api/products?page={page}&recordsnumber={RecordsNumber}";

if (!string.IsNullOrEmpty(Filter))

{

url += $"&filter={Filter}";

}

var response = await Repository.GetAsync<List<Product>>(url);

if (response.Error)

{

var message = await response.GetErrorMessageAsync();

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

return false;

}

Products = response.Response;

return true;

}

private async Task LoadPagesAsync()

{

ValidateRecordsNumber(RecordsNumber);

var url = $"api/products/totalPages?recordsnumber={RecordsNumber}";

if (!string.IsNullOrEmpty(Filter))

{

url += $"&filter={Filter}";

}

var response = await Repository.GetAsync<int>(url);

if (response.Error)

{

var message = await response.GetErrorMessageAsync();

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

return;

}

totalPages = response.Response;

}

private async Task Delete(int productId)

{

var result = await SweetAlertService.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.DeleteAsync<Product>($"api/products/{productId}");

if (responseHttp.Error)

{

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

{

NavigationManager.NavigateTo("/");

return;

}

var mensajeError = await responseHttp.GetErrorMessageAsync();

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

return;

}

await LoadAsync(1);

}

private async Task ApplyFilterAsync()

{

int page = 1;

await LoadAsync(page);

await SelectedPageAsync(page);

}

}

}

1. Luego modificamos el **ProductsIndex.razor**:

@page "/products"

<div class="card">

<div class="card-header">

<span>

<i class="bi bi-box2" /> Productos

<a class="btn btn-sm btn-primary float-end" href="/products/create"><i class="bi bi-plus-square" /> Nuevo Producto</a>

</span>

</div>

<div class="card-body">

<Filter PlaceHolder="Buscar producto..." Callback=@FilterCallBack />

<GenericList MyList="Products">

<Body>

<Pagination CurrentPage="currentPage"

TotalPages="totalPages"

SelectedPage="SelectedPageAsync"

RecordsNumber="SelectedRecordsNumberAsync" />

<table class="table table-striped">

<thead>

<tr>

<th>Nombre</th>

<th>Descripción</th>

<th>Precio</th>

<th>Inventario</th>

<th>Categorías</th>

<th>Imagenes</th>

<th>Imagen Principal</th>

<th style="width:168px"></th>

</tr>

</thead>

<tbody>

@foreach (var product in Products!)

{

<tr>

<td>

@product.Name

</td>

<td>

@product.Description

</td>

<td>

@($"{product.Price:C2}")

</td>

<td>

@($"{product.Stock:N2}")

</td>

<td>

@product.ProductCategoriesNumber

</td>

<td>

@product.ProductImagesNumber

</td>

<td>

<img src="@product.MainImage" style="width:100px;" />

</td>

<td>

<a href="/products/edit/@product.Id" class="btn btn-warning btn-sm"><i class="bi bi-pencil" /> Editar</a>

<button class="btn btn-danger btn-sm" @onclick=@(() => Delete(product.Id))><i class="bi bi-trash" /> Borrar</button>

</td>

</tr>

}

</tbody>

</table>

</Body>

</GenericList>

</div>

</div>

1. Modificamos el **NavMenu.razor.css**:

.bi-box2-fill-nav-menu {

background-image: url("data:image/svg+xml,%3Csvg xmlns='http://www.w3.org/2000/svg' width='16' height='16' fill='white' class='bi bi-list-nested' viewBox='0 0 16 16'%3E%3Cpath fill-rule='evenodd' d='M2.95.4a1 1 0 0 1 .8-.4h8.5a1 1 0 0 1 .8.4l2.85 3.8a.5.5 0 0 1 .1.3V15a1 1 0 0 1-1 1H1a1 1 0 0 1-1-1V4.5a.5.5 0 0 1 .1-.3zM7.5 1H3.75L1.5 4h6zm1 0v3h6l-2.25-3zM15 5H1v10h14z'/%3E%3C/svg%3E");

}

1. Modificamos el **NavMenu.razor**:

…

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

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

<span class="bi bi-globe-americas-fill-nav-menu" aria-hidden="true"></span> Paises

</NavLink>

</div>

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

<NavLink class="nav-link" href="/products">

<span class="bi bi-box2-fill-nav-menu" aria-hidden="true"></span> Productos

</NavLink>

</div>

…

1. Probamos y hacemos el **commit** de lo que llevamos.

## Creando nuevos productos

1. Creamos el componente genérico para poder seleccionar varitas categorías. Primero creamos en **Orders. Frontend.Helpers** la clase **MultipleSelectorModel**:

namespace Orders. Frontend.Helpers

{

public class MultipleSelectorModel

{

public MultipleSelectorModel(string key, string value)

{

Key = key;

Value = value;

}

public string Key { get; set; }

public string Value { get; set; }

}

}

1. Le agregamos estas líneas a nuestro archivo de estilos **app.css**:

.multiple-selector {

display: flex;

}

.selectable-ul {

height: 200px;

overflow-y: auto;

list-style-type: none;

width: 170px;

padding: 0;

border-radius: 3px;

border: 1px solid #ccc;

}

.selectable-ul li {

cursor: pointer;

border-bottom: 1px #eee solid;

padding: 2px 10px;

font-size: 14px;

}

.selectable-ul li:hover {

background-color: #08c

}

.multiple-selector-botones {

display: flex;

flex-direction: column;

justify-content: center;

padding: 5px

}

.multiple-selector-botones button {

margin: 5px;

}

1. Creamos en **Shared** nuestro **MultipleSelector.razor** y**MultipleSelector.razor.cs**:

using Microsoft.AspNetCore.Components;

using Orders.Frontend.Helpers;

namespace Orders.Frontend.Shared

{

public partial class MultipleSelector

{

private string addAllText = ">>";

private string removeAllText = "<<";

[Parameter]

public List<MultipleSelectorModel> NonSelected { get; set; } = new();

[Parameter]

public List<MultipleSelectorModel> Selected { get; set; } = new();

private void Select(MultipleSelectorModel item)

{

NonSelected.Remove(item);

Selected.Add(item);

}

private void Unselect(MultipleSelectorModel item)

{

Selected.Remove(item);

NonSelected.Add(item);

}

private void SelectAll()

{

Selected.AddRange(NonSelected);

NonSelected.Clear();

}

private void UnselectAll()

{

NonSelected.AddRange(Selected);

Selected.Clear();

}

}

}

1. Luego modificamos el **MultipleSelector.razor**:

<div class="multiple-selector">

<ul class="selectable-ul">

@foreach (var item in NonSelected)

{

<li @onclick=@(() => Select(item))>@item.Value</li>

}

</ul>

<div class="selector-multiple-botones">

<div class="mx-2 my-2">

<p><button type="button" @onclick="SelectAll">@addAllText</button></p>

</div>

<div class="mx-2 my-2">

<p><button type="button" @onclick="UnselectAll">@removeAllText</button></p>

</div>

</div>

<ul class="selectable-ul">

@foreach (var item in Selected)

{

<li @onclick=@(() => Unselect(item))>@item.Value</li>

}

</ul>

</div>

1. Dentro de **Pages/Products** creamos el **ProductForm.razor** y **ProductForm.razor.cs**:

using CurrieTechnologies.Razor.SweetAlert2;

using Microsoft.AspNetCore.Components.Forms;

using Microsoft.AspNetCore.Components.Routing;

using Microsoft.AspNetCore.Components;

using Orders.Frontend.Helpers;

using Orders.Shared.DTOs;

using Orders.Shared.Entities;

namespace Orders.Frontend.Pages.Products

{

public partial class ProductForm

{

private EditContext editContext = null!;

private string? imageUrl;

private List<MultipleSelectorModel> selected { get; set; } = new();

private List<MultipleSelectorModel> nonSelected { get; set; } = new();

[Inject] private SweetAlertService SweetAlertService { get; set; } = null!;

[Parameter, EditorRequired] public ProductDTO ProductDTO { get; set; } = null!;

[Parameter, EditorRequired] public EventCallback OnValidSubmit { get; set; }

[Parameter, EditorRequired] public EventCallback ReturnAction { get; set; }

[Parameter, EditorRequired] public List<Category> NonSelectedCategories { get; set; } = new();

[Parameter] public bool IsEdit { get; set; } = false;

[Parameter] public EventCallback AddImageAction { get; set; }

[Parameter] public EventCallback RemoveImageAction { get; set; }

[Parameter] public List<Category> SelectedCategories { get; set; } = new();

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

protected override void OnInitialized()

{

editContext = new(ProductDTO);

selected = SelectedCategories.Select(x => new MultipleSelectorModel(x.Id.ToString(), x.Name)).ToList();

nonSelected = NonSelectedCategories.Select(x => new MultipleSelectorModel(x.Id.ToString(), x.Name)).ToList();

}

private void ImageSelected(string imagenBase64)

{

if (ProductDTO.ProductImages is null)

{

ProductDTO.ProductImages = new List<string>();

}

ProductDTO.ProductImages!.Add(imagenBase64);

imageUrl = null;

}

private async Task OnDataAnnotationsValidatedAsync()

{

ProductDTO.ProductCategoryIds = selected.Select(x => int.Parse(x.Key)).ToList();

await OnValidSubmit.InvokeAsync();

}

private async Task OnBeforeInternalNavigation(LocationChangingContext context)

{

var formWasEdited = editContext.IsModified();

if (!formWasEdited)

{

return;

}

if (FormPostedSuccessfully)

{

return;

}

var result = await SweetAlertService.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. Luego modificamos el **ProductForm.razor**:

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

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

<DataAnnotationsValidator />

<div class="card">

<div class="card-header">

<span>

<i class="bi bi-box2" /> Crear Nuevo Producto

<a class="btn btn-sm btn-success float-end" href="/products"><i class="bi bi-arrow-left" /> Regresar</a>

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

</span>

</div>

<div class="card-body">

<div class="row">

<div class="col-6">

<div class="mb-3">

<label>Nombre:</label>

<div>

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

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

</div>

</div>

<div class="mb-3">

<label>Descripción:</label>

<div>

<InputText class="form-control" @bind-Value="@ProductDTO.Description" />

<ValidationMessage For="@(() => ProductDTO.Description)" />

</div>

</div>

<div class="mb-3">

<label>Precio:</label>

<div>

<InputNumber class="form-control" @bind-Value="@ProductDTO.Price" />

<ValidationMessage For="@(() => ProductDTO.Price)" />

</div>

</div>

<div class="mb-3">

<label>Inventario:</label>

<div>

<InputNumber class="form-control" @bind-Value="@ProductDTO.Stock" />

<ValidationMessage For="@(() => ProductDTO.Stock)" />

</div>

</div>

</div>

<div class="col-6">

<div class="mb-3">

<label>Categorías:</label>

<div>

<MultipleSelector NonSelected="nonSelected" Selected="selected" />

</div>

</div>

<div class="mb-3">

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

</div>

@if (IsEdit)

{

<div class="mb-3">

<button type="button" class="btn btn-outline-primary" @onclick="AddImageAction"><i class="bi bi-cart-plus" /> Agregar Imagenes</button>

<button type="button" class="btn btn-outline-danger" @onclick="RemoveImageAction"><i class="bi bi-trash" /> Eliminar Última Imagén</button>

</div>

}

</div>

</div>

</div>

</div>

</EditForm>

@\*@if (IsEdit && ProductDTO.ProductImages is not null)

{

<CarouselView Images="ProductDTO.ProductImages" />

}\*@

1. Dentro de **Pages/Products** creamos el **ProductCreate.razor** y **ProductCreate.razor.cs**:

using CurrieTechnologies.Razor.SweetAlert2;

using Microsoft.AspNetCore.Authorization;

using Microsoft.AspNetCore.Components;

using Orders.Frontend.Repositories;

using Orders.Shared.DTOs;

using Orders.Shared.Entities;

namespace Orders.Frontend.Pages.Products

{

[Authorize(Roles = "Admin")]

public partial class ProductCreate

{

private ProductDTO productDTO = new()

{

ProductCategoryIds = new List<int>(),

ProductImages = new List<string>()

};

private ProductForm? productForm;

private List<Category> selectedCategories = new();

private List<Category> nonSelectedCategories = new();

private bool loading = true;

[Inject] private IRepository Repository { get; set; } = null!;

[Inject] private NavigationManager NavigationManager { get; set; } = null!;

[Inject] private SweetAlertService SweetAlertService { get; set; } = null!;

protected override async Task OnInitializedAsync()

{

var httpActionResponse = await Repository.GetAsync<List<Category>>("/api/categories/combo");

loading = false;

if (httpActionResponse.Error)

{

var message = await httpActionResponse.GetErrorMessageAsync();

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

return;

}

nonSelectedCategories = httpActionResponse.Response!;

}

private async Task CreateAsync()

{

var httpActionResponse = await Repository.PostAsync("/api/products/full", productDTO);

if (httpActionResponse.Error)

{

var message = await httpActionResponse.GetErrorMessageAsync();

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

return;

}

Return();

}

private void Return()

{

productForm!.FormPostedSuccessfully = true;

NavigationManager.NavigateTo($"/products");

}

}

}

1. Modiicamos el **ProductCreate.razor**:

@page "/products/create"

@if (loading)

{

<Loading />

}

else

{

<ProductForm @ref="productForm" ProductDTO="productDTO" NonSelectedCategories="nonSelectedCategories" OnValidSubmit="CreateAsync" ReturnAction="Return" />

}

1. Probamos y hacemos el **commit** de lo que hemos logrado hasta el momento, corra la App con **Ctrl + F5**, para que tome los cambios en el CSS.

## Empezar con la edición de productos y colocar las imágenes en un carrusel

1. Para nuestro componente de Carrusel vamos a utilizar las librerías de **MudBlazor**, la documentación está en <https://mudblazor.com/getting-started/installation#prerequisites> primero procedemos con la instalación.
2. Agregamos el nuget **MudBlazor** al **Frontend**.
3. En el **\_Imports.razor** agregamos la línea:

@using MudBlazor

1. Agregamos al **index.html** la hoja de estilos y los scripts:

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="utf-8" />

<meta name="viewport" content="width=device-width, initial-scale=1.0" />

<title>Orders.Frontend</title>

<base href="/" />

<link rel="stylesheet" href="css/bootstrap/bootstrap.min.css" />

<link rel="stylesheet" href="css/app.css" />

<link rel="icon" type="image/png" href="favicon.png" />

<link href="Orders.Frontend.styles.css" rel="stylesheet" />

<link rel="stylesheet"

href="https://cdn.jsdelivr.net/npm/bootstrap-icons@1.11.1/font/bootstrap-icons.css"

integrity="sha384-4LISF5TTJX/fLmGSxO53rV4miRxdg84mZsxmO8Rx5jGtp/LbrixFETvWa5a6sESd"

crossorigin="anonymous">

<link href="https://fonts.googleapis.com/css?family=Roboto:300,400,500,700&display=swap"

rel="stylesheet" />

<link href="\_content/MudBlazor/MudBlazor.min.css"

rel="stylesheet" />

</head>

<body>

<div id="app">

<svg class="loading-progress">

<circle r="40%" cx="50%" cy="50%" />

<circle r="40%" cx="50%" cy="50%" />

</svg>

<div class="loading-progress-text"></div>

</div>

<div id="blazor-error-ui">

An unhandled error has occurred.

<a href="" class="reload">Reload</a>

<a class="dismiss">🗙</a>

</div>

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

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

<script src="\_content/MudBlazor/MudBlazor.min.js"></script>

</body>

</html>

1. Injectamos en el **Program** del proyecto  **Frontend**:

builder.Services.AddMudServices();

1. Creamos el componente compartido **CarouselView.razor** y **CarouselView.razor.cs**:

using Microsoft.AspNetCore.Components;

using MudBlazor;

namespace Orders.Frontend.Shared

{

public partial class CarouselView

{

private bool arrows = true;

private bool bullets = true;

private bool enableSwipeGesture = true;

private bool autocycle = true;

private Transition transition = Transition.Slide;

[Parameter, EditorRequired] public List<string> Images { get; set; } = null!;

}

}

1. Luego modificamos el **CarouselView.razor**:

<div class="my-2">

<MudCarousel Class="mud-width-full" Style="height:200px;" ShowArrows="@arrows" ShowBullets="@bullets" EnableSwipeGesture="@enableSwipeGesture" AutoCycle="@autocycle" TData="object">

@foreach (var image in Images)

{

<MudCarouselItem Transition="transition" Color="@Color.Primary">

<div class="d-flex" style="height:100%; justify-content:center">

<img src="@image" />

</div>

</MudCarouselItem>

}

</MudCarousel>

</div>

1. Modificamos el **ProductForm**:

…

</EditForm>

@if (IsEdit && ProductDTO.ProductImages is not null)

{

<CarouselView Images="ProductDTO.ProductImages" />

}

…

1. Creamos el **ProductEdit.razor** y **ProductEdit.razor.cs**:

using System.Data;

using CurrieTechnologies.Razor.SweetAlert2;

using Microsoft.AspNetCore.Authorization;

using Microsoft.AspNetCore.Components;

using Orders.Frontend.Repositories;

using Orders.Shared.DTOs;

using Orders.Shared.Entities;

namespace Orders.Frontend.Pages.Products

{

[Authorize(Roles = "Admin")]

public partial class ProductEdit

{

private ProductDTO productDTO = new()

{

ProductCategoryIds = new List<int>(),

ProductImages = new List<string>()

};

private ProductForm? productForm;

private List<Category> selectedCategories = new();

private List<Category> nonSelectedCategories = new();

private bool loading = true;

private Product? product;

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

[Inject] private IRepository Repository { get; set; } = null!;

[Inject] private NavigationManager NavigationManager { get; set; } = null!;

[Inject] private SweetAlertService SweetAlertService { get; set; } = null!;

protected override async Task OnInitializedAsync()

{

await LoadProductAsync();

await LoadCategoriesAsync();

}

private async Task AddImageAsync()

{

}

private async Task RemoveImageAsyc()

{

}

private async Task LoadProductAsync()

{

loading = true;

var httpActionResponse = await Repository.GetAsync<Product>($"/api/products/{ProductId}");

if (httpActionResponse.Error)

{

loading = false;

var message = await httpActionResponse.GetErrorMessageAsync();

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

return;

}

product = httpActionResponse.Response!;

productDTO = ToProductDTO(product);

loading = false;

}

private ProductDTO ToProductDTO(Product product)

{

return new ProductDTO

{

Description = product.Description,

Id = product.Id,

Name = product.Name,

Price = product.Price,

Stock = product.Stock,

ProductCategoryIds = product.ProductCategories!.Select(x => x.CategoryId).ToList(),

ProductImages = product.ProductImages!.Select(x => x.Image).ToList()

};

}

private async Task LoadCategoriesAsync()

{

loading = true;

var httpActionResponse = await Repository.GetAsync<List<Category>>("/api/categories/combo");

if (httpActionResponse.Error)

{

loading = false;

var message = await httpActionResponse.GetErrorMessageAsync();

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

return;

}

var categories = httpActionResponse.Response!;

foreach (var category in categories!)

{

var found = product!.ProductCategories!.FirstOrDefault(x => x.CategoryId == category.Id);

if (found == null)

{

nonSelectedCategories.Add(category);

}

else

{

selectedCategories.Add(category);

}

}

loading = false;

}

private async Task SaveChangesAsync()

{

var httpActionResponse = await Repository.PutAsync("/api/products/full", productDTO);

if (httpActionResponse.Error)

{

var message = await httpActionResponse.GetErrorMessageAsync();

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

return;

}

Return();

}

private void Return()

{

productForm!.FormPostedSuccessfully = true;

NavigationManager.NavigateTo($"/products");

}

}

}

1. Luego agregamos el **ProductEdit.razor.cs**:

@page "/products/edit/{ProductId:int}"

@if (loading)

{

<Loading />

}

else

{

<ProductForm @ref="productForm" ProductDTO="productDTO" SelectedCategories="selectedCategories" NonSelectedCategories="nonSelectedCategories" OnValidSubmit="SaveChangesAsync" ReturnAction="Return" IsEdit=true AddImageAction="AddImageAsync" RemoveImageAction="RemoveImageAsyc" />

}

1. Probamos y hacemos el **commit** de lo que hemos logrado hasta el momento, corra la App con **Ctrl + F5**, para que tome los cambios en el CSS.

## Agregando y eliminando imágenes a los productos y terminando la edición de producto

1. Dento de **Orders.Shared.DTOs** creamos el **ImageDTO**.

using System.ComponentModel.DataAnnotations;

namespace Orders.Shared.DTOs

{

public class ImageDTO

{

[Required]

public int ProductId { get; set; }

[Required]

public List<string> Images { get; set; } = null!;

}

}

1. Modificamos el **IProductsRepository**:

Task<ActionResponse<ImageDTO>> AddImageAsync(ImageDTO imageDTO);

Task<ActionResponse<ImageDTO>> RemoveLastImageAsync(ImageDTO imageDTO);

1. Modificamos el **ProductsRepository**:

public async Task<ActionResponse<ImageDTO>> AddImageAsync(ImageDTO imageDTO)

{

var product = await \_context.Products

.Include(x => x.ProductImages)

.FirstOrDefaultAsync(x => x.Id == imageDTO.ProductId);

if (product == null)

{

return new ActionResponse<ImageDTO>

{

WasSuccess = false,

Message = "Producto no existe"

};

}

for (int i = 0; i < imageDTO.Images.Count; i++)

{

if (!imageDTO.Images[i].StartsWith("https://"))

{

var photoProduct = Convert.FromBase64String(imageDTO.Images[i]);

imageDTO.Images[i] = await \_fileStorage.SaveFileAsync(photoProduct, ".jpg", "products");

product.ProductImages!.Add(new ProductImage { Image = imageDTO.Images[i] });

}

}

\_context.Update(product);

await \_context.SaveChangesAsync();

return new ActionResponse<ImageDTO>

{

WasSuccess = true,

Result = imageDTO

};

}

public async Task<ActionResponse<ImageDTO>> RemoveLastImageAsync(ImageDTO imageDTO)

{

var product = await \_context.Products

.Include(x => x.ProductImages)

.FirstOrDefaultAsync(x => x.Id == imageDTO.ProductId);

if (product == null)

{

return new ActionResponse<ImageDTO>

{

WasSuccess = false,

Message = "Producto no existe"

};

}

if (product.ProductImages is null || product.ProductImages.Count == 0)

{

return new ActionResponse<ImageDTO>

{

WasSuccess = true,

Result = imageDTO

};

}

var lastImage = product.ProductImages.LastOrDefault();

await \_fileStorage.RemoveFileAsync(lastImage!.Image, "products");

\_context.ProductImages.Remove(lastImage);

await \_context.SaveChangesAsync();

imageDTO.Images = product.ProductImages.Select(x => x.Image).ToList();

return new ActionResponse<ImageDTO>

{

WasSuccess = true,

Result = imageDTO

};

}

1. Modificamos el **IProductsUnitOfWork**:

Task<ActionResponse<ImageDTO>> AddImageAsync(ImageDTO imageDTO);

Task<ActionResponse<ImageDTO>> RemoveLastImageAsync(ImageDTO imageDTO);

1. Modificamos el **ProductsUnitOfWork**:

public async Task<ActionResponse<ImageDTO>> AddImageAsync(ImageDTO imageDTO) => await \_productsRepository.AddImageAsync(imageDTO);

public async Task<ActionResponse<ImageDTO>> RemoveLastImageAsync(ImageDTO imageDTO) => await \_productsRepository.RemoveLastImageAsync(imageDTO);

1. Modificamos el **ProductsController**.

[HttpPost("addImages")]

public async Task<IActionResult> PostAddImagesAsync(ImageDTO imageDTO)

{

var action = await \_productsUnitOfWork.AddImageAsync(imageDTO);

if (action.WasSuccess)

{

return Ok(action.Result);

}

return BadRequest(action.Message);

}

[HttpPost("removeLastImage")]

public async Task<IActionResult> PostRemoveLastImageAsync(ImageDTO imageDTO)

{

var action = await \_productsUnitOfWork.RemoveLastImageAsync(imageDTO);

if (action.WasSuccess)

{

return Ok(action.Result);

}

return BadRequest(action.Message);

}

1. Modificamos el **CarouselView.razor**.

<div class="my-2">

<MudCarousel Class="mud-width-full" Style="height:200px;" ShowArrows="@arrows" ShowBullets="@bullets" EnableSwipeGesture="@enableSwipeGesture" AutoCycle="@autocycle" TData="object">

@foreach (var image in Images)

{

@if (image.StartsWith("https://"))

{

<MudCarouselItem Transition="transition" Color="@Color.Primary">

<div class="d-flex" style="height:100%; justify-content:center">

<img src="@image" />

</div>

</MudCarouselItem>

}

}

</MudCarousel>

</div>

1. Modificamos el **ProductEdit.razor.cs**.

private async Task AddImageAsync()

{

if (productDTO.ProductImages is null || productDTO.ProductImages.Count == 0)

{

return;

}

var imageDTO = new ImageDTO

{

ProductId = ProductId,

Images = productDTO.ProductImages!

};

var httpActionResponse = await Repository.PostAsync<ImageDTO, ImageDTO>("/api/products/addImages", imageDTO);

if (httpActionResponse.Error)

{

var message = await httpActionResponse.GetErrorMessageAsync();

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

return;

}

productDTO.ProductImages = httpActionResponse.Response!.Images;

var toast = SweetAlertService.Mixin(new SweetAlertOptions

{

Toast = true,

Position = SweetAlertPosition.BottomEnd,

ShowConfirmButton = true,

Timer = 3000

});

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

}

private async Task RemoveImageAsyc()

{

if (productDTO.ProductImages is null || productDTO.ProductImages.Count == 0)

{

return;

}

var imageDTO = new ImageDTO

{

ProductId = ProductId,

Images = productDTO.ProductImages!

};

var httpActionResponse = await Repository.PostAsync<ImageDTO, ImageDTO>("/api/products/removeLastImage", imageDTO);

if (httpActionResponse.Error)

{

var message = await httpActionResponse.GetErrorMessageAsync();

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

return;

}

productDTO.ProductImages = httpActionResponse.Response!.Images;

var toast = SweetAlertService.Mixin(new SweetAlertOptions

{

Toast = true,

Position = SweetAlertPosition.BottomEnd,

ShowConfirmButton = true,

Timer = 3000

});

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

}

1. Probamos y hacemos el **commit** de lo que hemos logrado hasta el momento, corra la App con **Ctrl + F5**, para que tome los cambios en el CSS.

## Borrar registros relacionados de productos

1. Si itentemos borrar un registro. Nos genera error por los registros relacionados. Vamos a corregir eso.
2. Modicamos el **IProductsRepository**:

Task<ActionResponse<Product>> DeleteAsync(int id);

1. Modicamos el **ProductsRepository**:

public override async Task<ActionResponse<Product>> DeleteAsync(int id)

{

var product = await \_context.Products

.Include(x => x.ProductCategories)

.Include(x => x.ProductImages)

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

if (product == null)

{

return new ActionResponse<Product>

{

WasSuccess = false,

Message = "Producto no encontrado"

};

}

foreach (var productImage in product.ProductImages!)

{

await \_fileStorage.RemoveFileAsync(productImage.Image, "products");

}

try

{

\_context.ProductCategories.RemoveRange(product.ProductCategories!);

\_context.ProductImages.RemoveRange(product.ProductImages!);

\_context.Products.Remove(product);

await \_context.SaveChangesAsync();

return new ActionResponse<Product>

{

WasSuccess = true,

};

}

catch

{

return new ActionResponse<Product>

{

WasSuccess = false,

Message = "No se puede borrar el producto, porque tiene registros relacionados"

};

}

}

1. Modicamos el **IProductsUnitOfWork**:

Task<ActionResponse<Product>> DeleteAsync(int id);

1. Modicamos el **ProductsUnitOfWork**:

public override async Task<ActionResponse<Product>> DeleteAsync(int id) => await \_productsRepository.DeleteAsync(id);

1. Modicamos el **ProductsController**:

[HttpDelete("{id}")]

public override async Task<IActionResult> DeleteAsync(int id)

{

var action = await \_productsUnitOfWork.DeleteAsync(id);

if (!action.WasSuccess)

{

return NotFound();

}

return NoContent();

}

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

## Creando el “Home” de nuestra aplicación

1. Modificamos el **ProductsController** y le colocamos el **[AllowAnonymous]** a todos los **GET** de este controlador.
2. Agregamos el **Home.razor.css**:

.card {

display: flex;

flex-direction: column;

justify-content: space-between;

border: 1px solid lightgray;

box-shadow: 2px 2px 8px 4px #d3d3d3d1;

border-radius: 15px;

font-family: sans-serif;

margin: 5px;

}

1. Agregamos el **Home.razor.cs**:

using CurrieTechnologies.Razor.SweetAlert2;

using Microsoft.AspNetCore.Components;

using Orders.Frontend.Repositories;

using Orders.Shared.Entities;

namespace Orders.Frontend.Pages

{

public partial class Home

{

private int currentPage = 1;

private int totalPages;

public List<Product>? Products { get; set; }

[Parameter, SupplyParameterFromQuery] public string Page { get; set; } = string.Empty;

[Parameter, SupplyParameterFromQuery] public string Filter { get; set; } = string.Empty;

[Inject] private NavigationManager NavigationManager { get; set; } = null!;

[Inject] private SweetAlertService SweetAlertService { get; set; } = null!;

[Inject] private IRepository Repository { get; set; } = null!;

[Parameter, SupplyParameterFromQuery] public int RecordsNumber { get; set; } = 8;

protected override async Task OnInitializedAsync()

{

await LoadAsync();

}

private async Task SelectedRecordsNumberAsync(int recordsnumber)

{

RecordsNumber = recordsnumber;

int page = 1;

await LoadAsync(page);

await SelectedPageAsync(page);

}

private async Task FilterCallBack(string filter)

{

Filter = filter;

await ApplyFilterAsync();

StateHasChanged();

}

private async Task SelectedPageAsync(int page)

{

currentPage = page;

await LoadAsync(page);

}

private async Task LoadAsync(int page = 1)

{

if (!string.IsNullOrWhiteSpace(Page))

{

page = Convert.ToInt32(Page);

}

var ok = await LoadListAsync(page);

if (ok)

{

await LoadPagesAsync();

}

}

private void ValidateRecordsNumber(int recordsnumber)

{

if (recordsnumber == 0)

{

RecordsNumber = 8;

}

}

private async Task<bool> LoadListAsync(int page)

{

ValidateRecordsNumber(RecordsNumber);

var url = $"api/products?page={page}&RecordsNumber={RecordsNumber}";

if (!string.IsNullOrEmpty(Filter))

{

url += $"&filter={Filter}";

}

var response = await Repository.GetAsync<List<Product>>(url);

if (response.Error)

{

var message = await response.GetErrorMessageAsync();

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

return false;

}

Products = response.Response;

return true;

}

private async Task LoadPagesAsync()

{

ValidateRecordsNumber(RecordsNumber);

var url = $"api/products/totalPages/?RecordsNumber={RecordsNumber}";

if (!string.IsNullOrEmpty(Filter))

{

url += $"&filter={Filter}";

}

var response = await Repository.GetAsync<int>(url);

if (response.Error)

{

var message = await response.GetErrorMessageAsync();

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

return;

}

totalPages = response.Response;

}

private async Task ApplyFilterAsync()

{

int page = 1;

await LoadAsync(page);

await SelectedPageAsync(page);

}

private void AddToCartAsync(int productId)

{

}

}

}

1. Modificamos el **Home.razor**:

@page "/"

@if (Products is null)

{

<Loading />

}

else

{

<Filter PlaceHolder="Buscar producto..." Callback=@FilterCallBack />

<Pagination CurrentPage="currentPage"

TotalPages="totalPages"

SelectedPage="SelectedPageAsync"

RecordsNumber="SelectedRecordsNumberAsync" />

<div class="row row-cols-1 row-cols-md-4 g-4 mt-1">

@foreach (var product in Products!)

{

<div class="col">

<div class="card h-100">

<div class="text-center zoom">

<img src="@product.MainImage" style="height:150px; max-width:200px;" class="text-center" alt=@product.Name />

</div>

<div class="card-body">

<h5 class="card-title text-navy"> @product.Name</h5>

<p class="card-text smfnt">@product.Description</p>

<h5 class="text-muted">@($"{product.Price:C2}")</h5>

</div>

<div class="card-footer text-center">

<a href="/products/details/@product.Id" class="btn btn-sm btn-secondary"><i class="bi bi-info-circle" /> Detalles</a>

<button class="btn btn-sm btn-primary" @onclick=@(() => AddToCartAsync(product.Id))><i class="bi bi-cart-plus" /> Agregar al Carro</button>

</div>

</div>

</div>

}

</div>

}

1. Probamos, pero para el **Home** la paginación de 10, 25 y 50 no se ve del todo bien. Vamos a modificar el paginador parta que ofrezca una paginación diferente en esta página. Modificamos el **Pagination.razor.cs**:

…

[Parameter] public bool IsHome { get; set; } = false;

…

private void BuildOptions()

{

if (IsHome)

{

options =

[

new OptionModel { Value = 8, Name = "8" },

new OptionModel { Value = 16, Name = "16" },

new OptionModel { Value = 32, Name = "32" },

new OptionModel { Value = int.MaxValue, Name = "Todos" },

];

}

else

{

options =

[

new OptionModel { Value = 10, Name = "10" },

new OptionModel { Value = 25, Name = "25" },

new OptionModel { Value = 50, Name = "50" },

new OptionModel { Value = int.MaxValue, Name = "Todos" },

];

}

}

…

1. Modificamos el **Home.razor**:

…

<Pagination CurrentPage="currentPage"

TotalPages="totalPages"

SelectedPage="SelectedPageAsync"

RecordsNumber="SelectedRecordsNumberAsync"

IsHome />

…

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

## Agregando productos al carro de compras

1. Creamos la entidad **TemporalOrder**:

using System.ComponentModel.DataAnnotations;

namespace Orders.Shared.Entities

{

public class TemporalOrder

{

public int Id { get; set; }

public User? User { get; set; }

public string? UserId { get; set; }

public Product? Product { get; set; }

public int ProductId { get; set; }

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

[Display(Name = "Cantidad")]

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

public float Quantity { get; set; }

[DataType(DataType.MultilineText)]

[Display(Name = "Comentarios")]

public string? Remarks { get; set; }

public decimal Value => Product == null ? 0 : Product.Price \* (decimal)Quantity;

}

}

1. Modificmos la entidad **Product** agregando esta propiedad:

public ICollection<TemporalOrder>? TemporalOrders { get; set; }

1. Modificmos la entidad **User** agregando esta propiedad:

public ICollection<TemporalOrder>? TemporalOrders { get; set; }

1. La adicionamos en el **DataContext**:

public DbSet<TemporalOrder> TemporalOrders { get; set; }

1. Creamos la migración y actualizamos la base de datos.
2. En **Orders.Shared.DTOs** creamos el **TemporalOrderDTO**.

namespace Orders.Shared.DTOs

{

public class TemporalOrderDTO

{

public int ProductId { get; set; }

public float Quantity { get; set; } = 1;

public string Remarks { get; set; } = string.Empty;

}

}

1. Creamos el **ITemporalOrdersRepository**:

using Orders.Shared.DTOs;

using Orders.Shared.Entities;

using Orders.Shared.Responses;

namespace Orders.Backend.Repositories.Interfaces

{

public interface ITemporalOrdersRepository

{

Task<ActionResponse<TemporalOrderDTO>> AddFullAsync(string email, TemporalOrderDTO temporalOrderDTO);

Task<ActionResponse<IEnumerable<TemporalOrder>>> GetAsync(string email);

Task<ActionResponse<int>> GetCountAsync(string email);

}

}

1. Creamos el **TemporalOrdersRepository**:

using Microsoft.EntityFrameworkCore;

using Orders.Backend.Data;

using Orders.Backend.Repositories.Interfaces;

using Orders.Shared.DTOs;

using Orders.Shared.Entities;

using Orders.Shared.Responses;

namespace Orders.Backend.Repositories.Implementations

{

public class TemporalOrdersRepository : GenericRepository<TemporalOrder>, ITemporalOrdersRepository

{

private readonly DataContext \_context;

private readonly IUsersRepository \_usersRepository;

public TemporalOrdersRepository(DataContext context, IUsersRepository usersRepository) : base(context)

{

\_context = context;

\_usersRepository = usersRepository;

}

public async Task<ActionResponse<TemporalOrderDTO>> AddFullAsync(string email, TemporalOrderDTO temporalOrderDTO)

{

var product = await \_context.Products.FirstOrDefaultAsync(x => x.Id == temporalOrderDTO.ProductId);

if (product == null)

{

return new ActionResponse<TemporalOrderDTO>

{

WasSuccess = false,

Message = "Producto no existe"

};

}

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

if (user == null)

{

return new ActionResponse<TemporalOrderDTO>

{

WasSuccess = false,

Message = "Usuario no existe"

};

}

var temporalOrder = new TemporalOrder

{

Product = product,

Quantity = temporalOrderDTO.Quantity,

Remarks = temporalOrderDTO.Remarks,

User = user

};

try

{

\_context.Add(temporalOrder);

await \_context.SaveChangesAsync();

return new ActionResponse<TemporalOrderDTO>

{

WasSuccess = true,

Result = temporalOrderDTO

};

}

catch (Exception ex)

{

return new ActionResponse<TemporalOrderDTO>

{

WasSuccess = false,

Message = ex.Message

};

}

}

public async Task<ActionResponse<IEnumerable<TemporalOrder>>> GetAsync(string email)

{

var temporalOrders = await \_context.TemporalOrders

.Include(ts => ts.User!)

.Include(ts => ts.Product!)

.ThenInclude(p => p.ProductCategories!)

.ThenInclude(pc => pc.Category)

.Include(ts => ts.Product!)

.ThenInclude(p => p.ProductImages)

.Where(x => x.User!.Email == email)

.ToListAsync();

return new ActionResponse<IEnumerable<TemporalOrder>>

{

WasSuccess = true,

Result = temporalOrders

};

}

public async Task<ActionResponse<int>> GetCountAsync(string email)

{

var count = await \_context.TemporalOrders

.Where(x => x.User!.Email == email)

.SumAsync(x => x.Quantity);

return new ActionResponse<int>

{

WasSuccess = true,

Result = (int)count

};

}

}

}

1. Creamos el **ITemporalOrdersUnitOfWork**:

using Orders.Shared.DTOs;

using Orders.Shared.Entities;

using Orders.Shared.Responses;

namespace Orders.Backend.UnitsOfWork.Interfaces

{

public interface ITemporalOrdersUnitOfWork

{

Task<ActionResponse<TemporalOrderDTO>> AddFullAsync(string email, TemporalOrderDTO temporalOrderDTO);

Task<ActionResponse<IEnumerable<TemporalOrder>>> GetAsync(string email);

Task<ActionResponse<int>> GetCountAsync(string email);

}

}

1. Creamos el **TemporalOrdersUnitOfWork**:

using Orders.Backend.Repositories.Interfaces;

using Orders.Backend.UnitsOfWork.Implementations;

using Orders.Backend.UnitsOfWork.Interfaces;

using Orders.Shared.DTOs;

using Orders.Shared.Entities;

using Orders.Shared.Responses;

namespace Orders.Backend.Repositories.Implementations

{

public class TemporalOrdersUnitOfWork : GenericUnitOfWork<TemporalOrder>, ITemporalOrdersUnitOfWork

{

private readonly ITemporalOrdersRepository \_temporalOrdersRepository;

public TemporalOrdersUnitOfWork(IGenericRepository<TemporalOrder> repository, ITemporalOrdersRepository temporalOrdersRepository) : base(repository)

{

\_temporalOrdersRepository = temporalOrdersRepository;

}

public async Task<ActionResponse<TemporalOrderDTO>> AddFullAsync(string email, TemporalOrderDTO temporalOrderDTO) => await \_temporalOrdersRepository.AddFullAsync(email, temporalOrderDTO);

public async Task<ActionResponse<IEnumerable<TemporalOrder>>> GetAsync(string email) => await \_temporalOrdersRepository.GetAsync(email);

public async Task<ActionResponse<int>> GetCountAsync(string email) => await \_temporalOrdersRepository.GetCountAsync(email);

}

}

1. Agregamos las nueva inyecciones el **Program**:

builder.Services.AddScoped<ICategoriesRepository, CategoriesRepository>();

builder.Services.AddScoped<ICitiesRepository, CitiesRepository>();

builder.Services.AddScoped<ICountriesRepository, CountriesRepository>();

builder.Services.AddScoped<IProductsRepository, ProductsRepository>();

builder.Services.AddScoped<IStatesRepository, StatesRepository>();

builder.Services.AddScoped<ITemporalOrdersRepository, TemporalOrdersRepository>();

builder.Services.AddScoped<IUsersRepository, UsersRepository>();

builder.Services.AddScoped<ICategoriesUnitOfWork, CategoriesUnitOfWork>();

builder.Services.AddScoped<ICitiesUnitOfWork, CitiesUnitOfWork>();

builder.Services.AddScoped<ICountriesUnitOfWork, CountriesUnitOfWork>();

builder.Services.AddScoped<IProductsUnitOfWork, ProductsUnitOfWork>();

builder.Services.AddScoped<IStatesUnitOfWork, StatesUnitOfWork>();

builder.Services.AddScoped<ITemporalOrdersUnitOfWork, TemporalOrdersUnitOfWork>();

builder.Services.AddScoped<IUsersUnitOfWork, UsersUnitOfWork>();

1. Creamos el **TemporalOrdersController**:

using Microsoft.AspNetCore.Authentication.JwtBearer;

using Microsoft.AspNetCore.Authorization;

using Microsoft.AspNetCore.Mvc;

using Orders.Backend.UnitsOfWork.Interfaces;

using Orders.Shared.DTOs;

using Orders.Shared.Entities;

namespace Orders.Backend.Controllers

{

[ApiController]

[Authorize(AuthenticationSchemes = JwtBearerDefaults.AuthenticationScheme)]

[Route("api/[controller]")]

public class TemporalOrdersController : GenericController<TemporalOrder>

{

private readonly ITemporalOrdersUnitOfWork \_temporalOrdersUnitOfWork;

public TemporalOrdersController(IGenericUnitOfWork<TemporalOrder> unitOfWork, ITemporalOrdersUnitOfWork temporalOrdersUnitOfWork) : base(unitOfWork)

{

\_temporalOrdersUnitOfWork = temporalOrdersUnitOfWork;

}

[HttpPost("full")]

public async Task<IActionResult> PostAsync(TemporalOrderDTO temporalOrderDTO)

{

var action = await \_temporalOrdersUnitOfWork.AddFullAsync(User.Identity!.Name!, temporalOrderDTO);

if (action.WasSuccess)

{

return Ok(action.Result);

}

return BadRequest(action.Message);

}

[HttpGet("my")]

public override async Task<IActionResult> GetAsync()

{

var action = await \_temporalOrdersUnitOfWork.GetAsync(User.Identity!.Name!);

if (action.WasSuccess)

{

return Ok(action.Result);

}

return BadRequest(action.Message);

}

[HttpGet("count")]

public async Task<IActionResult> GetCountAsync()

{

var action = await \_temporalOrdersUnitOfWork.GetCountAsync(User.Identity!.Name!);

if (action.WasSuccess)

{

return Ok(action.Result);

}

return BadRequest(action.Message);

}

}

}

1. Modificamos el **Home.razor.cs**:

…

private int currentPage = 1;

private int totalPages;

private int counter = 0;

private bool isAuthenticated;

…

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

[CascadingParameter] private IModalService Modal { get; set; } = default!;

…

protected async override Task OnParametersSetAsync()

{

await CheckIsAuthenticatedAsync();

await LoadCounterAsync();

}

private async Task CheckIsAuthenticatedAsync()

{

var authenticationState = await authenticationStateTask;

isAuthenticated = authenticationState.User.Identity!.IsAuthenticated;

}

private async Task LoadCounterAsync()

{

if (!isAuthenticated)

{

return;

}

var responseHttp = await Repository.GetAsync<int>("/api/temporalOrders/count");

if (responseHttp.Error)

{

return;

}

counter = responseHttp.Response;

}

private async Task AddToCartAsync(int productId)

{

if (!isAuthenticated)

{

Modal.Show<Login>();

var toast1 = SweetAlertService.Mixin(new SweetAlertOptions

{

Toast = true,

Position = SweetAlertPosition.BottomEnd,

ShowConfirmButton = false,

Timer = 3000

});

await toast1.FireAsync(icon: SweetAlertIcon.Error, message: "Debes haber iniciado sesión para poder agregar productos al carro de compras.");

return;

}

var temporalOrderDTO = new TemporalOrderDTO

{

ProductId = productId

};

var httpActionResponse = await Repository.PostAsync("/api/temporalOrders/full", temporalOrderDTO);

if (httpActionResponse.Error)

{

var message = await httpActionResponse.GetErrorMessageAsync();

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

return;

}

await LoadCounterAsync();

var toast2 = SweetAlertService.Mixin(new SweetAlertOptions

{

Toast = true,

Position = SweetAlertPosition.BottomEnd,

ShowConfirmButton = true,

Timer = 3000

});

await toast2.FireAsync(icon: SweetAlertIcon.Success, message: "Producto agregado al carro de compras.");

}

…

1. Modificamos el **Home.razor**:

…

<div class="d-flex align-items-center justify-content-between">

<Filter PlaceHolder="Buscar producto..." Callback=@FilterCallBack />

<AuthorizeView>

<Authorized>

@if (counter > 0)

{

<a href="/Cart/ShowCart" class="btn btn-primary"><i class="bi bi-cart-fill" /> Ver Carro de Compras (@counter)</a>

}

</Authorized>

</AuthorizeView>

</div>

…

1. Dentro de **Pages** creamos la carpeta **Cart** y dentro de esta creamos el **ShowCart.razor** y **ShowCart.razor.cs** temporal.

@page "/Cart/ShowCart"

<h3>ShowCart</h3>

1. Probamos lo que llevamos hasta el momento.
2. Ahora vamos a mostrar los detalles del producto y dar la oportunidad de agregar al carro de compras ingresando una cantidad y un comentario. Primero creamos el **ProductDetails.razor** y **ProductDetails.razor.cs** dentro de **Pages/Products**:

@page "/products/details/{ProductId:int}"

@if (loading)

{

<Loading />

}

else

{

<div class="card">

<div class="card-header">

<span>

<i class="bi bi-star" /> @product!.Name

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

</span>

</div>

<div class="card-body">

<div class="row">

<div class="col-6">

<div class="mb-3">

<label>Nombre:</label>

<div>

<b>@product.Name</b>

</div>

</div>

<div class="mb-3">

<label>Descripción:</label>

<div>

<b>@product.Description</b>

</div>

</div>

<div class="mb-3">

<label>Precio:</label>

<div>

<b>@($"{product.Price:C2}")</b>

</div>

</div>

<div class="mb-3">

<label>Inventario:</label>

<div>

<b>@($"{product.Stock:N2}")</b>

</div>

</div>

<div class="mb-3">

<label>Categorías:</label>

<div>

@foreach (var category in categories!)

{

<div class="mx-2">

<b>@category</b>

</div>

}

</div>

</div>

</div>

<div class="col-6">

<EditForm Model="TemporalOrderDTO" OnValidSubmit="AddToCartAsync">

<DataAnnotationsValidator />

<div class="mb-3">

<label>Cantidad:</label>

<div>

<InputNumber class="form-control" @bind-Value="@TemporalOrderDTO.Quantity" />

<ValidationMessage For="@(() => TemporalOrderDTO.Quantity)" />

</div>

<label>Comentarios:</label>

<div>

<InputText class="form-control" @bind-Value="@TemporalOrderDTO.Remarks" />

<ValidationMessage For="@(() => TemporalOrderDTO.Remarks)" />

</div>

</div>

<button class="btn btn-primary" type="submit"><i class="bi bi-cart-plus" /> Agregar Al Carro de Compras</button>

</EditForm>

</div>

</div>

<CarouselView Images="images" />

</div>

</div>

}

1. Ahora creamos el **ProductDetails.razor.cs**:

using CurrieTechnologies.Razor.SweetAlert2;

using Microsoft.AspNetCore.Components;

using Microsoft.AspNetCore.Components.Authorization;

using Orders.Frontend.Repositories;

using Orders.Shared.DTOs;

using Orders.Shared.Entities;

namespace Orders.Frontend.Pages.Products

{

public partial class ProductDetails

{

private List<string>? categories;

private List<string>? images;

private bool loading = true;

private Product? product;

private bool isAuthenticated;

[Inject] private NavigationManager navigationManager { get; set; } = null!;

[Inject] private IRepository repository { get; set; } = null!;

[Inject] private SweetAlertService sweetAlertService { get; set; } = null!;

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

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

public TemporalOrderDTO TemporalOrderDTO { get; set; } = new();

protected override async Task OnParametersSetAsync()

{

await CheckIsAuthenticatedAsync();

}

private async Task CheckIsAuthenticatedAsync()

{

var authenticationState = await authenticationStateTask;

isAuthenticated = authenticationState.User.Identity!.IsAuthenticated;

}

protected override async Task OnInitializedAsync()

{

await LoadProductAsync();

}

private async Task LoadProductAsync()

{

loading = true;

var httpActionResponse = await repository.GetAsync<Product>($"/api/products/{ProductId}");

if (httpActionResponse.Error)

{

loading = false;

var message = await httpActionResponse.GetErrorMessageAsync();

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

return;

}

product = httpActionResponse.Response!;

categories = product.ProductCategories!.Select(x => x.Category!.Name).ToList();

images = product.ProductImages!.Select(x => x.Image).ToList();

loading = false;

}

public async Task AddToCartAsync()

{

if (!isAuthenticated)

{

navigationManager.NavigateTo("/Login");

var toast1 = sweetAlertService.Mixin(new SweetAlertOptions

{

Toast = true,

Position = SweetAlertPosition.BottomEnd,

ShowConfirmButton = true,

Timer = 3000

});

await toast1.FireAsync(icon: SweetAlertIcon.Error, message: "Debes haber iniciado sesión para poder agregar productos al carro de compras.");

return;

}

TemporalOrderDTO.ProductId = ProductId;

var httpActionResponse = await repository.PostAsync("/api/temporalOrders/full", TemporalOrderDTO);

if (httpActionResponse.Error)

{

var message = await httpActionResponse.GetErrorMessageAsync();

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

return;

}

var toast2 = sweetAlertService.Mixin(new SweetAlertOptions

{

Toast = true,

Position = SweetAlertPosition.BottomEnd,

ShowConfirmButton = true,

Timer = 3000

});

await toast2.FireAsync(icon: SweetAlertIcon.Success, message: "Producto agregado al carro de compras.");

navigationManager.NavigateTo("/");

}

}

}

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

## Mostrando y modificando el carro de compras

1. Agregamos este campo al **TemporalOrderDTO**:

public int Id { get; set; }

1. Agregamos la enumeración **OrderStatus**:

using System.ComponentModel;

namespace Orders.Shared.Enums

{

public enum OrderStatus

{

[Description("Nuevo")]

New,

[Description("Despachado")]

Dispatched,

[Description("Enviado")]

Sent,

[Description("Confirmado")]

Confirmed,

[Description("Cancelado")]

Cancelled

}

}

1. Agregamos el **OrderDTO**:

using Orders.Shared.Enums;

namespace Orders.Shared.DTOs

{

public class OrderDTO

{

public int Id { get; set; }

public OrderStatus OrderStatus { get; set; }

public string Remarks { get; set; } = string.Empty;

}

}

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

Task<ActionResponse<TemporalOrder>> GetAsync(int id);

Task<ActionResponse<TemporalOrder>> PutFullAsync(TemporalOrderDTO temporalOrderDTO);

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

public async Task<ActionResponse<TemporalOrder>> PutFullAsync(TemporalOrderDTO temporalOrderDTO)

{

var currentTemporalOrder = await \_context.TemporalOrders.FirstOrDefaultAsync(x => x.Id == temporalOrderDTO.Id);

if (currentTemporalOrder == null)

{

return new ActionResponse<TemporalOrder>

{

WasSuccess = false,

Message = "Registro no encontrado"

};

}

currentTemporalOrder!.Remarks = temporalOrderDTO.Remarks;

currentTemporalOrder.Quantity = temporalOrderDTO.Quantity;

\_context.Update(currentTemporalOrder);

await \_context.SaveChangesAsync();

return new ActionResponse<TemporalOrder>

{

WasSuccess = true,

Result = currentTemporalOrder

};

}

public override async Task<ActionResponse<TemporalOrder>> GetAsync(int id)

{

var temporalOrder = await \_context.TemporalOrders

.Include(ts => ts.User!)

.Include(ts => ts.Product!)

.ThenInclude(p => p.ProductCategories!)

.ThenInclude(pc => pc.Category)

.Include(ts => ts.Product!)

.ThenInclude(p => p.ProductImages)

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

if (temporalOrder == null)

{

return new ActionResponse<TemporalOrder>

{

WasSuccess = false,

Message = "Registro no encontrado"

};

}

return new ActionResponse<TemporalOrder>

{

WasSuccess = true,

Result = temporalOrder

};

}

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

Task<ActionResponse<TemporalOrder>> GetAsync(int id);

Task<ActionResponse<TemporalOrder>> PutFullAsync(TemporalOrderDTO temporalOrderDTO);

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

public async Task<ActionResponse<TemporalOrder>> PutFullAsync(TemporalOrderDTO temporalOrderDTO) => await \_temporalOrdersRepository.PutFullAsync(temporalOrderDTO);

public override async Task<ActionResponse<TemporalOrder>> GetAsync(int id) => await \_temporalOrdersRepository.GetAsync(id);

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

[HttpGet("{id}")]

public override async Task<IActionResult> GetAsync(int id)

{

var response = await \_temporalOrdersUnitOfWork.GetAsync(id);

if (response.WasSuccess)

{

return Ok(response.Result);

}

return NotFound(response.Message);

}

[HttpPut("full")]

public async Task<IActionResult> PutFullAsync(TemporalOrderDTO temporalOrderDTO)

{

var action = await \_temporalOrdersUnitOfWork.PutFullAsync(temporalOrderDTO);

if (action.WasSuccess)

{

return Ok(action.Result);

}

return NotFound(action.Message);

}

1. Agregamos el **ShowCart.razor.cs**:

using CurrieTechnologies.Razor.SweetAlert2;

using Microsoft.AspNetCore.Authorization;

using Microsoft.AspNetCore.Components;

using Orders.Frontend.Repositories;

using Orders.Shared.DTOs;

using Orders.Shared.Entities;

namespace Orders.Frontend.Pages.Cart

{

[Authorize(Roles = "Admin, User")]

public partial class ShowCart

{

public List<TemporalOrder>? temporalOrders { get; set; }

private float sumQuantity;

private decimal sumValue;

[Inject] private NavigationManager NavigationManager { get; set; } = null!;

[Inject] private IRepository Repository { get; set; } = null!;

[Inject] private SweetAlertService SweetAlertService { get; set; } = null!;

public OrderDTO OrderDTO { get; set; } = new();

protected override async Task OnInitializedAsync()

{

await LoadAsync();

}

private async Task LoadAsync()

{

try

{

var responseHppt = await Repository.GetAsync<List<TemporalOrder>>("api/temporalOrders/my");

temporalOrders = responseHppt.Response!;

sumQuantity = temporalOrders.Sum(x => x.Quantity);

sumValue = temporalOrders.Sum(x => x.Value);

}

catch (Exception ex)

{

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

}

}

private void ConfirmOrderAsync()

{

//TODO: Pending to implement

}

private async Task Delete(int temporalOrderId)

{

var result = await SweetAlertService.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.DeleteAsync<TemporalOrder>($"api/temporalOrders/{temporalOrderId}");

if (responseHttp.Error)

{

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

{

NavigationManager.NavigateTo("/");

return;

}

var mensajeError = await responseHttp.GetErrorMessageAsync();

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

return;

}

await LoadAsync();

var toast = SweetAlertService.Mixin(new SweetAlertOptions

{

Toast = true,

Position = SweetAlertPosition.BottomEnd,

ShowConfirmButton = false,

Timer = 3000

});

await toast.FireAsync(icon: SweetAlertIcon.Success, message: "Producto eliminado del carro de compras.");

}

}

}

1. Modificamos nuestro **ShowCart.razor**:

@page "/Cart/ShowCart"

@if (temporalOrders is null)

{

<Loading />

}

else

{

<GenericList MyList="temporalOrders">

<Body>

<div class="card">

<div class="card-header">

<span>

<i class="bi bi-cart" /> Carro de Compras

</span>

</div>

<div class="card-body">

<div class="row mb-2">

<div class="col-4">

<h3>Cantidad productos: <strong>@($"{sumQuantity:N2}")</strong></h3>

</div>

<div class="col-4">

<h3>Valor: <strong>@($"{sumValue:C2}")</strong></h3>

</div>

</div>

<EditForm Model="OrderDTO" OnValidSubmit="ConfirmOrderAsync">

<DataAnnotationsValidator />

<div class="mb-3">

<label>Comentarios:</label>

<div>

<InputText class="form-control" @bind-Value="@OrderDTO.Remarks" />

<ValidationMessage For="@(() => OrderDTO.Remarks)" />

</div>

</div>

<button class="btn btn-primary mb-3" type="submit"><i class="bi bi-check" /> Confirmar Pedido</button>

</EditForm>

<table class="table table-striped">

<thead>

<tr>

<th>Nombre</th>

<th>Descripción</th>

<th>Cantidad</th>

<th>Precio</th>

<th>Valor</th>

<th>Comentarios</th>

<th>Imagén</th>

<th style="width:168px"></th>

</tr>

</thead>

<tbody>

@foreach (var temporalOrder in temporalOrders)

{

<tr>

<td>

@temporalOrder.Product!.Name

</td>

<td>

@temporalOrder.Product!.Description

</td>

<td>

@($"{temporalOrder.Quantity:N2}")

</td>

<td>

@($"{temporalOrder.Product!.Price:C2}")

</td>

<td>

@($"{temporalOrder.Value:C2}")

</td>

<td>

@temporalOrder.Remarks

</td>

<td>

<img src="@temporalOrder.Product!.MainImage" style="width:100px;" />

</td>

<td>

<a href="/Cart/ModifyTemporalOrder/@temporalOrder.Id" class="btn btn-warning btn-sm"><i class="bi bi-pencil" /> Editar</a>

<button class="btn btn-danger btn-sm" @onclick=@(() => Delete(temporalOrder.Id))><i class="bi bi-trash" /> Borrar</button>

</td>

</tr>

}

</tbody>

</table>

</div>

</div>

</Body>

</GenericList>

}

1. Probamos lo que llevamos hasta el momento.
2. Dentro de **Pages/Cart** creamos el **ModifyTemporalOrder.razor** y **ModifyTemporalOrder.razor.cs**:

using CurrieTechnologies.Razor.SweetAlert2;

using Microsoft.AspNetCore.Authorization;

using Microsoft.AspNetCore.Components;

using Orders.Frontend.Repositories;

using Orders.Shared.DTOs;

using Orders.Shared.Entities;

namespace Orders.Frontend.Pages.Cart

{

[Authorize(Roles = "Admin, User")]

public partial class ModifyTemporalOrder

{

private List<string>? categories;

private List<string>? images;

private bool loading = true;

private Product? product;

private TemporalOrderDTO? temporalOrderDTO;

[Inject] private NavigationManager NavigationManager { get; set; } = null!;

[Inject] private IRepository Repository { get; set; } = null!;

[Inject] private SweetAlertService SweetAlertService { get; set; } = null!;

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

protected override async Task OnInitializedAsync()

{

await LoadTemporalOrderAsync();

}

private async Task LoadTemporalOrderAsync()

{

loading = true;

var httpResponse = await Repository.GetAsync<TemporalOrder>($"/api/temporalOrders/{TemporalOrderId}");

if (httpResponse.Error)

{

loading = false;

var message = await httpResponse.GetErrorMessageAsync();

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

return;

}

var temporalOrder = httpResponse.Response!;

temporalOrderDTO = new TemporalOrderDTO

{

Id = temporalOrder.Id,

ProductId = temporalOrder.ProductId,

Remarks = temporalOrder.Remarks!,

Quantity = temporalOrder.Quantity

};

product = temporalOrder.Product;

categories = product!.ProductCategories!.Select(x => x.Category.Name).ToList();

images = product.ProductImages!.Select(x => x.Image).ToList();

loading = false;

}

public async Task UpdateCartAsync()

{

var httpResponse = await Repository.PutAsync("/api/temporalOrders/full", temporalOrderDTO);

if (httpResponse.Error)

{

var message = await httpResponse.GetErrorMessageAsync();

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

return;

}

var toast2 = SweetAlertService.Mixin(new SweetAlertOptions

{

Toast = true,

Position = SweetAlertPosition.BottomEnd,

ShowConfirmButton = true,

Timer = 3000

});

await toast2.FireAsync(icon: SweetAlertIcon.Success, message: "Producto modificado en el de compras.");

NavigationManager.NavigateTo("/");

}

}

}

1. Modificamos el **ModifyTemporalOrder.razor**:

@page "/Cart/ModifyTemporalOrder/{TemporalOrderId:int}"

@if (loading)

{

<Loading />

}

else

{

<div class="card">

<div class="card-header">

<span>

<i class="bi bi-star" /> @product!.Name

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

</span>

</div>

<div class="card-body">

<div class="row">

<div class="col-6">

<div class="mb-3">

<label>Nombre:</label>

<div>

<b>@product.Name</b>

</div>

</div>

<div class="mb-3">

<label>Descripción:</label>

<div>

<b>@product.Description</b>

</div>

</div>

<div class="mb-3">

<label>Precio:</label>

<div>

<b>@($"{product.Price:C2}")</b>

</div>

</div>

<div class="mb-3">

<label>Inventario:</label>

<div>

<b>@($"{product.Stock:N2}")</b>

</div>

</div>

<div class="mb-3">

<label>Categorías:</label>

<div>

@foreach (var category in categories!)

{

<div class="mx-2">

<b>@category</b>

</div>

}

</div>

</div>

</div>

<div class="col-6">

<EditForm Model="temporalOrderDTO" OnValidSubmit="UpdateCartAsync">

<DataAnnotationsValidator />

<div class="mb-3">

<label>Cantidad:</label>

<div>

<InputNumber class="form-control" @bind-Value="@temporalOrderDTO!.Quantity" />

<ValidationMessage For="@(() => temporalOrderDTO.Quantity)" />

</div>

<label>Comentarios:</label>

<div>

<InputText class="form-control" @bind-Value="@temporalOrderDTO.Remarks" />

<ValidationMessage For="@(() => temporalOrderDTO.Remarks)" />

</div>

</div>

<button class="btn btn-primary" type="submit"><i class="bi bi-check" /> Actualizar Carro de Compras</button>

</EditForm>

</div>

</div>

<CarouselView Images="images" />

</div>

</div>

}

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

## Procesando el pedido

1. Agregamos la entidad **Order**:

using Orders.Shared.Enums;

using System.ComponentModel.DataAnnotations;

namespace Orders.Shared.Entities

{

public class Order

{

public int Id { get; set; }

[DisplayFormat(DataFormatString = "{0:yyyy/MM/dd hh:mm tt}")]

[Display(Name = "Inventario")]

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

public DateTime Date { get; set; }

public User? User { get; set; }

public string? UserId { get; set; }

[DataType(DataType.MultilineText)]

[Display(Name = "Comentarios")]

public string? Remarks { get; set; }

public OrderStatus OrderStatus { get; set; }

public ICollection<OrderDetail>? OrderDetails { get; set; }

[DisplayFormat(DataFormatString = "{0:N0}")]

[Display(Name = "Líneas")]

public int Lines => OrderDetails == null || OrderDetails.Count == 0 ? 0 : OrderDetails.Count;

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

[Display(Name = "Cantidad")]

public float Quantity => OrderDetails == null || OrderDetails.Count == 0 ? 0 : OrderDetails.Sum(sd => sd.Quantity);

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

[Display(Name = "Valor")]

public decimal Value => OrderDetails == null || OrderDetails.Count == 0 ? 0 : OrderDetails.Sum(sd => sd.Value);

}

}

1. Agregamos la entidad **OrderDetail**:

using System.ComponentModel.DataAnnotations;

namespace Orders.Shared.Entities

{

public class OrderDetail

{

public int Id { get; set; }

public Order? Order { get; set; }

public int OrderId { get; set; }

[DataType(DataType.MultilineText)]

[Display(Name = "Comentarios")]

public string? Remarks { get; set; }

public Product? Product { get; set; }

public int ProductId { get; set; }

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

[Display(Name = "Cantidad")]

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

public float Quantity { get; set; }

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

[Display(Name = "Valor")]

public decimal Value => Product == null ? 0 : (decimal)Quantity \* Product.Price;

}

}

1. Modificamos la entidad **Product**:

public ICollection<OrderDetail>? OrderDetails { get; set; }

1. Modificamos la entidad **User**:

public ICollection<Order>? Orders { get; set; }

1. Agregamos las nuevas entidades al **DataContext**:

public DbSet<Order> Orders { get; set; }

public DbSet<OrderDetail> OrderDetails { get; set; }

1. Agregamos la migración y actualizamos la base de datos.
2. Creamos el **IOrdersRepository**:

using Orders.Shared.DTOs;

using Orders.Shared.Entities;

using Orders.Shared.Responses;

namespace Orders.Backend.Repositories.Interfaces

{

public interface IOrdersRepository

{

Task<ActionResponse<IEnumerable<Order>>> GetAsync(string email, PaginationDTO pagination);

Task<ActionResponse<int>> GetTotalPagesAsync(string email, PaginationDTO pagination);

Task<ActionResponse<Order>> GetAsync(int id);

Task<ActionResponse<Order>> UpdateFullAsync(string email, OrderDTO orderDTO);

}

}

1. Creamos el **OrdersRepository**:

using Microsoft.EntityFrameworkCore;

using Orders.Backend.Data;

using Orders.Backend.Helpers;

using Orders.Backend.Repositories.Interfaces;

using Orders.Shared.DTOs;

using Orders.Shared.Entities;

using Orders.Shared.Enums;

using Orders.Shared.Responses;

namespace Orders.Backend.Repositories.Implementations

{

public class OrdersRepository : GenericRepository<Order>, IOrdersRepository

{

private readonly DataContext \_context;

private readonly IUsersRepository \_usersRepository;

public OrdersRepository(DataContext context, IUsersRepository usersRepository) : base(context)

{

\_context = context;

\_usersRepository = usersRepository;

}

public async Task<ActionResponse<IEnumerable<Order>>> GetAsync(string email, PaginationDTO pagination)

{

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

if (user == null)

{

return new ActionResponse<IEnumerable<Order>>

{

WasSuccess = false,

Message = "Usuario no válido",

};

}

var queryable = \_context.Orders

.Include(s => s.User!)

.Include(s => s.OrderDetails!)

.ThenInclude(sd => sd.Product)

.AsQueryable();

var isAdmin = await \_usersRepository.IsUserInRoleAsync(user, UserType.Admin.ToString());

if (!isAdmin)

{

queryable = queryable.Where(s => s.User!.Email == email);

}

return new ActionResponse<IEnumerable<Order>>

{

WasSuccess = true,

Result = await queryable

.OrderByDescending(x => x.Date)

.Paginate(pagination)

.ToListAsync()

};

}

public async Task<ActionResponse<int>> GetTotalPagesAsync(string email, PaginationDTO pagination)

{

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

if (user == null)

{

return new ActionResponse<int>

{

WasSuccess = false,

Message = "Usuario no válido",

};

}

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

var isAdmin = await \_usersRepository.IsUserInRoleAsync(user, UserType.Admin.ToString());

if (!isAdmin)

{

queryable = queryable.Where(s => s.User!.Email == email);

}

double count = await queryable.CountAsync();

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

return new ActionResponse<int>

{

WasSuccess = true,

Result = (int)totalPages

};

}

public override async Task<ActionResponse<Order>> GetAsync(int id)

{

var order = await \_context.Orders

.Include(s => s.User!)

.ThenInclude(u => u.City!)

.ThenInclude(c => c.State!)

.ThenInclude(s => s.Country)

.Include(s => s.OrderDetails!)

.ThenInclude(sd => sd.Product)

.ThenInclude(p => p.ProductImages)

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

if (order == null)

{

return new ActionResponse<Order>

{

WasSuccess = false,

Message = "Pedido no existe"

};

}

return new ActionResponse<Order>

{

WasSuccess = true,

Result = order

};

}

public async Task<ActionResponse<Order>> UpdateFullAsync(string email, OrderDTO orderDTO)

{

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

if (user == null)

{

return new ActionResponse<Order>

{

WasSuccess = false,

Message = "Usuario no existe"

};

}

var isAdmin = await \_usersRepository.IsUserInRoleAsync(user, UserType.Admin.ToString());

if (!isAdmin && orderDTO.OrderStatus != OrderStatus.Cancelled)

{

return new ActionResponse<Order>

{

WasSuccess = false,

Message = "Solo permitido para administradores."

};

}

var order = await \_context.Orders

.Include(s => s.OrderDetails)

.FirstOrDefaultAsync(s => s.Id == orderDTO.Id);

if (order == null)

{

return new ActionResponse<Order>

{

WasSuccess = false,

Message = "Pedido no existe"

};

}

if (orderDTO.OrderStatus == OrderStatus.Cancelled)

{

await ReturnStockAsync(order);

}

order.OrderStatus = orderDTO.OrderStatus;

\_context.Update(order);

await \_context.SaveChangesAsync();

return new ActionResponse<Order>

{

WasSuccess = true,

Result = order

};

}

private async Task ReturnStockAsync(Order order)

{

foreach (var orderDetail in order.OrderDetails!)

{

var product = await \_context.Products.FirstOrDefaultAsync(p => p.Id == orderDetail.ProductId);

if (product != null)

{

product.Stock += orderDetail.Quantity;

}

}

await \_context.SaveChangesAsync();

}

}

}

1. Creamos el **IOrdersUnitOfWork**:

using Orders.Shared.DTOs;

using Orders.Shared.Entities;

using Orders.Shared.Responses;

namespace Orders.Backend.UnitsOfWork.Interfaces

{

public interface IOrdersUnitOfWork

{

Task<ActionResponse<IEnumerable<Order>>> GetAsync(string email, PaginationDTO pagination);

Task<ActionResponse<int>> GetTotalPagesAsync(string email, PaginationDTO pagination);

Task<ActionResponse<Order>> GetAsync(int id);

Task<ActionResponse<Order>> UpdateFullAsync(string email, OrderDTO orderDTO);

}

}

1. Creamos el **OrdersUnitOfWork**:

using Orders.Backend.Repositories.Interfaces;

using Orders.Backend.UnitsOfWork.Interfaces;

using Orders.Shared.DTOs;

using Orders.Shared.Entities;

using Orders.Shared.Responses;

namespace Orders.Backend.UnitsOfWork.Implementations

{

public class OrdersUnitOfWork : GenericUnitOfWork<Order>, IOrdersUnitOfWork

{

private readonly IOrdersRepository \_ordersRepository;

public OrdersUnitOfWork(IGenericRepository<Order> repository, IOrdersRepository ordersRepository) : base(repository)

{

\_ordersRepository = ordersRepository;

}

public async Task<ActionResponse<IEnumerable<Order>>> GetAsync(string email, PaginationDTO pagination) => await \_ordersRepository.GetAsync(email, pagination);

public async Task<ActionResponse<int>> GetTotalPagesAsync(string email, PaginationDTO pagination) => await \_ordersRepository.GetTotalPagesAsync(email, pagination);

public async Task<ActionResponse<Order>> UpdateFullAsync(string email, OrderDTO orderDTO) => await \_ordersRepository.UpdateFullAsync(email, orderDTO);

public override async Task<ActionResponse<Order>> GetAsync(int id) => await \_ordersRepository.GetAsync(id);

}

}

1. Modificamos el **IProductsUnitOfWork**, no hay que implementar nada, porque lo toma del genérico. Solo se matricula en la intarfaz para exponerlo:

Task<ActionResponse<Product>> UpdateAsync(Product product);

1. Modificamos el **ITemporalOrdersUnitOfWork**, no hay que implementar nada, porque lo toma del genérico. Solo se matricula en la intarfaz para exponerlo.

Task<ActionResponse<TemporalOrder>> DeleteAsync(int id);

1. Modificamos el **IOrdersUnitOfWork**, no hay que implementar nada, porque lo toma del genérico. Solo se matricula en la intarfaz para exponerlo.

Task<ActionResponse<Order>> AddAsync(Order order);

1. En **Backend/Helpers** creamos el **IOrdersHelper**:

using Orders.Shared.Responses;

namespace Orders.Backend.Helpers

{

public interface IOrdersHelper

{

Task<ActionResponse<bool>> ProcessOrderAsync(string email, string remarks);

}

}

1. Luego hacemos la implementación en el **OrdersHelper**:

using Orders.Backend.UnitsOfWork.Interfaces;

using Orders.Shared.Entities;

using Orders.Shared.Enums;

using Orders.Shared.Responses;

namespace Orders.Backend.Helpers

{

public class OrdersHelper : IOrdersHelper

{

private readonly IUsersUnitOfWork \_usersUnitOfWork;

private readonly ITemporalOrdersUnitOfWork \_temporalOrdersUnitOfWork;

private readonly IProductsUnitOfWork \_productsUnitOfWork;

private readonly IOrdersUnitOfWork \_ordersUnitOfWork;

public OrdersHelper(IUsersUnitOfWork usersUnitOfWork, ITemporalOrdersUnitOfWork temporalOrdersUnitOfWork, IProductsUnitOfWork productsUnitOfWork, IOrdersUnitOfWork ordersUnitOfWork)

{

\_usersUnitOfWork = usersUnitOfWork;

\_temporalOrdersUnitOfWork = temporalOrdersUnitOfWork;

\_productsUnitOfWork = productsUnitOfWork;

\_ordersUnitOfWork = ordersUnitOfWork;

}

public async Task<ActionResponse<bool>> ProcessOrderAsync(string email, string remarks)

{

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

if (user == null)

{

return new ActionResponse<bool>

{

WasSuccess = false,

Message = "Usuario no válido"

};

}

var actionTemporalOrders = await \_temporalOrdersUnitOfWork.GetAsync(email);

if (!actionTemporalOrders.WasSuccess)

{

return new ActionResponse<bool>

{

WasSuccess = false,

Message = "No hay detalle en la orden"

};

}

var temporalOrders = actionTemporalOrders.Result as List<TemporalOrder>;

var response = await CheckInventoryAsync(temporalOrders!);

if (!response.WasSuccess)

{

return response;

}

var order = new Order

{

Date = DateTime.UtcNow,

User = user,

Remarks = remarks,

OrderDetails = new List<OrderDetail>(),

OrderStatus = OrderStatus.New

};

foreach (var temporalOrder in temporalOrders!)

{

order.OrderDetails.Add(new OrderDetail

{

Product = temporalOrder.Product,

Quantity = temporalOrder.Quantity,

Remarks = temporalOrder.Remarks,

});

var actionProduct = await \_productsUnitOfWork.GetAsync(temporalOrder.Product!.Id);

if (actionProduct.WasSuccess)

{

var product = actionProduct.Result;

if (product != null)

{

product.Stock -= temporalOrder.Quantity;

await \_productsUnitOfWork.UpdateAsync(product);

}

}

await \_temporalOrdersUnitOfWork.DeleteAsync(temporalOrder.Id);

}

await \_ordersUnitOfWork.AddAsync(order);

return response;

}

private async Task<ActionResponse<bool>> CheckInventoryAsync(List<TemporalOrder> temporalOrders)

{

var response = new ActionResponse<bool>() { WasSuccess = true };

foreach (var temporalOrder in temporalOrders)

{

var actionProduct = await \_productsUnitOfWork.GetAsync(temporalOrder.Product!.Id);

if (!actionProduct.WasSuccess)

{

response.WasSuccess = false;

response.Message = $"El producto {temporalOrder.Product!.Id}, ya no está disponible";

return response;

}

var product = actionProduct.Result;

if (product == null)

{

response.WasSuccess = false;

response.Message = $"El producto {temporalOrder.Product!.Id}, ya no está disponible";

return response;

}

if (product.Stock < temporalOrder.Quantity)

{

response.WasSuccess = false;

response.Message = $"Lo sentimos no tenemos existencias suficientes del producto {temporalOrder.Product!.Name}, para tomar su pedido. Por favor disminuir la cantidad o sustituirlo por otro.";

return response;

}

}

return response;

}

}

}

1. Configuramos las nuevas inyecciones en el **Program** del **Backend**:

…

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

builder.Services.AddTransient<SeedDb>();

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

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

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

builder.Services.AddScoped<IOrdersHelper, OrdersHelper>();

…

builder.Services.AddScoped<ICategoriesRepository, CategoriesRepository>();

builder.Services.AddScoped<ICitiesRepository, CitiesRepository>();

builder.Services.AddScoped<ICountriesRepository, CountriesRepository>();

builder.Services.AddScoped<IOrdersRepository, OrdersRepository>();

builder.Services.AddScoped<IProductsRepository, ProductsRepository>();

builder.Services.AddScoped<IStatesRepository, StatesRepository>();

builder.Services.AddScoped<ITemporalOrdersRepository, TemporalOrdersRepository>();

builder.Services.AddScoped<IUsersRepository, UsersRepository>();

builder.Services.AddScoped<ICategoriesUnitOfWork, CategoriesUnitOfWork>();

builder.Services.AddScoped<ICitiesUnitOfWork, CitiesUnitOfWork>();

builder.Services.AddScoped<ICountriesUnitOfWork, CountriesUnitOfWork>();

builder.Services.AddScoped<IOrdersUnitOfWork, OrdersUnitOfWork>();

builder.Services.AddScoped<IProductsUnitOfWork, ProductsUnitOfWork>();

builder.Services.AddScoped<IStatesUnitOfWork, StatesUnitOfWork>();

builder.Services.AddScoped<ITemporalOrdersUnitOfWork, TemporalOrdersUnitOfWork>();

builder.Services.AddScoped<IUsersUnitOfWork, UsersUnitOfWork>();

…

1. Creamos el **OrdersController**:

using Microsoft.AspNetCore.Authentication.JwtBearer;

using Microsoft.AspNetCore.Authorization;

using Microsoft.AspNetCore.Mvc;

using Orders.Backend.Helpers;

using Orders.Shared.DTOs;

namespace Orders.Backend.Controllers

{

[ApiController]

[Authorize(AuthenticationSchemes = JwtBearerDefaults.AuthenticationScheme)]

[Route("api/[controller]")]

public class OrdersController : ControllerBase

{

private readonly IOrdersHelper \_ordersHelper;

public OrdersController(IOrdersHelper ordersHelper)

{

\_ordersHelper = ordersHelper;

}

[HttpPost]

public async Task<IActionResult> PostAsync(OrderDTO saleDTO)

{

var response = await \_ordersHelper.ProcessOrderAsync(User.Identity!.Name!, saleDTO.Remarks);

if (response.WasSuccess)

{

return NoContent();

}

return BadRequest(response.Message);

}

}

}

1. Copiamos las imagenes en el **WWWRoot** del **Frontend**.
2. Creamos la página de confirmación de pedido **Pages/Cart/OrderConfirmed.razor**:

@page "/Cart/OrderConfirmed"

<center>

<h3>Pedido Confirmado</h3>

<img src="images/Shopping.png" width="300" />

<p>Su peidido ha sido confirmado. En pronto recibirá sus productos, muchas gracias</p>

<a href="/" class="btn btn-primary"><i class="bi bi-house" /> Volver al inicio</a>

</center>

1. Modificamos **ConfirmOrderAsync** del **ShowCart.razor.cs**:

private async Task ConfirmOrderAsync()

{

var result = await SweetAlertService.FireAsync(new SweetAlertOptions

{

Title = "Confirmación",

Text = "¿Esta seguro que quieres confirmar el pedido?",

Icon = SweetAlertIcon.Question,

ShowCancelButton = true

});

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

if (confirm)

{

return;

}

var httpActionResponse = await Repository.PostAsync("/api/orders", OrderDTO);

if (httpActionResponse.Error)

{

var message = await httpActionResponse.GetErrorMessageAsync();

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

return;

}

NavigationManager.NavigateTo("/Cart/OrderConfirmed");

}

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

## Administrar pedidos

1. Para poder ver las descripciones de las enumeraciones creamos el **EnumHelper** en el **Frontend**:

using System.ComponentModel;

namespace Orders.Frontend.Helpers

{

public class EnumHelper

{

public static string GetEnumDescription(Enum value)

{

var field = value.GetType().GetField(value.ToString())!;

var attributes = (DescriptionAttribute[])field.GetCustomAttributes(typeof(DescriptionAttribute), false);

if (attributes.Length > 0)

{

return attributes[0].Description;

}

else

{

return value.ToString();

}

}

}

}

1. Modificamos el **OrdersController**:

using Microsoft.AspNetCore.Authentication.JwtBearer;

using Microsoft.AspNetCore.Authorization;

using Microsoft.AspNetCore.Mvc;

using Orders.Backend.Helpers;

using Orders.Backend.UnitsOfWork.Interfaces;

using Orders.Shared.DTOs;

namespace Orders.Backend.Controllers

{

[ApiController]

[Authorize(AuthenticationSchemes = JwtBearerDefaults.AuthenticationScheme)]

[Route("api/[controller]")]

public class OrdersController : ControllerBase

{

private readonly IOrdersHelper \_ordersHelper;

private readonly IOrdersUnitOfWork \_ordersUnitOfWork;

public OrdersController(IOrdersHelper ordersHelper, IOrdersUnitOfWork ordersUnitOfWork)

{

\_ordersHelper = ordersHelper;

\_ordersUnitOfWork = ordersUnitOfWork;

}

[HttpPost]

public async Task<IActionResult> PostAsync(OrderDTO saleDTO)

{

var response = await \_ordersHelper.ProcessOrderAsync(User.Identity!.Name!, saleDTO.Remarks);

if (response.WasSuccess)

{

return NoContent();

}

return BadRequest(response.Message);

}

[HttpGet]

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

{

var response = await \_ordersUnitOfWork.GetAsync(User.Identity!.Name!, pagination);

if (response.WasSuccess)

{

return Ok(response.Result);

}

return BadRequest();

}

[HttpGet("totalPages")]

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

{

var action = await \_ordersUnitOfWork.GetTotalPagesAsync(User.Identity!.Name!, pagination);

if (action.WasSuccess)

{

return Ok(action.Result);

}

return BadRequest();

}

}

}

1. Modificamos el **\_imports.razor**:

@using Orders.Frontend.Helpers;

1. Creamos en **Pages/Cart** el **OrdersIndex.razor** y **OrdersIndex.razor.cs**:

using CurrieTechnologies.Razor.SweetAlert2;

using Microsoft.AspNetCore.Authorization;

using Microsoft.AspNetCore.Components;

using Orders.Frontend.Repositories;

using Orders.Shared.Entities;

namespace Orders.Frontend.Pages.Cart

{

[Authorize(Roles = "Admin")]

public partial class OrdersIndex

{

[Inject] private IRepository repository { get; set; } = null!;

[Inject] private SweetAlertService sweetAlertService { get; set; } = null!;

private int currentPage = 1;

private int totalPages;

public List<Order>? Orders { get; set; }

[Parameter, SupplyParameterFromQuery] public int RecordsNumber { get; set; } = 10;

[Parameter, SupplyParameterFromQuery] public string Page { get; set; } = string.Empty;

protected override async Task OnInitializedAsync()

{

await LoadAsync();

}

private async Task SelectedRecordsNumberAsync(int recordsnumber)

{

RecordsNumber = recordsnumber;

int page = 1;

await LoadAsync(page);

await SelectedPageAsync(page);

}

private async Task SelectedPageAsync(int page)

{

currentPage = page;

await LoadAsync(page);

}

private async Task LoadAsync(int page = 1)

{

if (!string.IsNullOrWhiteSpace(Page))

{

page = Convert.ToInt32(Page);

}

var ok = await LoadListAsync(page);

if (ok)

{

await LoadPagesAsync();

}

}

private void ValidateRecordsNumber(int recordsnumber)

{

if (recordsnumber == 0)

{

RecordsNumber = 10;

}

}

private async Task<bool> LoadListAsync(int page)

{

ValidateRecordsNumber(RecordsNumber);

var url = $"api/orders?page={page}&recordsnumber={RecordsNumber}";

var response = await repository.GetAsync<List<Order>>(url);

if (response.Error)

{

var message = await response.GetErrorMessageAsync();

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

return false;

}

Orders = response.Response;

return true;

}

private async Task LoadPagesAsync()

{

ValidateRecordsNumber(RecordsNumber);

var url = $"api/orders/totalPages?recordsnumber={RecordsNumber}";

var response = await repository.GetAsync<int>(url);

if (response.Error)

{

var message = await response.GetErrorMessageAsync();

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

return;

}

totalPages = response.Response;

}

}

}

1. Modificamos el **OrdersIndex.razor**:

@page "/orders"

@if (Orders is null)

{

<Loading />

}

else

{

<GenericList MyList="Orders">

<Body>

<div class="card">

<div class="card-header">

<span>

<i class="bi bi-currency-dollar" /> Pedidos

</span>

</div>

<div class="card-body">

<Pagination CurrentPage="currentPage"

TotalPages="totalPages"

SelectedPage="SelectedPageAsync"

RecordsNumber="SelectedRecordsNumberAsync" />

<table class="table table-striped">

<thead>

<tr>

<th>Fecha</th>

<th>Usuario</th>

<th>Comentario</th>

<th>Estado</th>

<th>Líneas</th>

<th>Cantidad</th>

<th>Valor</th>

<th></th>

</tr>

</thead>

<tbody>

@foreach (var sale in Orders)

{

<tr>

<td>

@($"{sale.Date:yyyy/MM/dd hh:mm tt}")

</td>

<td>

@sale.User!.FullName

</td>

<td>

@sale.Remarks

</td>

<td>

@EnumHelper.GetEnumDescription(sale.OrderStatus)

</td>

<td>

@sale.Lines

</td>

<td>

@($"{sale.Quantity:N2}")

</td>

<td>

@($"{sale.Value:C2}")

</td>

<td>

<a href="/cart/orderDetails/@sale.Id" class="btn btn-info btn-sm"><i class="bi bi-info-circle" /> Detalles</a>

</td>

</tr>

}

</tbody>

</table>

</div>

</div>

</Body>

</GenericList>

}

1. Modificamos el **NavMenu.razor.css**:

.bi-currency-dollar-fill-nav-menu {

background-image: url("data:image/svg+xml,%3Csvg xmlns='http://www.w3.org/2000/svg' width='16' height='16' fill='white' class='bi bi-list-nested' viewBox='0 0 16 16'%3E%3Cpath fill-rule='evenodd' d='M4 10.781c.148 1.667 1.513 2.85 3.591 3.003V15h1.043v-1.216c2.27-.179 3.678-1.438 3.678-3.3 0-1.59-.947-2.51-2.956-3.028l-.722-.187V3.467c1.122.11 1.879.714 2.07 1.616h1.47c-.166-1.6-1.54-2.748-3.54-2.875V1H7.591v1.233c-1.939.23-3.27 1.472-3.27 3.156 0 1.454.966 2.483 2.661 2.917l.61.162v4.031c-1.149-.17-1.94-.8-2.131-1.718zm3.391-3.836c-1.043-.263-1.6-.825-1.6-1.616 0-.944.704-1.641 1.8-1.828v3.495l-.2-.05zm1.591 1.872c1.287.323 1.852.859 1.852 1.769 0 1.097-.826 1.828-2.2 1.939V8.73z'/%3E%3C/svg%3E");

}

1. Modificamos el **NavMenu.razor**:

…

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

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

<span class="bi bi-globe-americas-fill-nav-menu" aria-hidden="true"></span> Paises

</NavLink>

</div>

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

<NavLink class="nav-link" href="/orders">

<span class="bi bi-currency-dollar-fill-nav-menu" aria-hidden="true"></span> Pedidos

</NavLink>

</div>

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

<NavLink class="nav-link" href="/products">

<span class="bi bi-box2-fill-nav-menu" aria-hidden="true"></span> Productos

</NavLink>

</div>

…

1. Probamos lo que llevamos hasta el momento.
2. Adicionamos este método al **OrdersController**:

[HttpGet("{id}")]

public async Task<IActionResult> GetAsync(int id)

{

var response = await \_ordersUnitOfWork.GetAsync(id);

if (response.WasSuccess)

{

return Ok(response.Result);

}

return NotFound(response.Message);

}

1. Modificamos el **\_imports.cs**:

@using Orders.Shared.Enums

1. Creamos el **OrderDetails.razor** y **OrderDetails.razor.cs**:

using System.Net;

using CurrieTechnologies.Razor.SweetAlert2;

using Microsoft.AspNetCore.Authorization;

using Microsoft.AspNetCore.Components;

using Orders.Frontend.Repositories;

using Orders.Shared.Entities;

namespace Orders.Frontend.Pages.Cart

{

[Authorize(Roles = "Admin")]

public partial class OrderDetails

{

private Order? order;

[Inject] private NavigationManager NavigationManager { get; set; } = null!;

[Inject] private IRepository Repository { get; set; } = null!;

[Inject] private SweetAlertService SweetAlertService { get; set; } = null!;

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

protected override async Task OnInitializedAsync()

{

await LoadAsync();

}

private async Task LoadAsync()

{

var responseHppt = await Repository.GetAsync<Order>($"api/orders/{OrderId}");

if (responseHppt.Error)

{

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

{

NavigationManager.NavigateTo("/orders");

return;

}

var messageError = await responseHppt.GetErrorMessageAsync();

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

return;

}

order = responseHppt.Response;

}

private void CancelOrderAsync()

{

}

private void DispatchOrderAsync()

{

}

private void SendOrderAsync()

{

}

private void ConfirmOrderAsync()

{

}

}

}

1. Modificamos el **OrderDetails.razor**:

@page "/cart/orderDetails/{OrderId:int}"

@if (order is null)

{

<Loading />

}

else

{

<GenericList MyList="order.OrderDetails!.ToList()">

<Body>

<div class="card">

<div class="card-header">

<span>

<i class="bi bi-currency-dollar"></i> @order.User!.FullName

@if (order.OrderStatus == OrderStatus.New)

{

<button class="btn btn-sm btn-danger float-end mx-2" @onclick=@(() => CancelOrderAsync())><i class="bi bi-trash" /> Cancelar</button>

<button class="btn btn-sm btn-primary float-end mx-2" @onclick=@(() => DispatchOrderAsync())><i class="bi bi-truck" /> Despachar</button>

}

else if (order.OrderStatus == OrderStatus.Dispatched)

{

<button class="btn btn-sm btn-warning float-end mx-2" @onclick=@(() => SendOrderAsync())><i class="bi bi-send" /> Enviar</button>

}

else if (order.OrderStatus == OrderStatus.Sent)

{

<button class="btn btn-sm btn-dark float-end mx-2" @onclick=@(() => ConfirmOrderAsync())><i class="bi bi-hand-thumbs-up" /> Confirmar</button>

}

<a class="btn btn-sm btn-success float-end" href="/orders"><i class="bi bi-arrow-left" /> Regresar</a>

</span>

</div>

<div class="row mx-2 my-2">

<div class="col-2">

<p>Cliente</p>

<p>Documento</p>

<p>Teléfono</p>

<p>Email</p>

<p>Dirección</p>

</div>

<div class="col-4">

<p><strong>@order.User.FullName</strong></p>

<p><strong>@order.User.Document</strong></p>

<p><strong>@order.User.PhoneNumber</strong></p>

<p><strong>@order.User.UserName</strong></p>

<p><strong>@order.User.Address, @order.User.City!.Name, @order.User.City.State!.Name, @order.User.City.State.Country!.Name</strong></p>

</div>

<div class="col-2">

<p>Estado</p>

<p>Fecha</p>

<p>Comentarios</p>

<p>Líneas</p>

<p>Cantidad</p>

<p>Valor</p>

</div>

<div class="col-4">

<p><strong>@EnumHelper.GetEnumDescription(order.OrderStatus)</strong></p>

<p><strong>@($"{order.Date.ToLocalTime():yyyy/MM/dd hh:mm tt}")</strong></p>

<p><strong>@(string.IsNullOrEmpty(order.Remarks) ? "NA" : order.Remarks)</strong></p>

<p><strong>@order.Lines</strong></p>

<p><strong>@($"{order.Quantity:N2}")</strong></p>

<p><strong>@($"{order.Value:C2}")</strong></p>

</div>

</div>

<div class="card-body">

<table class="table table-striped">

<thead>

<tr>

<th>Producto</th>

<th>Imagen</th>

<th>Comentarios</th>

<th>Cantidad</th>

<th>Precio</th>

<th>Valor</th>

</tr>

</thead>

<tbody>

@foreach (var saleDetail in order.OrderDetails!)

{

<tr>

<td>@saleDetail.Product!.Name</td>

<td><img src="@saleDetail.Product!.MainImage" style="width:100px;" /></td>

<td>@saleDetail.Remarks</td>

<td>@($"{saleDetail.Quantity:N2}")</td>

<td>@($"{saleDetail.Product!.Price:C2}")</td>

<td>@($"{saleDetail.Value:C2}")</td>

</tr>

}

</tbody>

</table>

</div>

</div>

</Body>

</GenericList>

}

1. Probamos.
2. Agregamos estos métodos al **OrdersController**:

[HttpPut]

public async Task<IActionResult> PutAsync(OrderDTO orderDTO)

{

var response = await \_ordersUnitOfWork.UpdateFullAsync(User.Identity!.Name!, orderDTO);

if (response.WasSuccess)

{

return Ok(response.Result);

}

return BadRequest(response.Message);

}

1. Modificamos estos métodos al **OrdersDetails.razor.cs**:

private async Task CancelOrderAsync()

{

await ModifyTemporalOrder("cancelar", OrderStatus.Cancelled);

}

private async Task DispatchOrderAsync()

{

await ModifyTemporalOrder("despachar", OrderStatus.Dispatched);

}

private async Task SendOrderAsync()

{

await ModifyTemporalOrder("enviar", OrderStatus.Sent);

}

private async Task ConfirmOrderAsync()

{

await ModifyTemporalOrder("confirmar", OrderStatus.Confirmed);

}

private async Task ModifyTemporalOrder(string message, OrderStatus status)

{

var result = await SweetAlertService.FireAsync(new SweetAlertOptions

{

Title = "Confirmación",

Text = $"¿Esta seguro que quieres {message} el pedido?",

Icon = SweetAlertIcon.Question,

ShowCancelButton = true

});

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

if (confirm)

{

return;

}

var orderDTO = new OrderDTO

{

Id = OrderId,

OrderStatus = status

};

var responseHttp = await Repository.PutAsync("api/orders", orderDTO);

if (responseHttp.Error)

{

var mensajeError = await responseHttp.GetErrorMessageAsync();

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

return;

}

NavigationManager.NavigateTo("/orders");

}

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

## Ver estado de mis pedidos

1. Agregamos estas líneas al **NavMenu.razor**:

…

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

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

<span class="bi bi-star" aria-hidden="true"></span> Productos

</NavLink>

</div>

</Authorized>

</AuthorizeView>

<AuthorizeView Roles="User">

<Authorized>

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

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

<span class="bi bi-currency-dollar" aria-hidden="true"></span> Ver Mis Pedidos

</NavLink>

</div>

</Authorized>

</AuthorizeView>

</nav>

</div>

1. Modificamos el **OrderIndex.razor.cs**:

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

1. Modificamos el **OrderDetails.razor**:

<span>

<i class="bi bi-currency-dollar"></i> @order.User!.FullName

@if (order.OrderStatus == OrderStatus.New)

{

<button class="btn btn-sm btn-danger float-end mx-2" @onclick=@(() => CancelOrderAsync())><i class="bi bi-trash" /> Cancelar</button>

<AuthorizeView Roles="Admin">

<Authorized>

<button class="btn btn-sm btn-primary float-end mx-2" @onclick=@(() => DispatchOrderAsync())><i class="bi bi-truck" /> Despachar</button>

</Authorized>

</AuthorizeView>

}

<AuthorizeView Roles="Admin">

<Authorized>

@if (order.OrderStatus == OrderStatus.Dispatched)

{

<button class="btn btn-sm btn-warning float-end mx-2" @onclick=@(() => SendOrderAsync())><i class="bi bi-send" /> Enviar</button>

}

@if (order.OrderStatus == OrderStatus.Sent)

{

<button class="btn btn-sm btn-dark float-end mx-2" @onclick=@(() => ConfirmOrderAsync())><i class="bi bi-hand-thumbs-up" /> Confirmar</button>

}

</Authorized>

</AuthorizeView>

<a class="btn btn-sm btn-success float-end" href="/orders"><i class="bi bi-arrow-left" /> Regresar</a>

</span>

1. Modificamos el **OrderDetails.razor.cs**:

[Authorize(Roles = "Admin, User")]

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

## Administrar usuarios y crear nuevos administradores

1. Adicionamos estos métodos al **IUsersRepository**:

Task<ActionResponse<IEnumerable<User>>> GetAsync(PaginationDTO pagination);

Task<ActionResponse<int>> GetTotalPagesAsync(PaginationDTO pagination);

1. Adicionamos estos métodos al **UsersRepository**:

public async Task<ActionResponse<IEnumerable<User>>> GetAsync(PaginationDTO pagination)

{

var queryable = \_context.Users

.Include(u => u.City)

.ThenInclude(c => c!.State)

.ThenInclude(s => s!.Country)

.AsQueryable();

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

{

queryable = queryable.Where(x => x.FirstName.ToLower().Contains(pagination.Filter.ToLower()) ||

x.LastName.ToLower().Contains(pagination.Filter.ToLower()));

}

return new ActionResponse<IEnumerable<User>>

{

WasSuccess = true,

Result = await queryable

.OrderBy(x => x.FirstName)

.ThenBy(x => x.LastName)

.Paginate(pagination)

.ToListAsync()

};

}

public async Task<ActionResponse<int>> GetTotalPagesAsync(PaginationDTO pagination)

{

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

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

{

queryable = queryable.Where(x => x.FirstName.ToLower().Contains(pagination.Filter.ToLower()) ||

x.LastName.ToLower().Contains(pagination.Filter.ToLower()));

}

double count = await queryable.CountAsync();

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

return new ActionResponse<int>

{

WasSuccess = true,

Result = (int)totalPages

};

}

1. Adicionamos estos métodos al **IUsersUnitOfWork**:

Task<ActionResponse<IEnumerable<User>>> GetAsync(PaginationDTO pagination);

Task<ActionResponse<int>> GetTotalPagesAsync(PaginationDTO pagination);

1. Adicionamos estos métodos al **UsersUnitOfWork**:

public async Task<ActionResponse<IEnumerable<User>>> GetAsync(PaginationDTO pagination) => await \_usersRepository.GetAsync(pagination);

public async Task<ActionResponse<int>> GetTotalPagesAsync(PaginationDTO pagination) => await \_usersRepository.GetTotalPagesAsync(pagination);

1. Adicionamos estos métodos al **AccountController** (primero inyectamos el **IUsersRepository**):

[HttpGet("all")]

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

{

var response = await \_usersRepository.GetAsync(pagination);

if (response.WasSuccess)

{

return Ok(response.Result);

}

return BadRequest();

}

[HttpGet("totalPages")]

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

{

var action = await \_usersRepository.GetTotalPagesAsync(pagination);

if (action.WasSuccess)

{

return Ok(action.Result);

}

return BadRequest();

}

1. Adicionamos estas línea al **NavMenu.css**:

.bi-people-fill-nav-menu {

background-image: url("data:image/svg+xml,%3Csvg xmlns='http://www.w3.org/2000/svg' width='16' height='16' fill='white' class='bi bi-list-nested' viewBox='0 0 16 16'%3E%3Cpath fill-rule='evenodd' d='M7 14s-1 0-1-1 1-4 5-4 5 3 5 4-1 1-1 1zm4-6a3 3 0 1 0 0-6 3 3 0 0 0 0 6m-5.784 6A2.24 2.24 0 0 1 5 13c0-1.355.68-2.75 1.936-3.72A6.3 6.3 0 0 0 5 9c-4 0-5 3-5 4s1 1 1 1zM4.5 8a2.5 2.5 0 1 0 0-5 2.5 2.5 0 0 0 0 5'/%3E%3C/svg%3E");

}

1. Adicionamos estas línea al **NavMenu**:

…

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

<NavLink class="nav-link" href="/products">

<span class="bi bi-box2-fill-nav-menu" aria-hidden="true"></span> Productos

</NavLink>

</div>

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

<NavLink class="nav-link" href="/users">

<span class="bi bi-people-fill-nav-menu" aria-hidden="true"></span> Usuarios

</NavLink>

</div>

…

1. Creamos el **UserIndex.razor** y **UserIndex.razor.cs** dentro de **Pages/Auth**:

using CurrieTechnologies.Razor.SweetAlert2;

using Microsoft.AspNetCore.Authorization;

using Microsoft.AspNetCore.Components;

using Orders.Frontend.Repositories;

using Orders.Shared.Entities;

namespace Orders.Frontend.Pages.Auth

{

[Authorize(Roles = "Admin")]

public partial class UserIndex

{

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

private int currentPage = 1;

private int totalPages;

[Inject] private IRepository Repository { get; set; } = null!;

[Inject] private SweetAlertService SweetAlertService { get; set; } = null!;

[Parameter, SupplyParameterFromQuery] public string Page { get; set; } = string.Empty;

[Parameter, SupplyParameterFromQuery] public string Filter { get; set; } = string.Empty;

[Parameter, SupplyParameterFromQuery] public int RecordsNumber { get; set; } = 10;

protected override async Task OnInitializedAsync()

{

await LoadAsync();

}

private async Task SelectedRecordsNumberAsync(int recordsnumber)

{

RecordsNumber = recordsnumber;

int page = 1;

await LoadAsync(page);

await SelectedPageAsync(page);

}

private async Task FilterCallBack(string filter)

{

Filter = filter;

await ApplyFilterAsync();

StateHasChanged();

}

private async Task SelectedPageAsync(int page)

{

currentPage = page;

await LoadAsync(page);

}

private async Task LoadAsync(int page = 1)

{

if (!string.IsNullOrWhiteSpace(Page))

{

page = Convert.ToInt32(Page);

}

var ok = await LoadListAsync(page);

if (ok)

{

await LoadPagesAsync();

}

}

private void ValidateRecordsNumber(int recordsnumber)

{

if (recordsnumber == 0)

{

RecordsNumber = 10;

}

}

private async Task<bool> LoadListAsync(int page)

{

ValidateRecordsNumber(RecordsNumber);

var url = $"api/accounts/all?page={page}&recordsnumber={RecordsNumber}";

if (!string.IsNullOrEmpty(Filter))

{

url += $"&filter={Filter}";

}

var response = await Repository.GetAsync<List<User>>(url);

if (response.Error)

{

var message = await response.GetErrorMessageAsync();

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

return false;

}

Users = response.Response;

return true;

}

private async Task LoadPagesAsync()

{

ValidateRecordsNumber(RecordsNumber);

var url = $"api/accounts/totalPages?recordsnumber={RecordsNumber}";

if (!string.IsNullOrEmpty(Filter))

{

url += $"&filter={Filter}";

}

var response = await Repository.GetAsync<int>(url);

if (response.Error)

{

var message = await response.GetErrorMessageAsync();

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

return;

}

totalPages = response.Response;

}

private async Task ApplyFilterAsync()

{

await LoadAsync();

}

}

}

1. Modificamos el **UserIndex.razor** dentro de **Pages/Auth**:

@page "/users"

@if (Users is null)

{

<Loading />

}

else

{

<GenericList MyList="Users">

<Body>

<div class="card">

<div class="card-header">

<span>

<i class="bi bi-people" /> Usuarios

<a class="btn btn-sm btn-primary float-end" href="/register/?IsAdmin=true"><i class="bi bi-plus-circle" /> Adicionar Administrador</a>

</span>

</div>

<div class="card-body">

<Filter PlaceHolder="Buscar usuario..." Callback=@FilterCallBack />

<Pagination CurrentPage="currentPage"

TotalPages="totalPages"

SelectedPage="SelectedPageAsync"

RecordsNumber="SelectedRecordsNumberAsync" />

<table class="table table-striped">

<thead>

<tr>

<th>Imagén</th>

<th>Usuario</th>

<th>Documento</th>

<th>Teléfono</th>

<th>Email</th>

<th>Dirección</th>

<th>Confirmado</th>

<th>Tipo Usuario</th>

</tr>

</thead>

<tbody>

@foreach (var user in Users)

{

<tr>

<td><img src="@user.Photo" width="80" height="80" style="border-radius:50%" /></td>

<td>@user.FullName</td>

<td>@user.Document</td>

<td>@user.PhoneNumber</td>

<td>@user.Email</td>

<td>@user.Address, @user.City!.Name, @user.City!.State!.Name, @user.City!.State!.Country!.Name</td>

<td>@user.EmailConfirmed</td>

<td>@EnumHelper.GetEnumDescription(user.UserType)</td>

</tr>

}

</tbody>

</table>

</div>

</div>

</Body>

</GenericList>

}

1. Probamos.
2. Modificamos el **Register.razor.cs**:

…

[Parameter, SupplyParameterFromQuery] public bool IsAdmin { get; set; }

…

private async Task CreteUserAsync()

{

userDTO.UserName = userDTO.Email;

userDTO.UserType = UserType.User;

if (IsAdmin)

{

userDTO.UserType = UserType.Admin;

}

loading = true;

var responseHttp = await Repository.PostAsync<UserDTO>("/api/accounts/CreateUser", userDTO);

loading = false;

if (responseHttp.Error)

{

var message = await responseHttp.GetErrorMessageAsync();

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

return;

}

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**.

## Corrección para que corra el App en Mac

1. Modificamos el **SeedBd**:

…

foreach (string? image in images)

{

string filePath;

if (RuntimeInformation.IsOSPlatform(OSPlatform.Windows))

{

filePath = $"{Environment.CurrentDirectory}\\Images\\products\\{image}";

}

else

{

filePath = $"{Environment.CurrentDirectory}/Images/products/{image}";

}

var fileBytes = File.ReadAllBytes(filePath);

var imagePath = await \_fileStorage.SaveFileAsync(fileBytes, "jpg", "products");

prodcut.ProductImages.Add(new ProductImage { Image = imagePath });

}

…

var city = await \_context.Cities.FirstOrDefaultAsync(x => x.Name == "Medellín");

if (city == null)

{

city = await \_context.Cities.FirstOrDefaultAsync();

}

string filePath;

if (RuntimeInformation.IsOSPlatform(OSPlatform.Windows))

{

filePath = $"{Environment.CurrentDirectory}\\Images\\users\\{image}";

}

else

{

filePath = $"{Environment.CurrentDirectory}/Images/users/{image}";

}

var fileBytes = File.ReadAllBytes(filePath);

var imagePath = await \_fileStorage.SaveFileAsync(fileBytes, "jpg", "users");

…

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

## Fitros por categorías

De encima, no me quedo contento si no implementamos esto, luego de haber echo el esfuerzo de incluir categorías y asignarle una o varas categorías a un producto.

1. Adicionamos esta propiedad al **PaginationDTO**:

public string? CategoryFilter { get; set; }

1. Modificamos estos métodos en el **ProductsRepository**:

public override async Task<ActionResponse<IEnumerable<Product>>> GetAsync(PaginationDTO pagination)

{

var queryable = \_context.Products

.Include(x => x.ProductImages)

.Include(x => x.ProductCategories)

.AsQueryable();

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

{

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

}

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

{

queryable = queryable.Where(x => x.ProductCategories!.Any(y => y.Category.Name == pagination.CategoryFilter));

}

return new ActionResponse<IEnumerable<Product>>

{

WasSuccess = true,

Result = await queryable

.OrderBy(x => x.Name)

.Paginate(pagination)

.ToListAsync()

};

}

public override async Task<ActionResponse<int>> GetTotalPagesAsync(PaginationDTO pagination)

{

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

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

{

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

}

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

{

queryable = queryable.Where(x => x.ProductCategories!.Any(y => y.Category.Name == pagination.CategoryFilter));

}

double count = await queryable.CountAsync();

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

return new ActionResponse<int>

{

WasSuccess = true,

Result = totalPages

};

}

1. Modificamos el **Home.razor.cs**:

…

private int currentPage = 1;

private int totalPages;

private int counter = 0;

private bool isAuthenticated;

private string allCategories = "all\_categories\_list";

public List<Product>? Products { get; set; }

public List<Category>? Categories { get; set; }

public string CategoryFilter { get; set; } = string.Empty;

…

protected async override Task OnParametersSetAsync()

{

await CheckIsAuthenticatedAsync();

await LoadCounterAsync();

await LoadCategoriesAsync();

}

private async Task LoadCategoriesAsync()

{

var responseHttp = await Repository.GetAsync<List<Category>>("api/categories/combo");

if (responseHttp.Error)

{

var message = await responseHttp.GetErrorMessageAsync();

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

}

Categories = responseHttp.Response;

}

…

private async Task LoadAsync(int page = 1, string category = "")

{

if (!string.IsNullOrWhiteSpace(category))

{

if (category == allCategories)

{

CategoryFilter = string.Empty;

}

else

{

CategoryFilter = category;

}

}

if (!string.IsNullOrWhiteSpace(Page))

{

page = Convert.ToInt32(Page);

}

var ok = await LoadListAsync(page);

if (ok)

{

await LoadPagesAsync();

}

}

…

private async Task<bool> LoadListAsync(int page)

{

ValidateRecordsNumber(RecordsNumber);

var url = $"api/products?page={page}&recordsnumber={RecordsNumber}";

if (!string.IsNullOrEmpty(Filter))

{

url += $"&filter={Filter}";

}

if (!string.IsNullOrEmpty(CategoryFilter))

{

url += $"&CategoryFilter={CategoryFilter}";

}

var response = await Repository.GetAsync<List<Product>>(url);

…

private async Task LoadPagesAsync()

{

ValidateRecordsNumber(RecordsNumber);

var url = $"api/products/totalPages?recordsnumber={RecordsNumber}";

if (!string.IsNullOrEmpty(Filter))

{

url += $"&filter={Filter}";

}

if (!string.IsNullOrEmpty(CategoryFilter))

{

url += $"&CategoryFilter={CategoryFilter}";

}

var response = await Repository.GetAsync<int>(url);

…

1. Modificamos el **Home.razor**:

@page "/"

@if (Products is null)

{

<Loading />

}

else

{

if (Categories != null)

{

<div class="d-flex flex-wrap justify-content-center mb-4 mt-2">

@foreach (var category in Categories)

{

<a class="btn btn-link" style="cursor: pointer" @onclick=@(() => LoadAsync(1, category.Name))>@category.Name</a>

}

<a class="btn btn-link" style="cursor: pointer" @onclick=@(() => LoadAsync(1, allCategories))>Todos</a>

</div>

}

<div class="d-flex align-items-center justify-content-between">

<Filter PlaceHolder="Buscar producto..." Callback=@FilterCallBack />

<AuthorizeView>

<Authorized>

@if (counter > 0)

{

<a href="/Cart/ShowCart" class="btn btn-primary"><i class="bi bi-cart-fill" /> Ver Carro de Compras (@counter)</a>

}

</Authorized>

</AuthorizeView>

</div>

if (Products.Count > 0)

{

<Pagination CurrentPage="currentPage"

TotalPages="totalPages"

SelectedPage="SelectedPageAsync"

RecordsNumber="SelectedRecordsNumberAsync"

IsHome />

<div class="row row-cols-1 row-cols-md-4 g-4 mt-1">

@foreach (var product in Products!)

{

<div class="col">

<div class="card h-100">

<div class="text-center zoom">

<img src="@product.MainImage" style="height:150px; max-width:200px;" class="text-center" alt=@product.Name />

</div>

<div class="card-body">

<h5 class="card-title text-navy"> @product.Name</h5>

<p class="card-text smfnt">@product.Description</p>

<h5 class="text-muted">@($"{product.Price:C2}")</h5>

</div>

<div class="card-footer text-center">

<a href="/products/details/@product.Id" class="btn btn-sm btn-secondary"><i class="bi bi-info-circle" /> Detalles</a>

<button class="btn btn-sm btn-primary" @onclick=@(() => AddToCartAsync(product.Id))><i class="bi bi-cart-plus" /> Agregar al Carro</button>

</div>

</div>

</div>

}

</div>

}

else

{

<div class="d-flex justify-content-center align-items-center" style="height: 30vh;">

<h1>Lo siento, no hay productos con estos criterios de búsqueda</h1>

</div>

}

}

1. Probamos.

## Creando pruebas unitarias

### Generales

1. Agreguele estos paquetes al nuevo proyecto **Orders.Test**:

**Microsoft.EntityFrameworkCore.InMemory**

**Moq**

1. Y actualizamos los paquetes del proyecto.
2. Instalamos las extensiones **Fine Code Coverage** y **Run Coverlet Report VS2022**. Para poder medir la cobertura de nuestras pruebas unitarias.

### Categorias

#### Controlador

1. Cree la carpeta **Controllers** y dentro de este adicione la clase **CategoriesControllerTests**:

using Microsoft.AspNetCore.Mvc;

using Moq;

using Orders.Backend.Controllers;

using Orders.Backend.UnitsOfWork;

using Orders.Shared.DTOs;

using Orders.Shared.Entites;

using Orders.Shared.Responses;

namespace Orders.Tests.Controllers

{

[TestClass]

public class CategoriesControllerTests

{

private Mock<IGenericUnitOfWork<Category>> \_mockGenericUnitOfWork = null!;

private Mock<ICategoriesUnitOfWork> \_mockCategoriesUnitOfWork = null!;

private CategoriesController \_controller = null!;

[TestInitialize]

public void Setup()

{

\_mockGenericUnitOfWork = new Mock<IGenericUnitOfWork<Category>>();

\_mockCategoriesUnitOfWork = new Mock<ICategoriesUnitOfWork>();

\_controller = new CategoriesController(\_mockGenericUnitOfWork.Object, \_mockCategoriesUnitOfWork.Object);

}

[TestMethod]

public async Task GetComboAsync\_ReturnsOkObjectResult()

{

// Arrange

var comboData = new List<Category> { new Category() };

\_mockCategoriesUnitOfWork.Setup(x => x.GetComboAsync()).ReturnsAsync(comboData);

// Act

var result = await \_controller.GetComboAsync();

// Assert

Assert.IsInstanceOfType(result, typeof(OkObjectResult));

var okResult = result as OkObjectResult;

Assert.AreEqual(comboData, okResult!.Value);

\_mockCategoriesUnitOfWork.Verify(x => x.GetComboAsync(), Times.Once());

}

[TestMethod]

public async Task GetAsync\_ReturnsOkObjectResult\_WhenWasSuccessIsTrue()

{

// Arrange

var pagination = new PaginationDTO();

var response = new ActionResponse<IEnumerable<Category>> { WasSuccess = true };

\_mockCategoriesUnitOfWork.Setup(x => x.GetAsync(pagination)).ReturnsAsync(response);

// Act

var result = await \_controller.GetAsync(pagination);

// Assert

Assert.IsInstanceOfType(result, typeof(OkObjectResult));

var okResult = result as OkObjectResult;

Assert.AreEqual(response.Result, okResult!.Value);

\_mockCategoriesUnitOfWork.Verify(x => x.GetAsync(pagination), Times.Once());

}

[TestMethod]

public async Task GetAsync\_ReturnsBadRequestResult\_WhenWasSuccessIsFalse()

{

// Arrange

var pagination = new PaginationDTO();

var response = new ActionResponse<IEnumerable<Category>> { WasSuccess = false };

\_mockCategoriesUnitOfWork.Setup(x => x.GetAsync(pagination)).ReturnsAsync(response);

// Act

var result = await \_controller.GetAsync(pagination);

// Assert

Assert.IsInstanceOfType(result, typeof(BadRequestResult));

\_mockCategoriesUnitOfWork.Verify(x => x.GetAsync(pagination), Times.Once());

}

[TestMethod]

public async Task GetPagesAsync\_ReturnsOkObjectResult\_WhenWasSuccessIsTrue()

{

// Arrange

var pagination = new PaginationDTO();

var action = new ActionResponse<int> { WasSuccess = true, Result = 5 };

\_mockCategoriesUnitOfWork.Setup(x => x.GetTotalPagesAsync(pagination)).ReturnsAsync(action);

// Act

var result = await \_controller.GetPagesAsync(pagination);

// Assert

Assert.IsInstanceOfType(result, typeof(OkObjectResult));

var okResult = result as OkObjectResult;

Assert.AreEqual(action.Result, okResult!.Value);

\_mockCategoriesUnitOfWork.Verify(x => x.GetTotalPagesAsync(pagination), Times.Once());

}

[TestMethod]

public async Task GetPagesAsync\_ReturnsBadRequestResult\_WhenWasSuccessIsFalse()

{

// Arrange

var pagination = new PaginationDTO();

var action = new ActionResponse<int> { WasSuccess = false };

\_mockCategoriesUnitOfWork.Setup(x => x.GetTotalPagesAsync(pagination)).ReturnsAsync(action);

// Act

var result = await \_controller.GetPagesAsync(pagination);

// Assert

Assert.IsInstanceOfType(result, typeof(BadRequestResult));

\_mockCategoriesUnitOfWork.Verify(x => x.GetTotalPagesAsync(pagination), Times.Once());

}

}

}

1. Corra los test y verifique que todo está funcionando correctamente.
2. Verificamos la cobertura del código.
3. Hacemos commit.

#### Unidad de Trabajo

1. Creamos la carpeta **UnitsOfWork** y dentro de esta adicione la clase **CategoriesUnitOfWorkTests**:

using Moq;

using Orders.Backend.Repositories;

using Orders.Backend.UnitsOfWork;

using Orders.Shared.DTOs;

using Orders.Shared.Entites;

using Orders.Shared.Responses;

namespace Orders.Tests.UnitsOfWork

{

[TestClass]

public class CategoriesUnitOfWorkTests

{

private Mock<IGenericRepository<Category>> \_mockGenericRepository = null!;

private Mock<ICategoriesRepository> \_mockCategoriesRepository = null!;

private CategoriesUnitOfWork \_unitOfWork = null!;

[TestInitialize]

public void Setup()

{

\_mockGenericRepository = new Mock<IGenericRepository<Category>>();

\_mockCategoriesRepository = new Mock<ICategoriesRepository>();

\_unitOfWork = new CategoriesUnitOfWork(\_mockGenericRepository.Object, \_mockCategoriesRepository.Object);

}

[TestMethod]

public async Task GetAsync\_CallsRepositoryAndReturnsResult()

{

// Arrange

var pagination = new PaginationDTO();

var expectedActionResponse = new ActionResponse<IEnumerable<Category>> { Result = new List<Category>() };

\_mockCategoriesRepository.Setup(x => x.GetAsync(pagination)).ReturnsAsync(expectedActionResponse);

// Act

var result = await \_unitOfWork.GetAsync(pagination);

// Assert

Assert.AreEqual(expectedActionResponse, result);

\_mockCategoriesRepository.Verify(x => x.GetAsync(pagination), Times.Once);

}

[TestMethod]

public async Task GetComboAsync\_CallsRepositoryAndReturnsResult()

{

// Arrange

var expectedCategories = new List<Category> { new Category() };

\_mockCategoriesRepository.Setup(x => x.GetComboAsync()).ReturnsAsync(expectedCategories);

// Act

var result = await \_unitOfWork.GetComboAsync();

// Assert

Assert.AreEqual(expectedCategories, result);

\_mockCategoriesRepository.Verify(x => x.GetComboAsync(), Times.Once);

}

[TestMethod]

public async Task GetTotalPagesAsync\_CallsRepositoryAndReturnsResult()

{

// Arrange

var pagination = new PaginationDTO();

var expectedActionResponse = new ActionResponse<int> { Result = 5 };

\_mockCategoriesRepository.Setup(x => x.GetTotalPagesAsync(pagination)).ReturnsAsync(expectedActionResponse);

// Act

var result = await \_unitOfWork.GetTotalPagesAsync(pagination);

// Assert

Assert.AreEqual(expectedActionResponse, result);

\_mockCategoriesRepository.Verify(x => x.GetTotalPagesAsync(pagination), Times.Once);

}

}

}

1. Corra los test y verifique que todo está funcionando correctamente.
2. Verificamos la cobertura del código.
3. Hacemos commit.

#### Repositorio

1. Cree la carpeta **Repositories** y dentro de esta adicione la clase **CategoriesRepositoryTests**:

using Microsoft.EntityFrameworkCore;

using Orders.Backend.Data;

using Orders.Backend.Repositories.Implementations;

using Orders.Shared.DTOs;

using Orders.Shared.Entities;

namespace Orders.Tests.Repositories

{

[TestClass]

public class CategoriesRepositoryTests

{

private DataContext \_context = null!;

private CategoriesRepository \_repository = null!;

[TestInitialize]

public void Setup()

{

var options = new DbContextOptionsBuilder<DataContext>()

.UseInMemoryDatabase(databaseName: Guid.NewGuid().ToString())

.Options;

\_context = new DataContext(options);

\_repository = new CategoriesRepository(\_context);

\_context.Categories.AddRange(new List<Category>

{

new Category { Id = 1, Name = "Electronics" },

new Category { Id = 2, Name = "Books" },

new Category { Id = 3, Name = "Clothing" },

});

\_context.SaveChanges();

}

[TestCleanup]

public void Cleanup()

{

\_context.Database.EnsureDeleted();

\_context.Dispose();

}

[TestMethod]

public async Task GetAsync\_ReturnsFilteredCategories()

{

// Arrange

var pagination = new PaginationDTO { Filter = "Book", RecordsNumber = 10, Page = 1 };

// Act

var response = await \_repository.GetAsync(pagination);

// Assert

Assert.IsTrue(response.WasSuccess);

var categories = response.Result!.ToList();

Assert.AreEqual(1, categories.Count);

Assert.AreEqual("Books", categories.First().Name);

}

[TestMethod]

public async Task GetAsync\_ReturnsAllCategories\_WhenNoFilterIsProvided()

{

// Arrange

var pagination = new PaginationDTO { RecordsNumber = 10, Page = 1 };

// Act

var response = await \_repository.GetAsync(pagination);

// Assert

Assert.IsTrue(response.WasSuccess);

var categories = response.Result!.ToList();

Assert.AreEqual(3, categories.Count);

}

[TestMethod]

public async Task GetComboAsync\_ReturnsAllCategories()

{

// Act

var categories = await \_repository.GetComboAsync();

// Assert

Assert.AreEqual(3, categories.Count());

}

[TestMethod]

public async Task GetTotalPagesAsync\_ReturnsCorrectNumberOfPages()

{

// Arrange

var pagination = new PaginationDTO { RecordsNumber = 2, Page = 1 };

// Act

var response = await \_repository.GetTotalPagesAsync(pagination);

// Assert

Assert.IsTrue(response.WasSuccess);

Assert.AreEqual(2, response.Result);

}

[TestMethod]

public async Task GetTotalPagesAsync\_WithFilter\_ReturnsCorrectNumberOfPages()

{

// Arrange

var pagination = new PaginationDTO { RecordsNumber = 2, Page = 1, Filter = "Bo" };

// Act

var response = await \_repository.GetTotalPagesAsync(pagination);

// Assert

Assert.IsTrue(response.WasSuccess);

Assert.AreEqual(1, response.Result);

}

}

}

1. Corra los test y verifique que todo está funcionando correctamente.
2. Verificamos la cobertura del código.
3. Hacemos commit.

### Genérico

#### Controlador

1. Adicione la clase **GenericControllerTests**:

using Microsoft.AspNetCore.Mvc;

using Orders.Backend.UnitsOfWork.Interfaces;

using Orders.Shared.DTOs;

namespace Orders.Backend.Controllers

{

public class GenericController<T> : Controller where T : class

{

private readonly IGenericUnitOfWork<T> \_unitOfWork;

public GenericController(IGenericUnitOfWork<T> unitOfWork)

{

\_unitOfWork = unitOfWork;

}

[HttpGet("full")]

public virtual async Task<IActionResult> GetAsync()

{

var action = await \_unitOfWork.GetAsync();

if (action.WasSuccess)

{

return Ok(action.Result);

}

return BadRequest();

}

[HttpGet]

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

{

var action = await \_unitOfWork.GetAsync(pagination);

if (action.WasSuccess)

{

return Ok(action.Result);

}

return BadRequest();

}

[HttpGet("totalPages")]

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

{

var action = await \_unitOfWork.GetTotalPagesAsync(pagination);

if (action.WasSuccess)

{

return Ok(action.Result);

}

return BadRequest();

}

[HttpGet("{id}")]

public virtual async Task<IActionResult> GetAsync(int id)

{

var action = await \_unitOfWork.GetAsync(id);

if (action.WasSuccess)

{

return Ok(action.Result);

}

return NotFound();

}

[HttpPost]

public virtual async Task<IActionResult> PostAsync(T model)

{

var action = await \_unitOfWork.AddAsync(model);

if (action.WasSuccess)

{

return Ok(action.Result);

}

return BadRequest(action.Message);

}

[HttpPut]

public virtual async Task<IActionResult> PutAsync(T model)

{

var action = await \_unitOfWork.UpdateAsync(model);

if (action.WasSuccess)

{

return Ok(action.Result);

}

return BadRequest(action.Message);

}

[HttpDelete("{id}")]

public virtual async Task<IActionResult> DeleteAsync(int id)

{

var action = await \_unitOfWork.DeleteAsync(id);

if (action.WasSuccess)

{

return NoContent();

}

return BadRequest(action.Message);

}

}

}

1. Corra los test y verifique que todo está funcionando correctamente.
2. Verificamos la cobertura del código.
3. Hacemos commit.

#### Unidad de Trabajo

1. Adicione la clase **GenericUnitOfWorkTests**:

using Moq;

using Orders.Backend.Repositories.Interfaces;

using Orders.Backend.UnitsOfWork.Implementations;

using Orders.Shared.DTOs;

using Orders.Shared.Responses;

namespace Orders.Tests.UnitsOfWork

{

[TestClass]

public class GenericUnitOfWorkTests

{

private Mock<IGenericRepository<object>> \_mockRepository = null!;

private GenericUnitOfWork<object> \_unitOfWork = null!;

private object \_testModel = null!;

private int \_testId;

private PaginationDTO \_paginationDTO = null!;

[TestInitialize]

public void Initialize()

{

\_mockRepository = new Mock<IGenericRepository<object>>();

\_unitOfWork = new GenericUnitOfWork<object>(\_mockRepository.Object);

\_testModel = new object();

\_testId = 1;

\_paginationDTO = new PaginationDTO();

}

[TestMethod]

public async Task AddAsync\_Success()

{

\_mockRepository.Setup(x => x.AddAsync(It.IsAny<object>()))

.ReturnsAsync(new ActionResponse<object> { Result = \_testModel });

var result = await \_unitOfWork.AddAsync(\_testModel);

Assert.IsNotNull(result);

Assert.AreEqual(\_testModel, result.Result);

}

[TestMethod]

public async Task DeleteAsync\_Success()

{

\_mockRepository.Setup(x => x.DeleteAsync(It.IsAny<int>()))

.ReturnsAsync(new ActionResponse<object> { Result = \_testModel });

var result = await \_unitOfWork.DeleteAsync(\_testId);

Assert.IsNotNull(result);

Assert.AreEqual(\_testModel, result.Result);

}

[TestMethod]

public async Task GetAsync\_Pagination\_Success()

{

\_mockRepository.Setup(x => x.GetAsync(It.IsAny<PaginationDTO>()))

.ReturnsAsync(new ActionResponse<IEnumerable<object>> { Result = new List<object> { \_testModel } });

var result = await \_unitOfWork.GetAsync(\_paginationDTO);

Assert.IsNotNull(result);

Assert.AreEqual(1, result.Result!.Count());

}

[TestMethod]

public async Task GetTotalPagesAsync\_Success()

{

\_mockRepository.Setup(x => x.GetTotalPagesAsync(It.IsAny<PaginationDTO>()))

.ReturnsAsync(new ActionResponse<int> { Result = 5 });

var result = await \_unitOfWork.GetTotalPagesAsync(\_paginationDTO);

Assert.IsNotNull(result);

Assert.AreEqual(5, result.Result);

}

[TestMethod]

public async Task GetAsync\_Id\_Success()

{

\_mockRepository.Setup(x => x.GetAsync(It.IsAny<int>()))

.ReturnsAsync(new ActionResponse<object> { Result = \_testModel });

var result = await \_unitOfWork.GetAsync(\_testId);

Assert.IsNotNull(result);

Assert.AreEqual(\_testModel, result.Result);

}

[TestMethod]

public async Task GetAsync\_Success()

{

\_mockRepository.Setup(x => x.GetAsync())

.ReturnsAsync(new ActionResponse<IEnumerable<object>> { Result = new List<object> { \_testModel } });

var result = await \_unitOfWork.GetAsync();

Assert.IsNotNull(result);

}

[TestMethod]

public async Task UpdateAsync\_Success()

{

\_mockRepository.Setup(x => x.UpdateAsync(It.IsAny<object>()))

.ReturnsAsync(new ActionResponse<object> { Result = \_testModel });

var result = await \_unitOfWork.UpdateAsync(\_testModel);

Assert.IsNotNull(result);

Assert.AreEqual(\_testModel, result.Result);

}

}

}

1. Corra los test y verifique que todo está funcionando correctamente.
2. Verificamos la cobertura del código.
3. Hacemos commit.

#### Repositorio

1. Cree la carpeta **Shared** y dentro de esta, adicione la clase **ExceptionalDataContext**:

using Microsoft.EntityFrameworkCore;

using Orders.Backend.Data;

namespace Orders.Tests.Shared

{

public class ExceptionalDataContext : DataContext

{

public ExceptionalDataContext(DbContextOptions<DataContext> options)

: base(options)

{ }

public override Task<int> SaveChangesAsync(CancellationToken cancellationToken = default)

{

throw new InvalidOperationException("Test Exception");

}

}

}

1. Adicione la clase **ExceptionalDBUpdateDataContext**:

using Microsoft.EntityFrameworkCore;

using Orders.Backend.Data;

namespace Orders.Tests.Shared

{

public class ExceptionalDBUpdateDataContext : DataContext

{

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

{

}

public override Task<int> SaveChangesAsync(CancellationToken cancellationToken = default)

{

throw new DbUpdateException("Test Exception");

}

}

}

1. Adicione la clase **ExceptionalDBUpdateDataContextWithInnerException**:

using Microsoft.EntityFrameworkCore;

using Orders.Backend.Data;

namespace Orders.Tests.Shared

{

public class ExceptionalDBUpdateDataContextWithInnerException : DataContext

{

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

{

}

public override Task<int> SaveChangesAsync(CancellationToken cancellationToken = default)

{

var innerException = new Exception("duplicate record");

throw new DbUpdateException("Test Exception", innerException);

}

}

}

1. Adicione la clase **GenericRepositoryTests**:

using Microsoft.EntityFrameworkCore;

using Orders.Backend.Data;

using Orders.Backend.Repositories.Implementations;

using Orders.Shared.DTOs;

using Orders.Shared.Entities;

using Orders.Tests.Shared;

namespace Orders.Tests.Repositories

{

[TestClass]

public class GenericRepositoryTests

{

private DataContext \_context = null!;

private DbContextOptions<DataContext> \_options = null!;

private GenericRepository<Category> \_repository = null!;

[TestInitialize]

public void Initialize()

{

\_options = new DbContextOptionsBuilder<DataContext>()

.UseInMemoryDatabase(Guid.NewGuid().ToString())

.Options;

\_context = new DataContext(\_options);

\_repository = new GenericRepository<Category>(\_context);

}

[TestCleanup]

public void Cleanup()

{

\_context.Database.EnsureDeleted();

\_context.Dispose();

}

[TestMethod]

public async Task AddAsync\_ShouldAddEntity()

{

// Arrange

var testEntity = new Category { Name = "Test" };

// Act

var response = await \_repository.AddAsync(testEntity);

// Assert

Assert.IsTrue(response.WasSuccess);

Assert.IsNotNull(response.Result);

Assert.AreEqual("Test", response.Result.Name);

}

[TestMethod]

public async Task AddAsync\_GeneralExceptionThrown\_ReturnsError()

{

// Arrange

var exceptionalContext = new ExceptionalDataContext(\_options);

var repository = new GenericRepository<Category>(exceptionalContext);

var testEntity = new Category { Name = "Test" };

// Act

var response = await repository.AddAsync(testEntity);

// Assert

Assert.IsFalse(response.WasSuccess);

Assert.AreEqual("Test Exception", response.Message);

}

[TestMethod]

public async Task AddAsync\_DbUpdateExceptionThrown\_ReturnsError()

{

// Arrange

var exceptionalContext = new ExceptionalDBUpdateDataContext(\_options);

var repository = new GenericRepository<Category>(exceptionalContext);

var testEntity = new Category { Name = "Test" };

// Act

var response = await repository.AddAsync(testEntity);

// Assert

Assert.IsFalse(response.WasSuccess);

Assert.AreEqual("Test Exception", response.Message);

}

[TestMethod]

public async Task AddAsync\_DuplicateExceptionThrown\_ReturnsError()

{

// Arrange

var exceptionalContext = new ExceptionalDBUpdateDataContextWithInnerException(\_options);

var repository = new GenericRepository<Category>(exceptionalContext);

var testEntity = new Category { Name = "Test" };

// Act

var response = await repository.AddAsync(testEntity);

// Assert

Assert.IsFalse(response.WasSuccess);

Assert.AreEqual("Ya existe el registro que estas intentando crear.", response.Message);

}

[TestMethod]

public async Task DeleteAsync\_DbUpdateExceptionThrown\_ReturnsError()

{

// Arrange

var category = new Category { Id = 1, Name = "Test" };

await \_context.Set<Category>().AddAsync(category);

var product = new Product { Id = 1, Name = "Test", Description = "Test" };

await \_context.Set<Product>().AddAsync(product);

var productCategory = new ProductCategory { Category = category, Product = product };

await \_context.Set<ProductCategory>().AddAsync(productCategory);

await \_context.SaveChangesAsync();

// Act

var response = await \_repository.DeleteAsync(category.Id);

// Assert

Assert.IsFalse(response.WasSuccess);

}

[TestMethod]

public async Task DeleteAsync\_ShouldDeleteEntity()

{

// Arrange

var testEntity = new Category { Name = "Test" };

await \_context.Set<Category>().AddAsync(testEntity);

await \_context.SaveChangesAsync();

// Act

var response = await \_repository.DeleteAsync(testEntity.Id);

// Assert

Assert.IsTrue(response.WasSuccess);

}

[TestMethod]

public async Task DeleteAsync\_EntityNotFound\_ShouldReturnErrorActionResponse()

{

// Act

var response = await \_repository.DeleteAsync(1);

// Assert

Assert.IsFalse(response.WasSuccess);

Assert.AreEqual("Registro no encontrado", response.Message);

}

[TestMethod]

public async Task GetAsync\_ById\_ShouldReturnEntity()

{

// Arrange

var testEntity = new Category { Name = "Test" };

await \_context.Set<Category>().AddAsync(testEntity);

await \_context.SaveChangesAsync();

// Act

var response = await \_repository.GetAsync(testEntity.Id);

// Assert

Assert.IsTrue(response.WasSuccess);

Assert.IsNotNull(response.Result);

Assert.AreEqual("Test", response.Result.Name);

}

[TestMethod]

public async Task GetAsync\_ById\_EntityNotFound\_ShouldReturnErrorActionResponse()

{

// Act

var response = await \_repository.GetAsync(1);

// Assert

Assert.IsFalse(response.WasSuccess);

Assert.AreEqual("Registro no encontrado", response.Message);

}

[TestMethod]

public async Task GetAsync\_Pagination\_ShouldReturnEntities()

{

// Arrange

await \_context.Set<Category>().AddRangeAsync(new List<Category>

{

new Category { Name = "Test1" },

new Category { Name = "Test2" },

new Category { Name = "Test3" },

});

await \_context.SaveChangesAsync();

// Act

var paginationDTO = new PaginationDTO { RecordsNumber = 2 };

var response = await \_repository.GetAsync(paginationDTO);

// Assert

Assert.IsTrue(response.WasSuccess);

Assert.IsNotNull(response.Result);

Assert.AreEqual(2, response.Result.Count());

}

[TestMethod]

public async Task GetAsync\_ShouldReturnEntities()

{

// Arrange

await \_context.Set<Category>().AddRangeAsync(new List<Category>

{

new Category { Name = "Test1" },

new Category { Name = "Test2" },

new Category { Name = "Test3" },

});

await \_context.SaveChangesAsync();

// Act

var response = await \_repository.GetAsync();

// Assert

Assert.IsTrue(response.WasSuccess);

Assert.IsNotNull(response.Result);

Assert.AreEqual(3, response.Result.Count());

}

[TestMethod]

public async Task GetTotalPagesAsync\_ShouldReturnTotalPages()

{

// Arrange

await \_context.Set<Category>().AddRangeAsync(new List<Category>

{

new Category { Name = "Test1" },

new Category { Name = "Test2" },

new Category { Name = "Test3" },

});

await \_context.SaveChangesAsync();

var paginationDTO = new PaginationDTO { RecordsNumber = 2 };

// Act

var response = await \_repository.GetTotalPagesAsync(paginationDTO);

// Assert

Assert.IsTrue(response.WasSuccess);

Assert.AreEqual(2, response.Result);

}

[TestMethod]

public async Task UpdateAsync\_ShouldUpdateEntity()

{

// Arrange

var testEntity = new Category { Name = "Test" };

await \_context.Set<Category>().AddAsync(testEntity);

await \_context.SaveChangesAsync();

testEntity.Name = "UpdatedTest";

// Act

var response = await \_repository.UpdateAsync(testEntity);

// Assert

Assert.IsTrue(response.WasSuccess);

Assert.IsNotNull(response.Result);

Assert.AreEqual("UpdatedTest", response.Result.Name);

}

[TestMethod]

public async Task UpdateAsync\_GeneralExceptionThrown\_ReturnsError()

{

// Arrange

var exceptionalContext = new ExceptionalDataContext(\_options);

var testEntity = new Category { Name = "Test" };

await exceptionalContext.Set<Category>().AddAsync(testEntity);

exceptionalContext.SaveChanges();

var repository = new GenericRepository<Category>(exceptionalContext);

testEntity.Name = "UpdatedTest";

// Act

var response = await repository.UpdateAsync(testEntity);

// Assert

Assert.IsFalse(response.WasSuccess);

Assert.AreEqual("Test Exception", response.Message);

}

[TestMethod]

public async Task UpdateAsync\_DbUpdateExceptionThrown\_ReturnsError()

{

// Arrange

var exceptionalContext = new ExceptionalDBUpdateDataContext(\_options);

var testEntity = new Category { Name = "Test" };

await exceptionalContext.Set<Category>().AddAsync(testEntity);

exceptionalContext.SaveChanges();

var repository = new GenericRepository<Category>(exceptionalContext);

testEntity.Name = "UpdatedTest";

// Act

var response = await repository.UpdateAsync(testEntity);

// Assert

Assert.IsFalse(response.WasSuccess);

Assert.AreEqual("Test Exception", response.Message);

}

[TestMethod]

public async Task UpdateAsync\_DuplicateExceptionThrown\_ReturnsError()

{

// Arrange

var exceptionalContext = new ExceptionalDBUpdateDataContextWithInnerException(\_options);

var testEntity = new Category { Name = "Test" };

await exceptionalContext.Set<Category>().AddAsync(testEntity);

exceptionalContext.SaveChanges();

var repository = new GenericRepository<Category>(exceptionalContext);

testEntity.Name = "UpdatedTest";

// Act

var response = await repository.UpdateAsync(testEntity);

// Assert

Assert.IsFalse(response.WasSuccess);

Assert.AreEqual("Ya existe el registro que estas intentando crear.", response.Message);

}

}

}

1. Corra los test y verifique que todo está funcionando correctamente.
2. Verificamos la cobertura del código.
3. Hacemos commit.

### Paises

#### Controlador

1. Adicione la clase **CountriesControllerTests**:

using Microsoft.AspNetCore.Mvc;

using Moq;

using Orders.Backend.Controllers;

using Orders.Backend.UnitsOfWork.Interfaces;

using Orders.Shared.DTOs;

using Orders.Shared.Entities;

using Orders.Shared.Responses;

namespace Orders.Tests.Controllers

{

[TestClass]

public class CountriesControllerTests

{

private Mock<IGenericUnitOfWork<Country>> \_mockGenericUnitOfWork = null!;

private Mock<ICountriesUnitOfWork> \_mockCountriesUnitOfWork = null!;

private CountriesController \_controller = null!;

[TestInitialize]

public void Initialize()

{

\_mockGenericUnitOfWork = new Mock<IGenericUnitOfWork<Country>>();

\_mockCountriesUnitOfWork = new Mock<ICountriesUnitOfWork>();

\_controller = new CountriesController(\_mockGenericUnitOfWork.Object, \_mockCountriesUnitOfWork.Object);

}

[TestMethod]

public async Task GetComboAsync\_ShouldReturnOk()

{

// Arrange

var response = new List<Country> { new Country { Id = 1, Name = "Country" } };

\_mockCountriesUnitOfWork.Setup(x => x.GetComboAsync())

.ReturnsAsync(response);

// Act

var result = await \_controller.GetComboAsync();

// Assert

Assert.IsInstanceOfType(result, typeof(OkObjectResult));

var okResult = result as OkObjectResult;

Assert.AreEqual(response, okResult?.Value);

\_mockCountriesUnitOfWork.Verify(x => x.GetComboAsync(), Times.Once());

}

[TestMethod]

public async Task GetAsync\_Pagination\_ShouldReturnOk()

{

// Arrange

var pagination = new PaginationDTO();

var countries = new List<Country>

{

new Country { Id = 1, Name = "Country1" },

new Country { Id = 2, Name = "Country2" }

};

var response = new ActionResponse<IEnumerable<Country>> { WasSuccess = true, Result = countries };

\_mockCountriesUnitOfWork.Setup(x => x.GetAsync(pagination))

.ReturnsAsync(response);

// Act

var result = await \_controller.GetAsync(pagination);

// Assert

Assert.IsInstanceOfType(result, typeof(OkObjectResult));

var okResult = result as OkObjectResult;

Assert.AreEqual(countries, okResult?.Value);

\_mockCountriesUnitOfWork.Verify(x => x.GetAsync(pagination), Times.Once());

}

[TestMethod]

public async Task GetAsync\_ShouldReturnOk()

{

// Arrange

var countries = new List<Country>

{

new Country { Id = 1, Name = "Country1" },

new Country { Id = 2, Name = "Country2" }

};

var response = new ActionResponse<IEnumerable<Country>> { WasSuccess = true, Result = countries };

\_mockCountriesUnitOfWork.Setup(x => x.GetAsync())

.ReturnsAsync(response);

// Act

var result = await \_controller.GetAsync();

// Assert

Assert.IsInstanceOfType(result, typeof(OkObjectResult));

var okResult = result as OkObjectResult;

Assert.AreEqual(countries, okResult?.Value);

\_mockCountriesUnitOfWork.Verify(x => x.GetAsync(), Times.Once());

}

[TestMethod]

public async Task GetAsync\_ShouldReturnError()

{

// Arrange

var response = new ActionResponse<IEnumerable<Country>> { WasSuccess = false };

\_mockCountriesUnitOfWork.Setup(x => x.GetAsync())

.ReturnsAsync(response);

// Act

var result = await \_controller.GetAsync();

// Assert

Assert.IsInstanceOfType(result, typeof(BadRequestResult));

\_mockCountriesUnitOfWork.Verify(x => x.GetAsync(), Times.Once());

}

[TestMethod]

public async Task GetAsync\_Pagination\_ShouldReturnBadRequest()

{

// Arrange

var pagination = new PaginationDTO();

var countries = new List<Country>

{

new Country { Id = 1, Name = "Country1" },

new Country { Id = 2, Name = "Country2" }

};

var response = new ActionResponse<IEnumerable<Country>> { WasSuccess = false, Result = countries };

\_mockCountriesUnitOfWork.Setup(x => x.GetAsync(pagination))

.ReturnsAsync(response);

// Act

var result = await \_controller.GetAsync(pagination);

// Assert

Assert.IsInstanceOfType(result, typeof(BadRequestResult));

\_mockCountriesUnitOfWork.Verify(x => x.GetAsync(pagination), Times.Once());

}

[TestMethod]

public async Task GetPagesAsync\_ShouldReturnOk()

{

// Arrange

var pagination = new PaginationDTO();

var totalPages = 5;

var response = new ActionResponse<int> { WasSuccess = true, Result = totalPages };

\_mockCountriesUnitOfWork.Setup(x => x.GetTotalPagesAsync(pagination)).ReturnsAsync(response);

// Act

var result = await \_controller.GetPagesAsync(pagination);

// Assert

Assert.IsInstanceOfType(result, typeof(OkObjectResult));

var okResult = result as OkObjectResult;

Assert.AreEqual(totalPages, okResult?.Value);

\_mockCountriesUnitOfWork.Verify(x => x.GetTotalPagesAsync(pagination), Times.Once());

}

[TestMethod]

public async Task GetPagesAsync\_ShouldReturnBadRequest()

{

// Arrange

var pagination = new PaginationDTO();

var totalPages = 5;

var response = new ActionResponse<int> { WasSuccess = false };

\_mockCountriesUnitOfWork.Setup(x => x.GetTotalPagesAsync(pagination)).ReturnsAsync(response);

// Act

var result = await \_controller.GetPagesAsync(pagination);

// Assert

Assert.IsInstanceOfType(result, typeof(BadRequestResult));

\_mockCountriesUnitOfWork.Verify(x => x.GetTotalPagesAsync(pagination), Times.Once());

}

[TestMethod]

public async Task GetAsync\_ById\_ShouldReturnOk()

{

// Arrange

var countryId = 1;

var country = new Country { Id = countryId, Name = "Country1" };

var response = new ActionResponse<Country> { WasSuccess = true, Result = country };

\_mockCountriesUnitOfWork.Setup(x => x.GetAsync(countryId)).ReturnsAsync(response);

// Act

var result = await \_controller.GetAsync(countryId);

// Assert

Assert.IsInstanceOfType(result, typeof(OkObjectResult));

var okResult = result as OkObjectResult;

Assert.AreEqual(country, okResult?.Value);

\_mockCountriesUnitOfWork.Verify(x => x.GetAsync(countryId), Times.Once());

}

[TestMethod]

public async Task GetAsync\_ById\_ShouldReturnNotFound()

{

// Arrange

var countryId = 1;

var response = new ActionResponse<Country> { WasSuccess = false, Message = "Not Found" };

\_mockCountriesUnitOfWork.Setup(x => x.GetAsync(countryId)).ReturnsAsync(response);

// Act

var result = await \_controller.GetAsync(countryId);

// Assert

Assert.IsInstanceOfType(result, typeof(NotFoundObjectResult));

var notFoundResult = result as NotFoundObjectResult;

Assert.AreEqual("Not Found", notFoundResult?.Value);

\_mockCountriesUnitOfWork.Verify(x => x.GetAsync(countryId), Times.Once());

}

}

}

1. Corra los test y verifique que todo está funcionando correctamente.
2. Verificamos la cobertura del código.
3. Hacemos commit.

#### Unidad de Trabajo

1. Adicione la clase **CountriesUnitOfWorkTests**:

using Moq;

using Orders.Backend.Repositories.Interfaces;

using Orders.Backend.UnitsOfWork.Implementations;

using Orders.Shared.DTOs;

using Orders.Shared.Entities;

using Orders.Shared.Responses;

namespace Orders.Tests.UnitsOfWork

{

[TestClass]

public class CountriesUnitOfWorkTests

{

private Mock<IGenericRepository<Country>> \_mockGenericRepository = null!;

private Mock<ICountriesRepository> \_mockCountriesRepository = null!;

private CountriesUnitOfWork \_unitOfWork = null!;

[TestInitialize]

public void Initialize()

{

\_mockGenericRepository = new Mock<IGenericRepository<Country>>();

\_mockCountriesRepository = new Mock<ICountriesRepository>();

\_unitOfWork = new CountriesUnitOfWork(\_mockGenericRepository.Object, \_mockCountriesRepository.Object);

}

[TestMethod]

public async Task GetAsync\_WithPagination\_ShouldReturnData()

{

// Arrange

var pagination = new PaginationDTO();

var expectedResponse = new ActionResponse<IEnumerable<Country>> { WasSuccess = true };

\_mockCountriesRepository.Setup(x => x.GetAsync(pagination))

.ReturnsAsync(expectedResponse);

// Act

var result = await \_unitOfWork.GetAsync(pagination);

// Assert

Assert.AreEqual(expectedResponse, result);

\_mockCountriesRepository.Verify(x => x.GetAsync(pagination), Times.Once);

}

[TestMethod]

public async Task GetAsync\_ShouldReturnData()

{

// Arrange

var expectedResponse = new ActionResponse<IEnumerable<Country>> { WasSuccess = true };

\_mockCountriesRepository.Setup(x => x.GetAsync())

.ReturnsAsync(expectedResponse);

// Act

var result = await \_unitOfWork.GetAsync();

// Assert

Assert.AreEqual(expectedResponse, result);

\_mockCountriesRepository.Verify(x => x.GetAsync(), Times.Once);

}

[TestMethod]

public async Task GetTotalPagesAsync\_ShouldReturnTotalPages()

{

// Arrange

var pagination = new PaginationDTO();

var expectedResponse = new ActionResponse<int> { WasSuccess = true };

\_mockCountriesRepository.Setup(x => x.GetTotalPagesAsync(pagination))

.ReturnsAsync(expectedResponse);

// Act

var result = await \_unitOfWork.GetTotalPagesAsync(pagination);

// Assert

Assert.AreEqual(expectedResponse, result);

\_mockCountriesRepository.Verify(x => x.GetTotalPagesAsync(pagination), Times.Once);

}

[TestMethod]

public async Task GetAsync\_WithId\_ShouldReturnData()

{

// Arrange

int id = 1;

var expectedResponse = new ActionResponse<Country> { WasSuccess = true };

\_mockCountriesRepository.Setup(x => x.GetAsync(id))

.ReturnsAsync(expectedResponse);

// Act

var result = await \_unitOfWork.GetAsync(id);

// Assert

Assert.AreEqual(expectedResponse, result);

\_mockCountriesRepository.Verify(x => x.GetAsync(id), Times.Once);

}

[TestMethod]

public async Task GetComboAsync\_ShouldReturnData()

{

// Arrange

var expectedCountries = new List<Country> { new Country { Id = 1, Name = "Country1" } };

\_mockCountriesRepository.Setup(x => x.GetComboAsync())

.ReturnsAsync(expectedCountries);

// Act

var result = await \_unitOfWork.GetComboAsync();

// Assert

CollectionAssert.AreEqual(expectedCountries, new List<Country>(result));

\_mockCountriesRepository.Verify(x => x.GetComboAsync(), Times.Once);

}

}

}

1. Corra los test y verifique que todo está funcionando correctamente.
2. Verificamos la cobertura del código.
3. Hacemos commit.

#### Repositorio

1. Adicione la clase **CountriesRepositoryTests**:

using Microsoft.EntityFrameworkCore;

using Orders.Backend.Data;

using Orders.Backend.Repositories.Implementations;

using Orders.Shared.DTOs;

using Orders.Shared.Entities;

namespace Orders.Tests.Repositories

{

[TestClass]

public class CountriesRepositoryTests

{

private DataContext \_context = null!;

private CountriesRepository \_repository = null!;

[TestInitialize]

public void Initialize()

{

var options = new DbContextOptionsBuilder<DataContext>()

.UseInMemoryDatabase(databaseName: Guid.NewGuid().ToString())

.Options;

\_context = new DataContext(options);

\_repository = new CountriesRepository(\_context);

SeedDatabase();

}

private void SeedDatabase()

{

var countries = new[]

{

new Country { Id = 1, Name = "USA" },

new Country { Id = 2, Name = "Canada" },

new Country { Id = 3, Name = "Mexico" },

};

\_context.Countries.AddRange(countries);

\_context.SaveChanges();

}

[TestMethod]

public async Task GetAsync\_Pagination\_ShouldReturnPaginatedCountries()

{

// Arrange

var pagination = new PaginationDTO { Page = 1, RecordsNumber = 2, Filter = "USA" };

// Act

var response = await \_repository.GetAsync(pagination);

// Assert

Assert.IsTrue(response.WasSuccess);

Assert.AreEqual(1, response!.Result!.Count());

}

[TestMethod]

public async Task GetAsync\_\_ShouldReturnCountries()

{

// Act

var response = await \_repository.GetAsync();

// Assert

Assert.IsTrue(response.WasSuccess);

Assert.AreEqual(3, response!.Result!.Count());

}

[TestMethod]

public async Task GetTotalPagesAsync\_ShouldReturnTotalPages()

{

// Arrange

var pagination = new PaginationDTO { RecordsNumber = 2, Filter = "Mexico" };

// Act

var response = await \_repository.GetTotalPagesAsync(pagination);

// Assert

Assert.IsTrue(response.WasSuccess);

Assert.AreEqual(1, response.Result);

}

[TestMethod]

public async Task GetAsync\_ById\_ShouldReturnCountry()

{

// Arrange

var countryId = 1;

// Act

var response = await \_repository.GetAsync(countryId);

// Assert

Assert.IsTrue(response.WasSuccess);

Assert.IsNotNull(response.Result);

Assert.AreEqual("USA", response.Result.Name);

}

[TestMethod]

public async Task GetAsync\_ById\_ShouldReturnNotFoundForInvalidId()

{

// Arrange

var countryId = 10;

// Act

var response = await \_repository.GetAsync(countryId);

// Assert

Assert.IsFalse(response.WasSuccess);

Assert.IsNull(response.Result);

Assert.AreEqual("País no existe", response.Message);

}

[TestMethod]

public async Task GetComboAsync\_ShouldReturnAllCountries()

{

// Act

var countries = await \_repository.GetComboAsync();

// Assert

Assert.AreEqual(3, countries.Count());

}

[TestCleanup]

public void Cleanup()

{

\_context.Database.EnsureDeleted();

\_context.Dispose();

}

}

}

1. Corra los test y verifique que todo está funcionando correctamente.
2. Verificamos la cobertura del código.
3. Hacemos commit.

### Estados / Departamentos

#### Controlador

1. Adicione la clase **StatesControllerTests**:

using Microsoft.AspNetCore.Mvc;

using Moq;

using Orders.Backend.Controllers;

using Orders.Backend.UnitsOfWork.Interfaces;

using Orders.Shared.DTOs;

using Orders.Shared.Entities;

using Orders.Shared.Responses;

namespace Orders.Tests.Controllers

{

[TestClass]

public class StatesControllerTests

{

private Mock<IGenericUnitOfWork<State>> \_mockUnitOfWork = null!;

private Mock<IStatesUnitOfWork> \_mockStatesUnitOfWork = null!;

private StatesController \_controller = null!;

[TestInitialize]

public void Initialize()

{

\_mockUnitOfWork = new Mock<IGenericUnitOfWork<State>>();

\_mockStatesUnitOfWork = new Mock<IStatesUnitOfWork>();

\_controller = new StatesController(\_mockUnitOfWork.Object, \_mockStatesUnitOfWork.Object);

}

[TestMethod]

public async Task GetComboAsync\_ShouldReturnOk()

{

// Arrange

var countryId = 1;

var states = new List<State> { new State(), new State() };

\_mockStatesUnitOfWork.Setup(x => x.GetComboAsync(countryId)).ReturnsAsync(states);

// Act

var result = await \_controller.GetComboAsync(countryId);

// Assert

Assert.IsInstanceOfType(result, typeof(OkObjectResult));

var okResult = (OkObjectResult)result;

Assert.AreEqual(states, okResult.Value);

\_mockStatesUnitOfWork.Verify(x => x.GetComboAsync(countryId), Times.Once());

}

[TestMethod]

public async Task GetAsync\_Paginated\_ShouldReturnOk()

{

// Arrange

var pagination = new PaginationDTO();

var states = new List<State> { new State(), new State() };

\_mockStatesUnitOfWork.Setup(x => x.GetAsync(pagination))

.ReturnsAsync(new ActionResponse<IEnumerable<State>>

{

WasSuccess = true,

Result = states

});

// Act

var result = await \_controller.GetAsync(pagination);

// Assert

Assert.IsInstanceOfType(result, typeof(OkObjectResult));

var okResult = (OkObjectResult)result;

Assert.AreEqual(states, okResult.Value);

\_mockStatesUnitOfWork.Verify(x => x.GetAsync(pagination), Times.Once());

}

[TestMethod]

public async Task GetAsync\_ShouldReturnOk()

{

// Arrange

var states = new List<State> { new State(), new State() };

\_mockStatesUnitOfWork.Setup(x => x.GetAsync())

.ReturnsAsync(new ActionResponse<IEnumerable<State>>

{

WasSuccess = true,

Result = states

});

// Act

var result = await \_controller.GetAsync();

// Assert

Assert.IsInstanceOfType(result, typeof(OkObjectResult));

var okResult = (OkObjectResult)result;

Assert.AreEqual(states, okResult.Value);

\_mockStatesUnitOfWork.Verify(x => x.GetAsync(), Times.Once());

}

[TestMethod]

public async Task GetAsync\_ShouldReturnError()

{

// Arrange

\_mockStatesUnitOfWork.Setup(x => x.GetAsync())

.ReturnsAsync(new ActionResponse<IEnumerable<State>>

{

WasSuccess = false,

});

// Act

var result = await \_controller.GetAsync();

// Assert

Assert.IsInstanceOfType(result, typeof(BadRequestResult));

\_mockStatesUnitOfWork.Verify(x => x.GetAsync(), Times.Once());

}

[TestMethod]

public async Task GetAsync\_ShouldReturnBadRequest()

{

// Arrange

var pagination = new PaginationDTO();

\_mockStatesUnitOfWork.Setup(x => x.GetAsync(pagination))

.ReturnsAsync(new ActionResponse<IEnumerable<State>> { WasSuccess = false });

// Act

var result = await \_controller.GetAsync(pagination);

// Assert

Assert.IsInstanceOfType(result, typeof(BadRequestResult));

\_mockStatesUnitOfWork.Verify(x => x.GetAsync(pagination), Times.Once());

}

[TestMethod]

public async Task GetPagesAsync\_ShouldReturnOk()

{

// Arrange

var pagination = new PaginationDTO();

var totalPages = 5;

\_mockStatesUnitOfWork.Setup(x => x.GetTotalPagesAsync(pagination))

.ReturnsAsync(new ActionResponse<int>

{

WasSuccess = true,

Result = totalPages

});

// Act

var result = await \_controller.GetPagesAsync(pagination);

// Assert

Assert.IsInstanceOfType(result, typeof(OkObjectResult));

var okResult = (OkObjectResult)result;

Assert.AreEqual(totalPages, okResult.Value);

\_mockStatesUnitOfWork.Verify(x => x.GetTotalPagesAsync(pagination), Times.Once());

}

[TestMethod]

public async Task GetPagesAsync\_ShouldReturnBadRequest()

{

// Arrange

var pagination = new PaginationDTO();

\_mockStatesUnitOfWork.Setup(x => x.GetTotalPagesAsync(pagination))

.ReturnsAsync(new ActionResponse<int> { WasSuccess = false });

// Act

var result = await \_controller.GetPagesAsync(pagination);

// Assert

Assert.IsInstanceOfType(result, typeof(BadRequestResult));

\_mockStatesUnitOfWork.Verify(x => x.GetTotalPagesAsync(pagination), Times.Once());

}

[TestMethod]

public async Task GetAsync\_ById\_ShouldReturnOk()

{

// Arrange

var stateId = 1;

var state = new State();

\_mockStatesUnitOfWork.Setup(x => x.GetAsync(stateId))

.ReturnsAsync(new ActionResponse<State>

{

WasSuccess = true,

Result = state

});

// Act

var result = await \_controller.GetAsync(stateId);

// Assert

Assert.IsInstanceOfType(result, typeof(OkObjectResult));

var okResult = (OkObjectResult)result;

Assert.AreEqual(state, okResult.Value);

\_mockStatesUnitOfWork.Verify(x => x.GetAsync(stateId), Times.Once());

}

[TestMethod]

public async Task GetAsync\_ById\_ShouldReturnNotFound()

{

// Arrange

var stateId = 1;

var message = "State not found";

\_mockStatesUnitOfWork.Setup(x => x.GetAsync(stateId))

.ReturnsAsync(new ActionResponse<State>

{

WasSuccess = false,

Message = message

});

// Act

var result = await \_controller.GetAsync(stateId);

// Assert

Assert.IsInstanceOfType(result, typeof(NotFoundObjectResult));

var notFoundResult = (NotFoundObjectResult)result;

Assert.AreEqual(message, notFoundResult.Value);

\_mockStatesUnitOfWork.Verify(x => x.GetAsync(stateId), Times.Once());

}

}

}

1. Corra los test y verifique que todo está funcionando correctamente.
2. Verificamos la cobertura del código.
3. Hacemos commit.

#### Unidad de Trabajo

1. Adicione la clase **StatesUnitOfWorkTests**:

using Moq;

using Orders.Backend.Repositories.Interfaces;

using Orders.Backend.UnitsOfWork.Implementations;

using Orders.Shared.DTOs;

using Orders.Shared.Entities;

using Orders.Shared.Responses;

namespace Orders.Tests.UnitsOfWork

{

[TestClass]

public class StatesUnitOfWorkTests

{

private Mock<IGenericRepository<State>> \_mockGenericRepository = null!;

private Mock<IStatesRepository> \_mockStatesRepository = null!;

private StatesUnitOfWork \_unitOfWork = null!;

[TestInitialize]

public void Initialize()

{

\_mockGenericRepository = new Mock<IGenericRepository<State>>();

\_mockStatesRepository = new Mock<IStatesRepository>();

\_unitOfWork = new StatesUnitOfWork(\_mockGenericRepository.Object, \_mockStatesRepository.Object);

}

[TestMethod]

public async Task GetAsync\_Paginated\_ShouldReturnStates()

{

// Arrange

var pagination = new PaginationDTO();

var states = new List<State> { new State(), new State() };

\_mockStatesRepository.Setup(x => x.GetAsync(pagination))

.ReturnsAsync(new ActionResponse<IEnumerable<State>>

{

WasSuccess = true,

Result = states

});

// Act

var result = await \_unitOfWork.GetAsync(pagination);

// Assert

Assert.IsTrue(result.WasSuccess);

Assert.AreEqual(states, result.Result);

\_mockStatesRepository.Verify(x => x.GetAsync(pagination), Times.Once());

}

[TestMethod]

public async Task GetAsync\_ShouldReturnStates()

{

// Arrange

var states = new List<State> { new State(), new State() };

\_mockStatesRepository.Setup(x => x.GetAsync())

.ReturnsAsync(new ActionResponse<IEnumerable<State>>

{

WasSuccess = true,

Result = states

});

// Act

var result = await \_unitOfWork.GetAsync();

// Assert

Assert.IsTrue(result.WasSuccess);

Assert.AreEqual(states, result.Result);

\_mockStatesRepository.Verify(x => x.GetAsync(), Times.Once());

}

[TestMethod]

public async Task GetTotalPagesAsync\_ShouldReturnTotalPages()

{

// Arrange

var pagination = new PaginationDTO();

var totalPages = 5;

\_mockStatesRepository.Setup(x => x.GetTotalPagesAsync(pagination))

.ReturnsAsync(new ActionResponse<int>

{

WasSuccess = true,

Result = totalPages

});

// Act

var result = await \_unitOfWork.GetTotalPagesAsync(pagination);

// Assert

Assert.IsTrue(result.WasSuccess);

Assert.AreEqual(totalPages, result.Result);

\_mockStatesRepository.Verify(x => x.GetTotalPagesAsync(pagination), Times.Once());

}

[TestMethod]

public async Task GetAsync\_ById\_ShouldReturnState()

{

// Arrange

var stateId = 1;

var state = new State();

\_mockStatesRepository.Setup(x => x.GetAsync(stateId))

.ReturnsAsync(new ActionResponse<State>

{

WasSuccess = true,

Result = state

});

// Act

var result = await \_unitOfWork.GetAsync(stateId);

// Assert

Assert.IsTrue(result.WasSuccess);

Assert.AreEqual(state, result.Result);

\_mockStatesRepository.Verify(x => x.GetAsync(stateId), Times.Once());

}

[TestMethod]

public async Task GetComboAsync\_ShouldReturnStates()

{

// Arrange

var countryId = 1;

var states = new List<State> { new State(), new State() };

\_mockStatesRepository.Setup(x => x.GetComboAsync(countryId))

.ReturnsAsync(states);

// Act

var result = await \_unitOfWork.GetComboAsync(countryId);

// Assert

Assert.AreEqual(states, result);

\_mockStatesRepository.Verify(x => x.GetComboAsync(countryId), Times.Once());

}

}

}

1. Corra los test y verifique que todo está funcionando correctamente.
2. Verificamos la cobertura del código.
3. Hacemos commit.

#### Repositorio

1. Adicione la clase **StatesRepositoryTests**:

using Microsoft.EntityFrameworkCore;

using Orders.Backend.Data;

using Orders.Backend.Repositories.Implementations;

using Orders.Shared.DTOs;

using Orders.Shared.Entities;

namespace Orders.Tests.Repositories

{

[TestClass]

public class StatesRepositoryTests

{

private DataContext \_context = null!;

private StatesRepository \_repository = null!;

[TestInitialize]

public void Initialize()

{

var options = new DbContextOptionsBuilder<DataContext>()

.UseInMemoryDatabase(databaseName: "OrdersDb")

.Options;

\_context = new DataContext(options);

\_repository = new StatesRepository(\_context);

}

[TestMethod]

public async Task GetAsync\_ShouldReturnStates()

{

// Arrange

PopulateTestData();

// Act

var result = await \_repository.GetAsync();

// Assert

Assert.IsTrue(result.WasSuccess);

Assert.AreEqual(4, result.Result!.Count());

Assert.AreEqual("TestState1", result.Result!.First().Name);

Assert.AreEqual("TestState4", result.Result!.Last().Name);

}

[TestMethod]

public async Task GetAsync\_ShouldReturnFilteredAndPaginatedStates()

{

// Arrange

PopulateTestData();

var pagination = new PaginationDTO

{

Filter = "test",

RecordsNumber = 2,

Page = 1,

Id = 1

};

// Act

var result = await \_repository.GetAsync(pagination);

// Assert

Assert.IsTrue(result.WasSuccess);

Assert.AreEqual(2, result.Result!.Count());

Assert.AreEqual("TestState1", result.Result!.First().Name);

Assert.AreEqual("TestState2", result.Result!.Last().Name);

}

[TestMethod]

public async Task GetTotalPagesAsync\_ShouldReturnCorrectTotalPages()

{

// Arrange

PopulateTestData();

var pagination = new PaginationDTO

{

RecordsNumber = 2,

Id = 1,

Filter = "Test"

};

// Act

var result = await \_repository.GetTotalPagesAsync(pagination);

// Assert

Assert.IsTrue(result.WasSuccess);

Assert.AreEqual(2, result.Result);

}

[TestMethod]

public async Task GetAsync\_ById\_ShouldReturnState()

{

// Arrange

PopulateTestData();

var stateId = 1;

// Act

var result = await \_repository.GetAsync(stateId);

// Assert

Assert.IsTrue(result.WasSuccess);

Assert.AreEqual("TestState1", result.Result!.Name);

}

[TestMethod]

public async Task GetAsync\_ById\_ShouldReturnError()

{

// Arrange

PopulateTestData();

var stateId = 999;

// Act

var result = await \_repository.GetAsync(stateId);

// Assert

Assert.IsFalse(result.WasSuccess);

Assert.AreEqual("Estado no existe", result.Message);

}

[TestMethod]

public async Task GetComboAsync\_ShouldReturnStatesForCountry()

{

// Arrange

PopulateTestData();

var countryId = 1;

// Act

var result = await \_repository.GetComboAsync(countryId);

// Assert

Assert.AreEqual(4, result.Count());

}

private void PopulateTestData()

{

if (\_context.Countries.Any())

{

return;

}

var country = new Country { Id = 1, Name = "TestCountry" };

\_context.Countries.Add(country);

var states = new List<State>

{

new State { Id = 1, Name = "TestState1", Country = country },

new State { Id = 2, Name = "TestState2", Country = country },

new State { Id = 3, Name = "TestState3", Country = country },

new State { Id = 4, Name = "TestState4", Country = country }

};

\_context.States.AddRange(states);

\_context.SaveChanges();

}

[TestCleanup]

public void Cleanup()

{

\_context.Database.EnsureDeleted();

\_context.Dispose();

}

}

}

1. Corra los test y verifique que todo está funcionando correctamente.
2. Verificamos la cobertura del código.
3. Hacemos commit.

### Ciudades

#### Controlador

1. Adicione la clase **CitiesControllerTests**:

using Microsoft.AspNetCore.Mvc;

using Moq;

using Orders.Backend.Controllers;

using Orders.Backend.UnitsOfWork.Interfaces;

using Orders.Shared.DTOs;

using Orders.Shared.Entities;

using Orders.Shared.Responses;

namespace Orders.Tests.Controllers

{

[TestClass]

public class CitiesControllerTests

{

private Mock<IGenericUnitOfWork<City>> \_mockGenericUnitOfWork = null!;

private Mock<ICitiesUnitOfWork> \_mockCitiesUnitOfWork = null!;

private CitiesController \_controller = null!;

[TestInitialize]

public void Initialize()

{

\_mockGenericUnitOfWork = new Mock<IGenericUnitOfWork<City>>();

\_mockCitiesUnitOfWork = new Mock<ICitiesUnitOfWork>();

\_controller = new CitiesController(\_mockGenericUnitOfWork.Object, \_mockCitiesUnitOfWork.Object);

}

[TestMethod]

public async Task GetComboAsync\_ShouldReturnOkResult()

{

// Arrange

var stateId = 1;

var cities = new List<City> { new City { Id = 1, Name = "City1" }, new City { Id = 2, Name = "City2" } };

\_mockCitiesUnitOfWork.Setup(x => x.GetComboAsync(stateId)).ReturnsAsync(cities);

// Act

var result = await \_controller.GetComboAsync(stateId);

// Assert

var okResult = result as OkObjectResult;

Assert.IsNotNull(okResult);

var resultValue = okResult.Value as IEnumerable<City>;

Assert.IsNotNull(resultValue);

Assert.AreEqual(2, resultValue.Count());

new List<City> { new City { Id = 1, Name = "City1" }, new City { Id = 2, Name = "City2" } };

\_mockCitiesUnitOfWork.Verify(x => x.GetComboAsync(stateId), Times.Once());

}

[TestMethod]

public async Task GetAsync\_ShouldReturnOkResult\_WhenActionResponseIsSuccess()

{

// Arrange

var pagination = new PaginationDTO();

var response = new ActionResponse<IEnumerable<City>> { WasSuccess = true, Result = new List<City>() };

\_mockCitiesUnitOfWork.Setup(x => x.GetAsync(pagination)).ReturnsAsync(response);

// Act

var result = await \_controller.GetAsync(pagination);

// Assert

var okResult = result as OkObjectResult;

Assert.IsNotNull(okResult);

\_mockCitiesUnitOfWork.Verify(x => x.GetAsync(pagination), Times.Once());

}

[TestMethod]

public async Task GetAsync\_ShouldReturnBadRequest\_WhenActionResponseIsNotSuccess()

{

// Arrange

var pagination = new PaginationDTO();

var response = new ActionResponse<IEnumerable<City>> { WasSuccess = false };

\_mockCitiesUnitOfWork.Setup(x => x.GetAsync(pagination)).ReturnsAsync(response);

// Act

var result = await \_controller.GetAsync(pagination);

// Assert

var badRequestResult = result as BadRequestResult;

Assert.IsNotNull(badRequestResult);

\_mockCitiesUnitOfWork.Verify(x => x.GetAsync(pagination), Times.Once());

}

[TestMethod]

public async Task GetPagesAsync\_ShouldReturnOkResult\_WhenActionResponseIsSuccess()

{

// Arrange

var pagination = new PaginationDTO();

var response = new ActionResponse<int> { WasSuccess = true, Result = 1 };

\_mockCitiesUnitOfWork.Setup(x => x.GetTotalPagesAsync(pagination)).ReturnsAsync(response);

// Act

var result = await \_controller.GetPagesAsync(pagination);

// Assert

var okResult = result as OkObjectResult;

Assert.IsNotNull(okResult);

Assert.AreEqual(1, okResult.Value);

\_mockCitiesUnitOfWork.Verify(x => x.GetTotalPagesAsync(pagination), Times.Once());

}

[TestMethod]

public async Task GetPagesAsync\_ShouldReturnBadRequest\_WhenActionResponseIsNotSuccess()

{

// Arrange

var pagination = new PaginationDTO();

var response = new ActionResponse<int> { WasSuccess = false };

\_mockCitiesUnitOfWork.Setup(x => x.GetTotalPagesAsync(pagination)).ReturnsAsync(response);

// Act

var result = await \_controller.GetPagesAsync(pagination);

// Assert

var badRequestResult = result as BadRequestResult;

Assert.IsNotNull(badRequestResult);

\_mockCitiesUnitOfWork.Verify(x => x.GetTotalPagesAsync(pagination), Times.Once());

}

}

}

1. Corra los test y verifique que todo está funcionando correctamente.
2. Verificamos la cobertura del código.
3. Hacemos commit.

#### Unidad de Trabajo

1. Adicione la clase **CitiesUnitOfWorkTests**:

using Moq;

using Orders.Backend.Repositories.Interfaces;

using Orders.Backend.UnitsOfWork.Implementations;

using Orders.Shared.DTOs;

using Orders.Shared.Entities;

using Orders.Shared.Responses;

namespace Orders.Tests.UnitsOfWork

{

[TestClass]

public class CitiesUnitOfWorkTests

{

private Mock<ICitiesRepository> \_mockCitiesRepository = null!;

private CitiesUnitOfWork \_unitOfWork = null!;

[TestInitialize]

public void Initialize()

{

\_mockCitiesRepository = new Mock<ICitiesRepository>();

\_unitOfWork = new CitiesUnitOfWork(null, \_mockCitiesRepository.Object);

}

[TestMethod]

public async Task GetAsync\_ShouldReturnCities()

{

// Arrange

var pagination = new PaginationDTO();

var expectedActionResponse = new ActionResponse<IEnumerable<City>> { WasSuccess = true, Result = new List<City>() };

\_mockCitiesRepository.Setup(x => x.GetAsync(pagination))

.ReturnsAsync(expectedActionResponse);

// Act

var result = await \_unitOfWork.GetAsync(pagination);

// Assert

Assert.IsTrue(result.WasSuccess);

Assert.AreEqual(expectedActionResponse.Result, result.Result);

\_mockCitiesRepository.Verify(x => x.GetAsync(pagination), Times.Once);

}

[TestMethod]

public async Task GetTotalPagesAsync\_ShouldReturnTotalPages()

{

// Arrange

var pagination = new PaginationDTO();

var expectedActionResponse = new ActionResponse<int> { WasSuccess = true, Result = 5 };

\_mockCitiesRepository.Setup(x => x.GetTotalPagesAsync(pagination))

.ReturnsAsync(expectedActionResponse);

// Act

var result = await \_unitOfWork.GetTotalPagesAsync(pagination);

// Assert

Assert.IsTrue(result.WasSuccess);

Assert.AreEqual(expectedActionResponse.Result, result.Result);

\_mockCitiesRepository.Verify(x => x.GetTotalPagesAsync(pagination), Times.Once);

}

[TestMethod]

public async Task GetComboAsync\_ShouldReturnCities()

{

// Arrange

var stateId = 1;

var expectedCities = new List<City> { new City { Id = 1, Name = "City1" }, new City { Id = 2, Name = "City2" } };

\_mockCitiesRepository.Setup(x => x.GetComboAsync(stateId))

.ReturnsAsync(expectedCities);

// Act

var result = await \_unitOfWork.GetComboAsync(stateId);

// Assert

Assert.AreEqual(expectedCities, result);

\_mockCitiesRepository.Verify(x => x.GetComboAsync(stateId), Times.Once);

}

}

}

1. Corra los test y verifique que todo está funcionando correctamente.
2. Verificamos la cobertura del código.
3. Hacemos commit.

#### Repositorio

1. Adicione la clase **CitiesRepositoryTests**:

using Microsoft.EntityFrameworkCore;

using Orders.Backend.Data;

using Orders.Backend.Repositories.Implementations;

using Orders.Shared.DTOs;

using Orders.Shared.Entities;

namespace Orders.Tests.Repositories

{

[TestClass]

public class CitiesRepositoryTests

{

private DataContext \_context = null!;

private CitiesRepository \_repository = null!;

[TestInitialize]

public void Initialize()

{

var options = new DbContextOptionsBuilder<DataContext>()

.UseInMemoryDatabase(databaseName: "InMemoryDatabase")

.Options;

\_context = new DataContext(options);

\_repository = new CitiesRepository(\_context);

\_context.Countries.Add(new Country { Id = 1, Name = "Country" });

\_context.States.AddRange(

new State { Id = 1, Name = "State1", CountryId = 1 },

new State { Id = 2, Name = "State2", CountryId = 1 });

\_context.Cities.AddRange(

new City { Id = 1, Name = "City1", StateId = 1 },

new City { Id = 2, Name = "City2", StateId = 1 },

new City { Id = 3, Name = "City3", StateId = 2 }

);

\_context.SaveChanges();

}

[TestMethod]

public async Task GetAsync\_ShouldReturnAllCitiesInStateWithPagination()

{

// Arrange

var pagination = new PaginationDTO { Id = 1, RecordsNumber = 2, Page = 1, Filter = "City" };

// Act

var response = await \_repository.GetAsync(pagination);

// Assert

Assert.IsTrue(response.WasSuccess);

Assert.AreEqual(2, response.Result!.Count());

}

[TestMethod]

public async Task GetAsync\_ShouldReturnFilteredCities()

{

// Arrange

var pagination = new PaginationDTO { Id = 1, Filter = "City1", RecordsNumber = 10, Page = 1 };

// Act

var response = await \_repository.GetAsync(pagination);

// Assert

Assert.IsTrue(response.WasSuccess);

Assert.AreEqual(1, response.Result!.Count());

Assert.AreEqual("City1", response.Result!.First().Name);

}

[TestMethod]

public async Task GetComboAsync\_ShouldReturnAllCitiesInState()

{

// Arrange

var stateId = 1;

// Act

var cities = await \_repository.GetComboAsync(stateId);

// Assert

Assert.AreEqual(2, cities.Count());

}

[TestMethod]

public async Task GetTotalPagesAsync\_ShouldReturnTotalPages()

{

// Arrange

var pagination = new PaginationDTO { Id = 1, RecordsNumber = 1, Page = 1, Filter = "City" };

// Act

var response = await \_repository.GetTotalPagesAsync(pagination);

// Assert

Assert.IsTrue(response.WasSuccess);

Assert.AreEqual(2, response.Result);

}

[TestCleanup]

public void Cleanup()

{

\_context.Database.EnsureDeleted();

\_context.Dispose();

}

}

}

1. Corra los test y verifique que todo está funcionando correctamente.
2. Verificamos la cobertura del código.
3. Hacemos commit.

### Pedidos

#### Controlador

1. Adicione la clase **OrdersControllerTests**:

using System.Security.Claims;

using Microsoft.AspNetCore.Http;

using Microsoft.AspNetCore.Mvc;

using Moq;

using Orders.Backend.Controllers;

using Orders.Backend.Helpers;

using Orders.Backend.UnitsOfWork.Interfaces;

using Orders.Shared.DTOs;

using Orders.Shared.Entities;

using Orders.Shared.Responses;

namespace Orders.Tests.Controllers

{

[TestClass]

public class OrdersControllerTests

{

private Mock<IOrdersHelper> \_mockOrdersHelper = null!;

private Mock<IOrdersUnitOfWork> \_mockOrdersUnitOfWork = null!;

private OrdersController \_controller = null!;

[TestInitialize]

public void Initialize()

{

\_mockOrdersHelper = new Mock<IOrdersHelper>();

\_mockOrdersUnitOfWork = new Mock<IOrdersUnitOfWork>();

\_controller = new OrdersController(\_mockOrdersHelper.Object, \_mockOrdersUnitOfWork.Object);

}

private void SetupUser(string username)

{

var user = new ClaimsPrincipal(new ClaimsIdentity(new Claim[]

{

new Claim(ClaimTypes.Name, username)

}, "mock"));

\_controller.ControllerContext = new ControllerContext()

{

HttpContext = new DefaultHttpContext() { User = user }

};

}

[TestMethod]

public async Task PostAsync\_ShouldReturnBadRequest\_WhenOrderIsNotProcessed()

{

// Arrange

SetupUser("testuser");

var orderDto = new OrderDTO();

\_mockOrdersHelper.Setup(x => x.ProcessOrderAsync("testuser", It.IsAny<string>()))

.ReturnsAsync(new ActionResponse<bool> { WasSuccess = false });

// Act

var result = await \_controller.PostAsync(orderDto);

// Assert

Assert.IsInstanceOfType(result, typeof(BadRequestObjectResult));

\_mockOrdersHelper.Verify(x => x.ProcessOrderAsync("testuser", It.IsAny<string>()), Times.Once());

}

[TestMethod]

public async Task PostAsync\_ShouldReturnNoContent\_WhenOrderIsProcessed()

{

// Arrange

SetupUser("testuser");

var orderDto = new OrderDTO();

\_mockOrdersHelper.Setup(x => x.ProcessOrderAsync("testuser", It.IsAny<string>()))

.ReturnsAsync(new ActionResponse<bool> { WasSuccess = true });

// Act

var result = await \_controller.PostAsync(orderDto);

// Assert

Assert.IsInstanceOfType(result, typeof(NoContentResult));

\_mockOrdersHelper.Verify(x => x.ProcessOrderAsync("testuser", It.IsAny<string>()), Times.Once());

}

[TestMethod]

public async Task GetAsync\_ShouldReturnOk\_WhenOrdersAreRetrievedSuccessfully()

{

// Arrange

SetupUser("testuser");

var paginationDto = new PaginationDTO();

\_mockOrdersUnitOfWork.Setup(x => x.GetAsync("testuser", paginationDto))

.ReturnsAsync(new ActionResponse<IEnumerable<Order>> { WasSuccess = true, Result = new List<Order>() });

// Act

var result = await \_controller.GetAsync(paginationDto);

// Assert

Assert.IsInstanceOfType(result, typeof(OkObjectResult));

\_mockOrdersUnitOfWork.Verify(x => x.GetAsync("testuser", paginationDto), Times.Once());

}

[TestMethod]

public async Task GetAsync\_ShouldReturnBadRequest\_WhenOrdersRetrievalFails()

{

// Arrange

SetupUser("testuser");

var paginationDto = new PaginationDTO();

\_mockOrdersUnitOfWork.Setup(x => x.GetAsync("testuser", paginationDto))

.ReturnsAsync(new ActionResponse<IEnumerable<Order>> { WasSuccess = false });

// Act

var result = await \_controller.GetAsync(paginationDto);

// Assert

Assert.IsInstanceOfType(result, typeof(BadRequestResult));

\_mockOrdersUnitOfWork.Verify(x => x.GetAsync("testuser", paginationDto), Times.Once());

}

[TestMethod]

public async Task GetPagesAsync\_ShouldReturnOk\_WhenTotalPagesAreRetrievedSuccessfully()

{

// Arrange

SetupUser("testuser");

var paginationDto = new PaginationDTO();

\_mockOrdersUnitOfWork.Setup(x => x.GetTotalPagesAsync("testuser", paginationDto))

.ReturnsAsync(new ActionResponse<int> { WasSuccess = true, Result = 5 });

// Act

var result = await \_controller.GetPagesAsync(paginationDto);

// Assert

Assert.IsInstanceOfType(result, typeof(OkObjectResult));

var okResult = result as OkObjectResult;

Assert.AreEqual(5, okResult!.Value);

\_mockOrdersUnitOfWork.Verify(x => x.GetTotalPagesAsync("testuser", paginationDto), Times.Once());

}

[TestMethod]

public async Task GetPagesAsync\_ShouldReturnBadRequest\_WhenRetrievalFails()

{

// Arrange

SetupUser("testuser");

var paginationDto = new PaginationDTO();

\_mockOrdersUnitOfWork.Setup(x => x.GetTotalPagesAsync("testuser", paginationDto))

.ReturnsAsync(new ActionResponse<int> { WasSuccess = false });

// Act

var result = await \_controller.GetPagesAsync(paginationDto);

// Assert

Assert.IsInstanceOfType(result, typeof(BadRequestResult));

\_mockOrdersUnitOfWork.Verify(x => x.GetTotalPagesAsync("testuser", paginationDto), Times.Once());

}

[TestMethod]

public async Task GetAsync\_WithId\_ShouldReturnOk\_WhenOrderIsRetrievedSuccessfully()

{

// Arrange

SetupUser("testuser");

int orderId = 1;

\_mockOrdersUnitOfWork.Setup(x => x.GetAsync(orderId))

.ReturnsAsync(new ActionResponse<Order> { WasSuccess = true, Result = new Order() });

// Act

var result = await \_controller.GetAsync(orderId);

// Assert

Assert.IsInstanceOfType(result, typeof(OkObjectResult));

\_mockOrdersUnitOfWork.Verify(x => x.GetAsync(orderId), Times.Once());

}

[TestMethod]

public async Task GetAsync\_WithId\_ShouldReturnNotFound\_WhenOrderIsNotFound()

{

// Arrange

SetupUser("testuser");

int orderId = 1;

\_mockOrdersUnitOfWork.Setup(x => x.GetAsync(orderId))

.ReturnsAsync(new ActionResponse<Order> { WasSuccess = false, Message = "Order not found" });

// Act

var result = await \_controller.GetAsync(orderId);

// Assert

Assert.IsInstanceOfType(result, typeof(NotFoundObjectResult));

var notFoundResult = result as NotFoundObjectResult;

Assert.AreEqual("Order not found", notFoundResult!.Value);

\_mockOrdersUnitOfWork.Verify(x => x.GetAsync(orderId), Times.Once());

}

[TestMethod]

public async Task PutAsync\_ShouldReturnOk\_WhenOrderIsUpdatedSuccessfully()

{

// Arrange

SetupUser("testuser");

var orderDto = new OrderDTO();

\_mockOrdersUnitOfWork.Setup(x => x.UpdateFullAsync("testuser", orderDto))

.ReturnsAsync(new ActionResponse<Order> { WasSuccess = true });

// Act

var result = await \_controller.PutAsync(orderDto);

// Assert

Assert.IsInstanceOfType(result, typeof(OkObjectResult));

\_mockOrdersUnitOfWork.Verify(x => x.UpdateFullAsync("testuser", orderDto), Times.Once());

}

[TestMethod]

public async Task PutAsync\_ShouldReturnBadRequest\_WhenUpdateFails()

{

// Arrange

SetupUser("testuser");

var orderDto = new OrderDTO();

\_mockOrdersUnitOfWork.Setup(x => x.UpdateFullAsync("testuser", orderDto))

.ReturnsAsync(new ActionResponse<Order> { WasSuccess = false, Message = "Update failed" });

// Act

var result = await \_controller.PutAsync(orderDto);

// Assert

Assert.IsInstanceOfType(result, typeof(BadRequestObjectResult));

var badRequestResult = result as BadRequestObjectResult;

Assert.AreEqual("Update failed", badRequestResult!.Value);

\_mockOrdersUnitOfWork.Verify(x => x.UpdateFullAsync("testuser", orderDto), Times.Once());

}

}

}

1. Corra los test y verifique que todo está funcionando correctamente.
2. Verificamos la cobertura del código.
3. Hacemos commit.

#### Unidad de Trabajo

1. Adicione la clase **OrdersUnitOfWorkTests**:

using Moq;

using Orders.Backend.Repositories.Interfaces;

using Orders.Backend.UnitsOfWork.Implementations;

using Orders.Shared.DTOs;

using Orders.Shared.Entities;

using Orders.Shared.Responses;

namespace Orders.Tests.UnitsOfWork

{

[TestClass]

public class OrdersUnitOfWorkTests

{

private Mock<IGenericRepository<Order>> \_mockGenericRepository = null!;

private Mock<IOrdersRepository> \_mockOrdersRepository = null!;

private OrdersUnitOfWork \_ordersUnitOfWork = null!;

[TestInitialize]

public void SetUp()

{

\_mockGenericRepository = new Mock<IGenericRepository<Order>>();

\_mockOrdersRepository = new Mock<IOrdersRepository>();

\_ordersUnitOfWork = new OrdersUnitOfWork(\_mockGenericRepository.Object, \_mockOrdersRepository.Object);

}

[TestMethod]

public async Task GetAsync\_ShouldReturnOrders\_WhenCalled()

{

// Arrange

var email = "test@example.com";

var paginationDTO = new PaginationDTO();

var response = new ActionResponse<IEnumerable<Order>> { WasSuccess = true };

\_mockOrdersRepository.Setup(x => x.GetAsync(email, paginationDTO))

.ReturnsAsync(response);

// Act

var result = await \_ordersUnitOfWork.GetAsync(email, paginationDTO);

// Assert

Assert.AreEqual(response, result);

}

[TestMethod]

public async Task GetTotalPagesAsync\_ShouldReturnTotalPages\_WhenCalled()

{

// Arrange

var email = "test@example.com";

var paginationDTO = new PaginationDTO();

var response = new ActionResponse<int> { WasSuccess = true };

\_mockOrdersRepository.Setup(x => x.GetTotalPagesAsync(email, paginationDTO))

.ReturnsAsync(response);

// Act

var result = await \_ordersUnitOfWork.GetTotalPagesAsync(email, paginationDTO);

// Assert

Assert.AreEqual(response, result);

\_mockOrdersRepository.Verify(x => x.GetTotalPagesAsync(email, paginationDTO), Times.Once());

}

[TestMethod]

public async Task GetAsync\_WithId\_ShouldReturnOrder\_WhenCalled()

{

// Arrange

var orderId = 1;

var response = new ActionResponse<Order> { WasSuccess = true };

\_mockOrdersRepository.Setup(x => x.GetAsync(orderId))

.ReturnsAsync(response);

// Act

var result = await \_ordersUnitOfWork.GetAsync(orderId);

// Assert

Assert.AreEqual(response, result);

\_mockOrdersRepository.Verify(x => x.GetAsync(orderId), Times.Once());

}

[TestMethod]

public async Task UpdateFullAsync\_ShouldUpdateOrder\_WhenCalled()

{

// Arrange

var email = "test@example.com";

var orderDTO = new OrderDTO();

var response = new ActionResponse<Order> { WasSuccess = true };

\_mockOrdersRepository.Setup(x => x.UpdateFullAsync(email, orderDTO))

.ReturnsAsync(response);

// Act

var result = await \_ordersUnitOfWork.UpdateFullAsync(email, orderDTO);

// Assert

Assert.AreEqual(response, result);

\_mockOrdersRepository.Verify(x => x.UpdateFullAsync(email, orderDTO), Times.Once());

}

}

}

1. Corra los test y verifique que todo está funcionando correctamente.
2. Verificamos la cobertura del código.
3. Hacemos commit.

#### Repositorio

1. Adicione la clase **OrdersRepositoryTests**:

using Microsoft.EntityFrameworkCore;

using Moq;

using Orders.Backend.Data;

using Orders.Backend.Repositories.Implementations;

using Orders.Backend.Repositories.Interfaces;

using Orders.Shared.DTOs;

using Orders.Shared.Entities;

using Orders.Shared.Enums;

namespace Orders.Tests.Repositories

{

[TestClass]

public class OrdersRepositoryTests

{

private DataContext \_context = null!;

private OrdersRepository \_repository = null!;

private Mock<IUsersRepository> \_mockUserRepository = null!;

[TestInitialize]

public void Initialize()

{

var options = new DbContextOptionsBuilder<DataContext>()

.UseInMemoryDatabase(databaseName: Guid.NewGuid().ToString())

.Options;

\_context = new DataContext(options);

\_mockUserRepository = new Mock<IUsersRepository>();

\_repository = new OrdersRepository(\_context, \_mockUserRepository.Object);

}

[TestCleanup]

public void Cleanup()

{

\_context.Dispose();

}

[TestMethod]

public async Task GetAsync\_UserDoesNotExist\_ReturnsFailedActionResponse()

{

// Act

var response = await \_repository.GetAsync("nonexistentuser@example.com", new PaginationDTO());

// Assert

Assert.IsFalse(response.WasSuccess);

Assert.AreEqual("Usuario no válido", response.Message);

}

[TestMethod]

public async Task GetAsync\_ValidUserAndOrder\_ReturnsOrders()

{

// Arrange

var email = "test@example.com";

var user = await CreateTestUser(email, UserType.User);

await CreateTestOrder(user);

\_mockUserRepository.Setup(x => x.GetUserAsync(email))

.ReturnsAsync(user);

\_mockUserRepository.Setup(x => x.IsUserInRoleAsync(user, UserType.Admin.ToString()))

.ReturnsAsync(false);

// Act

var response = await \_repository.GetAsync(email, new PaginationDTO());

// Assert

Assert.IsTrue(response.WasSuccess);

Assert.IsNotNull(response.Result);

Assert.AreEqual(1, response.Result.Count());

\_mockUserRepository.Verify(x => x.GetUserAsync(email), Times.Once());

\_mockUserRepository.Verify(x => x.IsUserInRoleAsync(user, UserType.Admin.ToString()), Times.Once());

}

[TestMethod]

public async Task GetTotalPagesAsync\_UserDoesNotExist\_ReturnsFailedActionResponse()

{

// Act

var response = await \_repository.GetTotalPagesAsync("nonexistentuser@example.com", new PaginationDTO());

// Assert

Assert.IsFalse(response.WasSuccess);

Assert.AreEqual("Usuario no válido", response.Message);

}

[TestMethod]

public async Task GetTotalPagesAsync\_ReturnsCorrectNumberOfPages()

{

// Arrange

var email = "test@example.com";

var user = await CreateTestUser(email, UserType.User);

await CreateTestOrder(user);

\_mockUserRepository.Setup(x => x.GetUserAsync(email))

.ReturnsAsync(user);

\_mockUserRepository.Setup(x => x.IsUserInRoleAsync(user, UserType.Admin.ToString()))

.ReturnsAsync(false);

var pagination = new PaginationDTO { RecordsNumber = 2, Page = 1 };

// Act

var response = await \_repository.GetTotalPagesAsync(email, pagination);

// Assert

Assert.IsTrue(response.WasSuccess);

Assert.AreEqual(1, response.Result);

\_mockUserRepository.Verify(x => x.GetUserAsync(email), Times.Once());

\_mockUserRepository.Verify(x => x.IsUserInRoleAsync(user, UserType.Admin.ToString()), Times.Once());

}

[TestMethod]

public async Task GetAsync\_OrderDoesNotExist\_ReturnsFailedActionResponse()

{

// Act

var response = await \_repository.GetAsync(999);

// Assert

Assert.IsFalse(response.WasSuccess);

Assert.AreEqual("Pedido no existe", response.Message);

}

[TestMethod]

public async Task GetAsync\_OrderExists\_ReturnsOrder()

{

// Arrange

var email = "test@example.com";

var user = await CreateTestUser(email, UserType.User);

var order = await CreateTestOrder(user);

// Act

var response = await \_repository.GetAsync(order.Id);

// Assert

Assert.IsTrue(response.WasSuccess);

Assert.IsNotNull(response.Result);

Assert.AreEqual(order.Id, response.Result.Id);

}

[TestMethod]

public async Task UpdateFullAsync\_UserDoesNotExist\_ReturnsFailedActionResponse()

{

// Arrange

var orderDTO = new OrderDTO { Id = 1, OrderStatus = OrderStatus.Sent };

// Act

var response = await \_repository.UpdateFullAsync("nonexistentuser@example.com", orderDTO);

// Assert

Assert.IsFalse(response.WasSuccess);

Assert.AreEqual("Usuario no existe", response.Message);

}

[TestMethod]

public async Task UpdateFullAsync\_OrderDoesNotExist\_ReturnsFailedActionResponse()

{

// Arrange

var email = "test@example.com";

var user = await CreateTestUser(email, UserType.User);

\_mockUserRepository.Setup(x => x.GetUserAsync(email))

.ReturnsAsync(user);

\_mockUserRepository.Setup(x => x.IsUserInRoleAsync(user, UserType.Admin.ToString()))

.ReturnsAsync(true);

var orderDTO = new OrderDTO { Id = 999, OrderStatus = OrderStatus.Sent };

// Act

var response = await \_repository.UpdateFullAsync(email, orderDTO);

// Assert

Assert.IsFalse(response.WasSuccess);

Assert.AreEqual("Pedido no existe", response.Message);

\_mockUserRepository.Verify(x => x.GetUserAsync(email), Times.Once());

\_mockUserRepository.Verify(x => x.IsUserInRoleAsync(user, UserType.Admin.ToString()), Times.Once());

}

[TestMethod]

public async Task UpdateFullAsync\_ValidData\_UpdatesOrder()

{

// Arrange

var email = "admin@example.com";

var user = await CreateTestUser(email, UserType.Admin);

var order = await CreateTestOrder(user);

\_mockUserRepository.Setup(x => x.GetUserAsync(email))

.ReturnsAsync(user);

\_mockUserRepository.Setup(x => x.IsUserInRoleAsync(user, UserType.Admin.ToString()))

.ReturnsAsync(true);

var orderDTO = new OrderDTO { Id = order.Id, OrderStatus = OrderStatus.Sent };

// Act

var response = await \_repository.UpdateFullAsync(email, orderDTO);

// Assert

Assert.IsTrue(response.WasSuccess);

Assert.AreEqual(OrderStatus.Sent, response.Result!.OrderStatus);

\_mockUserRepository.Verify(x => x.GetUserAsync(email), Times.Once());

\_mockUserRepository.Verify(x => x.IsUserInRoleAsync(user, UserType.Admin.ToString()), Times.Once());

}

[TestMethod]

public async Task UpdateFullAsync\_UserNoAdmin\_ReturnError()

{

// Arrange

var email = "user@example.com";

var user = await CreateTestUser(email, UserType.User);

var order = await CreateTestOrder(user);

\_mockUserRepository.Setup(x => x.GetUserAsync(email))

.ReturnsAsync(user);

\_mockUserRepository.Setup(x => x.IsUserInRoleAsync(user, UserType.Admin.ToString()))

.ReturnsAsync(false);

var orderDTO = new OrderDTO { Id = order.Id, OrderStatus = OrderStatus.Sent };

// Act

var response = await \_repository.UpdateFullAsync(email, orderDTO);

// Assert

Assert.IsFalse(response.WasSuccess);

\_mockUserRepository.Verify(x => x.GetUserAsync(email), Times.Once());

\_mockUserRepository.Verify(x => x.IsUserInRoleAsync(user, UserType.Admin.ToString()), Times.Once());

}

[TestMethod]

public async Task UpdateFullAsync\_CancelOrder\_UpdatesOrderAndReturnInventory()

{

// Arrange

var email = "admin@example.com";

var user = await CreateTestUser(email, UserType.Admin);

var order = await CreateTestOrderForCancel(user);

\_mockUserRepository.Setup(x => x.GetUserAsync(email))

.ReturnsAsync(user);

\_mockUserRepository.Setup(x => x.IsUserInRoleAsync(user, UserType.Admin.ToString()))

.ReturnsAsync(true);

var orderDTO = new OrderDTO { Id = order.Id, OrderStatus = OrderStatus.Cancelled };

// Act

var response = await \_repository.UpdateFullAsync(email, orderDTO);

// Assert

Assert.IsTrue(response.WasSuccess);

Assert.AreEqual(OrderStatus.Cancelled, response.Result!.OrderStatus);

\_mockUserRepository.Verify(x => x.GetUserAsync(email), Times.Once());

\_mockUserRepository.Verify(x => x.IsUserInRoleAsync(user, UserType.Admin.ToString()), Times.Once());

}

private async Task<User> CreateTestUser(string email, UserType userType)

{

var user = new User { Email = email, UserType = userType, Address = "Any", Document = "Any", FirstName = "John", LastName = "Doe" };

await \_context.Users.AddAsync(user);

await \_context.SaveChangesAsync();

return user;

}

private async Task<Order> CreateTestOrder(User user)

{

var order = new Order { User = user };

await \_context.Orders.AddAsync(order);

await \_context.SaveChangesAsync();

return order;

}

private async Task<Order> CreateTestOrderForCancel(User user)

{

await \_context.Products.AddAsync(new Product { Id = 1, Name = "Some", Description = "Some" });

var order = new Order

{

User = user,

OrderDetails = new List<OrderDetail>

{

new OrderDetail { Id = 1, ProductId = 1 }

}

};

await \_context.Orders.AddAsync(order);

await \_context.SaveChangesAsync();

return order;

}

}

}

1. Corra los test y verifique que todo está funcionando correctamente.
2. Verificamos la cobertura del código.
3. Hacemos commit.

### PedidosTemporales

#### Controlador

1. Adicione la clase **TemporalOrdersControllerTests**:

using System.Security.Claims;

using Microsoft.AspNetCore.Http;

using Microsoft.AspNetCore.Mvc;

using Moq;

using Orders.Backend.Controllers;

using Orders.Backend.UnitsOfWork.Interfaces;

using Orders.Shared.DTOs;

using Orders.Shared.Entities;

using Orders.Shared.Responses;

namespace Orders.Tests.Controllers

{

[TestClass]

public class TemporalOrdersControllerTests

{

private TemporalOrdersController \_controller = null!;

private Mock<ITemporalOrdersUnitOfWork> \_temporalOrdersUnitOfWorkMock = null!;

private Mock<IGenericUnitOfWork<TemporalOrder>> \_unitOfWorkMock = null!;

private DefaultHttpContext \_httpContext = null!;

[TestInitialize]

public void Initialize()

{

\_temporalOrdersUnitOfWorkMock = new Mock<ITemporalOrdersUnitOfWork>();

\_unitOfWorkMock = new Mock<IGenericUnitOfWork<TemporalOrder>>();

\_controller = new TemporalOrdersController(\_unitOfWorkMock.Object, \_temporalOrdersUnitOfWorkMock.Object);

\_httpContext = new DefaultHttpContext();

\_controller.ControllerContext.HttpContext = \_httpContext;

\_httpContext.User = new ClaimsPrincipal(new ClaimsIdentity(new Claim[] { new Claim(ClaimTypes.Name, "testUser") }));

}

[TestMethod]

public async Task PostAsync\_Success\_ReturnsOkObjectResult()

{

// Arrange

var temporalOrderDTO = new TemporalOrderDTO();

\_temporalOrdersUnitOfWorkMock.Setup(x => x.AddFullAsync(It.IsAny<string>(), It.IsAny<TemporalOrderDTO>()))

.ReturnsAsync(new ActionResponse<TemporalOrderDTO> { WasSuccess = true });

// Act

var result = await \_controller.PostAsync(temporalOrderDTO);

// Assert

Assert.IsInstanceOfType(result, typeof(OkObjectResult));

\_temporalOrdersUnitOfWorkMock.Verify(x => x.AddFullAsync(It.IsAny<string>(), It.IsAny<TemporalOrderDTO>()), Times.Once());

}

[TestMethod]

public async Task PostAsync\_Failure\_ReturnsBadRequestObjectResult()

{

// Arrange

var temporalOrderDTO = new TemporalOrderDTO();

\_temporalOrdersUnitOfWorkMock.Setup(x => x.AddFullAsync(It.IsAny<string>(), It.IsAny<TemporalOrderDTO>()))

.ReturnsAsync(new ActionResponse<TemporalOrderDTO> { WasSuccess = false });

// Act

var result = await \_controller.PostAsync(temporalOrderDTO);

// Assert

Assert.IsInstanceOfType(result, typeof(BadRequestObjectResult));

\_temporalOrdersUnitOfWorkMock.Verify(x => x.AddFullAsync(It.IsAny<string>(), It.IsAny<TemporalOrderDTO>()), Times.Once());

}

[TestMethod]

public async Task GetAsync\_Success\_ReturnsOkObjectResult()

{

// Arrange

var userName = "testUser";

\_temporalOrdersUnitOfWorkMock.Setup(x => x.GetAsync(userName))

.ReturnsAsync(new ActionResponse<IEnumerable<TemporalOrder>> { WasSuccess = true });

// Act

var result = await \_controller.GetAsync();

// Assert

Assert.IsInstanceOfType(result, typeof(OkObjectResult));

\_temporalOrdersUnitOfWorkMock.Verify(x => x.GetAsync(userName), Times.Once());

}

[TestMethod]

public async Task GetAsync\_Failure\_ReturnsBadRequestObjectResult()

{

// Arrange

var userName = "testUser";

\_temporalOrdersUnitOfWorkMock.Setup(x => x.GetAsync(userName))

.ReturnsAsync(new ActionResponse<IEnumerable<TemporalOrder>> { WasSuccess = false });

// Act

var result = await \_controller.GetAsync();

// Assert

Assert.IsInstanceOfType(result, typeof(BadRequestObjectResult));

\_temporalOrdersUnitOfWorkMock.Verify(x => x.GetAsync(userName), Times.Once());

}

[TestMethod]

public async Task GetCountAsync\_Success\_ReturnsOkObjectResult()

{

// Arrange

var userName = "testUser";

\_temporalOrdersUnitOfWorkMock.Setup(x => x.GetCountAsync(userName))

.ReturnsAsync(new ActionResponse<int> { WasSuccess = true, Result = 5 });

// Act

var result = await \_controller.GetCountAsync();

// Assert

Assert.IsInstanceOfType(result, typeof(OkObjectResult));

\_temporalOrdersUnitOfWorkMock.Verify(x => x.GetCountAsync(userName), Times.Once());

}

[TestMethod]

public async Task GetCountAsync\_Failure\_ReturnsBadRequestObjectResult()

{

// Arrange

var userName = "testUser";

\_temporalOrdersUnitOfWorkMock.Setup(x => x.GetCountAsync(userName))

.ReturnsAsync(new ActionResponse<int> { WasSuccess = false, Message = "Failed" });

// Act

var result = await \_controller.GetCountAsync();

// Assert

Assert.IsInstanceOfType(result, typeof(BadRequestObjectResult));

\_temporalOrdersUnitOfWorkMock.Verify(x => x.GetCountAsync(userName), Times.Once());

}

[TestMethod]

public async Task GetAsync\_ById\_Success\_ReturnsOkObjectResult()

{

// Arrange

\_temporalOrdersUnitOfWorkMock.Setup(x => x.GetAsync(It.IsAny<int>()))

.ReturnsAsync(new ActionResponse<TemporalOrder> { WasSuccess = true, Result = new TemporalOrder() });

// Act

var result = await \_controller.GetAsync(1);

// Assert

Assert.IsInstanceOfType(result, typeof(OkObjectResult));

\_temporalOrdersUnitOfWorkMock.Verify(x => x.GetAsync(It.IsAny<int>()), Times.Once());

}

[TestMethod]

public async Task GetAsync\_ById\_Failure\_ReturnsNotFoundObjectResult()

{

// Arrange

\_temporalOrdersUnitOfWorkMock.Setup(x => x.GetAsync(It.IsAny<int>()))

.ReturnsAsync(new ActionResponse<TemporalOrder> { WasSuccess = false, Message = "Not Found" });

// Act

var result = await \_controller.GetAsync(1);

// Assert

Assert.IsInstanceOfType(result, typeof(NotFoundObjectResult));

\_temporalOrdersUnitOfWorkMock.Verify(x => x.GetAsync(It.IsAny<int>()), Times.Once());

}

[TestMethod]

public async Task PutFullAsync\_Success\_ReturnsOkObjectResult()

{

// Arrange

var temporalOrderDTO = new TemporalOrderDTO();

\_temporalOrdersUnitOfWorkMock.Setup(x => x.PutFullAsync(temporalOrderDTO))

.ReturnsAsync(new ActionResponse<TemporalOrder> { WasSuccess = true, Result = new TemporalOrder() });

// Act

var result = await \_controller.PutFullAsync(temporalOrderDTO);

// Assert

Assert.IsInstanceOfType(result, typeof(OkObjectResult));

\_temporalOrdersUnitOfWorkMock.Verify(x => x.PutFullAsync(temporalOrderDTO), Times.Once());

}

[TestMethod]

public async Task PutFullAsync\_Failure\_ReturnsNotFoundObjectResult()

{

// Arrange

var temporalOrderDTO = new TemporalOrderDTO();

\_temporalOrdersUnitOfWorkMock.Setup(x => x.PutFullAsync(temporalOrderDTO))

.ReturnsAsync(new ActionResponse<TemporalOrder> { WasSuccess = false, Message = "Not Found" });

// Act

var result = await \_controller.PutFullAsync(temporalOrderDTO);

// Assert

Assert.IsInstanceOfType(result, typeof(NotFoundObjectResult));

\_temporalOrdersUnitOfWorkMock.Verify(x => x.PutFullAsync(temporalOrderDTO), Times.Once());

}

}

}

1. Corra los test y verifique que todo está funcionando correctamente.
2. Verificamos la cobertura del código.
3. Hacemos commit.

#### Unidad de Trabajo

1. Adicione la clase **TemporalOrdersUnitOfWorkTests**:

using Moq;

using Orders.Backend.Repositories.Interfaces;

using Orders.Backend.UnitsOfWork.Implementations;

using Orders.Shared.DTOs;

using Orders.Shared.Entities;

using Orders.Shared.Responses;

namespace Orders.Tests.UnitsOfWork

{

[TestClass]

public class TemporalOrdersUnitOfWorkTests

{

private TemporalOrdersUnitOfWork \_unitOfWork = null!;

private Mock<IGenericRepository<TemporalOrder>> \_genericRepositoryMock = null!;

private Mock<ITemporalOrdersRepository> \_temporalOrdersRepositoryMock = null!;

[TestInitialize]

public void Initialize()

{

\_genericRepositoryMock = new Mock<IGenericRepository<TemporalOrder>>();

\_temporalOrdersRepositoryMock = new Mock<ITemporalOrdersRepository>();

\_unitOfWork = new TemporalOrdersUnitOfWork(\_genericRepositoryMock.Object, \_temporalOrdersRepositoryMock.Object);

}

[TestMethod]

public async Task AddFullAsync\_CallsRepository\_ReturnsResult()

{

var email = "test@example.com";

var dto = new TemporalOrderDTO();

var response = new ActionResponse<TemporalOrderDTO>();

\_temporalOrdersRepositoryMock.Setup(repo => repo.AddFullAsync(email, dto))

.ReturnsAsync(response);

var result = await \_unitOfWork.AddFullAsync(email, dto);

Assert.AreEqual(response, result);

\_temporalOrdersRepositoryMock.Verify(repo => repo.AddFullAsync(email, dto), Times.Once);

}

[TestMethod]

public async Task GetAsync\_CallsRepository\_ReturnsResult()

{

var email = "test@example.com";

var response = new ActionResponse<IEnumerable<TemporalOrder>>();

\_temporalOrdersRepositoryMock.Setup(repo => repo.GetAsync(email))

.ReturnsAsync(response);

var result = await \_unitOfWork.GetAsync(email);

Assert.AreEqual(response, result);

\_temporalOrdersRepositoryMock.Verify(repo => repo.GetAsync(email), Times.Once);

}

[TestMethod]

public async Task GetCountAsync\_CallsRepository\_ReturnsResult()

{

var email = "test@example.com";

var response = new ActionResponse<int>();

\_temporalOrdersRepositoryMock.Setup(repo => repo.GetCountAsync(email))

.ReturnsAsync(response);

var result = await \_unitOfWork.GetCountAsync(email);

Assert.AreEqual(response, result);

\_temporalOrdersRepositoryMock.Verify(repo => repo.GetCountAsync(email), Times.Once);

}

[TestMethod]

public async Task PutFullAsync\_CallsRepository\_ReturnsResult()

{

var dto = new TemporalOrderDTO();

var response = new ActionResponse<TemporalOrder>();

\_temporalOrdersRepositoryMock.Setup(repo => repo.PutFullAsync(dto))

.ReturnsAsync(response);

var result = await \_unitOfWork.PutFullAsync(dto);

Assert.AreEqual(response, result);

\_temporalOrdersRepositoryMock.Verify(repo => repo.PutFullAsync(dto), Times.Once);

}

[TestMethod]

public async Task GetAsync\_ById\_CallsRepository\_ReturnsResult()

{

int id = 1;

var response = new ActionResponse<TemporalOrder>();

\_temporalOrdersRepositoryMock.Setup(repo => repo.GetAsync(id))

.ReturnsAsync(response);

var result = await \_unitOfWork.GetAsync(id);

Assert.AreEqual(response, result);

\_temporalOrdersRepositoryMock.Verify(repo => repo.GetAsync(id), Times.Once);

}

}

}

1. Corra los test y verifique que todo está funcionando correctamente.
2. Verificamos la cobertura del código.
3. Hacemos commit.

#### Repositorio

1. Adicione la clase **TemporalOrdersRepositoryTests**:

using Microsoft.EntityFrameworkCore;

using Moq;

using Orders.Backend.Data;

using Orders.Backend.Repositories.Implementations;

using Orders.Backend.Repositories.Interfaces;

using Orders.Shared.DTOs;

using Orders.Shared.Entities;

using Orders.Tests.Shared;

namespace Orders.Tests.Repositories

{

[TestClass]

public class TemporalOrdersRepositoryTests

{

private TemporalOrdersRepository \_repository = null!;

private DataContext \_context = null!;

private Mock<IUsersRepository> \_userRepositoryMock = null!;

private DbContextOptions<DataContext> \_options = null!;

[TestInitialize]

public void Initialize()

{

\_options = new DbContextOptionsBuilder<DataContext>()

.UseInMemoryDatabase(databaseName: Guid.NewGuid().ToString())

.Options;

\_context = new DataContext(\_options);

\_userRepositoryMock = new Mock<IUsersRepository>();

\_repository = new TemporalOrdersRepository(\_context, \_userRepositoryMock.Object);

}

[TestCleanup]

public void Cleanup()

{

\_context.Database.EnsureDeleted();

\_context.Dispose();

}

[TestMethod]

public async Task AddFullAsync\_ValidData\_AddsTemporalOrder()

{

// Arrange

var email = "test@example.com";

var user = new User { Email = email, Address = "Any", Document = "Any", FirstName = "John", LastName = "Doe" };

\_context.Users.Add(user);

\_context.SaveChanges();

var product = new Product { Id = 1, Name = "Some", Description = "Some" };

\_context.Products.Add(product);

\_context.SaveChanges();

var dto = new TemporalOrderDTO

{

ProductId = product.Id,

Quantity = 1

};

\_userRepositoryMock.Setup(x => x.GetUserAsync(email))

.ReturnsAsync(user);

// Act

var result = await \_repository.AddFullAsync(email, dto);

// Assert

Assert.IsTrue(result.WasSuccess);

Assert.AreEqual(1, \_context.TemporalOrders.Count());

var temporalOrder = \_context.TemporalOrders.First();

Assert.AreEqual(product.Id, temporalOrder.ProductId);

Assert.AreEqual(1, temporalOrder.Quantity);

}

[TestMethod]

public async Task AddFullAsync\_WithException\_ReturnsError()

{

// Arrange

var exceptionalContext = new ExceptionalDataContext(\_options);

var email = "test@example.com";

var user = new User { Email = email, Address = "Any", Document = "Any", FirstName = "John", LastName = "Doe" };

exceptionalContext.Users.Add(user);

exceptionalContext.SaveChanges();

var product = new Product { Id = 1, Name = "Some", Description = "Some" };

exceptionalContext.Products.Add(product);

exceptionalContext.SaveChanges();

var dto = new TemporalOrderDTO

{

ProductId = product.Id,

Quantity = 1

};

\_userRepositoryMock.Setup(x => x.GetUserAsync(email))

.ReturnsAsync(user);

var repository = new TemporalOrdersRepository(exceptionalContext, \_userRepositoryMock.Object);

// Act

var result = await repository.AddFullAsync(email, dto);

// Assert

Assert.IsFalse(result.WasSuccess);

}

[TestMethod]

public async Task AddFullAsync\_ValidUser\_ReturnsError()

{

// Arrange

var email = "test@example.com";

var product = new Product { Id = 1, Name = "Some", Description = "Some" };

\_context.Products.Add(product);

\_context.SaveChanges();

var dto = new TemporalOrderDTO

{

ProductId = product.Id,

Quantity = 1

};

// Act

var result = await \_repository.AddFullAsync(email, dto);

// Assert

Assert.IsFalse(result.WasSuccess);

Assert.AreEqual("Usuario no existe", result.Message);

}

[TestMethod]

public async Task AddFullAsync\_InvalidProduct\_ReturnsError()

{

// Arrange

var email = "test@example.com";

var dto = new TemporalOrderDTO

{

ProductId = 999,

Quantity = 1

};

// Act

var result = await \_repository.AddFullAsync(email, dto);

// Assert

Assert.IsFalse(result.WasSuccess);

Assert.AreEqual("Producto no existe", result.Message);

}

[TestMethod]

public async Task GetAsync\_UserExists\_ReturnsTemporalOrders()

{

// Arrange

var email = "test@example.com";

var product = new Product { Id = 1, Name = "Some", Description = "Some" };

\_context.Products.Add(product);

var user = new User { Email = email, Address = "Any", Document = "Any", FirstName = "John", LastName = "Doe" };

\_context.Users.Add(user);

\_context.SaveChanges();

var temporalOrders = new List<TemporalOrder>

{

new TemporalOrder { User = user, Product = product, Quantity = 1 },

new TemporalOrder { User = user, Product = product, Quantity = 2 }

};

\_context.TemporalOrders.AddRange(temporalOrders);

\_context.SaveChanges();

// Act

var result = await \_repository.GetAsync(email);

// Assert

Assert.IsTrue(result.WasSuccess);

Assert.AreEqual(2, result.Result!.Count());

}

[TestMethod]

public async Task GetCountAsync\_UserWithNoOrders\_ReturnsZero()

{

// Arrange

var email = "test@example.com";

// Act

var result = await \_repository.GetCountAsync(email);

// Assert

Assert.IsTrue(result.WasSuccess);

Assert.AreEqual(0, result.Result);

}

[TestMethod]

public async Task GetCountAsync\_UserDoesNotExist\_ReturnsZero()

{

// Arrange

var email = "nonexistent@example.com";

// Act

var result = await \_repository.GetCountAsync(email);

// Assert

Assert.IsTrue(result.WasSuccess);

Assert.AreEqual(0, result.Result);

}

[TestMethod]

public async Task PutFullAsync\_OrderExists\_UpdatesOrder()

{

// Arrange

var temporalOrder = new TemporalOrder { Id = 1, Remarks = "Old Remarks", Quantity = 5 };

\_context.TemporalOrders.Add(temporalOrder);

await \_context.SaveChangesAsync();

var updateDTO = new TemporalOrderDTO { Id = 1, Remarks = "New Remarks", Quantity = 10 };

// Act

var result = await \_repository.PutFullAsync(updateDTO);

// Assert

Assert.IsTrue(result.WasSuccess);

Assert.AreEqual(updateDTO.Remarks, result.Result!.Remarks);

Assert.AreEqual(updateDTO.Quantity, result.Result.Quantity);

}

[TestMethod]

public async Task PutFullAsync\_OrderDoesNotExist\_ReturnsErrorActionResponse()

{

// Arrange

var updateDTO = new TemporalOrderDTO { Id = 99, Remarks = "New Remarks", Quantity = 10 };

// Act

var result = await \_repository.PutFullAsync(updateDTO);

// Assert

Assert.IsFalse(result.WasSuccess);

Assert.AreEqual("Registro no encontrado", result.Message);

}

[TestMethod]

public async Task GetAsync\_OrderExists\_ReturnsOrder()

{

// Arrange

var email = "test@example.com";

var user = new User { Email = email, Address = "Any", Document = "Any", FirstName = "John", LastName = "Doe" };

\_context.Users.Add(user);

\_context.SaveChanges();

var product = new Product { Id = 1, Name = "Some", Description = "Some" };

\_context.Products.Add(product);

\_context.SaveChanges();

var temporalOrder = new TemporalOrder { Id = 1, User = user, Product = product };

\_context.TemporalOrders.Add(temporalOrder);

await \_context.SaveChangesAsync();

// Act

var result = await \_repository.GetAsync(1);

// Assert

Assert.IsTrue(result.WasSuccess);

Assert.IsNotNull(result.Result);

Assert.AreEqual(1, result.Result.Id);

}

[TestMethod]

public async Task GetAsync\_OrderDoesNotExist\_ReturnsErrorActionResponse()

{

// Act

var result = await \_repository.GetAsync(99);

// Assert

Assert.IsFalse(result.WasSuccess);

Assert.AreEqual("Registro no encontrado", result.Message);

}

}

}

1. Corra los test y verifique que todo está funcionando correctamente.
2. Verificamos la cobertura del código.
3. Hacemos commit.

### Productos

#### Controlador

1. Adicione la clase **ProductsControllerTests**:

using Microsoft.AspNetCore.Mvc;

using Moq;

using Orders.Backend.Controllers;

using Orders.Backend.UnitsOfWork.Interfaces;

using Orders.Shared.DTOs;

using Orders.Shared.Entities;

using Orders.Shared.Responses;

namespace Orders.Tests.Controllers

{

[TestClass]

public class ProductsControllerTests

{

private Mock<IGenericUnitOfWork<Product>> \_unitOfWorkMock = null!;

private Mock<IProductsUnitOfWork> \_productsUnitOfWorkMock = null!;

private ProductsController \_controller = null!;

[TestInitialize]

public void Initialize()

{

\_unitOfWorkMock = new Mock<IGenericUnitOfWork<Product>>();

\_productsUnitOfWorkMock = new Mock<IProductsUnitOfWork>();

\_controller = new ProductsController(\_unitOfWorkMock.Object, \_productsUnitOfWorkMock.Object);

}

[TestMethod]

public async Task GetAsync\_NoSuccess\_ReturnsError()

{

// Arrange

var pagination = new PaginationDTO();

\_productsUnitOfWorkMock.Setup(x => x.GetAsync(pagination))

.ReturnsAsync(new ActionResponse<IEnumerable<Product>>() { WasSuccess = false });

// Act

var result = await \_controller.GetAsync(pagination);

// Assert

Assert.IsInstanceOfType(result, typeof(BadRequestResult));

\_productsUnitOfWorkMock.Verify(x => x.GetAsync(pagination), Times.Once());

}

[TestMethod]

public async Task GetAsync\_WhenCalled\_ReturnsOkResult()

{

// Arrange

var pagination = new PaginationDTO();

\_productsUnitOfWorkMock.Setup(x => x.GetAsync(pagination))

.ReturnsAsync(new ActionResponse<IEnumerable<Product>>() { WasSuccess = true });

// Act

var result = await \_controller.GetAsync(pagination);

// Assert

Assert.IsInstanceOfType(result, typeof(OkObjectResult));

\_productsUnitOfWorkMock.Verify(x => x.GetAsync(pagination), Times.Once());

}

[TestMethod]

public async Task GetPagesAsync\_WhenCalled\_ReturnsOkResult()

{

// Arrange

var pagination = new PaginationDTO();

\_productsUnitOfWorkMock.Setup(x => x.GetTotalPagesAsync(pagination))

.ReturnsAsync(new ActionResponse<int>() { WasSuccess = true, Result = 5 });

// Act

var result = await \_controller.GetPagesAsync(pagination);

// Assert

Assert.IsInstanceOfType(result, typeof(OkObjectResult));

\_productsUnitOfWorkMock.Verify(x => x.GetTotalPagesAsync(pagination), Times.Once());

}

[TestMethod]

public async Task GetPagesAsync\_WhenFailed\_ReturnsBadRequest()

{

// Arrange

var pagination = new PaginationDTO();

\_productsUnitOfWorkMock.Setup(x => x.GetTotalPagesAsync(pagination))

.ReturnsAsync(new ActionResponse<int>() { WasSuccess = false });

// Act

var result = await \_controller.GetPagesAsync(pagination);

// Assert

Assert.IsInstanceOfType(result, typeof(BadRequestResult));

\_productsUnitOfWorkMock.Verify(x => x.GetTotalPagesAsync(pagination), Times.Once());

}

[TestMethod]

public async Task GetAsync\_ById\_WhenFound\_ReturnsOkResult()

{

// Arrange

int productId = 1;

\_productsUnitOfWorkMock.Setup(x => x.GetAsync(productId))

.ReturnsAsync(new ActionResponse<Product>() { WasSuccess = true });

// Act

var result = await \_controller.GetAsync(productId);

// Assert

Assert.IsInstanceOfType(result, typeof(OkObjectResult));

\_productsUnitOfWorkMock.Verify(x => x.GetAsync(productId), Times.Once());

}

[TestMethod]

public async Task GetAsync\_ById\_WhenNotFound\_ReturnsNotFound()

{

// Arrange

int productId = 1;

\_productsUnitOfWorkMock.Setup(x => x.GetAsync(productId))

.ReturnsAsync(new ActionResponse<Product>() { WasSuccess = false, Message = "Not Found" });

// Act

var result = await \_controller.GetAsync(productId);

// Assert

Assert.IsInstanceOfType(result, typeof(NotFoundObjectResult));

\_productsUnitOfWorkMock.Verify(x => x.GetAsync(productId), Times.Once());

}

[TestMethod]

public async Task PostFullAsync\_WhenAdded\_ReturnsOkResult()

{

// Arrange

var productDTO = new ProductDTO();

\_productsUnitOfWorkMock.Setup(x => x.AddFullAsync(productDTO))

.ReturnsAsync(new ActionResponse<Product>() { WasSuccess = true });

// Act

var result = await \_controller.PostFullAsync(productDTO);

// Assert

Assert.IsInstanceOfType(result, typeof(OkObjectResult));

\_productsUnitOfWorkMock.Verify(x => x.AddFullAsync(productDTO), Times.Once());

}

[TestMethod]

public async Task PostFullAsync\_WhenFailed\_ReturnsNotFound()

{

// Arrange

var productDTO = new ProductDTO();

\_productsUnitOfWorkMock.Setup(x => x.AddFullAsync(productDTO))

.ReturnsAsync(new ActionResponse<Product>() { WasSuccess = false, Message = "Not Found" });

// Act

var result = await \_controller.PostFullAsync(productDTO);

// Assert

Assert.IsInstanceOfType(result, typeof(NotFoundObjectResult));

\_productsUnitOfWorkMock.Verify(x => x.AddFullAsync(productDTO), Times.Once());

}

[TestMethod]

public async Task PutFullAsync\_WhenUpdated\_ReturnsOkResult()

{

// Arrange

var productDTO = new ProductDTO();

\_productsUnitOfWorkMock.Setup(x => x.UpdateFullAsync(productDTO))

.ReturnsAsync(new ActionResponse<Product>() { WasSuccess = true });

// Act

var result = await \_controller.PutFullAsync(productDTO);

// Assert

Assert.IsInstanceOfType(result, typeof(OkObjectResult));

\_productsUnitOfWorkMock.Verify(x => x.UpdateFullAsync(productDTO), Times.Once());

}

[TestMethod]

public async Task PutFullAsync\_WhenFailed\_ReturnsNotFound()

{

// Arrange

var productDTO = new ProductDTO();

\_productsUnitOfWorkMock.Setup(x => x.UpdateFullAsync(productDTO))

.ReturnsAsync(new ActionResponse<Product>() { WasSuccess = false, Message = "Not Found" });

// Act

var result = await \_controller.PutFullAsync(productDTO);

// Assert

Assert.IsInstanceOfType(result, typeof(NotFoundObjectResult));

\_productsUnitOfWorkMock.Verify(x => x.UpdateFullAsync(productDTO), Times.Once());

}

[TestMethod]

public async Task PostAddImagesAsync\_WhenSuccess\_ReturnsOkResult()

{

// Arrange

var imageDTO = new ImageDTO();

\_productsUnitOfWorkMock.Setup(x => x.AddImageAsync(imageDTO))

.ReturnsAsync(new ActionResponse<ImageDTO>() { WasSuccess = true });

// Act

var result = await \_controller.PostAddImagesAsync(imageDTO);

// Assert

Assert.IsInstanceOfType(result, typeof(OkObjectResult));

\_productsUnitOfWorkMock.Verify(x => x.AddImageAsync(imageDTO), Times.Once());

}

[TestMethod]

public async Task PostAddImagesAsync\_WhenFailed\_ReturnsBadRequest()

{

// Arrange

var imageDTO = new ImageDTO();

\_productsUnitOfWorkMock.Setup(x => x.AddImageAsync(imageDTO))

.ReturnsAsync(new ActionResponse<ImageDTO>() { WasSuccess = false, Message = "Failed to add image" });

// Act

var result = await \_controller.PostAddImagesAsync(imageDTO);

// Assert

Assert.IsInstanceOfType(result, typeof(BadRequestObjectResult));

\_productsUnitOfWorkMock.Verify(x => x.AddImageAsync(imageDTO), Times.Once());

}

[TestMethod]

public async Task PostRemoveLastImageAsync\_WhenSuccess\_ReturnsOkResult()

{

// Arrange

var imageDTO = new ImageDTO();

\_productsUnitOfWorkMock.Setup(x => x.RemoveLastImageAsync(imageDTO))

.ReturnsAsync(new ActionResponse<ImageDTO>() { WasSuccess = true });

// Act

var result = await \_controller.PostRemoveLastImageAsync(imageDTO);

// Assert

Assert.IsInstanceOfType(result, typeof(OkObjectResult));

\_productsUnitOfWorkMock.Verify(x => x.RemoveLastImageAsync(imageDTO), Times.Once());

}

[TestMethod]

public async Task PostRemoveLastImageAsync\_WhenFailed\_ReturnsBadRequest()

{

// Arrange

var imageDTO = new ImageDTO();

\_productsUnitOfWorkMock.Setup(x => x.RemoveLastImageAsync(imageDTO))

.ReturnsAsync(new ActionResponse<ImageDTO>() { WasSuccess = false, Message = "Failed to remove image" });

// Act

var result = await \_controller.PostRemoveLastImageAsync(imageDTO);

// Assert

Assert.IsInstanceOfType(result, typeof(BadRequestObjectResult));

\_productsUnitOfWorkMock.Verify(x => x.RemoveLastImageAsync(imageDTO), Times.Once());

}

[TestMethod]

public async Task DeleteAsync\_ExistingItem\_ReturnsNoContent()

{

// Arrange

int id = 1;

\_productsUnitOfWorkMock.Setup(x => x.DeleteAsync(id))

.ReturnsAsync(new ActionResponse<Product>() { WasSuccess = true });

// Act

var result = await \_controller.DeleteAsync(id);

// Assert

Assert.IsInstanceOfType(result, typeof(NoContentResult));

\_productsUnitOfWorkMock.Verify(x => x.DeleteAsync(id), Times.Once());

}

[TestMethod]

public async Task DeleteAsync\_NonExistingItem\_ReturnsNotFound()

{

// Arrange

int id = 999;

\_productsUnitOfWorkMock.Setup(x => x.DeleteAsync(id))

.ReturnsAsync(new ActionResponse<Product>() { WasSuccess = false });

// Act

var result = await \_controller.DeleteAsync(id);

// Assert

Assert.IsInstanceOfType(result, typeof(NotFoundResult));

\_productsUnitOfWorkMock.Verify(x => x.DeleteAsync(id), Times.Once());

}

}

}

1. Corra los test y verifique que todo está funcionando correctamente.
2. Verificamos la cobertura del código.
3. Hacemos commit.

#### Unidad de Trabajo

1. Adicione la clase **ProductsUnitOfWorkTests**:

using Microsoft.AspNetCore.Mvc;

using Moq;

using Orders.Backend.Repositories.Interfaces;

using Orders.Backend.UnitsOfWork.Implementations;

using Orders.Shared.DTOs;

using Orders.Shared.Entities;

using Orders.Shared.Responses;

namespace Orders.Tests.UnitsOfWork

{

[TestClass]

public class ProductsUnitOfWorkTests

{

private Mock<IGenericRepository<Product>> \_repositoryMock = null!;

private Mock<IProductsRepository> \_productsRepositoryMock = null!;

private ProductsUnitOfWork \_unitOfWork = null!;

[TestInitialize]

public void SetUp()

{

\_repositoryMock = new Mock<IGenericRepository<Product>>();

\_productsRepositoryMock = new Mock<IProductsRepository>();

\_unitOfWork = new ProductsUnitOfWork(\_repositoryMock.Object, \_productsRepositoryMock.Object);

}

[TestMethod]

public async Task GetAsync\_WithPagination\_ReturnsProducts()

{

// Arrange

var pagination = new PaginationDTO();

var expectedActionResponse = new ActionResponse<IEnumerable<Product>> { WasSuccess = true };

\_productsRepositoryMock.Setup(x => x.GetAsync(pagination))

.ReturnsAsync(expectedActionResponse);

// Act

var result = await \_unitOfWork.GetAsync(pagination);

// Assert

Assert.AreEqual(expectedActionResponse, result);

\_productsRepositoryMock.Verify(x => x.GetAsync(pagination), Times.Once);

}

[TestMethod]

public async Task GetTotalPagesAsync\_ReturnsTotalPages()

{

// Arrange

var pagination = new PaginationDTO();

var expectedActionResponse = new ActionResponse<int> { WasSuccess = true };

\_productsRepositoryMock.Setup(x => x.GetTotalPagesAsync(pagination))

.ReturnsAsync(expectedActionResponse);

// Act

var result = await \_unitOfWork.GetTotalPagesAsync(pagination);

// Assert

Assert.AreEqual(expectedActionResponse, result);

\_productsRepositoryMock.Verify(x => x.GetTotalPagesAsync(pagination), Times.Once);

}

[TestMethod]

public async Task GetAsync\_ById\_ReturnsProduct()

{

// Arrange

var productId = 1;

var expectedActionResponse = new ActionResponse<Product> { WasSuccess = true };

\_productsRepositoryMock.Setup(x => x.GetAsync(productId))

.ReturnsAsync(expectedActionResponse);

// Act

var result = await \_unitOfWork.GetAsync(productId);

// Assert

Assert.AreEqual(expectedActionResponse, result);

\_productsRepositoryMock.Verify(x => x.GetAsync(productId), Times.Once);

}

[TestMethod]

public async Task AddFullAsync\_ReturnsProduct()

{

// Arrange

var productDTO = new ProductDTO();

var expectedActionResponse = new ActionResponse<Product> { WasSuccess = true };

\_productsRepositoryMock.Setup(x => x.AddFullAsync(productDTO))

.ReturnsAsync(expectedActionResponse);

// Act

var result = await \_unitOfWork.AddFullAsync(productDTO);

// Assert

Assert.AreEqual(expectedActionResponse, result);

\_productsRepositoryMock.Verify(x => x.AddFullAsync(productDTO), Times.Once);

}

[TestMethod]

public async Task UpdateFullAsync\_ReturnsProduct()

{

// Arrange

var productDTO = new ProductDTO();

var expectedActionResponse = new ActionResponse<Product> { WasSuccess = true };

\_productsRepositoryMock.Setup(x => x.UpdateFullAsync(productDTO))

.ReturnsAsync(expectedActionResponse);

// Act

var result = await \_unitOfWork.UpdateFullAsync(productDTO);

// Assert

Assert.AreEqual(expectedActionResponse, result);

\_productsRepositoryMock.Verify(x => x.UpdateFullAsync(productDTO), Times.Once);

}

[TestMethod]

public async Task AddImageAsync\_ReturnsImage()

{

// Arrange

var imageDTO = new ImageDTO();

var expectedActionResponse = new ActionResponse<ImageDTO> { WasSuccess = true };

\_productsRepositoryMock.Setup(x => x.AddImageAsync(imageDTO))

.ReturnsAsync(expectedActionResponse);

// Act

var result = await \_unitOfWork.AddImageAsync(imageDTO);

// Assert

Assert.AreEqual(expectedActionResponse, result);

\_productsRepositoryMock.Verify(x => x.AddImageAsync(imageDTO), Times.Once);

}

[TestMethod]

public async Task RemoveLastImageAsync\_ReturnsImage()

{

// Arrange

var imageDTO = new ImageDTO();

var expectedActionResponse = new ActionResponse<ImageDTO> { WasSuccess = true };

\_productsRepositoryMock.Setup(x => x.RemoveLastImageAsync(imageDTO))

.ReturnsAsync(expectedActionResponse);

// Act

var result = await \_unitOfWork.RemoveLastImageAsync(imageDTO);

// Assert

Assert.AreEqual(expectedActionResponse, result);

\_productsRepositoryMock.Verify(x => x.RemoveLastImageAsync(imageDTO), Times.Once);

}

[TestMethod]

public async Task DeleteAsync\_ExistingItem\_ReturnsSuccessResponse()

{

// Arrange

int id = 1;

\_productsRepositoryMock.Setup(x => x.DeleteAsync(id))

.ReturnsAsync(new ActionResponse<Product> { WasSuccess = true });

// Act

var response = await \_unitOfWork.DeleteAsync(id);

// Assert

Assert.IsTrue(response.WasSuccess);

\_productsRepositoryMock.Verify(x => x.DeleteAsync(id), Times.Once);

}

[TestMethod]

public async Task DeleteAsync\_NonExistingItem\_ReturnsFailureResponse()

{

// Arrange

int id = 999; // Make sure this ID does not exist in your test data

\_productsRepositoryMock.Setup(x => x.DeleteAsync(id))

.ReturnsAsync(new ActionResponse<Product> { WasSuccess = false });

// Act

var response = await \_unitOfWork.DeleteAsync(id);

// Assert

Assert.IsFalse(response.WasSuccess);

\_productsRepositoryMock.Verify(x => x.DeleteAsync(id), Times.Once);

}

}

}

1. Corra los test y verifique que todo está funcionando correctamente.
2. Verificamos la cobertura del código.
3. Hacemos commit.

#### Repositorio

1. Adicione la clase **ProductsRepositoryTests**:

using Microsoft.EntityFrameworkCore;

using Moq;

using Orders.Backend.Data;

using Orders.Backend.Helpers;

using Orders.Backend.Repositories.Implementations;

using Orders.Shared.DTOs;

using Orders.Shared.Entities;

using Orders.Tests.Shared;

namespace Orders.Tests.Repositories

{

[TestClass]

public class ProductsRepositoryTests

{

private DataContext \_context = null!;

private ProductsRepository \_repository = null!;

private Mock<IFileStorage> \_fileStorageMock = null!;

private DbContextOptions<DataContext> \_options = null!;

private const string \_string64base = "U29tZVZhbGlkQmFzZTY0U3RyaW5n";

private const string \_container = "products";

[TestInitialize]

public void SetUp()

{

\_options = new DbContextOptionsBuilder<DataContext>()

.UseInMemoryDatabase(databaseName: "TestDatabase")

.Options;

\_context = new DataContext(\_options);

\_fileStorageMock = new Mock<IFileStorage>();

\_repository = new ProductsRepository(\_context, \_fileStorageMock.Object);

PopulateData();

}

[TestCleanup]

public void TearDown()

{

\_context.Database.EnsureDeleted();

\_context.Dispose();

}

[TestMethod]

public async Task AddImagesAsync\_ProductNotFound\_ReturnsError()

{

// Arrange

var imageDto = new ImageDTO { ProductId = 999 };

// Act

var result = await \_repository.AddImageAsync(imageDto);

// Assert

Assert.IsFalse(result.WasSuccess);

}

[TestMethod]

public async Task AddImageAsync\_WithValidData\_AddsImage()

{

// Arrange

var imageDTO = new ImageDTO

{

ProductId = 1,

Images = new List<string> { \_string64base }

};

\_fileStorageMock.Setup(fs => fs.SaveFileAsync(It.IsAny<byte[]>(), ".jpg", \_container))

.ReturnsAsync("storedImagePath");

// Act

var result = await \_repository.AddImageAsync(imageDTO);

// Assert

Assert.IsTrue(result.WasSuccess);

Assert.IsTrue(result.Result!.Images[0].Contains("storedImagePath"));

\_fileStorageMock.Verify(x => x.SaveFileAsync(It.IsAny<byte[]>(), ".jpg", \_container), Times.Once());

}

[TestMethod]

public async Task RemoveLastImageAsync\_ProductNotFound\_ReturnsError()

{

// Arrange

var imageDto = new ImageDTO { ProductId = 999 };

// Act

var result = await \_repository.RemoveLastImageAsync(imageDto);

// Assert

Assert.IsFalse(result.WasSuccess);

}

[TestMethod]

public async Task RemoveLastImageAsync\_NoImages\_ReturnsOk()

{

// Arrange

var imageDto = new ImageDTO { ProductId = 1 };

// Act

var result = await \_repository.RemoveLastImageAsync(imageDto);

// Assert

Assert.IsTrue(result.WasSuccess);

}

[TestMethod]

public async Task RemoveLastImageAsync\_RemovesLastImage\_ReturnsOk()

{

// Arrange

var imagePath = "https//image2.jpg";

\_fileStorageMock.Setup(fs => fs.RemoveFileAsync(imagePath, \_container))

.Returns(Task.CompletedTask);

var imageDto = new ImageDTO { ProductId = 2 };

// Act

var result = await \_repository.RemoveLastImageAsync(imageDto);

// Assert

Assert.IsTrue(result.WasSuccess);

Assert.AreEqual(1, result.Result!.Images.Count);

\_fileStorageMock.Verify(x => x.RemoveFileAsync(imagePath, \_container), Times.Once());

}

[TestMethod]

public async Task GetAsync\_WithoutFilter\_ReturnsAllProducts()

{

// Arrange

var pagination = new PaginationDTO { RecordsNumber = 10, Page = 1 };

// Act

var result = await \_repository.GetAsync(pagination);

// Assert

Assert.IsTrue(result.WasSuccess);

var products = result.Result as List<Product>;

Assert.AreEqual(2, products!.Count);

}

[TestMethod]

public async Task GetAsync\_WithPagination\_ReturnsProducts()

{

// Arrange

var pagination = new PaginationDTO { Filter = "Some", CategoryFilter = "Any" };

// Act

var result = await \_repository.GetAsync(pagination);

// Assert

Assert.IsTrue(result.WasSuccess);

}

[TestMethod]

public async Task GetTotalPagesAsync\_ReturnsTotalPages()

{

// Arrange

var pagination = new PaginationDTO { Filter = "Some", CategoryFilter = "Any" };

// Act

var result = await \_repository.GetTotalPagesAsync(pagination);

// Assert

Assert.IsTrue(result.WasSuccess);

}

[TestMethod]

public async Task GetAsync\_ValidId\_ReturnsProduct()

{

// Act

var result = await \_repository.GetAsync(1);

// Assert

Assert.IsTrue(result.WasSuccess);

Assert.AreEqual("Product A", result.Result!.Name);

}

[TestMethod]

public async Task GetAsync\_InvalidId\_ReturnsError()

{

// Act

var result = await \_repository.GetAsync(999);

// Assert

Assert.IsFalse(result.WasSuccess);

}

[TestMethod]

public async Task AddFullAsync\_ValidDTO\_ReturnsOk()

{

// Arrange

\_fileStorageMock.Setup(fs => fs.SaveFileAsync(It.IsAny<byte[]>(), ".jpg", \_container))

.ReturnsAsync("testImage.jpg");

var productDTO = new ProductDTO

{

Name = "TestProduct",

Description = "Description",

Price = 100.00M,

Stock = 10,

ProductImages = new List<string> { \_string64base },

ProductCategoryIds = new List<int> { 1 }

};

// Act

var result = await \_repository.AddFullAsync(productDTO);

// Assert

Assert.IsTrue(result.WasSuccess);

Assert.AreEqual("TestProduct", result.Result!.Name);

\_fileStorageMock.Verify(x => x.SaveFileAsync(It.IsAny<byte[]>(), ".jpg", \_container), Times.Once());

}

[TestMethod]

public async Task AddFullAsync\_DuplicateName\_ReturnsErrors()

{

// Arrange

var productDTO = new ProductDTO

{

Name = "Product A",

Description = "Product A",

Price = 100.00M,

Stock = 10,

ProductImages = new List<string> { \_string64base },

ProductCategoryIds = new List<int> { 1 }

};

// Act

var result = await \_repository.AddFullAsync(productDTO);

// Assert

Assert.IsFalse(result.WasSuccess);

Assert.AreEqual("Ya existe un producto con el mismo nombre.", result.Message);

}

[TestMethod]

public async Task AddFullAsync\_GeneralException\_ReturnsErrors()

{

// Arrange

var productDTO = new ProductDTO

{

Name = "Product A",

Description = "Product A",

Price = 100.00M,

Stock = 10,

ProductImages = new List<string> { \_string64base },

ProductCategoryIds = new List<int> { 1 }

};

var message = "Test exception";

\_fileStorageMock.Setup(fs => fs.SaveFileAsync(It.IsAny<byte[]>(), ".jpg", \_container))

.Throws(new Exception(message));

// Act

var result = await \_repository.AddFullAsync(productDTO);

// Assert

Assert.IsFalse(result.WasSuccess);

Assert.AreEqual(message, result.Message);

\_fileStorageMock.Verify(x => x.SaveFileAsync(It.IsAny<byte[]>(), ".jpg", \_container), Times.Once());

}

[TestMethod]

public async Task UpdateFullAsync\_ValidDTO\_UpdatesProduct()

{

// Arrange

var productDTO = new ProductDTO

{

Id = 1,

Name = "NewName",

Description = "NewDescription",

Price = 100.00M,

Stock = 10,

ProductCategoryIds = new List<int> { 2 }

};

// Act

var result = await \_repository.UpdateFullAsync(productDTO);

// Assert

//Assert.IsTrue(result.WasSuccess);

//Assert.AreEqual("NewName", result.Result!.Name);

}

[TestMethod]

public async Task UpdateFullAsync\_NonExistingProduct\_ReturnsError()

{

// Arrange

var productDTO = new ProductDTO

{

Id = 999,

Name = "TestName",

Description = "TestDescription",

Price = 100.00M,

Stock = 10

};

// Act

var result = await \_repository.UpdateFullAsync(productDTO);

// Assert

Assert.IsFalse(result.WasSuccess);

}

[TestMethod]

public async Task UpdateFullAsync\_GeneralException\_ReturnsError()

{

// Arrange

var exceptionalContext = new ExceptionalDataContext(\_options);

var repository = new ProductsRepository(exceptionalContext, \_fileStorageMock.Object);

var productDTO = new ProductDTO

{

Id = 1,

Name = "DuplicateName",

Description = "Description",

Price = 100.00M,

Stock = 10,

ProductCategoryIds = new List<int> { 2 }

};

// Act

var result = await repository.UpdateFullAsync(productDTO);

// Assert

Assert.IsFalse(result.WasSuccess);

Assert.AreEqual("Test Exception", result.Message);

}

[TestMethod]

public async Task UpdateFullAsync\_DbUpdateException\_ReturnsError()

{

// Arrange

var exceptionalContext = new ExceptionalDBUpdateDataContextWithInnerException(\_options);

var repository = new ProductsRepository(exceptionalContext, \_fileStorageMock.Object);

var productDTO = new ProductDTO

{

Id = 1,

Name = "DuplicateName",

Description = "Description",

Price = 100.00M,

Stock = 10,

ProductCategoryIds = new List<int> { 2 }

};

// Act

var result = await repository.UpdateFullAsync(productDTO);

// Assert

Assert.IsFalse(result.WasSuccess);

Assert.AreEqual("Ya existe un producto con el mismo nombre.", result.Message);

}

[TestMethod]

public async Task DeleteAsync\_ExistingItem\_ReturnsSuccessResponse()

{

// Arrange

int id = 2;

// Act

var response = await \_repository.DeleteAsync(id);

// Assert

Assert.IsTrue(response.WasSuccess);

}

[TestMethod]

public async Task DeleteAsync\_NonExistingItem\_ReturnsNotFoundResponse()

{

// Arrange

int nonExistingId = 999;

// Act

var response = await \_repository.DeleteAsync(nonExistingId);

// Assert

Assert.IsFalse(response.WasSuccess);

}

[TestMethod]

public async Task DeleteAsync\_FailureDueToRelatedRecords\_ReturnsFailureResponse()

{

// Arrange

int id = 1;

// Act

var response = await \_repository.DeleteAsync(id);

// Assert

Assert.IsFalse(response.WasSuccess);

}

private void PopulateData()

{

var category1 = new Category { Id = 1, Name = "Category1" };

var category2 = new Category { Id = 2, Name = "Category2" };

\_context.Categories.AddRange(category1, category2);

\_context.SaveChanges();

var product1 = new Product

{

Id = 1,

Name = "Product A",

Description = "Product A",

ProductCategories = new List<ProductCategory> { new ProductCategory { Category = category1 } }

};

var product2 = new Product

{

Id = 2,

Name = "Product B",

Description = "Product B",

ProductCategories = new List<ProductCategory> { new ProductCategory { Category = category1 } },

ProductImages = new List<ProductImage>

{

new ProductImage { Image = "https//image1.jpg" },

new ProductImage { Image = "https//image2.jpg" }

}

};

\_context.Products.AddRange(product1, product2);

var temporalOrder = new TemporalOrder

{

Product = product1,

Quantity = 1,

User = new User { Address = "some", Document = "any", FirstName = "John", LastName = "Doe" }

};

\_context.TemporalOrders.Add(temporalOrder);

\_context.SaveChanges();

}

}

}

1. Corra los test y verifique que todo está funcionando correctamente.
2. Verificamos la cobertura del código.
3. Hacemos commit.

### Cuentas

#### Controlador

1. Adicione la clase **AccountsControllerTests**:

using System.Security.Claims;

using Microsoft.AspNetCore.Http;

using Microsoft.AspNetCore.Identity;

using Microsoft.AspNetCore.Mvc;

using Microsoft.AspNetCore.Mvc.Routing;

using Microsoft.Extensions.Configuration;

using Moq;

using Orders.Backend.Controllers;

using Orders.Backend.Helpers;

using Orders.Backend.Repositories.Interfaces;

using Orders.Backend.UnitsOfWork.Interfaces;

using Orders.Shared.DTOs;

using Orders.Shared.Entities;

using Orders.Shared.Enums;

using Orders.Shared.Responses;

using SignInResult = Microsoft.AspNetCore.Identity.SignInResult;

namespace Orders.Tests.Controllers

{

[TestClass]

public class AccountsControllerTests

{

private Mock<IUsersUnitOfWork> \_mockUsersUnitOfWork = null!;

private Mock<IConfiguration> \_mockConfiguration = null!;

private Mock<IFileStorage> \_mockFileStorage = null!;

private Mock<IMailHelper> \_mockMailHelper = null!;

private Mock<IUsersRepository> \_mockUsersRepository = null!;

private AccountsController \_controller = null!;

private const string \_container = "userphotos";

private const string \_string64base = "U29tZVZhbGlkQmFzZTY0U3RyaW5n";

[TestInitialize]

public void Initialize()

{

\_mockUsersUnitOfWork = new Mock<IUsersUnitOfWork>();

\_mockConfiguration = new Mock<IConfiguration>();

\_mockFileStorage = new Mock<IFileStorage>();

\_mockMailHelper = new Mock<IMailHelper>();

\_mockUsersRepository = new Mock<IUsersRepository>();

\_mockConfiguration

.SetupGet(x => x["Url Frontend"])

.Returns("http://frontend-url.com");

\_mockConfiguration

.SetupGet(x => x["jwtKey"])

.Returns("xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzz");

var mockUrlHelper = new Mock<IUrlHelper>();

mockUrlHelper

.Setup(u => u.Action(It.IsAny<UrlActionContext>()))

.Returns("http://generated-link.com");

\_controller = new AccountsController(

\_mockUsersUnitOfWork.Object,

\_mockConfiguration.Object,

\_mockFileStorage.Object,

\_mockMailHelper.Object,

\_mockUsersRepository.Object)

{

Url = mockUrlHelper.Object

};

var mockHttpContext = new Mock<HttpContext>();

var mockHttpRequest = new Mock<HttpRequest>();

mockHttpRequest.Setup(req => req.Scheme)

.Returns("http");

mockHttpContext.Setup(ctx => ctx.Request)

.Returns(mockHttpRequest.Object);

\_controller.ControllerContext = new ControllerContext

{

HttpContext = mockHttpContext.Object

};

var user = new ClaimsPrincipal(new ClaimsIdentity(new Claim[]

{

new Claim(ClaimTypes.Name, "test@example.com"),

}, "mock"));

\_controller.ControllerContext.HttpContext = new DefaultHttpContext() { User = user };

}

[TestMethod]

public async Task GetAsync\_ShouldReturnOk\_WhenUsersAreFound()

{

// Arrange

var pagination = new PaginationDTO();

var response = new ActionResponse<IEnumerable<User>> { WasSuccess = true };

\_mockUsersRepository.Setup(x => x.GetAsync(pagination))

.ReturnsAsync(response);

// Act

var result = await \_controller.GetAsync(pagination);

// Assert

Assert.IsInstanceOfType(result, typeof(OkObjectResult));

\_mockUsersRepository.Verify(x => x.GetAsync(pagination), Times.Once());

}

[TestMethod]

public async Task GetAsync\_ShouldReturnBadRequest\_WhenUsersAreNotFound()

{

// Arrange

var pagination = new PaginationDTO();

var response = new ActionResponse<IEnumerable<User>> { WasSuccess = false };

\_mockUsersRepository.Setup(x => x.GetAsync(pagination))

.ReturnsAsync(response);

// Act

var result = await \_controller.GetAsync(pagination);

// Assert

Assert.IsInstanceOfType(result, typeof(BadRequestResult));

\_mockUsersRepository.Verify(x => x.GetAsync(pagination), Times.Once());

}

[TestMethod]

public async Task GetPagesAsync\_ShouldReturnOk\_WhenTotalPagesAreSuccessfullyRetrieved()

{

// Arrange

var pagination = new PaginationDTO();

var response = new ActionResponse<int> { WasSuccess = true, Result = 5 };

\_mockUsersRepository.Setup(x => x.GetTotalPagesAsync(pagination))

.ReturnsAsync(response);

// Act

var result = await \_controller.GetPagesAsync(pagination);

// Assert

var okResult = result as OkObjectResult;

Assert.IsNotNull(okResult);

Assert.AreEqual(200, okResult.StatusCode);

Assert.AreEqual(5, okResult.Value);

\_mockUsersRepository.Verify(x => x.GetTotalPagesAsync(pagination), Times.Once());

}

[TestMethod]

public async Task GetPagesAsync\_ShouldReturnBadRequest\_WhenUnableToRetrieveTotalPages()

{

// Arrange

var pagination = new PaginationDTO();

var response = new ActionResponse<int> { WasSuccess = false };

\_mockUsersRepository.Setup(x => x.GetTotalPagesAsync(pagination))

.ReturnsAsync(response);

// Act

var result = await \_controller.GetPagesAsync(pagination);

// Assert

Assert.IsInstanceOfType(result, typeof(BadRequestResult));

var badRequestResult = result as BadRequestResult;

Assert.IsNotNull(badRequestResult);

Assert.AreEqual(400, badRequestResult.StatusCode);

\_mockUsersRepository.Verify(x => x.GetTotalPagesAsync(pagination), Times.Once());

}

[TestMethod]

public async Task CreateUser\_ShouldReturnNoContent\_WhenUserIsCreatedSuccessfully()

{

// Arrange

var userDTO = new UserDTO

{

Password = "password123",

Photo = \_string64base,

Address = "Some",

CityId = 1,

Document = "Any",

Email = "Some",

FirstName = "Test",

Id = "123",

LastName = "Test",

PasswordConfirm = "password123",

PhoneNumber = "Any",

UserName = "Test",

UserType = UserType.User

};

var user = new User();

\_mockFileStorage.Setup(x => x.SaveFileAsync(It.IsAny<byte[]>(), ".jpg", \_container))

.ReturnsAsync("photoUrl");

\_mockUsersUnitOfWork.Setup(x => x.AddUserAsync(It.IsAny<User>(), userDTO.Password))

.ReturnsAsync(IdentityResult.Success);

\_mockUsersUnitOfWork.Setup(x => x.AddUserToRoleAsync(It.IsAny<User>(), It.IsAny<string>()))

.Returns(Task.CompletedTask);

\_mockUsersUnitOfWork.Setup(x => x.GenerateEmailConfirmationTokenAsync(It.IsAny<User>()))

.ReturnsAsync("token");

var response = new ActionResponse<string> { WasSuccess = true };

\_mockMailHelper.Setup(x => x.SendMail(It.IsAny<string>(), It.IsAny<string>(), It.IsAny<string>(), It.IsAny<string>()))

.Returns(response);

// Act

var result = await \_controller.CreateUser(userDTO);

// Assert

Assert.IsInstanceOfType(result, typeof(NoContentResult));

\_mockUsersUnitOfWork.Verify(x => x.AddUserAsync(It.IsAny<User>(), userDTO.Password), Times.Once());

\_mockUsersUnitOfWork.Verify(x => x.AddUserToRoleAsync(It.IsAny<User>(), It.IsAny<string>()), Times.Once());

\_mockUsersUnitOfWork.Verify(x => x.GenerateEmailConfirmationTokenAsync(It.IsAny<User>()), Times.Once());

\_mockMailHelper.Verify(x => x.SendMail(It.IsAny<string>(), It.IsAny<string>(), It.IsAny<string>(), It.IsAny<string>()), Times.Once());

}

[TestMethod]

public async Task CreateUser\_ShouldReturnBadRequest\_WhenUserCreationFails()

{

// Arrange

var userDTO = new UserDTO();

var identityErrors = new List<IdentityError> { new IdentityError { Description = "User creation failed" } };

\_mockUsersUnitOfWork.Setup(x => x.AddUserAsync(It.IsAny<User>(), It.IsAny<string>()))

.ReturnsAsync(IdentityResult.Failed(identityErrors.ToArray()));

// Act

var result = await \_controller.CreateUser(userDTO);

// Assert

Assert.IsInstanceOfType(result, typeof(BadRequestObjectResult));

\_mockUsersUnitOfWork.Verify(x => x.AddUserAsync(It.IsAny<User>(), It.IsAny<string>()), Times.Once());

}

[TestMethod]

public async Task CreateUser\_ShouldReturnBadRequest\_WhenEmailNotSent()

{

// Arrange

var userDTO = new UserDTO { Password = "password123", Photo = \_string64base };

var user = new User();

\_mockFileStorage.Setup(x => x.SaveFileAsync(It.IsAny<byte[]>(), ".jpg", \_container))

.ReturnsAsync("photoUrl");

\_mockUsersUnitOfWork.Setup(x => x.AddUserAsync(It.IsAny<User>(), userDTO.Password))

.ReturnsAsync(IdentityResult.Success);

\_mockUsersUnitOfWork.Setup(x => x.AddUserToRoleAsync(It.IsAny<User>(), It.IsAny<string>()))

.Returns(Task.CompletedTask);

\_mockUsersUnitOfWork.Setup(x => x.GenerateEmailConfirmationTokenAsync(It.IsAny<User>()))

.ReturnsAsync("token");

var response = new ActionResponse<string> { WasSuccess = false };

\_mockMailHelper.Setup(x => x.SendMail(It.IsAny<string>(), It.IsAny<string>(), It.IsAny<string>(), It.IsAny<string>()))

.Returns(response);

// Act

var result = await \_controller.CreateUser(userDTO);

// Assert

Assert.IsInstanceOfType(result, typeof(BadRequestObjectResult));

\_mockUsersUnitOfWork.Verify(x => x.AddUserAsync(It.IsAny<User>(), userDTO.Password), Times.Once());

\_mockUsersUnitOfWork.Verify(x => x.AddUserToRoleAsync(It.IsAny<User>(), It.IsAny<string>()), Times.Once());

\_mockUsersUnitOfWork.Verify(x => x.GenerateEmailConfirmationTokenAsync(It.IsAny<User>()), Times.Once());

\_mockMailHelper.Verify(x => x.SendMail(It.IsAny<string>(), It.IsAny<string>(), It.IsAny<string>(), It.IsAny<string>()), Times.Once());

}

[TestMethod]

public async Task ConfirmEmailAsync\_UserNotFound\_ReturnsNotFound()

{

// Act

var result = await \_controller.ConfirmEmailAsync(Guid.NewGuid().ToString(), "token");

// Assert

Assert.IsInstanceOfType(result, typeof(NotFoundResult));

}

[TestMethod]

public async Task ConfirmEmailAsync\_InvalidToken\_ReturnsBadRequest()

{

// Arrange

var user = new User();

var message = "Invalid token";

var token = "token";

var identityErrors = new List<IdentityError> { new IdentityError { Description = message } };

\_mockUsersUnitOfWork.Setup(x => x.GetUserAsync(It.IsAny<Guid>()))

.ReturnsAsync(user);

\_mockUsersUnitOfWork.Setup(x => x.ConfirmEmailAsync(user, token.Replace(" ", "+")))

.ReturnsAsync(IdentityResult.Failed(identityErrors.ToArray()));

// Act

var result = await \_controller.ConfirmEmailAsync(Guid.NewGuid().ToString(), token);

// Assert

var badRequestResult = result as BadRequestObjectResult;

Assert.IsNotNull(badRequestResult);

\_mockUsersUnitOfWork.Verify(x => x.GetUserAsync(It.IsAny<Guid>()), Times.Once());

\_mockUsersUnitOfWork.Verify(x => x.ConfirmEmailAsync(user, token.Replace(" ", "+")), Times.Once());

}

[TestMethod]

public async Task ConfirmEmailAsync\_ValidToken\_ReturnsNoContent()

{

// Arrange

var user = new User();

var token = "token";

\_mockUsersUnitOfWork.Setup(x => x.GetUserAsync(It.IsAny<Guid>()))

.ReturnsAsync(user);

\_mockUsersUnitOfWork.Setup(x => x.ConfirmEmailAsync(user, token.Replace(" ", "+")))

.ReturnsAsync(IdentityResult.Success);

// Act

var result = await \_controller.ConfirmEmailAsync(Guid.NewGuid().ToString(), token);

// Assert

Assert.IsInstanceOfType(result, typeof(NoContentResult));

\_mockUsersUnitOfWork.Verify(x => x.GetUserAsync(It.IsAny<Guid>()), Times.Once());

\_mockUsersUnitOfWork.Verify(x => x.ConfirmEmailAsync(user, token.Replace(" ", "+")), Times.Once());

}

[TestMethod]

public async Task Login\_Success\_ReturnsOk()

{

// Arrange

var user = new User

{

Email = "some@yopmail.com",

UserType = UserType.User,

Document = "123",

FirstName = "John",

LastName = "Doe",

Address = "Any",

Photo = \_string64base,

CityId = 1

};

var loginModel = new LoginDTO { Email = user.Email, Password = "123456" };

\_mockUsersUnitOfWork.Setup(x => x.LoginAsync(loginModel))

.ReturnsAsync(SignInResult.Success);

\_mockUsersUnitOfWork.Setup(x => x.GetUserAsync(user.Email))

.ReturnsAsync(user);

// Act

var result = await \_controller.LoginAsync(loginModel);

// Assert

Assert.IsInstanceOfType(result, typeof(OkObjectResult));

\_mockUsersUnitOfWork.Verify(x => x.LoginAsync(loginModel), Times.Once());

\_mockUsersUnitOfWork.Verify(x => x.GetUserAsync(user.Email), Times.Once());

}

[TestMethod]

public async Task Login\_LockedOut\_ReturnsBadRequest()

{

// Arrange

var loginDto = new LoginDTO { Email = "test@test.com", Password = "Test1234!" };

\_mockUsersUnitOfWork.Setup(x => x.LoginAsync(loginDto))

.ReturnsAsync(SignInResult.LockedOut);

// Act

var result = await \_controller.LoginAsync(loginDto);

// Assert

var badRequestResult = result as BadRequestObjectResult;

Assert.IsNotNull(badRequestResult);

Assert.AreEqual("Ha superado el máximo número de intentos, su cuenta está bloqueada, intente de nuevo en 5 minutos.", badRequestResult.Value);

\_mockUsersUnitOfWork.Verify(x => x.LoginAsync(loginDto), Times.Once());

}

[TestMethod]

public async Task Login\_NotAllowed\_ReturnsBadRequest()

{

// Arrange

var loginDto = new LoginDTO { Email = "test@test.com", Password = "Test1234!" };

\_mockUsersUnitOfWork.Setup(x => x.LoginAsync(loginDto))

.ReturnsAsync(SignInResult.NotAllowed);

// Act

var result = await \_controller.LoginAsync(loginDto);

// Assert

var badRequestResult = result as BadRequestObjectResult;

Assert.IsNotNull(badRequestResult);

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

\_mockUsersUnitOfWork.Verify(x => x.LoginAsync(loginDto), Times.Once());

}

[TestMethod]

public async Task Login\_InvalidCredentials\_ReturnsBadRequest()

{

// Arrange

var loginDto = new LoginDTO { Email = "test@test.com", Password = "Test1234!" };

\_mockUsersUnitOfWork.Setup(x => x.LoginAsync(loginDto))

.ReturnsAsync(SignInResult.Failed);

// Act

var result = await \_controller.LoginAsync(loginDto);

// Assert

var badRequestResult = result as BadRequestObjectResult;

Assert.IsNotNull(badRequestResult);

Assert.AreEqual("Email o contraseña incorrectos.", badRequestResult.Value);

\_mockUsersUnitOfWork.Verify(x => x.LoginAsync(loginDto), Times.Once());

}

[TestMethod]

public async Task PutAsync\_UserNotFound\_ReturnsNotFound()

{

// Arrange

var userName = "testuser";

\_controller.ControllerContext = GetControllerContext(userName);

// Act

var result = await \_controller.PutAsync(new User());

// Assert

Assert.IsInstanceOfType(result, typeof(NotFoundResult));

}

[TestMethod]

public async Task PutAsync\_ExceptionThrown\_ReturnsBadRequest()

{

// Arrange

var message = "Test exception";

var userName = "testuser";

\_controller.ControllerContext = GetControllerContext(userName);

\_mockUsersUnitOfWork.Setup(x => x.GetUserAsync(userName))

.Throws(new Exception(message));

// Act

var result = await \_controller.PutAsync(new User());

var badRequestResult = result as BadRequestObjectResult;

// Assert

Assert.IsInstanceOfType(result, typeof(BadRequestObjectResult));

Assert.AreEqual(message, badRequestResult!.Value);

}

[TestMethod]

public async Task PutAsync\_UserPhotoNotEmpty\_UpdatesPhoto()

{

// Arrange

var user = new User

{

Email = "some@yopmail.com",

UserType = UserType.User,

Document = "123",

FirstName = "John",

LastName = "Doe",

Address = "Any",

Photo = \_string64base,

CityId = 1

};

var currentUser = new User

{

Email = "some@yopmail.com",

UserType = UserType.User,

Document = "123",

FirstName = "John",

LastName = "Doe",

Address = "Any",

Photo = "oldPhoto",

CityId = 1

};

var userName = "testuser";

var newPhotoUrl = "newPhotoUrl";

var mockIdentityResult = IdentityResult.Success;

\_controller.ControllerContext = GetControllerContext(userName);

\_mockUsersUnitOfWork.Setup(x => x.GetUserAsync(userName))

.ReturnsAsync(currentUser);

\_mockFileStorage.Setup(fs => fs.SaveFileAsync(It.IsAny<byte[]>(), ".jpg", \_container))

.ReturnsAsync(newPhotoUrl);

\_mockUsersUnitOfWork.Setup(x => x.UpdateUserAsync(currentUser))

.ReturnsAsync(mockIdentityResult);

// Act

var result = await \_controller.PutAsync(user);

var okResult = result as OkObjectResult;

var token = okResult?.Value as TokenDTO;

// Assert

Assert.IsNotNull(token!.Token);

\_mockUsersUnitOfWork.Verify(x => x.GetUserAsync(userName), Times.Once());

\_mockUsersUnitOfWork.Verify(x => x.UpdateUserAsync(currentUser), Times.Once());

}

[TestMethod]

public async Task PutAsync\_PhotoUpdateException\_ReturnsBadRequest()

{

// Arrange

var user = new User { Photo = \_string64base };

var userName = "testuser";

var message = "Photo upload failed";

\_controller.ControllerContext = GetControllerContext(userName);

\_mockUsersUnitOfWork.Setup(x => x.GetUserAsync(userName))

.ReturnsAsync(new User());

\_mockFileStorage.Setup(fs => fs.SaveFileAsync(It.IsAny<byte[]>(), ".jpg", \_container))

.Throws(new Exception(message));

// Act

var result = await \_controller.PutAsync(user);

// Assert

Assert.IsInstanceOfType(result, typeof(BadRequestObjectResult));

\_mockUsersUnitOfWork.Verify(x => x.GetUserAsync(userName), Times.Once());

}

[TestMethod]

public async Task PutAsync\_UpdateUserFails\_ReturnsBadRequest()

{

// Arrange

var user = new User();

var currentUser = new User();

var identityError = new IdentityError { Description = "Update failed" };

var userName = "testuser";

\_controller.ControllerContext = GetControllerContext(userName);

\_mockUsersUnitOfWork.Setup(x => x.GetUserAsync(userName))

.ReturnsAsync(currentUser);

\_mockUsersUnitOfWork.Setup(x => x.UpdateUserAsync(It.IsAny<User>()))

.ReturnsAsync(IdentityResult.Failed(identityError));

// Act

var result = await \_controller.PutAsync(user);

// Assert

Assert.IsInstanceOfType(result, typeof(BadRequestObjectResult));

\_mockUsersUnitOfWork.Verify(x => x.GetUserAsync(userName), Times.Once());

\_mockUsersUnitOfWork.Verify(x => x.UpdateUserAsync(It.IsAny<User>()), Times.Once());

}

[TestMethod]

public async Task RecoverPassword\_UserNotFound\_ReturnsNotFound()

{

// Arrange

var userName = "test@example.com";

// Act

var result = await \_controller.RecoverPasswordAsync(new EmailDTO { Email = userName });

// Assert

Assert.IsInstanceOfType(result, typeof(NotFoundResult));

\_mockUsersUnitOfWork.Verify(x => x.GetUserAsync(userName), Times.Once());

}

[TestMethod]

public async Task RecoverPassword\_EmailSentSuccessfully\_ReturnsNoContent()

{

// Arrange

var user = new User { Email = "test@example.com" };

\_mockUsersUnitOfWork.Setup(x => x.GetUserAsync(user.Email))

.ReturnsAsync(user);

\_mockUsersUnitOfWork.Setup(x => x.GeneratePasswordResetTokenAsync(user))

.ReturnsAsync("GeneratedToken");

var response = new ActionResponse<string> { WasSuccess = true };

\_mockMailHelper.Setup(x => x.SendMail(It.IsAny<string>(), It.IsAny<string>(), It.IsAny<string>(), It.IsAny<string>()))

.Returns(response);

// Act

var result = await \_controller.RecoverPasswordAsync(new EmailDTO { Email = user.Email });

// Assert

Assert.IsInstanceOfType(result, typeof(NoContentResult));

\_mockUsersUnitOfWork.Verify(x => x.GetUserAsync(user.Email), Times.Once());

\_mockUsersUnitOfWork.Verify(x => x.GeneratePasswordResetTokenAsync(user), Times.Once());

\_mockMailHelper.Verify(x => x.SendMail(It.IsAny<string>(), It.IsAny<string>(), It.IsAny<string>(), It.IsAny<string>()), Times.Once());

}

[TestMethod]

public async Task RecoverPassword\_EmailFailedWithMessage\_ReturnsBadRequestWithMessage()

{

// Arrange

var user = new User { Email = "test@example.com" };

var message = "Failed to send";

\_mockUsersUnitOfWork.Setup(x => x.GetUserAsync(user.Email))

.ReturnsAsync(user);

\_mockUsersUnitOfWork.Setup(x => x.GeneratePasswordResetTokenAsync(user))

.ReturnsAsync("GeneratedToken");

var response = new ActionResponse<string> { WasSuccess = false, Message = message };

\_mockMailHelper.Setup(x => x.SendMail(It.IsAny<string>(), It.IsAny<string>(), It.IsAny<string>(), It.IsAny<string>()))

.Returns(response);

// Act

var result = await \_controller.RecoverPasswordAsync(new EmailDTO { Email = user.Email });

var badRequest = result as BadRequestObjectResult;

// Assert

Assert.IsInstanceOfType(result, typeof(BadRequestObjectResult));

Assert.AreEqual(message, badRequest!.Value);

\_mockUsersUnitOfWork.Verify(x => x.GetUserAsync(user.Email), Times.Once());

\_mockUsersUnitOfWork.Verify(x => x.GeneratePasswordResetTokenAsync(user), Times.Once());

\_mockMailHelper.Verify(x => x.SendMail(It.IsAny<string>(), It.IsAny<string>(), It.IsAny<string>(), It.IsAny<string>()), Times.Once());

}

[TestMethod]

public async Task GetAsync\_UserExists\_ReturnsOkWithUser()

{

// Arrange

var user = new User();

\_mockUsersUnitOfWork.Setup(x => x.GetUserAsync("test@example.com")).ReturnsAsync(user);

// Act

var result = await \_controller.GetAsync();

// Assert

var okResult = result as OkObjectResult;

Assert.IsNotNull(okResult);

Assert.AreEqual(user, okResult.Value);

}

[TestMethod]

public async Task GetAsync\_UserDoesNotExist\_ReturnsOkWithNull()

{

// Act

var result = await \_controller.GetAsync();

// Assert

var okResult = result as OkObjectResult;

Assert.IsNotNull(okResult);

Assert.IsNull(okResult.Value);

}

[TestMethod]

public async Task ResetPassword\_UserNotFound\_ReturnsNotFound()

{

// Act

var result = await \_controller.ResetPasswordAsync(new ResetPasswordDTO());

// Assert

Assert.IsInstanceOfType(result, typeof(NotFoundResult));

\_mockUsersUnitOfWork.Verify(x => x.GetUserAsync(It.IsAny<string>()), Times.Once());

}

[TestMethod]

public async Task ResetPassword\_ValidReset\_ReturnsNoContent()

{

// Arrange

var mockUser = new User();

var mockIdentityResult = IdentityResult.Success;

\_mockUsersUnitOfWork.Setup(x => x.GetUserAsync(It.IsAny<string>()))

.ReturnsAsync(mockUser);

\_mockUsersUnitOfWork.Setup(x => x.ResetPasswordAsync(It.IsAny<User>(), It.IsAny<string>(), It.IsAny<string>()))

.ReturnsAsync(mockIdentityResult);

// Act

var result = await \_controller.ResetPasswordAsync(new ResetPasswordDTO());

// Assert

Assert.IsInstanceOfType(result, typeof(NoContentResult));

\_mockUsersUnitOfWork.Verify(x => x.GetUserAsync(It.IsAny<string>()), Times.Once());

\_mockUsersUnitOfWork.Verify(x => x.ResetPasswordAsync(It.IsAny<User>(), It.IsAny<string>(), It.IsAny<string>()), Times.Once());

}

[TestMethod]

public async Task ResetPassword\_InvalidReset\_ReturnsBadRequest()

{

// Arrange

var description = "Test error";

var mockUser = new User();

var mockIdentityErrors = new List<IdentityError>

{

new IdentityError { Description = description }

};

var mockIdentityResult = IdentityResult.Failed(mockIdentityErrors.ToArray());

\_mockUsersUnitOfWork.Setup(x => x.GetUserAsync(It.IsAny<string>()))

.ReturnsAsync(mockUser);

\_mockUsersUnitOfWork.Setup(x => x.ResetPasswordAsync(It.IsAny<User>(), It.IsAny<string>(), It.IsAny<string>()))

.ReturnsAsync(mockIdentityResult);

// Act

var result = await \_controller.ResetPasswordAsync(new ResetPasswordDTO());

var badRequestResult = result as BadRequestObjectResult;

// Assert

Assert.IsInstanceOfType(result, typeof(BadRequestObjectResult));

Assert.AreEqual(description, badRequestResult!.Value);

\_mockUsersUnitOfWork.Verify(x => x.GetUserAsync(It.IsAny<string>()), Times.Once());

\_mockUsersUnitOfWork.Verify(x => x.ResetPasswordAsync(It.IsAny<User>(), It.IsAny<string>(), It.IsAny<string>()), Times.Once());

}

[TestMethod]

public async Task ChangePasswordAsync\_InvalidModel\_ReturnsBadRequest()

{

// Arrange

\_controller.ModelState.AddModelError("TestError", "Test error message");

// Act

var result = await \_controller.ChangePasswordAsync(new ChangePasswordDTO());

// Assert

Assert.IsInstanceOfType(result, typeof(BadRequestObjectResult));

}

[TestMethod]

public async Task ChangePasswordAsync\_UserNotFound\_ReturnsNotFound()

{

// Arrange

var userName = "testuser";

\_controller.ControllerContext = GetControllerContext(userName);

// Act

var result = await \_controller.ChangePasswordAsync(new ChangePasswordDTO());

// Assert

Assert.IsInstanceOfType(result, typeof(NotFoundResult));

}

[TestMethod]

public async Task ChangePasswordAsync\_ValidChange\_ReturnsNoContent()

{

// Arrange

var userName = "testuser";

var mockUser = new User();

var mockIdentityResult = IdentityResult.Success;

\_controller.ControllerContext = GetControllerContext(userName);

\_mockUsersUnitOfWork.Setup(x => x.GetUserAsync(userName))

.ReturnsAsync(mockUser);

\_mockUsersUnitOfWork.Setup(x => x.ChangePasswordAsync(It.IsAny<User>(), It.IsAny<string>(), It.IsAny<string>()))

.ReturnsAsync(mockIdentityResult);

// Act

var result = await \_controller.ChangePasswordAsync(new ChangePasswordDTO());

// Assert

Assert.IsInstanceOfType(result, typeof(NoContentResult));

\_mockUsersUnitOfWork.Verify(x => x.GetUserAsync(userName), Times.Once());

\_mockUsersUnitOfWork.Verify(x => x.ChangePasswordAsync(It.IsAny<User>(), It.IsAny<string>(), It.IsAny<string>()), Times.Once());

}

[TestMethod]

public async Task ResedToken\_UserNotFound\_ReturnsNotFound()

{

// Arrange

var emailModel = new EmailDTO { Email = "test@example.com" };

// Act

var result = await \_controller.ResedTokenAsync(emailModel);

// Assert

Assert.IsInstanceOfType(result, typeof(NotFoundResult));

\_mockUsersUnitOfWork.Verify(x => x.GetUserAsync(emailModel.Email), Times.Once());

}

[TestMethod]

public async Task ResedToken\_EmailSentSuccessfully\_ReturnsNoContent()

{

// Arrange

var emailModel = new EmailDTO

{

Email = "test@example.com"

};

var user = new User();

\_mockUsersUnitOfWork.Setup(x => x.GetUserAsync(emailModel.Email))

.ReturnsAsync(user);

\_mockUsersUnitOfWork.Setup(x => x.GenerateEmailConfirmationTokenAsync(user))

.ReturnsAsync("GeneratedToken");

var response = new ActionResponse<string> { WasSuccess = true };

\_mockMailHelper.Setup(x => x.SendMail(It.IsAny<string>(), It.IsAny<string>(), It.IsAny<string>(), It.IsAny<string>()))

.Returns(response);

// Act

var result = await \_controller.ResedTokenAsync(emailModel);

// Assert

Assert.IsInstanceOfType(result, typeof(NoContentResult));

\_mockUsersUnitOfWork.Verify(x => x.GetUserAsync(emailModel.Email), Times.Once());

\_mockUsersUnitOfWork.Verify(x => x.GenerateEmailConfirmationTokenAsync(user), Times.Once());

\_mockMailHelper.Verify(x => x.SendMail(It.IsAny<string>(), It.IsAny<string>(), It.IsAny<string>(), It.IsAny<string>()), Times.Once());

}

[TestMethod]

public async Task ResedToken\_EmailFailedToSend\_ReturnsBadRequest()

{

// Arrange

var emailModel = new EmailDTO

{

Email = "test@example.com"

};

var user = new User();

\_mockUsersUnitOfWork.Setup(x => x.GetUserAsync(emailModel.Email))

.ReturnsAsync(user);

\_mockUsersUnitOfWork.Setup(x => x.GenerateEmailConfirmationTokenAsync(user))

.ReturnsAsync("GeneratedToken");

var response = new ActionResponse<string> { WasSuccess = false, Message = "Email sending failed" };

\_mockMailHelper.Setup(x => x.SendMail(It.IsAny<string>(), It.IsAny<string>(), It.IsAny<string>(), It.IsAny<string>()))

.Returns(response);

// Act

var result = await \_controller.ResedTokenAsync(emailModel);

// Assert

Assert.IsInstanceOfType(result, typeof(BadRequestObjectResult));

\_mockUsersUnitOfWork.Verify(x => x.GetUserAsync(emailModel.Email), Times.Once());

\_mockUsersUnitOfWork.Verify(x => x.GenerateEmailConfirmationTokenAsync(user), Times.Once());

\_mockMailHelper.Verify(x => x.SendMail(It.IsAny<string>(), It.IsAny<string>(), It.IsAny<string>(), It.IsAny<string>()), Times.Once());

}

[TestMethod]

public async Task ChangePasswordAsync\_InvalidChange\_ReturnsBadRequest()

{

// Arrange

var userName = "testuser";

var description = "Test error";

var mockUser = new User();

var mockIdentityErrors = new List<IdentityError>

{

new IdentityError { Description = description }

};

var mockIdentityResult = IdentityResult.Failed(mockIdentityErrors.ToArray());

\_controller.ControllerContext = GetControllerContext(userName);

\_mockUsersUnitOfWork.Setup(x => x.GetUserAsync(userName))

.ReturnsAsync(mockUser);

\_mockUsersUnitOfWork.Setup(x => x.ChangePasswordAsync(It.IsAny<User>(), It.IsAny<string>(), It.IsAny<string>()))

.ReturnsAsync(mockIdentityResult);

// Act

var result = await \_controller.ChangePasswordAsync(new ChangePasswordDTO());

var badRequestResult = result as BadRequestObjectResult;

// Assert

Assert.IsInstanceOfType(result, typeof(BadRequestObjectResult));

Assert.AreEqual(description, badRequestResult!.Value);

\_mockUsersUnitOfWork.Verify(x => x.GetUserAsync(userName), Times.Once());

\_mockUsersUnitOfWork.Verify(x => x.ChangePasswordAsync(It.IsAny<User>(), It.IsAny<string>(), It.IsAny<string>()), Times.Once());

}

private ControllerContext GetControllerContext(string userName)

{

var claims = new[]

{

new Claim(ClaimTypes.Name, userName)

};

var identity = new ClaimsIdentity(claims, "test");

var claimsPrincipal = new ClaimsPrincipal(identity);

var httpContext = new DefaultHttpContext

{

User = claimsPrincipal

};

return new ControllerContext

{

HttpContext = httpContext

};

}

}

}

1. Corra los test y verifique que todo está funcionando correctamente.
2. Verificamos la cobertura del código.
3. Hacemos commit.

#### Unidad de Trabajo

1. Adicione la clase **UsersUnitOfWorkTest**:

using Microsoft.AspNetCore.Identity;

using Moq;

using Orders.Backend.Repositories.Interfaces;

using Orders.Backend.UnitsOfWork.Implementations;

using Orders.Shared.DTOs;

using Orders.Shared.Entities;

using Orders.Shared.Responses;

namespace Orders.Tests.UnitsOfWork

{

[TestClass]

public class UsersUnitOfWorkTest

{

private readonly Mock<IUsersRepository> \_mockUsersRepository = new Mock<IUsersRepository>();

private readonly UsersUnitOfWork \_usersUnitOfWork;

public UsersUnitOfWorkTest()

{

\_usersUnitOfWork = new UsersUnitOfWork(\_mockUsersRepository.Object);

}

[TestMethod]

public async Task AddUserAsync\_ShouldReturnSuccess()

{

// Arrange

var user = new User();

var password = "TestPassword123";

var expectedResult = IdentityResult.Success;

\_mockUsersRepository.Setup(repo => repo.AddUserAsync(user, password))

.ReturnsAsync(expectedResult);

// Act

var result = await \_usersUnitOfWork.AddUserAsync(user, password);

// Assert

Assert.AreEqual(expectedResult, result);

\_mockUsersRepository.Verify(repo => repo.AddUserAsync(user, password), Times.Once);

}

[TestMethod]

public async Task AddUserAsync\_ShouldReturnFailure()

{

// Arrange

var user = new User();

var password = "TestPassword123";

var expectedResult = IdentityResult.Failed(new IdentityError());

\_mockUsersRepository.Setup(repo => repo.AddUserAsync(user, password))

.ReturnsAsync(expectedResult);

// Act

var result = await \_usersUnitOfWork.AddUserAsync(user, password);

// Assert

Assert.AreEqual(expectedResult, result);

\_mockUsersRepository.Verify(repo => repo.AddUserAsync(user, password), Times.Once);

}

[TestMethod]

public async Task AddUserToRoleAsync\_CallsRepositoryMethod()

{

// Arrange

var user = new User();

var roleName = "TestRole";

\_mockUsersRepository.Setup(repo => repo.AddUserToRoleAsync(user, roleName))

.Returns(Task.CompletedTask);

// Act

await \_usersUnitOfWork.AddUserToRoleAsync(user, roleName);

// Assert

\_mockUsersRepository.Verify(repo => repo.AddUserToRoleAsync(user, roleName), Times.Once);

}

[TestMethod]

public async Task CheckRoleAsync\_CallsRepositoryMethod()

{

// Arrange

var roleName = "TestRole";

\_mockUsersRepository.Setup(repo => repo.CheckRoleAsync(roleName))

.Returns(Task.CompletedTask);

// Act

await \_usersUnitOfWork.CheckRoleAsync(roleName);

// Assert

\_mockUsersRepository.Verify(repo => repo.CheckRoleAsync(roleName), Times.Once);

}

[TestMethod]

public async Task GetUserAsync\_ReturnsUser\_WhenUserExists()

{

// Arrange

var email = "test@example.com";

var expectedUser = new User { Email = email };

\_mockUsersRepository.Setup(repo => repo.GetUserAsync(email))

.ReturnsAsync(expectedUser);

// Act

var result = await \_usersUnitOfWork.GetUserAsync(email);

// Assert

Assert.IsNotNull(result);

Assert.AreEqual(expectedUser, result);

\_mockUsersRepository.Verify(repo => repo.GetUserAsync(email), Times.Once);

}

[TestMethod]

public async Task GetUserAsync\_ReturnsNull\_WhenUserDoesNotExist()

{

// Arrange

var email = "nonexistent@example.com";

// Act

var result = await \_usersUnitOfWork.GetUserAsync(email);

// Assert

Assert.IsNull(result);

\_mockUsersRepository.Verify(repo => repo.GetUserAsync(email), Times.Once);

}

[TestMethod]

public async Task GetUserGuidAsync\_ReturnsUser\_WhenUserExists()

{

// Arrange

var userId = Guid.NewGuid();

var expectedUser = new User { Id = userId.ToString() };

\_mockUsersRepository.Setup(repo => repo.GetUserAsync(userId))

.ReturnsAsync(expectedUser);

// Act

var result = await \_usersUnitOfWork.GetUserAsync(userId);

// Assert

Assert.IsNotNull(result);

Assert.AreEqual(expectedUser, result);

\_mockUsersRepository.Verify(repo => repo.GetUserAsync(userId), Times.Once);

}

[TestMethod]

public async Task GetUserGuidAsync\_ReturnsNull\_WhenUserDoesNotExist()

{

// Arrange

var userId = Guid.NewGuid();

// Act

var result = await \_usersUnitOfWork.GetUserAsync(userId);

// Assert

Assert.IsNull(result);

\_mockUsersRepository.Verify(repo => repo.GetUserAsync(userId), Times.Once);

}

[TestMethod]

public async Task ChangePasswordAsync\_ReturnsSuccess\_WhenPasswordChanged()

{

// Arrange

var user = new User();

var currentPassword = "CurrentPassword123";

var newPassword = "NewPassword123";

var expectedResult = IdentityResult.Success;

\_mockUsersRepository.Setup(repo => repo.ChangePasswordAsync(user, currentPassword, newPassword))

.ReturnsAsync(expectedResult);

// Act

var result = await \_usersUnitOfWork.ChangePasswordAsync(user, currentPassword, newPassword);

// Assert

Assert.AreEqual(expectedResult, result);

\_mockUsersRepository.Verify(repo => repo.ChangePasswordAsync(user, currentPassword, newPassword), Times.Once);

}

[TestMethod]

public async Task ChangePasswordAsync\_ReturnsFailure\_WhenPasswordChangeFails()

{

// Arrange

var user = new User();

var currentPassword = "CurrentPassword123";

var newPassword = "NewPassword123";

var expectedResult = IdentityResult.Failed(new IdentityError { Description = "Password change failed." });

\_mockUsersRepository.Setup(repo => repo.ChangePasswordAsync(user, currentPassword, newPassword))

.ReturnsAsync(expectedResult);

// Act

var result = await \_usersUnitOfWork.ChangePasswordAsync(user, currentPassword, newPassword);

// Assert

Assert.AreEqual(expectedResult, result);

\_mockUsersRepository.Verify(repo => repo.ChangePasswordAsync(user, currentPassword, newPassword), Times.Once);

}

[TestMethod]

public async Task UpdateUserAsync\_ReturnsSuccess\_WhenUpdateIsSuccessful()

{

// Arrange

var user = new User();

var expectedResult = IdentityResult.Success;

\_mockUsersRepository.Setup(repo => repo.UpdateUserAsync(user))

.ReturnsAsync(expectedResult);

// Act

var result = await \_usersUnitOfWork.UpdateUserAsync(user);

// Assert

Assert.AreEqual(expectedResult, result);

\_mockUsersRepository.Verify(repo => repo.UpdateUserAsync(user), Times.Once);

}

[TestMethod]

public async Task UpdateUserAsync\_ReturnsFailure\_WhenUpdateFails()

{

// Arrange

var user = new User();

var expectedResult = IdentityResult.Failed(new IdentityError { Description = "Update failed." });

\_mockUsersRepository.Setup(repo => repo.UpdateUserAsync(user))

.ReturnsAsync(expectedResult);

// Act

var result = await \_usersUnitOfWork.UpdateUserAsync(user);

// Assert

Assert.AreEqual(expectedResult, result);

\_mockUsersRepository.Verify(repo => repo.UpdateUserAsync(user), Times.Once);

}

[TestMethod]

public async Task IsUserInRoleAsync\_ReturnsTrue\_WhenUserIsInRole()

{

// Arrange

var user = new User();

var roleName = "TestRole";

\_mockUsersRepository.Setup(repo => repo.IsUserInRoleAsync(user, roleName))

.ReturnsAsync(true);

// Act

var result = await \_usersUnitOfWork.IsUserInRoleAsync(user, roleName);

// Assert

Assert.IsTrue(result);

\_mockUsersRepository.Verify(repo => repo.IsUserInRoleAsync(user, roleName), Times.Once);

}

[TestMethod]

public async Task IsUserInRoleAsync\_ReturnsFalse\_WhenUserIsNotInRole()

{

// Arrange

var user = new User();

var roleName = "TestRole";

\_mockUsersRepository.Setup(repo => repo.IsUserInRoleAsync(user, roleName))

.ReturnsAsync(false);

// Act

var result = await \_usersUnitOfWork.IsUserInRoleAsync(user, roleName);

// Assert

Assert.IsFalse(result);

\_mockUsersRepository.Verify(repo => repo.IsUserInRoleAsync(user, roleName), Times.Once);

}

[TestMethod]

public async Task LoginAsync\_ReturnsSuccess\_WhenCredentialsAreValid()

{

// Arrange

var loginModel = new LoginDTO();

var expectedResult = SignInResult.Success;

\_mockUsersRepository.Setup(repo => repo.LoginAsync(loginModel))

.ReturnsAsync(expectedResult);

// Act

var result = await \_usersUnitOfWork.LoginAsync(loginModel);

// Assert

Assert.AreEqual(expectedResult, result);

\_mockUsersRepository.Verify(repo => repo.LoginAsync(loginModel), Times.Once);

}

[TestMethod]

public async Task LoginAsync\_ReturnsFailed\_WhenCredentialsAreInvalid()

{

// Arrange

var loginModel = new LoginDTO();

var expectedResult = SignInResult.Failed;

\_mockUsersRepository.Setup(repo => repo.LoginAsync(loginModel))

.ReturnsAsync(expectedResult);

// Act

var result = await \_usersUnitOfWork.LoginAsync(loginModel);

// Assert

Assert.AreEqual(expectedResult, result);

\_mockUsersRepository.Verify(repo => repo.LoginAsync(loginModel), Times.Once);

}

[TestMethod]

public async Task LogoutAsync\_CallsRepositoryMethod()

{

// Arrange

\_mockUsersRepository.Setup(repo => repo.LogoutAsync())

.Returns(Task.CompletedTask);

// Act

await \_usersUnitOfWork.LogoutAsync();

// Assert

\_mockUsersRepository.Verify(repo => repo.LogoutAsync(), Times.Once);

}

[TestMethod]

public async Task GenerateEmailConfirmationTokenAsync\_GeneratesTokenForUser()

{

// Arrange

var user = new User();

var expectedToken = "test-token";

\_mockUsersRepository.Setup(repo => repo.GenerateEmailConfirmationTokenAsync(user))

.ReturnsAsync(expectedToken);

// Act

var result = await \_usersUnitOfWork.GenerateEmailConfirmationTokenAsync(user);

// Assert

Assert.AreEqual(expectedToken, result);

\_mockUsersRepository.Verify(repo => repo.GenerateEmailConfirmationTokenAsync(user), Times.Once);

}

[TestMethod]

public async Task ConfirmEmailAsync\_ReturnsSuccess\_WhenEmailConfirmationIsSuccessful()

{

// Arrange

var user = new User();

var token = "confirmation-token";

var expectedResult = IdentityResult.Success;

\_mockUsersRepository.Setup(repo => repo.ConfirmEmailAsync(user, token))

.ReturnsAsync(expectedResult);

// Act

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

// Assert

Assert.AreEqual(expectedResult, result);

\_mockUsersRepository.Verify(repo => repo.ConfirmEmailAsync(user, token), Times.Once);

}

[TestMethod]

public async Task ConfirmEmailAsync\_ReturnsFailure\_WhenEmailConfirmationFails()

{

// Arrange

var user = new User();

var token = "invalid-token";

var expectedResult = IdentityResult.Failed(new IdentityError { Description = "Email confirmation failed." });

\_mockUsersRepository.Setup(repo => repo.ConfirmEmailAsync(user, token))

.ReturnsAsync(expectedResult);

// Act

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

// Assert

Assert.AreEqual(expectedResult, result);

\_mockUsersRepository.Verify(repo => repo.ConfirmEmailAsync(user, token), Times.Once);

}

[TestMethod]

public async Task GeneratePasswordResetTokenAsync\_GeneratesTokenForUser()

{

// Arrange

var user = new User();

var expectedToken = "reset-token";

\_mockUsersRepository.Setup(repo => repo.GeneratePasswordResetTokenAsync(user))

.ReturnsAsync(expectedToken);

// Act

var result = await \_usersUnitOfWork.GeneratePasswordResetTokenAsync(user);

// Assert

Assert.AreEqual(expectedToken, result);

\_mockUsersRepository.Verify(repo => repo.GeneratePasswordResetTokenAsync(user), Times.Once);

}

[TestMethod]

public async Task ResetPasswordAsync\_ReturnsSuccess\_WhenPasswordResetIsSuccessful()

{

// Arrange

var user = new User();

var token = "valid-token";

var newPassword = "NewPassword123";

var expectedResult = IdentityResult.Success;

\_mockUsersRepository.Setup(repo => repo.ResetPasswordAsync(user, token, newPassword))

.ReturnsAsync(expectedResult);

// Act

var result = await \_usersUnitOfWork.ResetPasswordAsync(user, token, newPassword);

// Assert

Assert.AreEqual(expectedResult, result);

\_mockUsersRepository.Verify(repo => repo.ResetPasswordAsync(user, token, newPassword), Times.Once);

}

[TestMethod]

public async Task ResetPasswordAsync\_ReturnsFailure\_WhenPasswordResetFails()

{

// Arrange

var user = new User();

var token = "invalid-token";

var newPassword = "NewPassword123";

var expectedResult = IdentityResult.Failed(new IdentityError { Description = "Password reset failed." });

\_mockUsersRepository.Setup(repo => repo.ResetPasswordAsync(user, token, newPassword))

.ReturnsAsync(expectedResult);

// Act

var result = await \_usersUnitOfWork.ResetPasswordAsync(user, token, newPassword);

// Assert

Assert.AreEqual(expectedResult, result);

\_mockUsersRepository.Verify(repo => repo.ResetPasswordAsync(user, token, newPassword), Times.Once);

}

[TestMethod]

public async Task GetAsync\_WithPagination\_ReturnsUsers()

{

// Arrange

var pagination = new PaginationDTO { Page = 1, RecordsNumber = 10 };

var response = new ActionResponse<IEnumerable<User>> { WasSuccess = true };

\_mockUsersRepository.Setup(repo => repo.GetAsync(pagination))

.ReturnsAsync(response);

// Act

var result = await \_usersUnitOfWork.GetAsync(pagination);

// Assert

Assert.AreEqual(response, result);

\_mockUsersRepository.Verify(repo => repo.GetAsync(pagination), Times.Once);

}

[TestMethod]

public async Task GetTotalPagesAsync\_WithPagination\_ReturnsTotalPages()

{

// Arrange

var pagination = new PaginationDTO { Page = 1, RecordsNumber = 10 };

var response = new ActionResponse<int> { WasSuccess = true, Result = 5 };

\_mockUsersRepository.Setup(repo => repo.GetTotalPagesAsync(pagination))

.ReturnsAsync(response);

// Act

var result = await \_usersUnitOfWork.GetTotalPagesAsync(pagination);

// Assert

Assert.AreEqual(response, result);

\_mockUsersRepository.Verify(repo => repo.GetTotalPagesAsync(pagination), Times.Once);

}

}

}

1. Corra los test y verifique que todo está funcionando correctamente.
2. Verificamos la cobertura del código.
3. Hacemos commit.

#### Repositorio

1. Adicione la clase **UsersRepositoryTest**:

using Microsoft.AspNetCore.Authentication;

using Microsoft.AspNetCore.Http;

using Microsoft.AspNetCore.Identity;

using Microsoft.EntityFrameworkCore;

using Microsoft.Extensions.Logging;

using Microsoft.Extensions.Options;

using Moq;

using Orders.Backend.Data;

using Orders.Backend.Repositories.Implementations;

using Orders.Shared.DTOs;

using Orders.Shared.Entities;

namespace Orders.Tests.Repositories;

[TestClass]

public class UsersRepositoryTests

{

private DataContext \_context = null!;

private UsersRepository \_usersRepository = null!;

private Mock<UserManager<User>> \_mockUserManager = null!;

private Mock<RoleManager<IdentityRole>> \_mockRoleManager = null!;

private Mock<SignInManager<User>> \_mockSignInManager = null!;

private readonly Guid \_guid = Guid.NewGuid();

[TestInitialize]

public void SetUp()

{

// Initialize the in-memory database

var options = new DbContextOptionsBuilder<DataContext>()

.UseInMemoryDatabase(databaseName: "TestDatabase")

.Options;

\_context = new DataContext(options);

// Mock the UserManager, RoleManager, SignInManager

var userStoreMock = new Mock<IUserStore<User>>();

\_mockUserManager = new Mock<UserManager<User>>(userStoreMock.Object, null, null, null, null, null, null, null, null);

var roleStoreMock = new Mock<IRoleStore<IdentityRole>>();

\_mockRoleManager = new Mock<RoleManager<IdentityRole>>(roleStoreMock.Object, null, null, null, null);

var optionsAccessorMock = new Mock<IOptions<IdentityOptions>>();

var loggerMock = new Mock<ILogger<SignInManager<User>>>();

var authenticationSchemeProviderMock = new Mock<IAuthenticationSchemeProvider>();

var userConfirmationMock = new Mock<IUserConfirmation<User>>();

var httpContextAccessorMock = new Mock<IHttpContextAccessor>();

var claimsFactoryMock = new Mock<IUserClaimsPrincipalFactory<User>>();

\_mockSignInManager = new Mock<SignInManager<User>>(

\_mockUserManager.Object,

httpContextAccessorMock.Object,

claimsFactoryMock.Object,

optionsAccessorMock.Object,

loggerMock.Object,

authenticationSchemeProviderMock.Object,

userConfirmationMock.Object);

\_usersRepository = new UsersRepository(\_context, \_mockUserManager.Object, \_mockRoleManager.Object, \_mockSignInManager.Object);

PopulateDatabase();

}

[TestCleanup]

public void TearDown()

{

\_context.Database.EnsureDeleted();

\_context.Dispose();

}

[TestMethod]

public async Task GetAsync\_WithEmail\_UserExists\_ReturnsUser()

{

// Arrange

var email = "john.doe@example.com";

// Act

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

// Assert

Assert.IsNotNull(user);

Assert.AreEqual("John", user.FirstName);

}

[TestMethod]

public async Task GetAsync\_WithEmail\_UserDoesNotExist\_ReturnsNull()

{

// Arrange

var email = "nonexistent@example.com";

// Act

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

// Assert

Assert.IsNull(user);

}

[TestMethod]

public async Task GetAsync\_WithUserId\_UserExists\_ReturnsUser()

{

// Act

var user = await \_usersRepository.GetUserAsync(\_guid);

// Assert

Assert.IsNotNull(user);

Assert.AreEqual("Jane", user.FirstName);

}

[TestMethod]

public async Task GetAsync\_WithUserId\_UserDoesNotExist\_ReturnsFailure()

{

// Arrange

var userId = Guid.NewGuid();

// Act

var user = await \_usersRepository.GetUserAsync(userId);

// Assert

Assert.IsNull(user);

}

[TestMethod]

public async Task GetAsync\_WithPagination\_ReturnsUsers()

{

// Arrange

var pagination = new PaginationDTO { Page = 1, RecordsNumber = 10, Filter = "J" };

// Act

var result = await \_usersRepository.GetAsync(pagination);

// Assert

Assert.IsTrue(result.WasSuccess);

Assert.IsNotNull(result.Result);

Assert.AreEqual(2, result.Result.Count());

}

[TestMethod]

public async Task GetTotalPagesAsync\_WithPagination\_ReturnsTotalPages()

{

// Arrange

var pagination = new PaginationDTO { Page = 1, RecordsNumber = 1, Filter = "J" };

// Act

var result = await \_usersRepository.GetTotalPagesAsync(pagination);

// Assert

Assert.IsTrue(result.WasSuccess);

Assert.AreEqual(2, result.Result);

}

[TestMethod]

public async Task GetTotalPagesAsync\_WithFilter\_ReturnsFilteredTotalPages()

{

// Arrange

var pagination = new PaginationDTO { Page = 1, RecordsNumber = 10, Filter = "John" };

// Act

var result = await \_usersRepository.GetTotalPagesAsync(pagination);

// Assert

Assert.IsTrue(result.WasSuccess);

Assert.AreEqual(1, result.Result);

}

[TestMethod]

public async Task GeneratePasswordResetTokenAsync\_ReturnsToken()

{

// Arrange

var user = new User();

var expectedToken = "fake-reset-token";

\_mockUserManager.Setup(x => x.GeneratePasswordResetTokenAsync(user))

.ReturnsAsync(expectedToken);

// Act

var result = await \_usersRepository.GeneratePasswordResetTokenAsync(user);

// Assert

Assert.AreEqual(expectedToken, result);

\_mockUserManager.Verify(x => x.GeneratePasswordResetTokenAsync(user), Times.Once());

}

[TestMethod]

public async Task ResetPasswordAsync\_ReturnsIdentityResult()

{

// Arrange

var user = new User();

var token = "valid-token";

var newPassword = "newPassword123!";

var expectedResult = IdentityResult.Success;

\_mockUserManager.Setup(x => x.ResetPasswordAsync(user, token, newPassword))

.ReturnsAsync(expectedResult);

// Act

var result = await \_usersRepository.ResetPasswordAsync(user, token, newPassword);

// Assert

Assert.AreEqual(expectedResult, result);

\_mockUserManager.Verify(x => x.ResetPasswordAsync(user, token, newPassword), Times.Once());

}

[TestMethod]

public async Task GenerateEmailConfirmationTokenAsync\_ReturnsToken()

{

// Arrange

var user = new User();

var expectedToken = "email-confirm-token";

\_mockUserManager.Setup(x => x.GenerateEmailConfirmationTokenAsync(user))

.ReturnsAsync(expectedToken);

// Act

var result = await \_usersRepository.GenerateEmailConfirmationTokenAsync(user);

// Assert

Assert.AreEqual(expectedToken, result);

\_mockUserManager.Verify(x => x.GenerateEmailConfirmationTokenAsync(user), Times.Once());

}

[TestMethod]

public async Task ConfirmEmailAsync\_ReturnsIdentityResult()

{

// Arrange

var user = new User();

var token = "valid-token";

var expectedResult = IdentityResult.Success;

\_mockUserManager.Setup(x => x.ConfirmEmailAsync(user, token))

.ReturnsAsync(expectedResult);

// Act

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

// Assert

Assert.AreEqual(expectedResult, result);

\_mockUserManager.Verify(x => x.ConfirmEmailAsync(user, token), Times.Once());

}

[TestMethod]

public async Task AddUserAsync\_ReturnsIdentityResult()

{

// Arrange

var user = new User();

var password = "StrongPassword123!";

var expectedResult = IdentityResult.Success;

\_mockUserManager.Setup(x => x.CreateAsync(user, password))

.ReturnsAsync(expectedResult);

// Act

var result = await \_usersRepository.AddUserAsync(user, password);

// Assert

Assert.AreEqual(expectedResult, result);

\_mockUserManager.Verify(x => x.CreateAsync(user, password), Times.Once());

}

[TestMethod]

public async Task AddUserToRoleAsync\_CallsAddToRoleAsync()

{

// Arrange

var user = new User();

var roleName = "Admin";

var expectedResult = IdentityResult.Success;

\_mockUserManager.Setup(x => x.AddToRoleAsync(user, roleName))

.ReturnsAsync(expectedResult);

// Act

await \_usersRepository.AddUserToRoleAsync(user, roleName);

// Assert

\_mockUserManager.Verify(x => x.AddToRoleAsync(user, roleName), Times.Once());

}

[TestMethod]

public async Task ChangePasswordAsync\_ReturnsIdentityResult()

{

// Arrange

var user = new User();

var currentPassword = "CurrentPassword123!";

var newPassword = "NewPassword123!";

var expectedResult = IdentityResult.Success;

\_mockUserManager.Setup(x => x.ChangePasswordAsync(user, currentPassword, newPassword))

.ReturnsAsync(expectedResult);

// Act

var result = await \_usersRepository.ChangePasswordAsync(user, currentPassword, newPassword);

// Assert

Assert.AreEqual(expectedResult, result);

\_mockUserManager.Verify(x => x.ChangePasswordAsync(user, currentPassword, newPassword), Times.Once());

}

[TestMethod]

public async Task CheckRoleAsync\_RoleExists\_DoesNothing()

{

// Arrange

var roleName = "Admin";

\_mockRoleManager.Setup(x => x.RoleExistsAsync(roleName))

.ReturnsAsync(true);

// Act

await \_usersRepository.CheckRoleAsync(roleName);

// Assert

\_mockRoleManager.Verify(x => x.RoleExistsAsync(roleName), Times.Once());

\_mockRoleManager.Verify(x => x.CreateAsync(It.IsAny<IdentityRole>()), Times.Never());

}

[TestMethod]

public async Task CheckRoleAsync\_RoleDoesNotExist\_CreatesRole()

{

// Arrange

var roleName = "Admin";

\_mockRoleManager.Setup(x => x.RoleExistsAsync(roleName))

.ReturnsAsync(false);

\_mockRoleManager.Setup(x => x.CreateAsync(It.IsAny<IdentityRole>()))

.ReturnsAsync(IdentityResult.Success);

// Act

await \_usersRepository.CheckRoleAsync(roleName);

// Assert

\_mockRoleManager.Verify(x => x.RoleExistsAsync(roleName), Times.Once());

\_mockRoleManager.Verify(x => x.CreateAsync(It.Is<IdentityRole>(r => r.Name == roleName)), Times.Once());

}

[TestMethod]

public async Task IsUserInRoleAsync\_UserIsInRole\_ReturnsTrue()

{

// Arrange

var user = new User();

var roleName = "Admin";

\_mockUserManager.Setup(x => x.IsInRoleAsync(user, roleName))

.ReturnsAsync(true);

// Act

var result = await \_usersRepository.IsUserInRoleAsync(user, roleName);

// Assert

Assert.IsTrue(result);

\_mockUserManager.Verify(x => x.IsInRoleAsync(user, roleName), Times.Once());

}

[TestMethod]

public async Task IsUserInRoleAsync\_UserIsNotInRole\_ReturnsFalse()

{

// Arrange

var user = new User();

var roleName = "Admin";

\_mockUserManager.Setup(x => x.IsInRoleAsync(user, roleName)).ReturnsAsync(false);

// Act

var result = await \_usersRepository.IsUserInRoleAsync(user, roleName);

// Assert

Assert.IsFalse(result);

\_mockUserManager.Verify(x => x.IsInRoleAsync(user, roleName), Times.Once());

}

[TestMethod]

public async Task LoginAsync\_ValidCredentials\_ReturnsSignInResultSuccess()

{

// Arrange

var model = new LoginDTO { Email = "user@example.com", Password = "password123" };

\_mockSignInManager.Setup(x => x.PasswordSignInAsync(model.Email, model.Password, false, true))

.ReturnsAsync(SignInResult.Success);

// Act

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

// Assert

Assert.IsTrue(result.Succeeded);

\_mockSignInManager.Verify(x => x.PasswordSignInAsync(model.Email, model.Password, false, true), Times.Once());

}

[TestMethod]

public async Task LoginAsync\_InvalidCredentials\_ReturnsSignInResultFailed()

{

// Arrange

var model = new LoginDTO { Email = "user@example.com", Password = "wrongPassword" };

\_mockSignInManager.Setup(x => x.PasswordSignInAsync(model.Email, model.Password, false, true))

.ReturnsAsync(SignInResult.Failed);

// Act

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

// Assert

Assert.IsFalse(result.Succeeded);

\_mockSignInManager.Verify(x => x.PasswordSignInAsync(model.Email, model.Password, false, true), Times.Once());

}

[TestMethod]

public async Task LogoutAsync\_CallsSignOutAsync()

{

// Arrange

\_mockSignInManager.Setup(x => x.SignOutAsync())

.Returns(Task.CompletedTask);

// Act

await \_usersRepository.LogoutAsync();

// Assert

\_mockSignInManager.Verify(x => x.SignOutAsync(), Times.Once());

}

[TestMethod]

public async Task UpdateUserAsync\_UserUpdated\_ReturnsIdentityResultSuccess()

{

// Arrange

var user = new User();

var expectedResult = IdentityResult.Success;

\_mockUserManager.Setup(x => x.UpdateAsync(user))

.ReturnsAsync(expectedResult);

// Act

var result = await \_usersRepository.UpdateUserAsync(user);

// Assert

Assert.AreEqual(expectedResult, result);

\_mockUserManager.Verify(x => x.UpdateAsync(user), Times.Once());

}

[TestMethod]

public async Task UpdateUserAsync\_UserUpdateFailed\_ReturnsIdentityResultFailed()

{

// Arrange

var user = new User();

var expectedResult = IdentityResult.Failed();

\_mockUserManager.Setup(x => x.UpdateAsync(user)).ReturnsAsync(expectedResult);

// Act

var result = await \_usersRepository.UpdateUserAsync(user);

// Assert

Assert.AreEqual(expectedResult, result);

\_mockUserManager.Verify(x => x.UpdateAsync(user), Times.Once());

}

private void PopulateDatabase()

{

var country = new Country

{

Name = "Country",

States = new List<State>

{

new State

{

Name = "State",

Cities = new List<City>

{

new City { Name = "City" }

}

}

}

};

\_context.Countries.Add(country);

\_context.SaveChanges();

var user1 = new User { Id = "1", FirstName = "John", LastName = "Doe", Email = "john.doe@example.com", Address = "Some", Document = "Any", CityId = 1 };

var user2 = new User { Id = \_guid.ToString(), FirstName = "Jane", LastName = "Doe", Email = "jane.doe@example.com", Address = "Some", Document = "Any", CityId = 1 };

\_context.Users.AddRange(user1, user2);

\_context.SaveChanges();

}

}

1. Corra los test y verifique que todo está funcionando correctamente.
2. Verificamos la cobertura del código.
3. Hacemos commit.

### Helpers

#### OrdersHelperTest

1. Adicione la clase **OrdersHelperTests**:

using Moq;

using Orders.Backend.Helpers;

using Orders.Backend.UnitsOfWork.Interfaces;

using Orders.Shared.Entities;

using Orders.Shared.Responses;

namespace Orders.Tests.Helpers

{

[TestClass]

public class OrdersHelperTests

{

private Mock<IUsersUnitOfWork> \_usersUnitOfWorkMock = null!;

private Mock<ITemporalOrdersUnitOfWork> \_temporalOrdersUoWMock = null!;

private Mock<IProductsUnitOfWork> \_productsUoWMock = null!;

private Mock<IOrdersUnitOfWork> \_ordersUoWMock = null!;

private OrdersHelper \_ordersHelper = null!;

[TestInitialize]

public void Initialize()

{

\_usersUnitOfWorkMock = new Mock<IUsersUnitOfWork>();

\_temporalOrdersUoWMock = new Mock<ITemporalOrdersUnitOfWork>();

\_productsUoWMock = new Mock<IProductsUnitOfWork>();

\_ordersUoWMock = new Mock<IOrdersUnitOfWork>();

\_ordersHelper = new OrdersHelper(\_usersUnitOfWorkMock.Object, \_temporalOrdersUoWMock.Object, \_productsUoWMock.Object, \_ordersUoWMock.Object);

}

[TestMethod]

public async Task ProcessOrderAsync\_UserDoesNotExist\_ReturnsFalseActionResponse()

{

// Arrange

string email = "test@test.com";

// Act

var result = await \_ordersHelper.ProcessOrderAsync(email, "remarks");

// Assert

Assert.IsFalse(result.WasSuccess);

Assert.AreEqual("Usuario no válido", result.Message);

}

[TestMethod]

public async Task ProcessOrderAsync\_TemporalOrdersNotFound\_ReturnsFalseActionResponse()

{

// Arrange

string email = "test@test.com";

var user = new User { Email = email };

\_usersUnitOfWorkMock.Setup(uh => uh.GetUserAsync(email)).ReturnsAsync(user);

\_temporalOrdersUoWMock.Setup(touw => touw.GetAsync(email))

.ReturnsAsync(new ActionResponse<IEnumerable<TemporalOrder>> { WasSuccess = false });

// Act

var result = await \_ordersHelper.ProcessOrderAsync(email, "remarks");

// Assert

Assert.IsFalse(result.WasSuccess);

Assert.AreEqual("No hay detalle en la orden", result.Message);

}

[TestMethod]

public async Task ProcessOrderAsync\_InventoryCheckFails\_ReturnsFalseActionResponse()

{

// Arrange

string email = "test@test.com";

var user = new User { Email = email };

var temporalOrders = new List<TemporalOrder>

{

new TemporalOrder { Quantity = 5, Product = new Product { Id = 1, Name = "Product1", Stock = 3 } }

};

\_usersUnitOfWorkMock.Setup(uh => uh.GetUserAsync(email)).ReturnsAsync(user);

\_temporalOrdersUoWMock.Setup(touw => touw.GetAsync(email))

.ReturnsAsync(new ActionResponse<IEnumerable<TemporalOrder>> { WasSuccess = true, Result = temporalOrders });

\_productsUoWMock.Setup(puw => puw.GetAsync(It.IsAny<int>()))

.ReturnsAsync(new ActionResponse<Product> { WasSuccess = true, Result = temporalOrders[0].Product });

// Act

var result = await \_ordersHelper.ProcessOrderAsync(email, "remarks");

// Assert

Assert.IsFalse(result.WasSuccess);

Assert.AreEqual($"Lo sentimos no tenemos existencias suficientes del producto {temporalOrders[0].Product!.Name}, para tomar su pedido. Por favor disminuir la cantidad o sustituirlo por otro.", result.Message);

}

[TestMethod]

public async Task ProcessOrderAsync\_HappyPath\_ReturnsTrueActionResponse()

{

// Arrange

string email = "test@test.com";

var user = new User { Email = email };

var temporalOrders = new List<TemporalOrder>

{

new TemporalOrder { Quantity = 2, Product = new Product { Id = 1, Name = "Product1", Stock = 5 }, Remarks = "Remarks1", Id = 1 }

};

\_usersUnitOfWorkMock.Setup(uh => uh.GetUserAsync(email))

.ReturnsAsync(user);

\_temporalOrdersUoWMock.Setup(touw => touw.GetAsync(email))

.ReturnsAsync(new ActionResponse<IEnumerable<TemporalOrder>> { WasSuccess = true, Result = temporalOrders });

\_productsUoWMock.Setup(puw => puw.GetAsync(It.IsAny<int>()))

.ReturnsAsync(new ActionResponse<Product> { WasSuccess = true, Result = temporalOrders[0].Product });

\_temporalOrdersUoWMock.Setup(touw => touw.DeleteAsync(It.IsAny<int>()))

.ReturnsAsync(new ActionResponse<TemporalOrder> { WasSuccess = true });

\_productsUoWMock.Setup(puw => puw.UpdateAsync(It.IsAny<Product>()))

.ReturnsAsync(new ActionResponse<Product> { WasSuccess = true });

\_ordersUoWMock.Setup(ouw => ouw.AddAsync(It.IsAny<Order>()))

.ReturnsAsync(new ActionResponse<Order> { WasSuccess = true });

// Act

var result = await \_ordersHelper.ProcessOrderAsync(email, "remarks");

// Assert

Assert.IsTrue(result.WasSuccess);

\_productsUoWMock.Verify(puw => puw.UpdateAsync(It.Is<Product>(p => p.Stock == 3)), Times.Once);

\_temporalOrdersUoWMock.Verify(touw => touw.DeleteAsync(1), Times.Once);

\_ordersUoWMock.Verify(ouw => ouw.AddAsync(It.Is<Order>(o => o.Remarks == "remarks")), Times.Once);

}

[TestMethod]

public async Task ProcessOrderAsync\_ProductNoAvailabe\_ReturnsError()

{

// Arrange

string email = "test@test.com";

var user = new User { Email = email };

var temporalOrders = new List<TemporalOrder>

{

new TemporalOrder { Quantity = 2, Product = new Product { Id = 1, Name = "Product1", Stock = 5 }, Remarks = "Remarks1", Id = 1 }

};

\_usersUnitOfWorkMock.Setup(uh => uh.GetUserAsync(email))

.ReturnsAsync(user);

\_temporalOrdersUoWMock.Setup(touw => touw.GetAsync(email))

.ReturnsAsync(new ActionResponse<IEnumerable<TemporalOrder>> { WasSuccess = true, Result = temporalOrders });

\_productsUoWMock.Setup(puw => puw.GetAsync(It.IsAny<int>()))

.ReturnsAsync(new ActionResponse<Product> { WasSuccess = false });

\_temporalOrdersUoWMock.Setup(touw => touw.DeleteAsync(It.IsAny<int>()))

.ReturnsAsync(new ActionResponse<TemporalOrder> { WasSuccess = true });

\_productsUoWMock.Setup(puw => puw.UpdateAsync(It.IsAny<Product>()))

.ReturnsAsync(new ActionResponse<Product> { WasSuccess = true });

\_ordersUoWMock.Setup(ouw => ouw.AddAsync(It.IsAny<Order>()))

.ReturnsAsync(new ActionResponse<Order> { WasSuccess = true });

// Act

var result = await \_ordersHelper.ProcessOrderAsync(email, "remarks");

// Assert

Assert.IsFalse(result.WasSuccess);

}

[TestMethod]

public async Task ProcessOrderAsync\_ProductNoAvailabeTwo\_ReturnsError()

{

// Arrange

string email = "test@test.com";

var user = new User { Email = email };

var temporalOrders = new List<TemporalOrder>

{

new TemporalOrder { Quantity = 2, Product = new Product { Id = 1, Name = "Product1", Stock = 5 }, Remarks = "Remarks1", Id = 1 }

};

\_usersUnitOfWorkMock.Setup(uh => uh.GetUserAsync(email))

.ReturnsAsync(user);

\_temporalOrdersUoWMock.Setup(touw => touw.GetAsync(email))

.ReturnsAsync(new ActionResponse<IEnumerable<TemporalOrder>> { WasSuccess = true, Result = temporalOrders });

\_productsUoWMock.Setup(puw => puw.GetAsync(It.IsAny<int>()))

.ReturnsAsync(new ActionResponse<Product> { WasSuccess = true });

\_temporalOrdersUoWMock.Setup(touw => touw.DeleteAsync(It.IsAny<int>()))

.ReturnsAsync(new ActionResponse<TemporalOrder> { WasSuccess = true });

\_productsUoWMock.Setup(puw => puw.UpdateAsync(It.IsAny<Product>()))

.ReturnsAsync(new ActionResponse<Product> { WasSuccess = true });

\_ordersUoWMock.Setup(ouw => ouw.AddAsync(It.IsAny<Order>()))

.ReturnsAsync(new ActionResponse<Order> { WasSuccess = true });

// Act

var result = await \_ordersHelper.ProcessOrderAsync(email, "remarks");

// Assert

Assert.IsFalse(result.WasSuccess);

}

}

}

1. Corra los test y verifique que todo está funcionando correctamente.
2. Verificamos la cobertura del código.
3. Hacemos commit.

#### MailHelperTest

1. Adicionamos el **ISmtpClient**:

using MimeKit;

namespace Orders.Backend.Helpers

{

public interface ISmtpClient

{

void Connect(string host, int port, bool useSsl);

void Authenticate(string username, string password);

void Send(MimeMessage message);

void Disconnect(bool quit);

}

}

1. Adicione la clase **SmtpClientWrapper**:

using MailKit.Net.Smtp;

using MimeKit;

namespace Orders.Backend.Helpers

{

public class SmtpClientWrapper : ISmtpClient

{

private readonly SmtpClient \_smtpClient = new SmtpClient();

public void Authenticate(string username, string password) => \_smtpClient.Authenticate(username, password);

public void Connect(string host, int port, bool useSsl) => \_smtpClient.Connect(host, port, useSsl);

public void Disconnect(bool quit) => \_smtpClient.Disconnect(quit);

public void Send(MimeMessage message) => \_smtpClient.Send(message);

}

}

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

builder.Services.AddScoped<ISmtpClient, SmtpClientWrapper>();

1. Modificamos el **MailHelper**, primero inyectamos el **ISmtpClient**:

public ActionResponse<string> 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();

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

\_smtpClient.Authenticate(from!, password!);

\_smtpClient.Send(message);

\_smtpClient.Disconnect(true);

return new ActionResponse<string> { WasSuccess = true };

}

catch (Exception ex)

{

return new ActionResponse<string>

{

WasSuccess = false,

Message = ex.Message,

};

}

}

1. Adicione la clase **MailHelperTests**:

using Microsoft.Extensions.Configuration;

using MimeKit;

using Moq;

using Orders.Backend.Helpers;

namespace Orders.Tests.Helpers

{

[TestClass]

public class MailHelperTests

{

private Mock<IConfiguration> \_configurationMock = null!;

private Mock<ISmtpClient> \_smtpClientMock = null!;

private MailHelper \_mailHelper = null!;

[TestInitialize]

public void Initialize()

{

\_configurationMock = new Mock<IConfiguration>();

\_smtpClientMock = new Mock<ISmtpClient>();

\_configurationMock.SetupGet(x => x["Mail:From"]).Returns("From");

\_configurationMock.SetupGet(x => x["Mail:Name"]).Returns("Name");

\_configurationMock.SetupGet(x => x["Mail:Smtp"]).Returns("Smtp");

\_configurationMock.SetupGet(x => x["Mail:Port"]).Returns("123");

\_configurationMock.SetupGet(x => x["Mail:Password"]).Returns("Password");

\_mailHelper = new MailHelper(\_configurationMock.Object, \_smtpClientMock.Object);

}

[TestMethod]

public void SendMail\_ShouldReturnSuccessActionResponse()

{

// Arrange

var toName = "John Doe";

var toEmail = "john.doe@example.com";

var subject = "Test Subject";

var body = "Test Body";

// Act

var response = \_mailHelper.SendMail(toName, toEmail, subject, body);

// Assert

Assert.IsTrue(response.WasSuccess);

\_smtpClientMock.Verify(x => x.Connect(It.IsAny<string>(), It.IsAny<int>(), It.IsAny<bool>()), Times.Once);

\_smtpClientMock.Verify(x => x.Authenticate(It.IsAny<string>(), It.IsAny<string>()), Times.Once);

\_smtpClientMock.Verify(x => x.Send(It.IsAny<MimeMessage>()), Times.Once);

\_smtpClientMock.Verify(x => x.Disconnect(It.IsAny<bool>()), Times.Once);

}

[TestMethod]

public void SendMail\_ShouldReturnErrorActionResponse\_WhenExceptionThrown()

{

// Arrange

var toName = "John Doe";

var toEmail = "john.doe@example.com";

var subject = "Test Subject";

var body = "Test Body";

var exceptionMessage = "SMTP error";

\_smtpClientMock.Setup(x => x.Send(It.IsAny<MimeMessage>())).Throws(new Exception(exceptionMessage));

// Act

var response = \_mailHelper.SendMail(toName, toEmail, subject, body);

// Assert

Assert.IsFalse(response.WasSuccess);

Assert.AreEqual(exceptionMessage, response.Message);

}

}

}

1. Corra los test y verifique que todo está funcionando correctamente.
2. Verificamos la cobertura del código.
3. Hacemos commit.

#### FileStorage

1. Adicionamos el **IBlobContainerClient**:

using Azure.Storage.Blobs;

using Azure.Storage.Blobs.Models;

namespace Orders.Backend.Helpers

{

public interface IBlobContainerClient

{

Task<BlobClient> GetBlobClientAsync(string name);

Task CreateIfNotExistsAsync();

Task SetAccessPolicyAsync(PublicAccessType accessType);

}

}

1. Adicionamos el **BlobContainerClientWrapper**:

using Azure.Storage.Blobs;

using Azure.Storage.Blobs.Models;

namespace Orders.Backend.Helpers

{

public class BlobContainerClientWrapper : IBlobContainerClient

{

private readonly BlobContainerClient \_blobContainerClient;

public BlobContainerClientWrapper(string connectionString, string containerName)

{

\_blobContainerClient = new BlobContainerClient(connectionString, containerName);

}

public Task<BlobClient> GetBlobClientAsync(string name) => Task.FromResult(\_blobContainerClient.GetBlobClient(name));

public Task CreateIfNotExistsAsync() => \_blobContainerClient.CreateIfNotExistsAsync();

public Task SetAccessPolicyAsync(PublicAccessType accessType) => \_blobContainerClient.SetAccessPolicyAsync(accessType);

}

}

1. Adicionamos el **IBlobContainerClientFactory**:

namespace Orders.Backend.Helpers

{

public interface IBlobContainerClientFactory

{

IBlobContainerClient CreateBlobContainerClient(string connectionString, string containerName);

}

}

1. Adicionamos el **BlobContainerClientFactory**:

using Azure.Storage.Blobs;

namespace Orders.Backend.Helpers

{

public class BlobContainerClientFactory : IBlobContainerClientFactory

{

public IBlobContainerClient CreateBlobContainerClient(string connectionString, string containerName) => new BlobContainerClientWrapper(connectionString, containerName);

}

}

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

…

builder.Services.AddScoped<ISmtpClient, SmtpClientWrapper>();

builder.Services.AddScoped<IBlobContainerClientFactory, BlobContainerClientFactory>();

…

1. Modificamos el **FileStorage**:

using Azure.Storage.Blobs.Models;

namespace Orders.Backend.Helpers

{

public class FileStorage : IFileStorage

{

private readonly string \_connectionString;

private readonly IBlobContainerClientFactory \_blobContainerClientFactory;

public FileStorage(IConfiguration configuration, IBlobContainerClientFactory blobContainerClientFactory)

{

\_connectionString = configuration["ConnectionStrings:AzureStorage"] ?? throw new InvalidOperationException("Connection string 'AzureStorage' not found.");

\_blobContainerClientFactory = blobContainerClientFactory;

}

public async Task RemoveFileAsync(string path, string containerName)

{

var client = \_blobContainerClientFactory.CreateBlobContainerClient(\_connectionString, containerName);

await client.CreateIfNotExistsAsync();

var fileName = Path.GetFileName(path);

var blob = await client.GetBlobClientAsync(fileName);

await blob.DeleteIfExistsAsync();

}

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

{

var client = \_blobContainerClientFactory.CreateBlobContainerClient(\_connectionString, containerName);

await client.CreateIfNotExistsAsync();

await client.SetAccessPolicyAsync(PublicAccessType.Blob);

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

var blob = await client.GetBlobClientAsync(fileName);

using (var ms = new MemoryStream(content))

{

await blob.UploadAsync(ms);

}

return blob.Uri.ToString();

}

}

}

1. Adicione la clase **FileStorageTests**:

using Azure;

using Azure.Storage.Blobs;

using Azure.Storage.Blobs.Models;

using Microsoft.Extensions.Configuration;

using Moq;

using Orders.Backend.Helpers;

namespace Orders.Tests.Helpers

{

[TestClass]

public class FileStorageTests

{

[TestMethod]

public async Task TestRemoveFileAsync()

{

// Arrange

var configurationMock = new Mock<IConfiguration>();

configurationMock.Setup(x => x["ConnectionStrings:AzureStorage"])

.Returns("fake\_connection\_string");

var blobClientMock = new Mock<BlobClient>();

blobClientMock.Setup(x => x.DeleteIfExistsAsync(It.IsAny<DeleteSnapshotsOption>(), It.IsAny<BlobRequestConditions>(), It.IsAny<CancellationToken>()))

.ReturnsAsync(Response.FromValue(true, Mock.Of<Response>()));

var blobContainerClientMock = new Mock<IBlobContainerClient>();

blobContainerClientMock.Setup(x => x.GetBlobClientAsync(It.IsAny<string>()))

.ReturnsAsync(blobClientMock.Object);

blobContainerClientMock.Setup(x => x.CreateIfNotExistsAsync())

.Returns(Task.CompletedTask);

var blobContainerClientFactoryMock = new Mock<IBlobContainerClientFactory>();

blobContainerClientFactoryMock.Setup(x => x.CreateBlobContainerClient(It.IsAny<string>(), It.IsAny<string>()))

.Returns(blobContainerClientMock.Object);

var fileStorage = new FileStorage(configurationMock.Object, blobContainerClientFactoryMock.Object);

// Act

await fileStorage.RemoveFileAsync("fake\_path", "fake\_container");

// Assert

blobClientMock.Verify(x => x.DeleteIfExistsAsync(It.IsAny<DeleteSnapshotsOption>(), It.IsAny<BlobRequestConditions>(), It.IsAny<CancellationToken>()), Times.Once);

}

[TestMethod]

public async Task TestSaveFileAsync\_Success()

{

// Arrange

var configurationMock = new Mock<IConfiguration>();

configurationMock.Setup(x => x["ConnectionStrings:AzureStorage"])

.Returns("fake\_connection\_string");

var blobClientMock = new Mock<BlobClient>();

var blobContentInfoMock = new Mock<BlobContentInfo>();

var responseMock = new Mock<Response<BlobContentInfo>>();

responseMock.Setup(x => x.Value)

.Returns(blobContentInfoMock.Object);

blobClientMock.Setup(x => x.UploadAsync(It.IsAny<Stream>(), true, default))

.ReturnsAsync(responseMock.Object);

blobClientMock.SetupGet(x => x.Uri)

.Returns(new Uri("http://fake.blob.url"));

var blobContainerClientMock = new Mock<IBlobContainerClient>();

blobContainerClientMock.Setup(x => x.GetBlobClientAsync(It.IsAny<string>()))

.ReturnsAsync(blobClientMock.Object);

blobContainerClientMock.Setup(x => x.CreateIfNotExistsAsync())

.Returns(Task.CompletedTask);

blobContainerClientMock.Setup(x => x.SetAccessPolicyAsync(PublicAccessType.Blob))

.Returns(Task.CompletedTask);

var blobContainerClientFactoryMock = new Mock<IBlobContainerClientFactory>();

blobContainerClientFactoryMock.Setup(x => x.CreateBlobContainerClient(It.IsAny<string>(), It.IsAny<string>()))

.Returns(blobContainerClientMock.Object);

var fileStorage = new FileStorage(configurationMock.Object, blobContainerClientFactoryMock.Object);

// Act

var result = await fileStorage.SaveFileAsync(new byte[] { }, ".txt", "fake\_container");

// Assert

Assert.AreEqual("http://fake.blob.url/", result);

}

}

}

1. Corra los test y verifique que todo está funcionando correctamente.
2. Verificamos la cobertura del código.
3. Hacemos commit.

### Otros

#### SeedDb

1. Creamos el **IRuntimeInformationWrapper**:

using System.Runtime.InteropServices;

namespace Orders.Backend.Helpers

{

public interface IRuntimeInformationWrapper

{

bool IsOSPlatform(OSPlatform osPlatform);

}

}

1. Creamos el **RuntimeInformationWrapper**:

using System.Runtime.InteropServices;

namespace Orders.Backend.Helpers

{

public class RuntimeInformationWrapper : IRuntimeInformationWrapper

{

public bool IsOSPlatform(OSPlatform osPlatform) => RuntimeInformation.IsOSPlatform(osPlatform);

}

}

1. Configuramos la nueva inyección en el **backend**:

…

builder.Services.AddScoped<IBlobContainerClientFactory, BlobContainerClientFactory>();

builder.Services.AddScoped<IRuntimeInformationWrapper, RuntimeInformationWrapper>();

…

1. Modificamos el **SeedDb** para que use la nueva inyección:

if (\_runtimeInformationWrapper.IsOSPlatform(OSPlatform.Windows))

1. Adicionamos la clase **SeedDbTests**:

using System.Runtime.InteropServices;

using Microsoft.EntityFrameworkCore;

using Moq;

using Orders.Backend.Data;

using Orders.Backend.Helpers;

using Orders.Backend.Services;

using Orders.Shared.Responses;

namespace Orders.Tests.Others

{

[TestClass]

public class SeedDbTests

{

private SeedDb \_seedDb = null!;

private Mock<IApiService> \_apiServiceMock = null!;

private Mock<IUserHelper> \_userHelperMock = null!;

private Mock<IFileStorage> \_fileStorageMock = null!;

private Mock<IRuntimeInformationWrapper> \_runtimeInformationMock = null!;

private DataContext \_context = null!;

[TestInitialize]

public void Initialize()

{

var options = new DbContextOptionsBuilder<DataContext>()

.UseInMemoryDatabase(databaseName: "OrdersDbTest")

.Options;

\_context = new DataContext(options);

\_apiServiceMock = new Mock<IApiService>();

\_userHelperMock = new Mock<IUserHelper>();

\_fileStorageMock = new Mock<IFileStorage>();

\_runtimeInformationMock = new Mock<IRuntimeInformationWrapper>();

\_seedDb = new SeedDb(\_context, \_apiServiceMock.Object, \_userHelperMock.Object, \_fileStorageMock.Object, \_runtimeInformationMock.Object);

}

[TestMethod]

public async Task SeedAsync\_WithNoAPiCountriesActionResponseAndWindowsOS\_ShouldSeedData()

{

// Arrange

\_runtimeInformationMock.Setup(r => r.IsOSPlatform(OSPlatform.Windows))

.Returns(true);

\_fileStorageMock.Setup(x => x.SaveFileAsync(It.IsAny<byte[]>(), It.IsAny<string>(), It.IsAny<string>()))

.ReturnsAsync("imageUrl");

\_apiServiceMock.Setup(x => x.GetAsync<List<CountryResponse>>(It.IsAny<string>(), It.IsAny<string>()))

.ReturnsAsync(new ActionResponse<List<CountryResponse>> { WasSuccess = false });

// Act

await \_seedDb.SeedAsync();

// Assert

Assert.IsTrue(await \_context.Countries.AnyAsync());

Assert.IsTrue(await \_context.Categories.AnyAsync());

Assert.IsTrue(await \_context.Products.AnyAsync());

Assert.IsTrue(await \_context.ProductCategories.AnyAsync());

Assert.IsTrue(await \_context.ProductImages.AnyAsync());

}

[TestMethod]

public async Task SeedAsync\_WithAPiCountriesActionResponseAndWindowsOS\_ShouldSeedData()

{

// Arrange

\_runtimeInformationMock.Setup(r => r.IsOSPlatform(OSPlatform.Windows))

.Returns(false);

\_fileStorageMock.Setup(x => x.SaveFileAsync(It.IsAny<byte[]>(), It.IsAny<string>(), It.IsAny<string>()))

.ReturnsAsync("imageUrl");

var CountryResponse = new ActionResponse<List<CountryResponse>>

{

WasSuccess = true,

Result = new List<CountryResponse>

{

new CountryResponse { Id = 1, Name = "Some", Iso2 = "SO" }

}

};

\_apiServiceMock.Setup(x => x.GetAsync<List<CountryResponse>>(It.IsAny<string>(), It.IsAny<string>()))

.ReturnsAsync(CountryResponse);

var StateResponse = new ActionResponse<List<StateResponse>>

{

WasSuccess = true,

Result = new List<StateResponse>

{

new StateResponse { Id = 1, Name = "Some", Iso2 = "SO" }

}

};

\_apiServiceMock.Setup(x => x.GetAsync<List<StateResponse>>(It.IsAny<string>(), It.IsAny<string>()))

.ReturnsAsync(StateResponse);

var CityResponse = new ActionResponse<List<CityResponse>>

{

WasSuccess = true,

Result = new List<CityResponse>

{

new CityResponse { Id = 1, Name = "Some" },

new CityResponse { Id = 2, Name = "Mosfellsbær" },

new CityResponse { Id = 3, Name = "Șăulița" }

}

};

\_apiServiceMock.Setup(x => x.GetAsync<List<CityResponse>>(It.IsAny<string>(), It.IsAny<string>()))

.ReturnsAsync(CityResponse);

// Act

await \_seedDb.SeedAsync();

// Assert

Assert.IsTrue(await \_context.Countries.AnyAsync());

Assert.IsTrue(await \_context.Categories.AnyAsync());

Assert.IsTrue(await \_context.Products.AnyAsync());

Assert.IsTrue(await \_context.ProductCategories.AnyAsync());

Assert.IsTrue(await \_context.ProductImages.AnyAsync());

}

[TestCleanup]

public void Cleanup()

{

\_context.Database.EnsureDeleted();

\_context.Dispose();

}

}

}

1. Corra los test y verifique que todo está funcionando correctamente.
2. Verificamos la cobertura del código.
3. Hacemos commit.

**Nota general**: para el resto de clases o métodos que no es posible probar, se puede colocar esta anotación:

[ExcludeFromCodeCoverage(Justification = "It is a wrapper used to test other classes. There is no way to prove it.")]

Y de esta forma podemos obtener una medición más real del código realmente cubierto.

## Publicación en Azure

Antes de publicar vamos hacer unos cambios, unos son mejoras sencillas y otros son necesarios para poder publicar con éxito.

1. Cambiemos el TimeOut de la base de datos. Agregamos este par de parámetros al string de conexión de la base de datos:

"ConnectionStrings": {

"LocalConnection": "Server=(localdb)\\MSSQLLocalDB;Database=VendaPues;Trusted\_Connection=True;MultipleActiveResultSets=true;Connection Timeout=600;Command Timeout=600;",

"AzureStorage": "DefaultEndpointsProtocol=https;AccountName=orderszulu2024;AccountKey=aUBtiF7GTURebDNoae/2mn3BxISYUe5GzpldozWo96SI07nPU/M3XUf3JjUdtdlX/nTsJ48/8EkM+AStm/YdLA==;EndpointSuffix=core.windows.net"

},

1. Modificamos el **DataContext**:

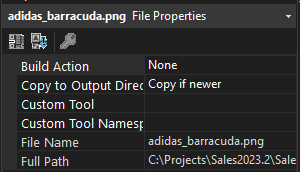
public DataContext(DbContextOptions<DataContext> options) : base(options)

{

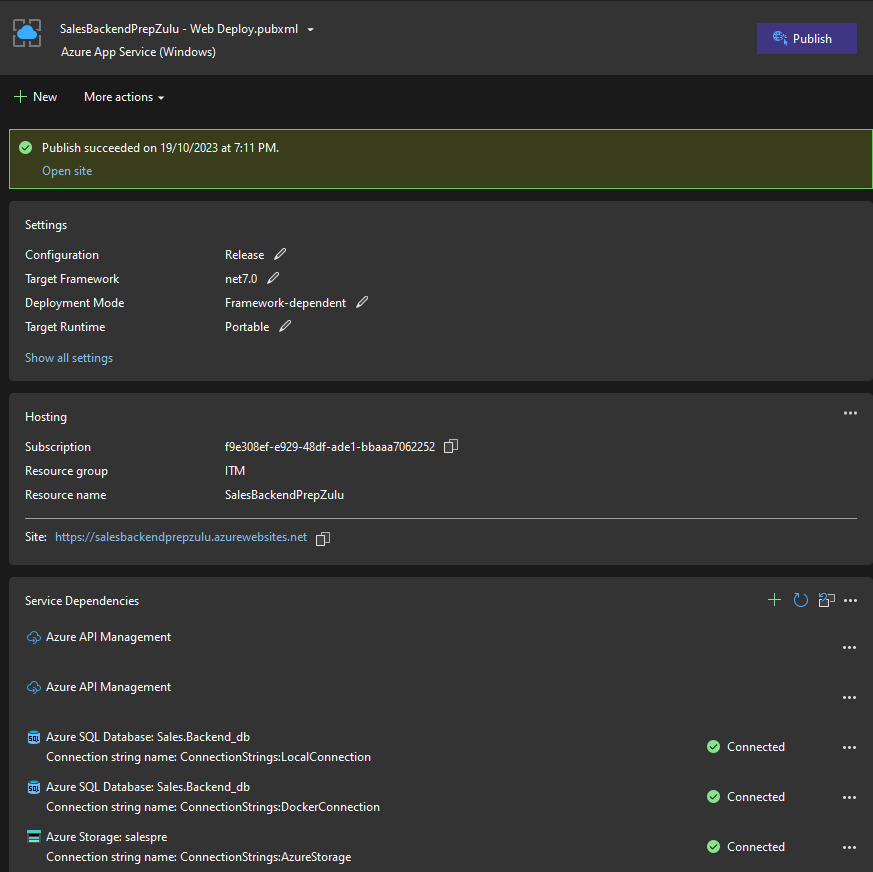
Database.SetCommandTimeout(600);

}

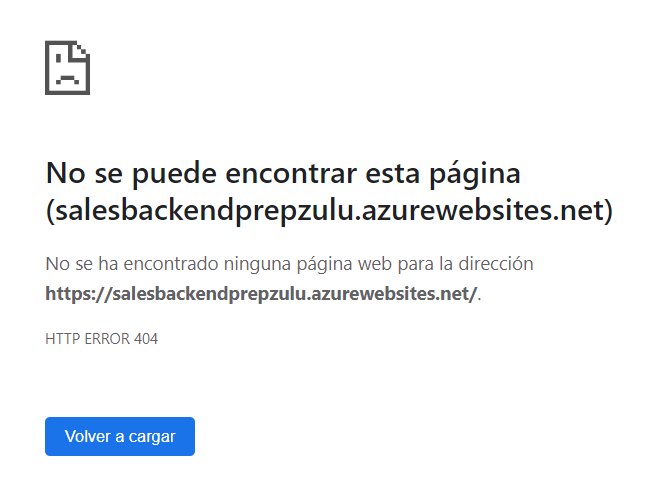
1. Revisemoe las propiedades de las imágenes puestas en el backend como: “**Copy if newer**”.



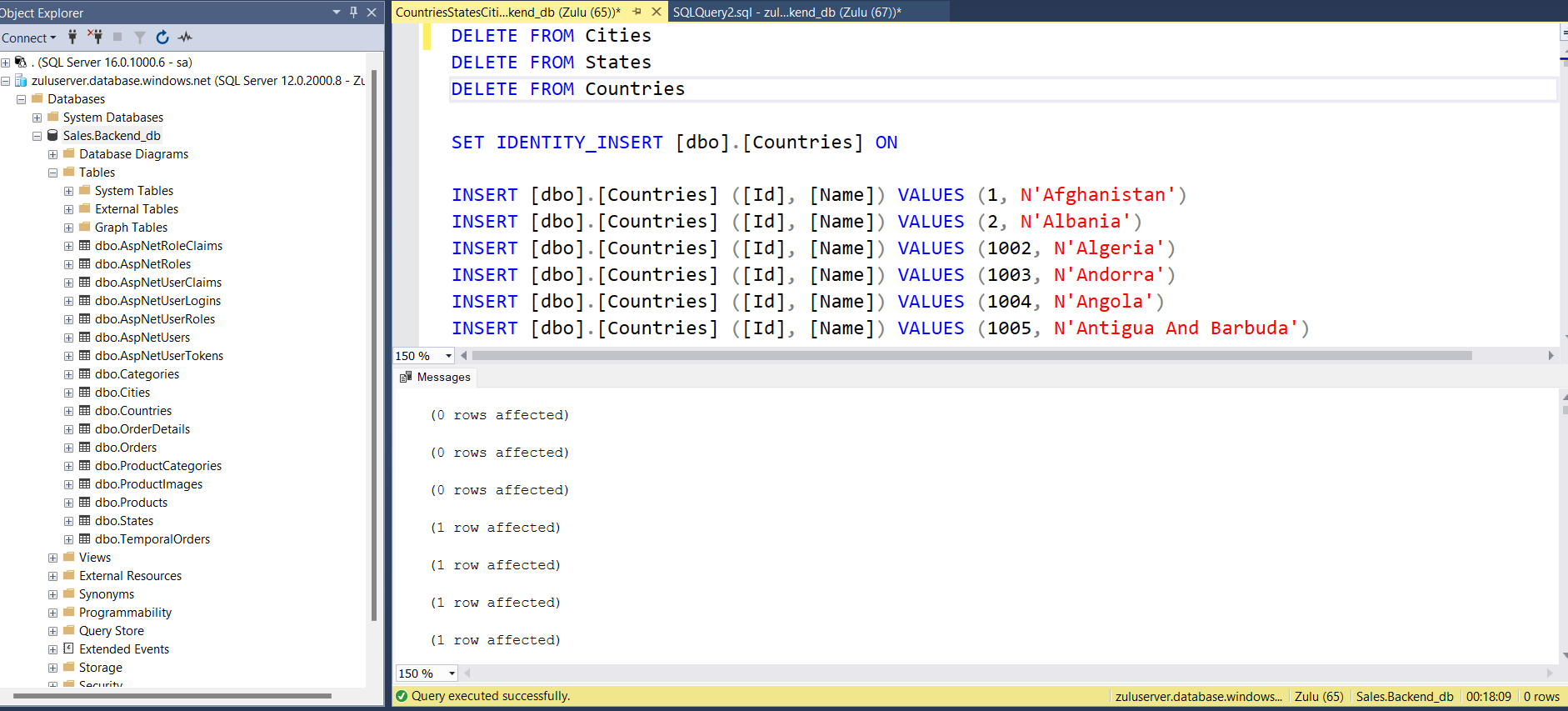
1. Publicar el backend en Azure, ver video para poder configurar todos los pasos correctamente:



1. Si todo estuvo bien te debe salir una pantalla similar a esta:



1. Como cambiamos el Seeder y solo ingresa a la ciudad de Medellín, conectemonos a la base de datos en Azure y corramos el Script que ingresa la mayoría de las ciudades del mundo:



1. Tome la dirección de publicación del Backend (según mi ejemplo es: <https://salesbackendprepzulu.azurewebsites.net>) y modifique el **Program** del Frontend. **Nota**: reemplace las URL por las suyas.

builder.RootComponents.Add<HeadOutlet>("head::after");

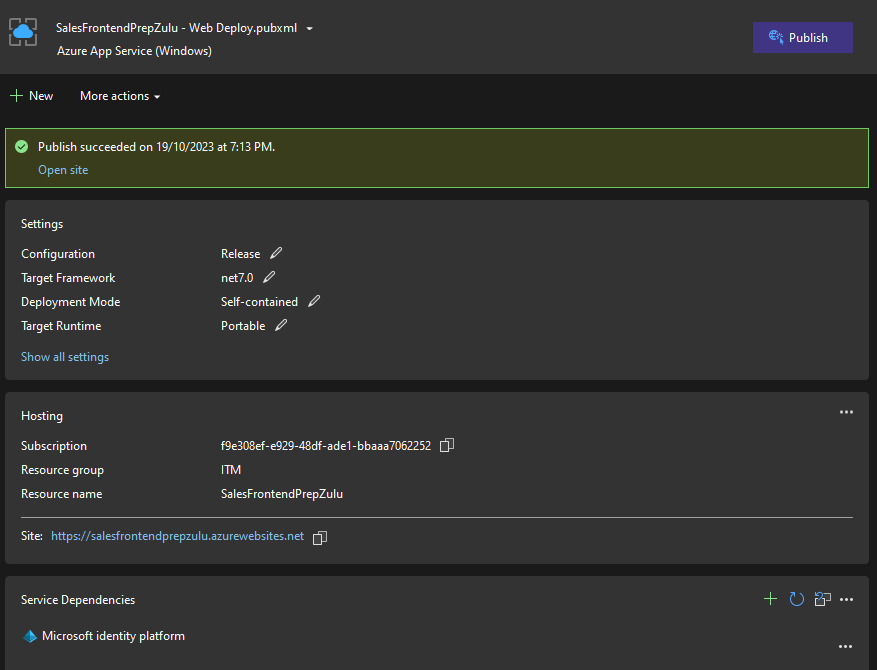
var uriBack = "https://salesbackendprepzulu.azurewebsites.net/";

//var uriBack = "https://localhost:7030/";

builder.Services.AddSingleton(sp => new HttpClient { BaseAddress = new Uri(uriBack) });

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

1. Publicar el frontend en Azure, ver video para poder configurar todos los pasos correctamente:



1. Tome la dirección de publicación del Frontend (según mi ejemplo es: <https://salesfrontendprepzulu.azurewebsites.net>) y modifique el **appsettings** del Backend. **Nota**: reemplace las URL por las suyas.

},

"Url Frontend": "salesfrontendprepzulu.azurewebsites.net",

//"Url Frontend": "localhost:7007",

"AllowedHosts": "\*",

"jwtKey": "sagdsadgfeSDF674545R5690kolsjdkljdDFKLJF!DLKJslkjsEFG$%FEfgdslkjfglkjhfgdkljhdR5454545\_4TGRGtyo!!kjytkljty",

"Mail": {

"From": "{Your gmail account}",

"Name": "Soporte Orders",

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

"Port": 587,

"Password": "{Your password}"

}

1. Publique de nuevo el Backend.
2. Entre al Frontend y verifique que todo esté funcionando correctamente.

## Fin